

Development of Data Acquisition Software for Time-of-Flight Photoelectron-Photoelectron Coincidence Spectrometers

Isak Bakken

September 23, 2008

Abstract

Within the scope of this thesis work, software has been developed for two time-of-flight multi-electron coincidence spectrometers at AlbaNova University Center in Stockholm. This software consists of a Linux driver for a special time-to-digital converter (TDC) card, and some software for data acquisition – a server and a client.

The server functions as a middleman between the driver and the client, and enables data acquisition from the spectrometers via TCP/IP (over a network). The software can be used to acquire multiple types of coincidence spectra. The most important details for using the software are documented in this report.

The developed data acquisition software and TDC card driver were used to record time-of-flight photoelectron spectra of O_2^+ , and time-of-flight photoelectron-photoelectron coincidence spectra of Xe^{2+} . Studies of these spectra revealed ionic states in accord with what is known in the literature, and indicate that the software developed works as intended.

Sammanfattning

Inom ramen för detta examensarbete har mjukvara utvecklats, för två flygtids-multi-elektron-koincidens-spektrometrar vid AlbaNova Universitetscentrum i Stockholm. Denna mjukvara består dels av en Linux-drivrutin för ett speciellt tid-till-digital-konverteringskort (TDC-kort), samt av ett par programvaror för datainsamling – en server och en klient.

Servern fungerar som mellanhand mellan drivrutinen och klienten, och möjliggör därmed datainsamling från en spektrometer via TCP/IP (över ett nätverk). Mjukvaran kan användas för att erhålla flera olika typer av koincidensspektra. De viktigaste detaljerna för användande av mjukvaran är dokumenterade i denna rapport.

Den utvecklade datainsamlingsmjukvaran och TDC-kortsdrivrutinen används för att erhålla flygtids-fotoelektron-spektra av O_2^+ , och flygtids-fotoelektron-fotoelektron-koincidens-spektra av Xe^{2+} . Studier av dessa spektra visar jontillstånd i överensstämmelse med vad som hittas i litteratur, och indikerar att den utvecklade mjukvaran fungerar som den ska.

(Svensk titel: *“Utveckling av datainsamlingsmjukvara för flygtids-fotoelektron-fotoelektron-koincidens-spektrometrar”*)

Acknowledgments

I would like to thank the following people for their support or contribution to this thesis: *Raimund Feifel*, who has been a very dedicated and helpful supervisor; *Leif Karlsson*, for his help in finding a suitable thesis project; *Mats Larsson*, for making this project possible; *Kerstin Jon-And*, for being a supportive contact-person; *John H. D. Eland*, for giving good answers to my questions about the TDC card and the coincidence spectrometers; *Christian Bohm*, for providing me with equipment used in pulse studies; *Egil Andersson*, for collaboration and discussions regarding the development of new software, and for his great ideas and solutions to many of the problems. Special thanks go to *Pamela Karlsson* for helping me find a good thesis project and introducing me to Leif Karlsson and Raimund Feifel.

In addition, the following people have been contributing to the result in various ways, such as discussions or practical help with the spectrometers or computer equipment: *Pelle Linusson*, *Tomasz Kloda*, *Tony Hansson*, *Henrik Öström*, *Mohamed Elshakre*, *Alexander Agapow*, and *Roger Tuomenoksa*.

This thesis is dedicated to the memory of *Fabian Österdahl* (tek. lic.), who was a brilliant cooperation partner with a very positive attitude, and good ideas for solutions to the problems. Before he passed away, he was helping me with the electronic part of a related project which was the original focus of this thesis. I would also like to thank *Uwe Kopp* (National Semiconductors), for donating an evaluation board and some integrated circuits for use in that particular project.

Nomenclature

ADC	An Analog-to-Digital Converter (ADC) converts continuous electronic signals to discrete digital numbers.	team of hackers across the Net. It aims towards POSIX and Single UNIX Specification compliance.
BIOS	Basic Input/Output System	
CPU	Central Processing Unit; executes programs in a computer, by fetching machine code instructions and performing the operations, such as calculations and data management.	OpenGL Open Graphics Library (OpenGL) is a standard specification of a application programming interface (API) for producing 2D and 3D computer graphics.
driver	A device driver works like a translating layer between a device and the applications or operating systems that use it. It allows applications to access devices by using high-level commands, and converts them into the low-level commands required by the device.	PCI Peripheral Component Interconnect is a computer bus standard for connecting peripheral devices to a motherboard. It is a 32-bit parallel bus with a maximum bandwidth of 132 MB/s.
FIFO	First-In-First-Out; a buffer where the first item put into the buffer is the first item read out of the buffer.	RAM Random Access Memory
FPGA	A field-programmable gate array (FPGA) is a type of logic chip which can be re-programmed easily to change the functioning of the chip.	Ruby Ruby is a dynamic, reflective, general purpose object-oriented programming language inspired by Perl and Smalltalk.
GNU	The GNU Project was launched in 1984 to develop a complete free Unix-like operating system.	TCP Transmission Control Protocol. The TCP/IP are network protocols on top of which the Internet is largely built.
IP	Internet Protocol	TDC Time-to-Digital Converter; a device that measures the duration between two events (such as two pulses), and converts the time to a digital value which can be read by a computer.
ISA	Industry Standard Architecture is an 8-bit or 16-bit parallel computer bus standard that is no longer available on most computer motherboards, since it has been superseded by PCI. The maximum bandwidth is 8 MB/s.	USB Universal Serial Bus is a serial bus standard which is today very common. It is used primarily for connecting external devices to a computer. The maximum bandwidth is 40 MB/s for USB 2.0.
Linux	Linux is a clone of the operating system Unix, written from scratch by Linus Torvalds with assistance from a loosely-knit	

The description of the terms as well as others not listed here can be found e.g. in an encyclopedia or by using one of the standard search engines on the Internet. Information about the different buses are found in [1].

Contents

1	Introduction	6
1.1	Historical overview	6
1.2	Development of new software	8
2	Photoelectron spectroscopy	10
2.1	The Photoelectric effect	10
2.2	Photoionization	10
2.2.1	Single ionization	10
2.2.2	Double ionization	11
2.3	Time-of-flight spectroscopy	12
3	Experimental set-up	13
3.1	Spectrometer	13
3.1.1	Spectrometer offsets	14
3.2	Electronics	14
3.3	Computer set-up	15
3.3.1	New computer set-up	15
4	Development of new software	17
4.1	Operating systems and device drivers	17
4.2	TDC card device driver	17
4.2.1	Development of the LKM	18
4.2.2	Using the module	19
4.2.2.1	Loading/unloading the TDC module	19
4.2.3	Controlling the TDC module	19
4.2.4	Reading data from the TDC module	19
4.3	Data Acquisition Software	20
4.3.1	Server	20
4.3.2	Client	21
4.3.2.1	Connecting to the server	21
4.3.2.2	Creating a new measurement	21
4.3.2.3	Running a measurement	22
4.3.2.4	The data files	22

5	Testing the DAQ-software and TDC-module by acquiring spectra	25
5.1	Single ionization of O_2 at 21.218 eV	25
5.2	Double photoionization of Xe at 40.814 eV	27
6	Conclusions and discussion	30
6.1	Improvements	30
6.1.1	TDC module	30
6.1.2	Server	30
6.1.3	Client	30
6.1.4	Position sensitivity	31
6.2	Alternatives	31
A	Details of the spectrometer calibration procedure	i
B	Spectral lines of O_2^+ photoelectron spectra	iii
C	TDC8 ISA and alternative timing cards	vi
D	Technical details for the TDC module	vii
D.1	Source code overview	vii
D.2	Module parameters	ix
D.3	The FIFO-buffer	ix
D.4	Scripts for loading / unloading the LKM	ix
D.5	Commanding the TDC module	x
D.5.1	Command list	xi
E	Protocol for Client–Server communication	xiii
E.1	The protocol at present	xiii
F	Analysis of spectra with Matlab	xix
	Bibliography	xxvii

Chapter 1

Introduction

1.1 Historical overview

Spectral analysis, first used in the 1750s by T. Melvill, has become an important tool to study and identify properties of matter by analyzing e.g. the wavelengths of the light emitted by them, since each element has unique spectral lines. As this thesis deals with a spectroscopic method which involves the analysis of electrons, some milestones for the development of this research field will be recalled. A summary of some of the important events in the history of electron spectroscopy is presented in Table 1.1 on page 7.

Experiments were performed in 1887 by H. Hertz, through which he realized the creation of an electric current in metal surfaces that were irradiated with light (electromagnetic radiation). A few years later, P. Lenard showed that this effect occurred only when the light had a wavelength less than a certain threshold value [3]. About 10 years after Hertz's pioneering experiments, J. J. Thomson discovered what is today known as the electron. Based on these findings, A. Einstein was able in 1905 to explain theoretically this photoelectric effect in deriving the *photoelectric law* (which is discussed in Section 2.1), through the radical postulation that quantization is an inherent property of electromagnetic radiation.¹ For this work, Einstein was awarded the Nobel price in Physics in 1921 [4].

In 1910, the idea of an atomistic structure of matter was strengthened by E. Rutherford who dis-

covered the positive charges of the atomic nucleus. Based on that, N. Bohr was able to present in 1913 a model of an atom where electrons are circulating around the nucleus in well-defined orbits [3]. Already Bohr himself knew that his model was limited to the hydrogen atom and hydrogen-like ions, and a more fundamental theory was needed. Such a theory was presented in 1926 by E. Schrödinger: *wave mechanics*,² which treated the electrons as particle waves around the atomic nucleus. As a typical result of the so-called *Schrödinger equation*, the spatial distribution of an atomic p-orbital is shown in Table 1.1 on page 7.

Einstein's photoelectric law is the theoretical basis for *photoelectron spectroscopy* (PES) [6]. Instead of using light as carrier of information about matter, photoelectrons released from the matter are studied. The idea – although simple in principle – turned out to be technically challenging: The first experiments were performed already in the 1920s, but no lines could be revealed in the spectra at the time. K. Siegbahn³ in Uppsala finally succeeded with his group in the 1950s, using X-rays, and their spectral analysis indeed revealed distinct line structure for the first time [3].

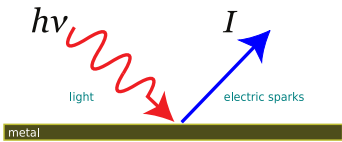

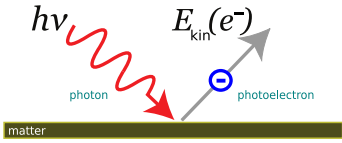

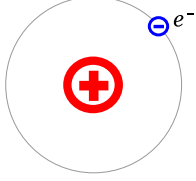
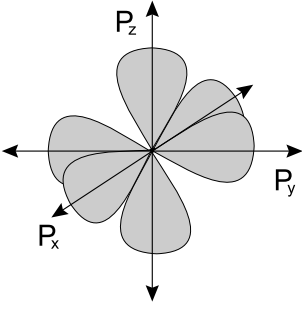
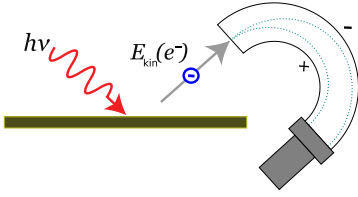
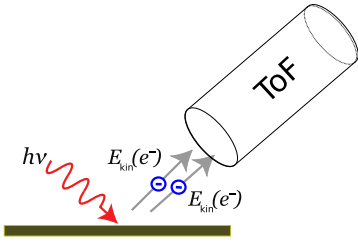
A new field of research started: PES enabled the direct measurements of electron binding energies at high precision and with a sensitivity to the chemical surrounding of the probed atom. It quickly became one of the most important methods

¹The energy quanta are nowadays called *photons*, following the nomenclature of G. N. Lewis in 1926 [3].

²Actually, in 1926 Schrödinger proved that his less abstract theory of wave mechanics was equivalent to Heisenberg's more abstract formulation of *quantum mechanics* or *matrix mechanics*, which had been presented in 1925 [5, pp.161,166].

³He was awarded the Nobel Prize in Physics in 1981 for this work [7, p.159].

Table 1.1: Overview of some important events in the history of photoelectron spectroscopy [2].

Year	Person	Concept	Model
1887	H. Hertz	Photocurrent	 <p>The diagram shows a horizontal metal surface. A red wavy arrow labeled $h\nu$ and 'light' points towards the surface. A blue arrow labeled I and 'electric sparks' points away from the surface.</p>
1897	J. J. Thomson	Electron	 <p>A blue circle with a minus sign inside, representing an electron.</p>
1905	A. Einstein	Photoelectrons	 <p>The diagram shows a horizontal matter surface. A red wavy arrow labeled $h\nu$ and 'photon' points towards the surface. A blue circle with a minus sign and 'photoelectron' is shown being ejected from the surface. A grey arrow labeled $E_{kin}(e^-)$ points away from the photoelectron.</p>
1910	E. Rutherford	Atomic nucleus	 <p>A red circle with a plus sign inside, representing an atomic nucleus.</p>
1913	N. Bohr	Atomic model	 <p>A central red circle with a plus sign is surrounded by a larger grey circle. A blue circle with a minus sign and e^- is positioned on the outer circle.</p>
1926	E. Schrödinger	Wave mechanics	 <p>A 3D coordinate system with axes P_x, P_y, and P_z. A probability distribution is shown as several grey lobes centered at the origin.</p>
1955	K. Siegbahn	Photoelectron spectroscopy	 <p>The diagram shows a horizontal sample surface. A red wavy arrow labeled $h\nu$ points towards the surface. A blue circle with a minus sign and $E_{kin}(e^-)$ is shown being ejected from the surface. A curved detector with '+' and '-' ends is positioned to the right.</p>
2003	J. H. D. Eland	Time-of-flight Photoelectron-coincidence spectroscopy	 <p>The diagram shows a horizontal sample surface. A red wavy arrow labeled $h\nu$ points towards the surface. A blue circle with a minus sign and $E_{kin}(e^-)$ is shown being ejected from the surface. A cylindrical detector labeled 'TOF' is positioned to the right.</p>

for studying fundamental properties of matter and dynamic processes [7, p.159].

It took almost another 50 years until John H. D. Eland at Oxford in 2003 demonstrated a highly sensitive and highly versatile electron spectroscopy technique which allows not only measuring one electron at a time, but the simultaneous detection of several electrons emitted upon single photon absorption. The Oxford technique thereby just opened up the hitherto almost unexplored area of multi-electron emission processes of atoms and molecules [8, 9].

1.2 Development of new software

The spectrometers originally developed in the laboratories of J. H. D. Eland at Oxford, which are now in use at the AlbaNova University Center in Stockholm, make use of certain timing cards – Time-to-Digital Converter (TDC) cards – which allow a precise timing of the electron flight-times. There are two TOF-PEPECO spectrometers in use: One utilizes a pulsed Helium lamp as ionization source, and another one utilizes a femtosecond laser system. The two spectrometers have some differences – e.g. the apparatus combined with the laser system has a longer flight-tube for electrons, which gives higher resolution for measuring the electrons’ flight-times. They both make use of the same TDC card model however.

These TDC cards have so far been used in conjunction with software written by J. H. D. Eland, in the QuickBasic programming language (see Figure 1.1). These programs are technically limited by shortcomings of the operating system⁴ and the programming language itself.⁵ It is also plight

to maintain the programs and enhance them with new functionality: They each have their source code in one individual file per program – there is no re-usage of source code between them. Lots of hardware instructions for the TDC card, which are not easily understood without advanced knowledge of how the TDC card works, make up large parts of the source code. Furthermore, the QuickBasic programming language is not so common anymore,⁶ i.e. only a few people have the knowledge required to modify the existing software.

In order to be able to use the TDC card with a modern operating system, a *device driver* was found to be very useful: Modern operating systems generally do not allow direct communication with the hardware from programs. A device driver is often required, and such a driver for the TDC card allows new software to treat it as a “black box”, with well-defined inputs and outputs, without having to deal with its technicalities. Since no driver existed for the TDC card in Linux, a suitable project for this thesis was to develop such a driver.

In addition to making a device driver, during course of the project, data acquisition (DAQ) software with a modern and user-friendly design have been developed in the Ruby programming language, making use of this device driver.⁷ These programs in combination with the device driver were used for recording sample spectra which are presented and discussed in this thesis. The spectrometer with a pulsed Helium lamp has been the main focus of my studies, and the expression “the spectrometer” hereafter refers to that particular set-up.

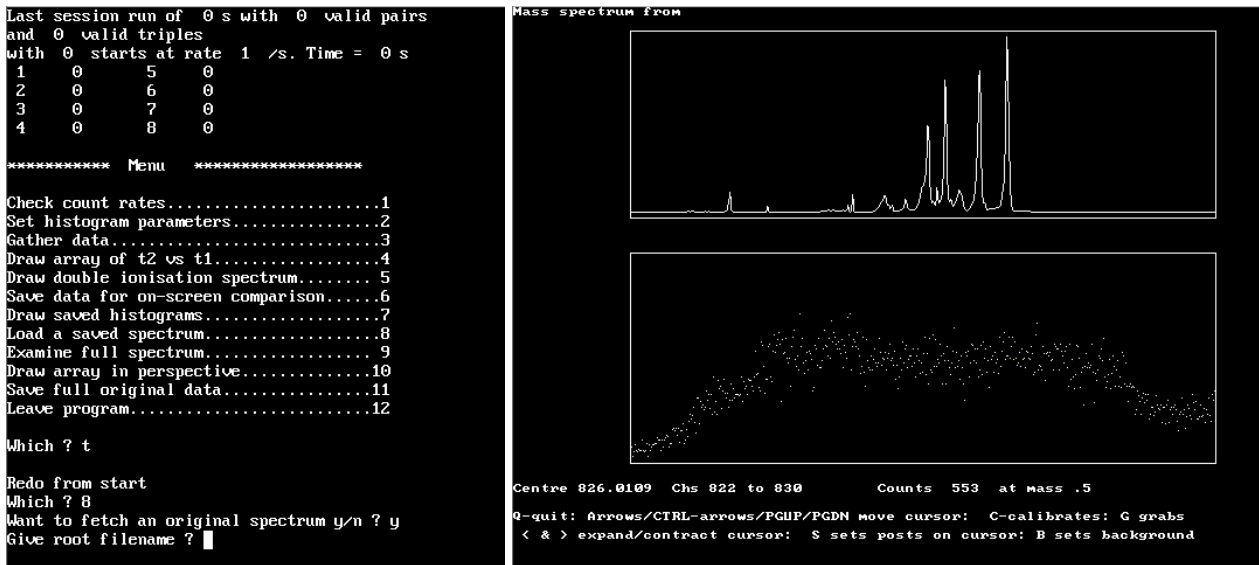
⁴Only operating systems based on MS DOS (e.g. Windows 98) can run the software.

⁵The times read out from the TDC card are formatted as 16-bit unsigned integer values (in units of 0.5 ns). In QuickBasic however, the only 16-bit integer format is signed. In the programs used hitherto, this inconvenience has resulted in a slightly complex process for decoding the data. The programs do not make use of the full resolution of the TDC card (0.5 ns) either, but instead have time units of 1 ns. Furthermore, the software does not handle exceptions. Instead it comes to a halt

or crashes if the user does something unintended.

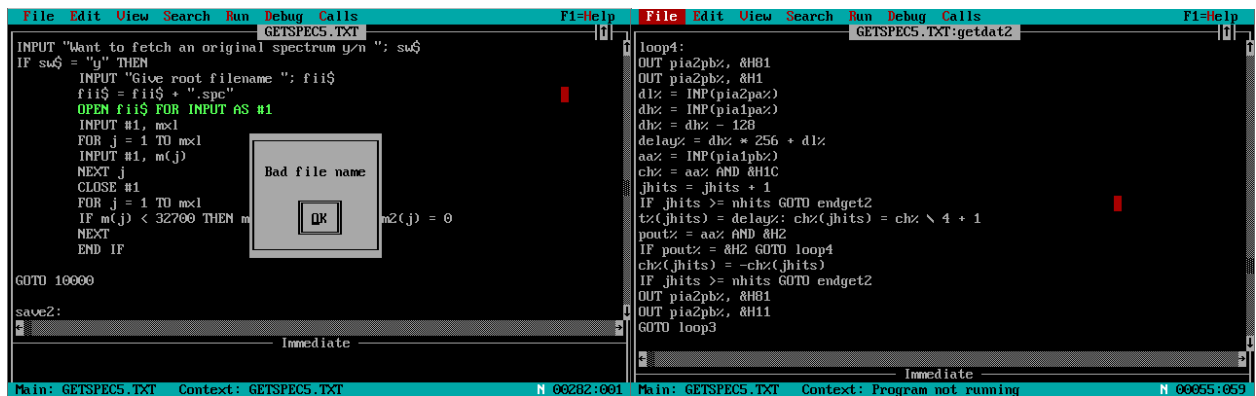
⁶It is at least not among the 50 most popular programming languages [10].

⁷Since both spectrometers make use of the same TDC card, the driver works for both set-ups. The data acquisition software have also been designed to handle the differences between the spectrometers through easily modifiable settings, and has been tested on both spectrometers.



(a) GETSPEC5's typical text-based main menu.

(b) The EEIITOF program, here plotting a mass spectrum.



(c) Exceptions may result in a halt or crash of the program.

(d) Without good knowledge of the software's purpose and the TDC-card's functionality, the source code would be very difficult to understand.

Figure 1.1: Screen-shots from the QuickBasic programs GETSPEC5 and EEIITOF and the source code. Interaction with the programs are by keyboard commands only: There is no support for using the mouse to pan or zoom in the spectra.

Chapter 2

Photoelectron spectroscopy

By analyzing the kinetic energy of electrons which leave a sample when particles interact with it, information about the properties of the sample and mechanisms of the interaction can be obtained. This method is called *electron spectroscopy*,¹ or *photoelectron spectroscopy* when photons are used as impinging particles [6].

Electron spectra of atoms typically reveal distinct lines corresponding to transitions between various electronic states involved. Basically, upon removal of an electron, the system goes from its neutral ground electronic state into one of the electronic states of a singly ionized system. Diatomic molecules have more complexity, since absorbed energy can also be stored in the system e.g. as inter-nuclear vibrations. The spectral lines in the photoelectron spectrum of O_2^+ , which will be used for calibration of the spectrometer, are discussed in Appendix B.

2.1 The Photoelectric effect

A photon γ has an energy proportional to its frequency ν , according to the Planck formula, $E_\gamma = h\nu$ [11, p.158]. The *work function* of a solid surface, ϕ , describes the minimum energy needed to release an electron from the surface. When an electron on the surface absorbs a photon the electron might leave the surface if $h\nu \geq \phi$.² This phenomenon is known as the *photoelectric effect* (see illustration in

¹The word *spectroscopy* is a paraphrase of the Greek words *spectrum* (picture) and *scopein* (see).

²Secondary interactions between the electron and other atoms in the surface might reduce the electron's energy and recapture it.

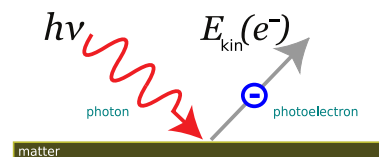


Figure 2.1: The photoelectric effect on a solid.

Figure 2.1).

The electron will leave the surface at a certain speed³ v , corresponding to a kinetic energy:

$$E_{kin} = m_e v^2 / 2. \quad (2.1)$$

The *photoelectric law*,

$$E_{kin} = h\nu - \phi, \quad (2.2)$$

gives an upper limit for the kinetic energy of electrons that leave the surface in this way.

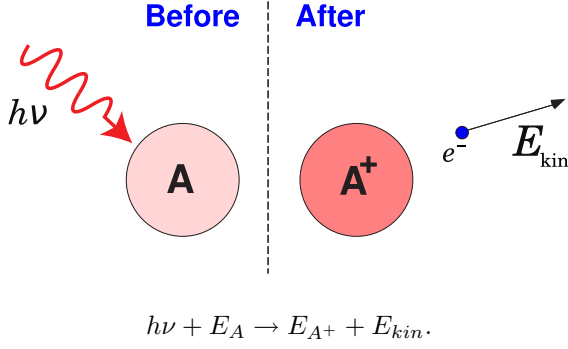
2.2 Photoionization

When photons ionize atoms or molecules, the process is generally called *photoionization*. Electrons released by photons through photoionization are referred to as *photoelectrons* (PE) [7, 12].

2.2.1 Single ionization

The photoelectric law mentioned above can be easily modified for studies of free atoms or molecules instead of macroscopic metal surfaces. If e.g. a photon of high enough energy $h\nu$ is absorbed by an

³Note that electrons in this study do not reach relativistic speeds, so classical mechanics can be used.

**Figure 2.2:** Photoionization (single).

A photon with a certain energy $h\nu$ is absorbed by an atom A . The atom becomes singly ionized; a photoelectron, e^- , leaves it with kinetic energy E_{kin} .

atom A (with total energy E_A), the atom can be ionized (the cation having total energy E_{A^+}). A photoelectron, e^- will then be realized, with a well-defined kinetic energy E_{kin} . The process is illustrated in Figure 2.2. Energy conservation of the total system⁴ can be formulated as the equation:

$$h\nu = \underbrace{E_{A^+} - E_A}_{E_B} + E_{kin}, \quad (2.3)$$

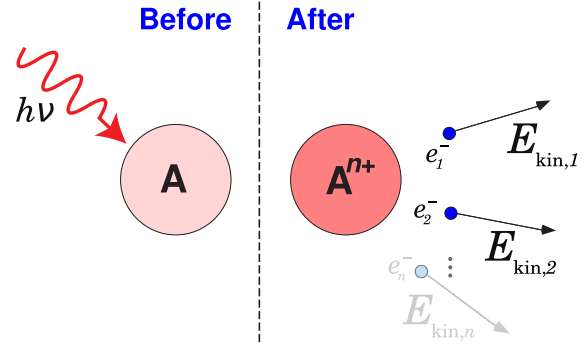
where E_B is the *electron binding energy*, which is equal to E_I , the first *ionization energy* (or ionization potential) – the energy required to release an electron from the neutral atom. The work function of the surface in equation (2.2) is hence replaced by E_I , and the equivalent photoelectric law for free atoms or molecules is found:

$$E_{kin} = h\nu - E_I \quad (2.4)$$

Koopmans' theorem states that the ionization energy is equal to the orbital energy of the electron in the *highest occupied molecular orbital* (HOMO) in the neutral molecule (or atom). The theorem is however an approximation – in general, ionization energies are slightly lower than the orbital energy of the electron [13, 14].⁵

⁴The ion will recoil as the electron leaves it, but since the electron mass is approximately 1/1836 of the proton mass, the recoil energies of most atomic nuclei are negligible (it is important in some cases however) [7, p.13]. In this thesis, where O₂ molecules and Xe atoms are dealt with, the recoil energy can safely be ignored.

⁵The theorem assumes e.g. that in a multi-electron atom,

**Figure 2.3:** Photoionization (multiple).

A photon with a certain energy $h\nu$ is absorbed by an atom A . The atom becomes multiply ionized; n photoelectrons: $e_1^-, e_2^-, \dots, e_n^-$, leave it with kinetic energies $E_{kin,1}, E_{kin,2}, \dots, E_{kin,n}$.

2.2.2 Double ionization

If sufficiently high photon energy is used for excitation, it might possibly lead to the release of two or more electrons at the same time, as illustrated in Figure 2.3. For an arbitrary number n of simultaneously emitted electrons due to a single photon absorption with energy $h\nu$, conservation of energy results in the relation:

$$h\nu = E_{A^{n+}} - E_A + \sum_i^n E_{kin,i}, \quad (2.5)$$

where the individual electrons are indexed by i . When two electrons are emitted in coincidence upon single photon absorption, it is called *double photoionization*. Accordingly, e.g. the *double ionization energy* or *double ionization potential* (DIP), is equal to the energy difference between the dication A^{2+} and the atom A , and hence from equation (2.5) we get the expression:

$$\text{DIP} = h\nu - (E_{kin,1} + E_{kin,2}). \quad (2.6)$$

In general, the studies of double or multiple ion-

the remaining electrons' wavefunctions are not altered upon removal of one electron from the system. In reality, removal of an electron results in a reconfiguration of the remaining electrons' orbitals – so-called *orbital relaxation*. (The difference between the ionization energy and the orbital energy of the electron is called the *reorganization energy*.)

ization are important for understanding e.g. chemical reactions in environments of intense radiation, such as the outer atmosphere and the interstellar space [3].

which the software has been developed, and we shall examine the double ionization spectrum of Xe as a good example.

2.3 Time-of-flight spectroscopy

Electron *Time-of-flight* (TOF) spectroscopy is a method in which the electrons' kinetic energies are measured by their speed in a flight-tube of fixed length. The kinetic energy of an electron is given by the formula (2.1). For electrons traveling at constant non-relativistic speed $v = d/t$, where t is the flight-time, and d is the distance traveled, the kinetic energy is thus:

$$E_{kin} = \frac{m_e(d/t)^2}{2} = \left(\frac{D}{t}\right)^2, \quad (2.7)$$

for a constant D which depends on the flight-length of electrons in the TOF-spectrometer. The constant D is typically determined in measuring a series of well-known single ionization spectra for each TOF-spectrometer. This has previously been done for the spectrometers at AlbaNova.

Methods for studying several photoelectrons in coincidence have been developed by J. H. D. Eland since the 1990s [3] and has resulted in the *Time-Of-Flight PhotoElectron-PhotoElectron COincidence* (TOF-PEPECO) spectroscopy method in 2003 [8]. This method – compared to related techniques such as *Threshold Photoelectron COincidence spectroscopy* (TPEsCO) – has the advantage that all electrons emitted in pairs through double photoionization are equally analyzed in energy [8].

The TOF-PEPECO method has very recently been further developed. In particular a method which can study an arbitrary number of photoions in coincidence with an arbitrary number of photoelectrons, abbreviated as TOF-PE(m)PI(n)CO, has been presented in 2006 by J. H. D. Eland and R. Feifel [15].

Within the framework of this thesis, we shall mainly focus on the TOF-PEPECO method for

Chapter 3

Experimental set-up

The experimental set-up has been presented in various works before [12, 9, 16, 8, 2], and is briefly outlined here.

3.1 Spectrometer

The spectrometer consists mainly of an interaction chamber, a flight tube, an electron detector and some electronic units. A pulsed lamp producing ionizing UV-radiation, a monochromator, and a gas inlet system are some additional necessary components of the experimental set-up. The monochromator, interaction chamber and flight-tube are evacuated by turbomolecular pumps.

From the pulsed UV-lamp, typically operated at a pulsed rate of a few kHz, having pulse widths of 5–10 ns, the He and He⁺ resonance lines of 21.218 eV and 40.814 eV, respectively, are selected with a monochromator, and are used for the spectra of O₂ and Xe respectively. The light pulses enter the interaction chamber of the spectrometer, and ionize the target gas atoms or molecules inserted through the gas inlet. Released photoelectrons fly toward the electron detector located on the other end of the several meter long flight-tube. The set-up is sketched in Figure 3.1 on page 14.

A magnetic bottle effect is achieved by a strong permanent magnet located right in front of the gas inlet needle which produces a convergent magnetic field, in conjunction with a solenoid which produces a parallel magnetic field in the flight-tube. The strong permanent magnet reflects photoelectrons emitted in the wrong direction and hence

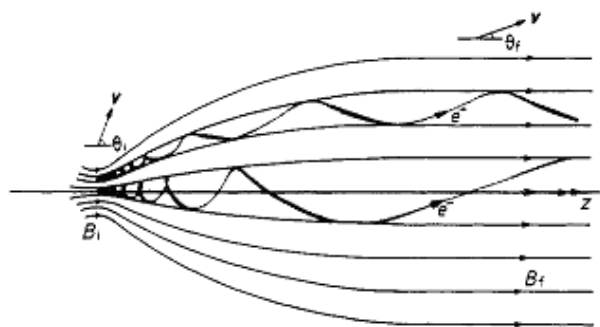


Figure 3.2: *The magnetic bottle: Photoelectrons follow spiraling trajectories along the magnetic field lines [17, Fig.1].*

guides them into the flight tube towards the electron detector, as illustrated in Figure 3.2. According to computer simulations performed by Kruit and Read [17], the parallellization mechanism in the magnetic bottle (see Figure 3.2) will make certain that 90–100 percent of the electrons emitted over almost the whole 4π solid angle are detected in this way.¹

The flight-tube is shielded from external/earth magnetic fields with a mu-metal screen. In one of the spectrometers, there is also an inner tube to which an electric field can be applied for decelerating or accelerating electrons slightly, and in this way to tune the resolution of the spectrometer. Some small DC potentials are applied to the permanent magnet and needle in order to ensure that all electrons reach the detector.

The permanent magnet's position is known to affect the electron count rate and resolution of

¹The multichannel plate detector only has approximately 65 percent detection efficiency however.

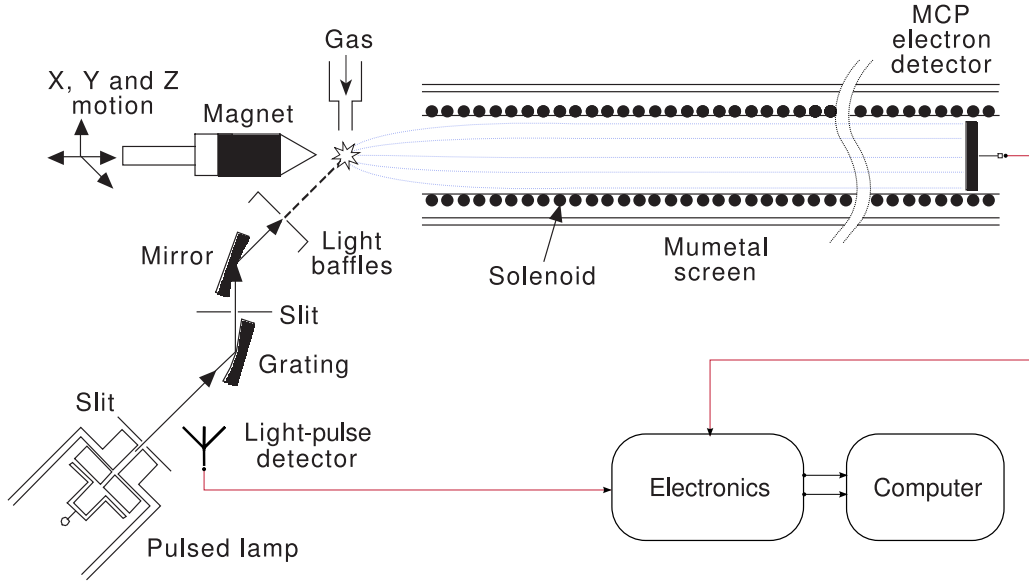


Figure 3.1: Diagram of an electron time-of-flight spectrometer

the spectrometer strongly, and it can therefore be moved in all directions with small adjustments.

3.1.1 Spectrometer offsets

The theoretical time-to-energy conversion formula (2.7) needs to be modified slightly for a real TOF-spectrometer. As known from the original work of Eland et al. [9], spectral lines from atomic and molecular time-of-flight electron spectra can be fit to:²

$$E_{kin} = \frac{D^2}{(t + t_0)^2} - E_0 \quad (3.1)$$

The time offset t_0 is partly due to the light-pulse width³ as well as due to a signal delay in the electronic equipment and cables. The energy offset E_0 is mainly attributed to surface potentials in the spectrometer. These offsets might drift during long runs, and the spectrometer must therefore be calibrated on a frequent basis. More details about the formulas used in the calibration are discussed in Appendix A.

²The time offset in the reference mentioned is typically negative compared with the t_0 used here; the formula is there described as: $t = t_0 + D / (E_{kin} + E_0)^{1/2}$.

³In the calibration for the spectra discussed later in this thesis, the photon energy is not the same for O_2 and Xe measurements. The changes to the light-pulse affect the value of t_0 , and this is taken into account in the calculations. Ideally, the calibration of t_0 should be performed using the same photon energy as the one used in the main spectrum.

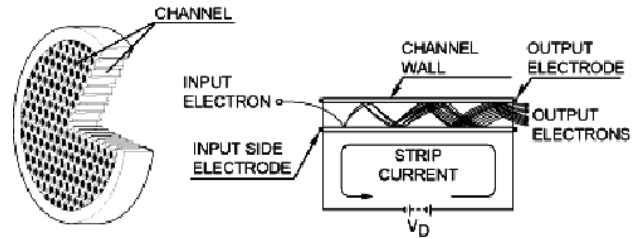


Figure 3.4: MCP electron detector.

3.2 Electronics

The spectrometer has a micro-channel plate (MCP) for detecting electrons at the end of the flight-tube. This MCP is position sensitive, and has a typical detection efficiency of about 65% [9]. The MCP has many parallel microchannels, each of which are continuous-dynode electron multipliers. When a photoelectron hits any of the channels' walls, electron multiplication occurs; under strong electric fields, an avalanche effect releases many more electrons which are propagated through the channel.⁴ The principle is illustrated in Figure 3.4 on page 14. Thus, the signal is amplified by several orders of magnitude before the cloud of electrons hit an anode. An electric pulse that is measurable by elec-

⁴The necessary recharging of the MCP after a hit results in a small dead-time, which is less than 50 ns.

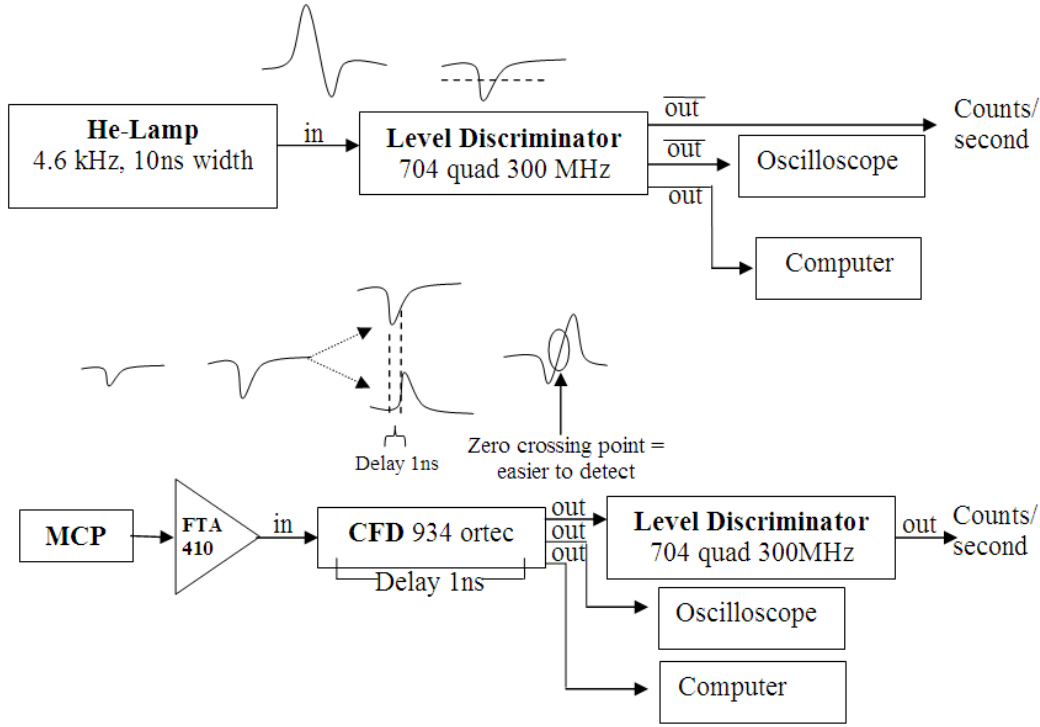


Figure 3.3: *Electronic scheme for one of the spectrometers.*

tronic equipment is then propagated from the anode.

An antenna at the light source detects the ionizing light-pulse, and this signal is fed through a logic level discriminator and is used as the trigger for flight time measurements. The weak electronic pulses from the MCP's anode are amplified and fed through a constant fraction discriminator (CFD), and a logic level discriminator which form the stop signals of the coincidence experiments. Oscilloscopes and counters are also in place to monitor the hits in real-time. A sketch of the pulses' routes are presented in Figure 3.3.

3.3 Computer set-up

The electronic equipment generate logic pulses corresponding to the light pulse, and to each of the detected electrons from the MCP. These pulses are handled by a timing device – the TDC card mentioned above – in the computer, which measures the delay between the lamp pulse and the electron pulses. The present TDC card requires that

the computer's motherboard has an ISA bus. Most motherboards available at present time do not support the ISA standard anymore; they use the PCI bus instead, since it gives better performance [1]. Details about the TDC card used and some other models are discussed in Appendix C.

The computer performs data analysis and filtering of the data, through special software. Data are saved to files, for later analysis in various applications. The software also has the ability to draw spectra and coincidence maps during acquisition.

3.3.1 New computer set-up

Within the framework of this thesis, I have built a new computer set-up, which is running newly developed software (more details on the new software are presented in Chapter 4). It was built mainly because the old computers used so far with the spectrometers (which support ISA-cards) are several generations old and cannot run modern software.

A modern computer with a fast processor was found to be very useful in the development of im-

proved data acquisition software, but since support for the ISA bus cards was required, options were very limited.⁵ Certain motherboards for industrial use, available at specialized retailers, still implement the ISA bus, and also support modern processor architectures. One such motherboard is the *BK775ISA*⁶ which has been installed in the new computer set-up. The cost of this computer was much less than the cost of a new PCI variant of the TDC card, and the computer has very good performance.⁷

A modern GNU/Linux-based operating system⁸ is installed on the new computer, mainly due to its open source nature which give the programmer more control over the system. The *linux-rt* package is also installed,⁹ in order to enable a patched Linux-kernel with support for high-resolution timers. Using these timers makes it possible to perform timing related tasks with much higher precision compared to what is possible on non-real-time operating systems. We shall come back to the necessity of accurate timing in Section 4.2.

⁵The possibility of using an adapter which connects the ISA-card to a USB-port was first investigated. The *ArsTech USB-ISA* is such an adapter, which is quite inexpensive [18]. I attempted to make the TDC card work with this adapter using software and drivers that were shipped with it. After several weeks of testing and after discussing the matter with ArsTech support personnel, the conclusion was drawn that the TDC card would not work with the adapter, and that possibility was abandoned.

⁶*BK478B800+* and *BK775ISA* are available at Binärteknik <http://www.binarteknik.se/produkter/0815.phtml>. The *BK775ISA* supports the Intel Pentium 4 or Celeron processor, and up to two gigabytes of Dual Channel DDR RAM. A word of caution: The card we received showed signs of faulty USB components (overcurrent on certain pins), so I had to disable USB support in the BIOS to stop Linux from complaining about this.

⁷The motherboard has been equipped with an *Intel Celeron D* processor with a clock-speed of 3.2 GHz, and two gigabytes of RAM, among other components.

⁸Ubuntu 8.04, developed by *Canonical Ltd.*, was chosen since it is currently one of the most widely used distributions.

⁹The console command to install this package is: `sudo apt-get install linux-rt`

Chapter 4

Development of new software

4.1 Operating systems and device drivers

An operating system provides an abstraction layer of the computer's hardware for programs. This allows the programs to treat the hardware in a consistent way, without requiring knowledge of every possible hardware configuration that can exist. Only well-known system calls are required, and the operating system handles the details of how to perform the actions on the hardware. This translation is performed by the help of *device drivers* – special types of modules responsible for communication between the processor and the devices in appropriate ways.

An operating system also manages the operations of programs, and protects against unauthorized access to resources – programs cannot generally communicate directly with the computer's hardware without making use of device drivers,¹ and neither can they access memory or system resources without the operating system's permission. The operating system allow potentially risky system calls to programs only under the right circumstances: The *CPU* has different levels of operation; some operations are allowed only at the highest level. In Unix/Linux, the kernel executes under this level, which is called *supervisor mode*, or *kernel space*, and everything else (programs etc.)

¹It is possible to write a program which communicates with the TDC card through special system calls with supervisor privileges, but the performance is then reduced. This was the first stage of the development of new software – prototype code written in C++, and when it worked, I continued with developing a driver for the TDC card – written in C.

executes under the lowest level, the *user mode* or *user space* [19, ch.2]. Modules, e.g. device drivers, run in kernel space, so they can directly communicate with computer hardware and devices (and programs).

4.2 TDC card device driver

A device driver for the TDC card has been developed, in the form of a Linux kernel module (LKM). The requirements to use this LKM (hereafter referred to as “the TDC module” or “the LKM”) are:

- A PC equipped with a TDC card similar to the RoentDek TDC8 ISA;
- An operating system with Linux kernel 2.6.x with real-time patch, that allows loading of kernel modules at runtime; recommended: Ubuntu 8.04 or newer;
- The necessary software needed to compile the source code; on Ubuntu: *linux-rt*, *build-essentials*, and *linux-headers* packages;

The TDC card does not notify the processor when a hit has been detected. Instead, the processor has to check with the TDC card over and over again if a hit has been detected or not, while a measurement is in progress. This loop can easily occupy the processor with useless work.² Ideally, the TDC card should notify the processor when a trigger signal has been detected. This can be done by modifying

²This was the case in the old QuickBasic software which in effect would freeze in a loop waiting for a COM signal until one had been detected.

the TDC card to send an interrupt request (IRQ) signal to the processor as soon as a hit is detected.³

However, the processor does not have to check the TDC card for data more often than the lamp pulse rate (which triggers the TDC card to start measuring times). In the device driver for the TDC card, a process periodically checks the card for data, at a rate similar to the supposed trigger rate. To be certain that no signals are missed, the DAQ software will check the card for data more often than necessary, typically four times per trigger pulse.⁴ On an operating system without high-resolution timers, the callback function would be able to check the TDC card for data at most approximately 100 times per second – another approach would have to be taken, such as the loop described above,⁵ or modifying the TDC card to support IRQ-signals. The high-resolution timer has a theoretical resolution of 1 ns, compared to a normal operating system's tick rate of 10 ms.⁶

4.2.1 Development of the LKM

The C programming language is the most widely used programming language for system programming (e.g. drivers, operating systems, and embedded system applications). It gives the programmer much control on a low level (close to the hardware), and generally⁷ compiles into highly optimized executable files, with respect to file-size, memory footprint, and/or execution speed. Therefore it is the best choice of programming language for the LKM.⁸

³The TDC card can in principle be modified quite easily in order to make use of this feature. The driver is also designed with this in mind, and can easily be modified to support interrupts.

⁴This guarantees that no signals are missed. In principle the rate could be made lower, i. e. twice the trigger rate, or even equal to it, if the trigger rate is very stable (which is the case on the laser system).

⁵One such version of the driver has also been developed, and is available on the subversion server as well. Its performance is much worse though.

⁶The practical resolution is lower though: The scheduler in the kernel make it possible to run multiple processes “in parallel” on a single CPU by means of time slicing. A process cannot be certain that it is indeed running at a precise time.

⁷Assuming that the programmer has sufficient knowledge in the programming language.

⁸The Linux kernel is almost entirely written in C, and some portions are written in assembler. C++ compilers are not trustworthy, especially for kernel programming, according to L. Torvalds and other developers [20].

However, the C programming language can be a dangerous choice for critical system code if the programmer is not experienced: It offers no protection against *memory leakage*⁹ which is one of the most difficult problems to solve in computer programming. Memory allocation and deallocation (release) has therefore been carefully designed in the code for the TDC module, since the device driver is supposed to remain in the kernel once it has been inserted.

In the development of the code, a subversion server has been used to track changes to the code. There are also different prototypes and branches stored on the server. The source code for all the branches, and all previous versions, can be found at the URI:

```
svn://simon2.fysik.uu.se/home/svn/repos/usx/tof/
TDC/
```

The folder `tags/` contains snapshots of stable versions of the driver at different times in the development. The folder `branches/` contains alternative development branches.¹⁰ The `trunk/` folder contains the main development version. A stable version of the driver, found in the folder `tags/20080827/`, is the one presently used. The necessary source code for the module can be obtained¹¹ with SSH tunneling and a subversion client with the command:

```
svn checkout svn+ssh://simon2.fysik.uu.se/home/
svn/repos/usx/tof/TDC/tags/20080827
```

As a basis for the device driver, example code for a basic character device and other types of drivers by Corbet et al. [19] were used, and extended with features necessary for the TDC card. More detailed

⁹Dynamically allocated memory remains in use until it is released explicitly by the module, or until the module is unloaded. If memory allocated in a kernel module is not released later on, the module will act as a “black hole” into which available memory just disappears from the system while the module is loaded into the kernel. The memory that has “disappeared” remains in use until the computer is restarted. Sometimes this also results in a complete system crash, e.g. if no more memory is available on the system.

¹⁰E.g. the branch `branches/old_version/` does not require a real-time Linux kernel. It gives poor performance though, compared to the version in the `trunk/` folder.

¹¹The subversion server requires that a valid user-name and password is supplied. Egil Andersson, `egil.andersson@fysik.uu.se`, can help with this matter.

information about the source code can be found in Section D.1.

4.2.2 Using the module

First, download all the source code from the subversion repository. If the TDC module `tdcmod.ko` is not found, or cannot be loaded, it must be compiled from the source code. The included file `Makefile` contains details necessary for compilation of the code. In order to compile the module, simply type `make` in the console and press the Enter key (assuming that the current working directory contains the downloaded source code).

4.2.2.1 Loading/unloading the TDC module

After successful compilation of the TDC module source code, there should be a file called `tdcmod.ko` in the working directory. This is the loadable LKM, i.e. the device driver, and it must be loaded into the kernel. This can be done automatically [21, sec.1.2] with the provided bash script `tdc_load` (explained in Section D.4), which must be executed in supervisor mode. Any module parameters (see Section D.2) that need alteration should be given as arguments to the `tdc_load` script [19, pp.22–37]. Example of usage:

- To load the module for a TDC card with default settings:

```
sudo ./tdc_load
```

- To load the module for a TDC card at base address 0x330 with a buffer size of 1024 bytes:

```
sudo ./tdc_load tdc_base_address=0x330
tdc_buffer_size=1024
```

In order to verify that the module has been successfully loaded, the `lsmod` command can be used to see a list of the loaded kernel modules. If the module is already loaded, it will fail to load. To unload the module manually, it is recommended to execute the bash script `tdc_unload` (see Section D.4):

```
sudo ./tdc_unload
```

Normally, there is no need to unload the module once it has been loaded.¹² If the computer is restarted, the module must be loaded into the kernel again manually.¹³

Once the TDC module has been loaded into the kernel, the data acquisition software developed – described in Chapter 4 – can be used for recording spectra with the TDC card. This software communicates with the TDC module through standard file system calls, as described in the rest of this chapter. The data acquisition software provides a more user-friendly interface, though.

4.2.3 Controlling the TDC module

The TDC module is controlled by writing to the device as to a file. Details on how commands are written to the device and a list of all possible commands are available in Section D.5.

The TDC card should first be configured to run in common stop mode if common start mode is not used, and if a special time window is desired (a range of allowed times – all events outside this time window are ignored), the minimum and maximum time should be configured. The desired rate at which the TDC card should be checked for new events must also be configured – this is the same as the timer callback rate.¹⁴

A measurement can then be started with a `start` command, and stopped with a `stop` command. If the data acquisition needs to be halted temporarily, it can be paused with the `pause` command.

4.2.4 Reading data from the TDC module

As events are read out from the TDC card by the device driver, details about them are stored

¹²If the source code has been changed however, the module must be unloaded before the new version can be loaded.

¹³It is possible to make the module load automatically on start-up of the computer, if preferred.

¹⁴Note: In the DAQ software, in the dialog for a new measurement, there is a setting for trigger rate. This number is the lamp pulse rate. The timer callback rate is in this software automatically set to four times the lamp pulse rate (to be certain that read-out of data will certainly take place before new pulses arrive).

in a special format (described in Section D.3) in a FIFO buffer in the computer's memory. When data are read from the device by a user process (from `/dev/tdc`), the data that has been read leaves the buffer, so memory for new data to arrive will be available.

If the buffer is empty, no (more) COM signals with hits belonging to them have been detected (yet). The reading process will then be put to sleep until data arrives into the buffer. The sleeping process will however wake up if data acquisition is ended before any data arrives.

If no more data are available in the buffer, and the data acquisition is ended by the user through a `stop` command, the EOF (end of file) character will be sent to the reading process.¹⁵

The device driver also calculates various statistics about ongoing data acquisition. This information can be accessed in the form of a (virtual) text file located at `/proc/tdc_measurement`. This file displays for instance counts of hits on each of the channels, as well as trigger (COM signal) rate and (e.g. electron channel) hit-rates. The count-rates are updated once per second, all other data are updated every time the file is read.

4.3 Data Acquisition Software

The data acquisition software can be found on the previously described subversion repository, at this location:

```
svn://simon2.fysik.uu.se/home/svn/repos/usx/tof/
    DAQ-suite/
```

There are a few different branches of the code, some of which are not used for this thesis.¹⁶ In the rest of this section, it is presumed that a stable version of the software, found in `tags/20080828/` has been downloaded into the working directory in some way. The software can be downloaded e.g. with the command:

¹⁵Note: This can only happen at a position in the stream where the user expects to read the number of hits belonging to the next COM signal.

¹⁶Further improvements of the software is presently performed by Egil Andersson, in the branch at `branches/egil/`. The branch used here is found in `branches/isak_new_for_updated_driver/`.

```
svn checkout svn+ssh://simon2.fysik.uu.se/home/
    svn/repos/usx/tof/DAQ-suite/tags/20080828
```

The software consists of Ruby source files, which need no compilation before execution – this makes it possible to easily modify the software in the future. A server and a client are the main components, and they can be run on different computers. The reason for this design was the reflection that most computers which support the ISA bus (and hence the TDC card) are old and unable to run advanced software at a reasonable speed. The server is therefore minimalistic and optimized for performance. The client performs most of the actual analysis of the acquired data, and requires a somewhat fast computer to run smoothly.

4.3.1 Server

The server is an intermediary software between the client and the TDC module. It can be thought of as a temporary storage place of the data read out from the TDC module, which allows one or many clients to access the same data, over a network (TCP/IP) or on the same computer.¹⁷

The software requirements for the computer running the server are:

Ruby version 1.8.5 or newer; version 1.9 preferred since it gives much better performance.

TDC module needs to be loaded before starting the server.

The server is started by entering the sub-directory `server/` and by executing the script `server.rb` there.

```
ruby1.9 ./server.rb
```

It is possible to give command-line arguments to the server, and these are described more thoroughly by the help argument:

```
ruby1.9 ./server.rb help
```

By default, the server will run locally and not be available on the network, but one can allow connections to the server from the network by supplying

¹⁷If many clients are connected, only the first one will be the “master” client which is allowed to control the TDC card. The other clients can only read data.

the argument `-host` followed by the IP address of the host on the network. If `auto` is given instead of an IP address, the script will try to find out the IP address automatically. A port can also be specified (default port is 10001).

It is possible to run the server on a computer that does not have a TDC card installed, by supplying the argument `simulated`. This is mainly useful for debugging purposes – in this mode the server will output pre-recorded data¹⁸ and will not interact with the TDC module at all. The command for starting the server in this mode, available on the network on port 80 would be:

```
ruby1.9 ./server.rb -host auto -port 80
simulated
```

When the server has been started, it will print to the command line the server's IP address and port in use. It will also display an input console where commands can be entered to the server. The command `quit` (or `exit`) is used to stop the server. Other commands are explained in the console.

4.3.2 Client

The client is the most advanced part in the DAQ software. It has an easily comprehensible graphical user interface. Prerequisites needed to run the client software are:

Ruby version 1.8.5

Gnuplot version 4.2.2 or newer.

Gtk2+/Gnome2 and Ruby bindings; in Ubuntu, the packages: `libgtk2-ruby` and `libgnome2-ruby` can be installed together with their dependencies automatically.

Glade3 version 3.4.5 or newer.

[**GSL** (GNU Scientific Library) version 1.11 or newer; and the package `libgsl-ruby` or `rb-gsl`: Enables viewing of 2D correlation maps of electron energies such as Figure 4.2 on page 23, but is optional.]

¹⁸Pre-recorded data were during the development of the software read from the TDC module during acquisitions of e.g. O_2^+ photoelectron spectra, and saved to binary files which were compacted and included in the software.

The client is started by entering the sub-directory `client/` and from this directory executing:¹⁹

```
ruby ./client_app.rb
```

4.3.2.1 Connecting to the server

In order to acquire new spectra, one must first connect to the server software. This is done by entering the server IP address and port in the format `host:port`, in the designated field in the top left of the client window, and then clicking on the *Connect* button next to the field. If one clears this field instead of entering an address and port, the default values will be used – the client assumes that the server is running with default settings on the same computer.²⁰

4.3.2.2 Creating a new measurement

When a connection to the DAQ server has been established, the *New measurement* button is enabled. When this button is pressed, a dialog window will appear (see Figure 4.1 on page 22), where the settings for the new measurement can be entered. It is important to set the *Trigger pulse rate* to a value which matches the light source pulse rate. Also, in order to reduce the amount of irrelevant events, a time window can be specified from t_{min} to t_{max} – events outside of this window are ignored completely. The field Δt is related to the double-hit dead-time of the (electron) detector and the TDC card. If multiple hits are detected on one of the TDC card's channels, they will be completely ignored if the minimum time difference between any of them is less than Δt . The fields E_0 and t_0 are used to specify the calibration offsets, and these are only used for the graphical view of the spectra.²¹ Under the tab *Advanced* can be found settings which are rarely necessary to modify: The

¹⁹Note that Ruby version 1.9 cannot execute the client, because some of the libraries used by the client did not have bindings to Ruby1.9 at the time of development. The client is therefore designed for Ruby version 1.8.5 which was the stable version at the time of development.

²⁰Default IP address 127.0.0.1 and port number 10001.

²¹It is not necessary to know the correct offsets in advance, since all spectrum and coincidence data are saved as flight-times to files that can later be analyzed and converted to energies.

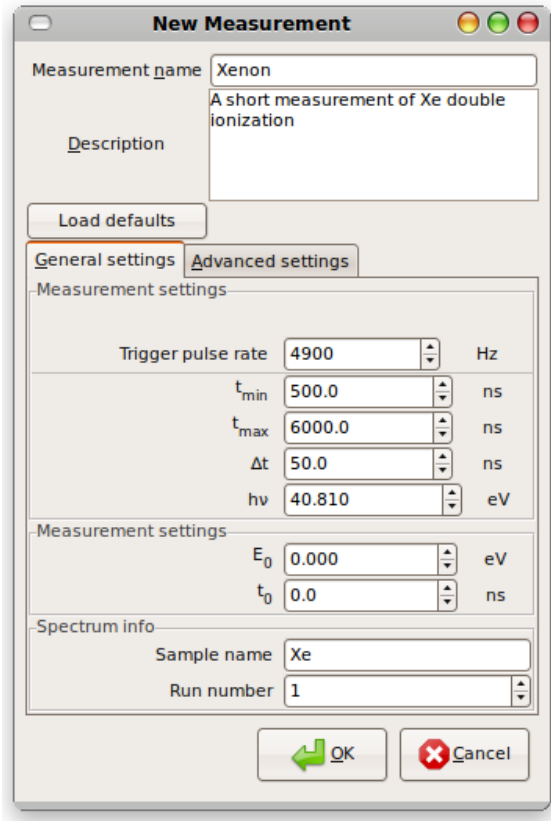


Figure 4.1: Dialog for setting up a new measurement.

spectrometer constant D , whether to use the TDC card in *common start* or *common stop* mode, and which TDC card channel that is connected to the the electron signal, etc. The time resolution to be used for the measurement can also be changed.²²

4.3.2.3 Running a measurement

Once a new measurement has been set up, data acquisition can be started by pressing the button *Start*. The *New measurement* button is disabled when a measurement has been started, and it becomes active again only when the measurement has ended by pressing the *Stop* button. Data acquisition can also be temporarily halted for some time by pressing the *Pause* button,²³ and restarted again with another press to the same button.

²²By default the time-of-flight spectra have bin-sizes of 1 ns. The highest resolution (lowest bin-size) possible is 0.5 ns – the resolution of the TDC card.

²³The *Start* button changes name to *Pause* as soon as a measurement has been started, and back again once a measurement has been paused.

All data entered in the *New measurement* dialog are automatically saved to a special metadata file²⁴ (`*_metadata.par`) in the folder `client/autosaved_data/YYYY-mm-dd/`.²⁵ One metadata file is created for each new measurement. The button *Open* in the top right of the client window can be used to load a metadata file (and also individual spectra or coincidence files) from previous measurements, to study the settings used for them, as well as their recorded spectra (see Figure 4.2 on page 23). No server connection is required for opening metadata or spectrum files.

Graphical spectra are continuously updated at a desired rate, which can be adjusted in the designated field, as illustrated in Figure 4.3 on page 24.

The tab labeled *Server details* contains detailed information about the server. The current version of the server displays all the information from the TDC module's `tdc_proc_measurement` function (details such as hit rate, measurement run-time etc.), described in Section D.1. These details are not automatically stored to any file.

4.3.2.4 The data files

While a measurement is running, the client is continuously acquiring data from the server, analyzing it, and with regular intervals storing binary files in the same directory as the metadata file. When the measurement is ended, the files are updated and closed. There are files with a histogram for the flight-times of single electron hits (`*_single_electrons-elToF.tofspc`) and a histogram of all hits disregarding multiplicity (`*_all_electrons-exToF.tofspc`). Coincidence files are also stored, i.e. when double electron hits occur on the detector, both electrons' flight-times are appended to a list of all double hits, and the whole list is saved to a binary file

²⁴The settings are also saved to the main configuration file when the application is terminated, and retrieved from there the next time the application is executed. This reduces the amount of unnecessary interaction between the user and the program – the dialog remembers the settings.

²⁵The asterisk in the file names of the auto-saved files are substituted with the date and time that the measurement started, in the format: `YYYY-mm-dd_HHMMSS`. The sub-directory `YYYY-mm-dd` also means the current date in that format.

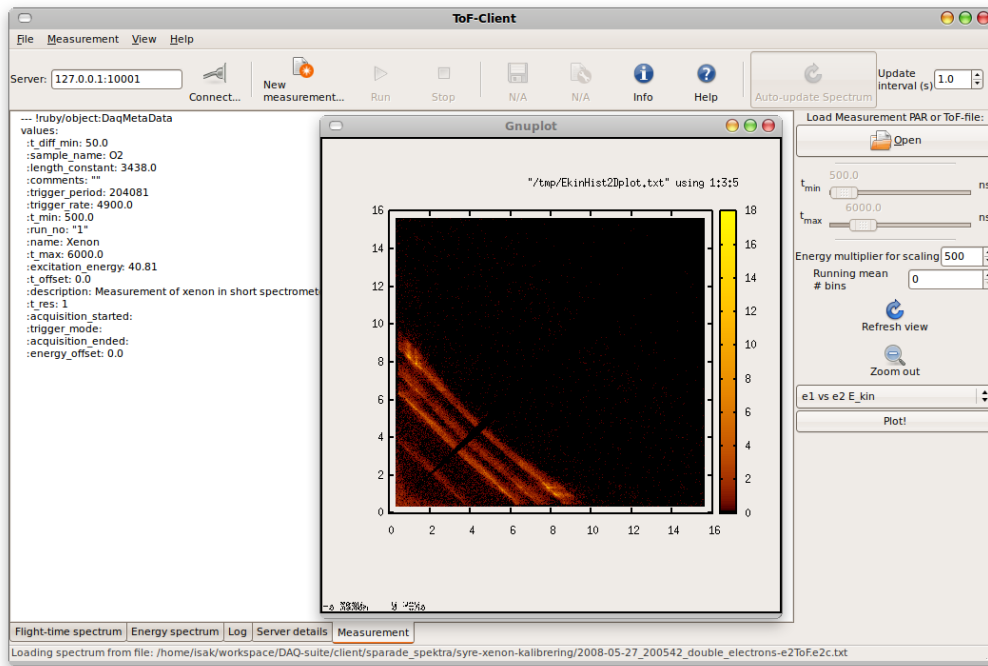


Figure 4.2: Previous measurement data loaded (metadata file).

Viewing the map of $E_{kin}(e_1)$ vs $E_{kin}(e_2)$ for an electron coincidence measurement of doubly ionized xenon, as well as the details of the measurement. Most buttons in the user interface are disabled until they are relevant (e.g. the start button cannot be pressed until a new measurement has been created), which is supposed to guide the user to click on the buttons in the right order, without having to learn the software first.

(*_double_electrons-e2ToF.e2c.txt). Individual files are saved with coincidences for electron hits with multiplicity 2, 3, 4 or 5, in the current version of the software, but this can be arbitrarily modified up to the TDC card's maximum number of hits per channel.

These binary data files are much smaller in size than the equivalent files from the previous QuickBasic software (which stored all numbers in plain text). For the spectrum files, there is one unsigned 16-bit integer for each flight-time, and following that is an unsigned 32-bit integer for the intensity of this time bin, and this pattern is repeated until the end of the file, without any other characters such as spaces or newlines. For an example of how to parse these files, see Listing F.1.

For the coincidence files, there are unsigned 32-bit integer values for each flight-time of each hit.²⁶

²⁶Note that it would suffice with a 16-bit value, since all flight-times from the TDC card are 16-bit values. This would reduce these coincidence file-sizes to 50% of their present sizes. The 32-bit format is used as the result of a typo in the code which saves the files, which was noticed a little too late in the project.

I.e. for a double hit coincidence file, the two first 32-bit unsigned integer values denote two related hits' flight-times, and the next two 32-bit unsigned integer values give the flight-times of the next two hits that were detected in coincidence, and so on until the end of the file. For an example of how to parse these coincidence files, see Listing F.3.

Changing it requires the analysis software and recorded spectra to be modified to fit the new data format – and this would break backward compatibility with previously saved spectra. Therefore I have left the format as it is, since the files are already quite small compared to the files that were saved by the original QuickBasic software.

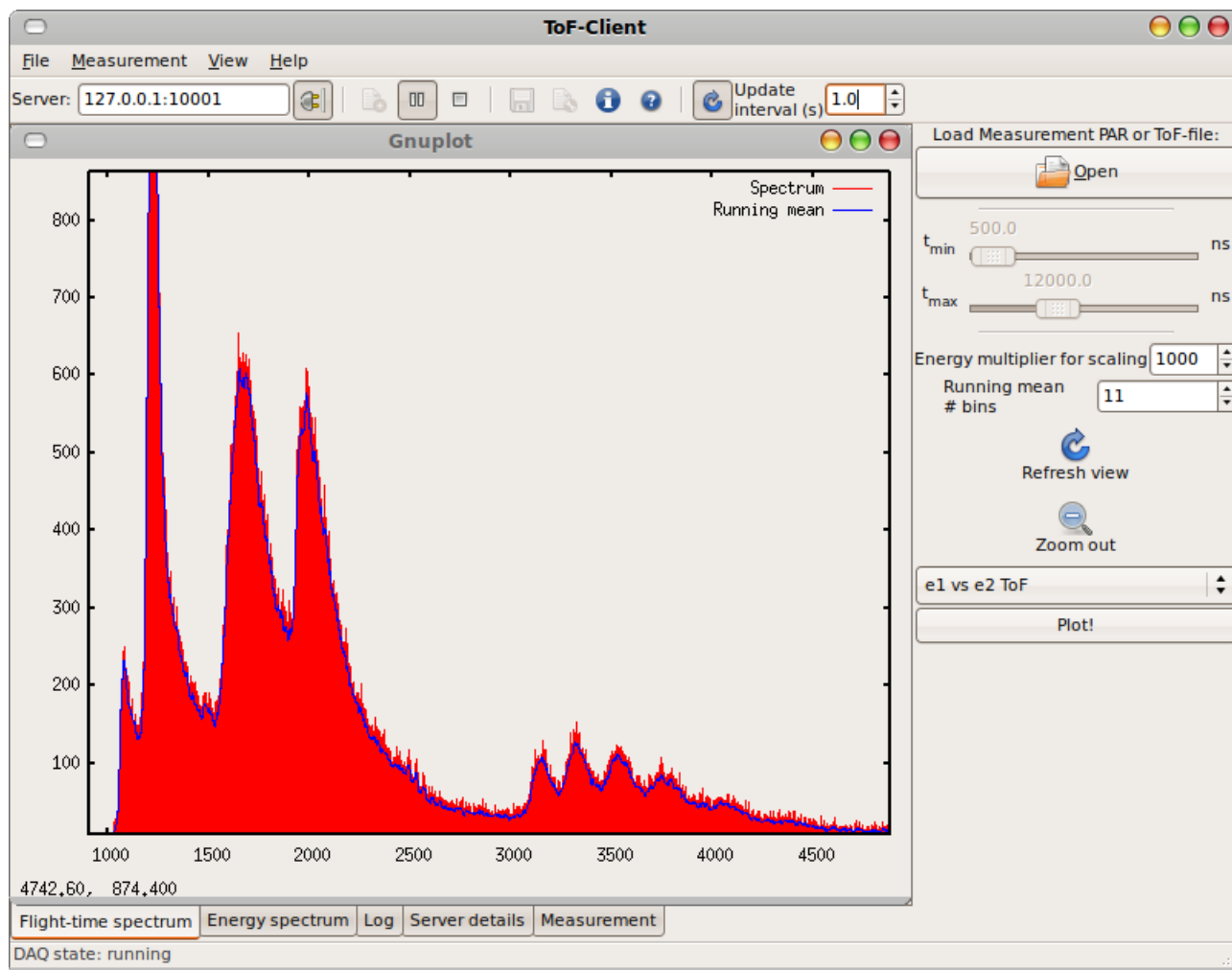


Figure 4.3: Data acquisition in progress, showing the time-of-flight photoelectron spectrum of molecular oxygen acquired from a server running in simulated mode.

Note that the buttons are smaller compared to the previous screen-shot. This setting can be changed from the *View* menu. The setting for how often the spectrum will be updated (the update interval) is here 1.0 s, and can be changed in the highlighted field in the top right of the window. The blue line in the spectrum is a smoothed spectrum (running mean of 11 adjacent bins from the raw histogram – the number of bins to use can be changed in the field with the label *Running mean # bins*). Gnuplot's default keyboard and mouse shortcuts work for zooming in the graphs etc.

Chapter 5

Testing the DAQ-software and TDC-module by acquiring spectra

Spectra have been obtained on both of the spectrometers using the software developed in the present thesis, and some of them will be shown below in order to demonstrate the working of the software. Since the focus of the present thesis is not so much to study the recorded data in itself, but to demonstrate that the software works as desired, we shall mainly discuss the results obtained on the instrument which uses the pulsed Helium lamp as ionization source.

The double ionization spectrum of Xe will be studied and compared to similar ones from the literature. Calibration must be performed both before and after the acquisition of the main spectrum, as discussed in Section 3.1.1. Here, the single ionization spectrum of molecular oxygen is used for this purpose. How to calculate theoretical kinetic energies for the photoelectron spectra of singly ionized O_2 is discussed in Appendix B. The following spectra are acquired in order:

1. Single ionization TOF-PE spectrum for O_2 , 40 minutes @ $h\nu = 21.218$ eV (O_2 , first run)
2. Double ionization TOF-PEPECO spectrum for Xe, 2 hours @ $h\nu = 40.814$ eV (Xe run)
3. Single ionization TOF-PE spectrum for O_2 , 30 minutes @ $h\nu = 21.218$ eV (O_2 , second run)

By carefully optimizing the spectrometer¹ and acquiring data for longer time the resolution of

¹I.e. by fine-tuning the shape of the magnetic bottle by careful adjustments of the permanent magnet's position, and by adjust-

the spectra can certainly be further improved compared to what is shown below.² Since the spectral quality itself is not the main focus of my thesis either, I have not attempted to optimize the resolution of the spectrometer to its best, and the acquisition time was comparatively short with respect to a normal run-time of 12–72 hours.³

5.1 Single ionization of O_2 at 21.218 eV

The time-of-flight photoelectron spectra from the first and second runs of singly ionized molecular oxygen are displayed together in Figure 5.1 on page 26. The graphs were produced in Matlab, and the code for the first run is displayed in Listing F.1. The code for the second run is not listed, since it is very similar.

As the intensities of the time-bins in the spectra are low, the noise makes it difficult to see the shape of some peaks in the spectrum (especially for the B -state where the intensities are low). The uncertainty in each observed intensity i per time bin is \sqrt{i} , assuming that the probability of an electron hit in that bin is given by a Poisson distribution. This

ing the DC voltage levels put on the magnet, needle, and the inner flight tube.

²Furthermore, the time resolution in the DAQ software, and thus the bin-size of the time-of-flight spectra, was set to 1 ns for these measurements.

³For other gases, one can determine the offsets while acquiring the spectrum, e.g. by recording the spectrum of a mixture of the sample and a noble gas such as Xe [22].

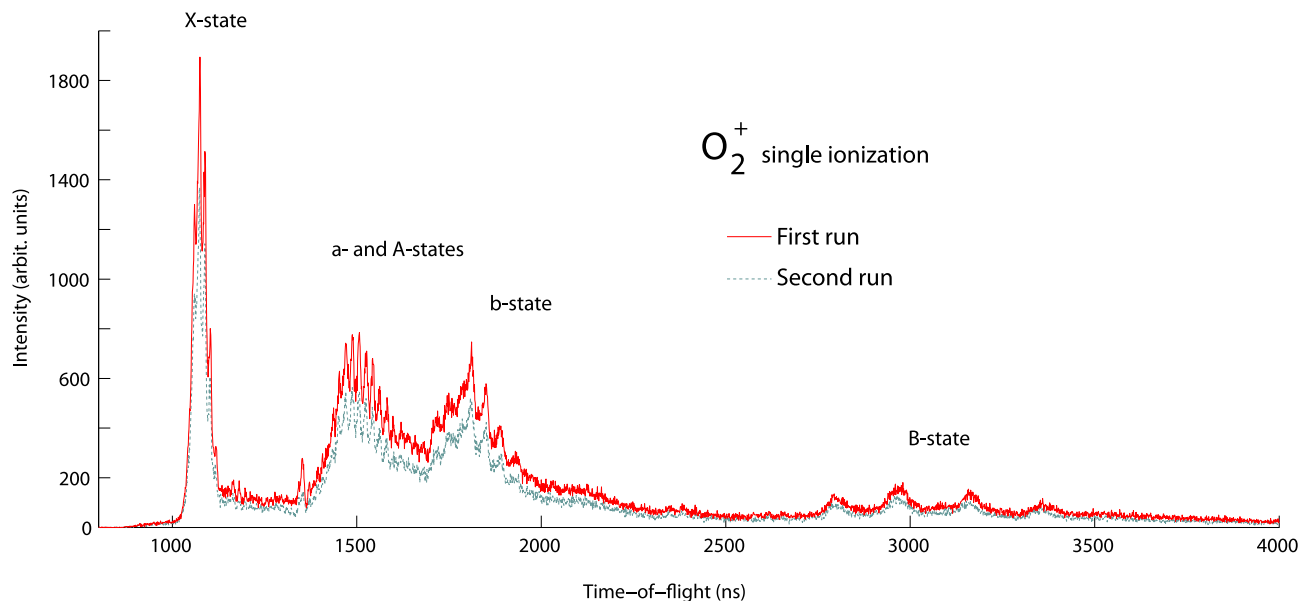


Figure 5.1: Time-of-flight photoelectron spectra of singly ionized O_2 , recorded at a photon energy of 21.218 eV. The first run was acquired for approximately 40 minutes prior to the Xenon measurement, and the second run was acquired right after the Xenon measurement, for approximately 30 minutes.

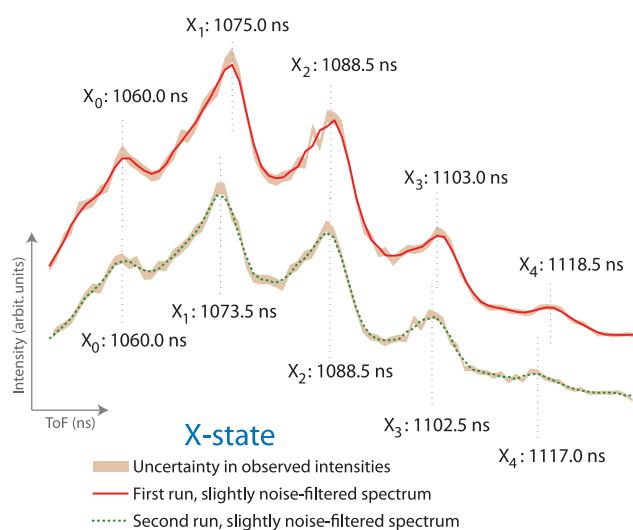


Figure 5.2: Detailed view of the vibrational fine structure lines of the X-electronic state of O_2^+ from Figure 5.1.

X-state vibrations from the time-of-flight photoelectron spectrum for the first and second acquisition of O_2 , singly photoionized using a photon energy of 21.218 eV. The graphs show slightly noise-filtered spectra, using a moving average from the histogram bins. The uncertainty margins of (unfiltered) histogram bins are represented by the lightly colored areas.

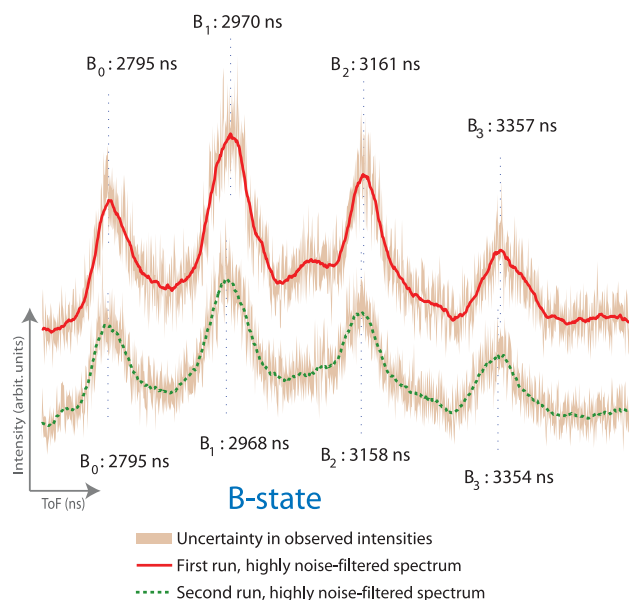


Figure 5.3: Detailed view of the vibrational fine structure lines of the B-state in O_2^+ from Figure 5.1.

Table 5.1: Kinetic energies (E_k) of O_2^+ as calculated in Appendix B, and the observed flight-times for the first run (t_1) and second run (t_2) of O_2 converted to energies (E_1 resp. E_2), using the converged values of E_0 and t_0 from each of the calculations (the first and second run) for the different vibrational energy levels of the X -, b - and B -electronic states of O_2^+ .

	E_k (eV)	t_1 (ns)	t_2 (ns)	E_1 (eV)	E_2 (eV)
X_o	9.149	1060	1060	9.169	9.157
X_1	8.915	1075	1073.5	8.912	8.925
X_2	8.685	1088.5	1088.5	8.457	8.677
X_3	8.459	1103	1102.5	8.457	8.454
X_4	8.236	1118.5	1117	8.219	8.232
b_0	3.049	1775	—	3.052	—
b_1	2.903	1811	1808	2.915	2.922
b_2	2.761	1850	1847	2.775	2.783
b_3	2.623	1889	1889	2.643	2.641
b_4	2.489	1931	—	2.510	—
B_0	0.925	2795	2795	0.948	0.947
B_1	0.784	2970	2968	0.782	0.783
B_2	0.649	3161	3158	0.632	0.633
B_3	0.520	3357	3354	0.503	0.504
$\sqrt{\sum (E_{kin}(t) - E_k)^2}$:				0.015	0.014

uncertainty is displayed in the graphs in the figures as a gray area between a lower and an upper limit, at values $i \pm \sqrt{i}$. Noise filtering is performed using the moving average of a number of adjacent bins in the spectrum (two bins for the sharp X -state peaks, and twelve bins for the B -state where the peaks are comparatively broad), in order to visualize better the shapes of the peaks. The filtered spectrum is located almost entirely within the uncertainty region. Detailed views of filtered X -state and B -state vibrations are illustrated in Figures 5.2 and 5.3.

The first five vibrational lines of the X -state and the first four vibrational lines of the B -state are resolved in the spectrum, and the corresponding peaks' centroids are located graphically. Using the procedure described in Appendix A and in the Matlab code in Listing F.2 (only the code for the first run is listed – the second run is analyzed in the same way), numerical values of the offsets E_0 and t_0 are calculated from these t -values. For both the first and second run, $E_0 = 0.518 \pm 0.06$ eV, and $t_0 = 45 \pm 1$ ns.⁴ Using these values in formula (3.1),

⁴Thus there was no offset drift during this short acquisition.

kinetic energies E_1 (first run) and E_2 (second run) were calculated and are given in Table 5.1, for each of the t -values of the states.

As a check on whether or not these offset values are reasonable, the standard deviations of the differences between the E_k -values from the literature (see Appendix B) and the observed values E_1 and E_2 are also displayed in the table. Their small magnitudes correspond well to the typical order of magnitude of the spectrometer resolution.

5.2 Double photoionization of Xe at 40.814 eV

The weighted averages of E_0 and t_0 for the first and second run of the O_2 spectra are used in estimating the offsets for the Xenon run: $E_0 = 0.518 \pm 0.004$ eV, $t_0 = 44.9 \pm 0.8$ ns. As mentioned in section 3.1.1, t_0 has a systematic shift, due to the higher photon energy which affects the lamp pulses. Therefore 5 ns is subtracted⁵ from the t_0 -value in the Matlab code in Listing F.3, which performs the analysis of the Xe data and creates the graphical spectra displayed here. The correlation maps of the electron pairs released from double photoionization are illustrated in Figures 5.4 and 5.5.

From equation (2.6) we get the DIP, after subtracting the photoelectron pairs' energies from the photon energy. We first create a histogram of the electrons' total kinetic energies, which is equivalent to integrating along the diagonal lines of Figure 5.5. We then negate the values of each bin and add the photon energy 40.814 eV to them. Such a histogram reveals the electronic states of the Xe^{2+} dications, akin to a conventional photoelectron spectrum, and is illustrated in Figure 5.6. Information about the resolved states are given by the peaks' relative intensities and their positions, and are presented in Table 5.2. The full width half maximum (FWHM) of the peaks give information about the resolution of the spectrum, which varies from 0.3 eV to 0.5 eV, as mentioned in Figure 5.6.

⁵The value 5 ns was suggested by R. Feifel, and appears to be valid according to the following analysis.

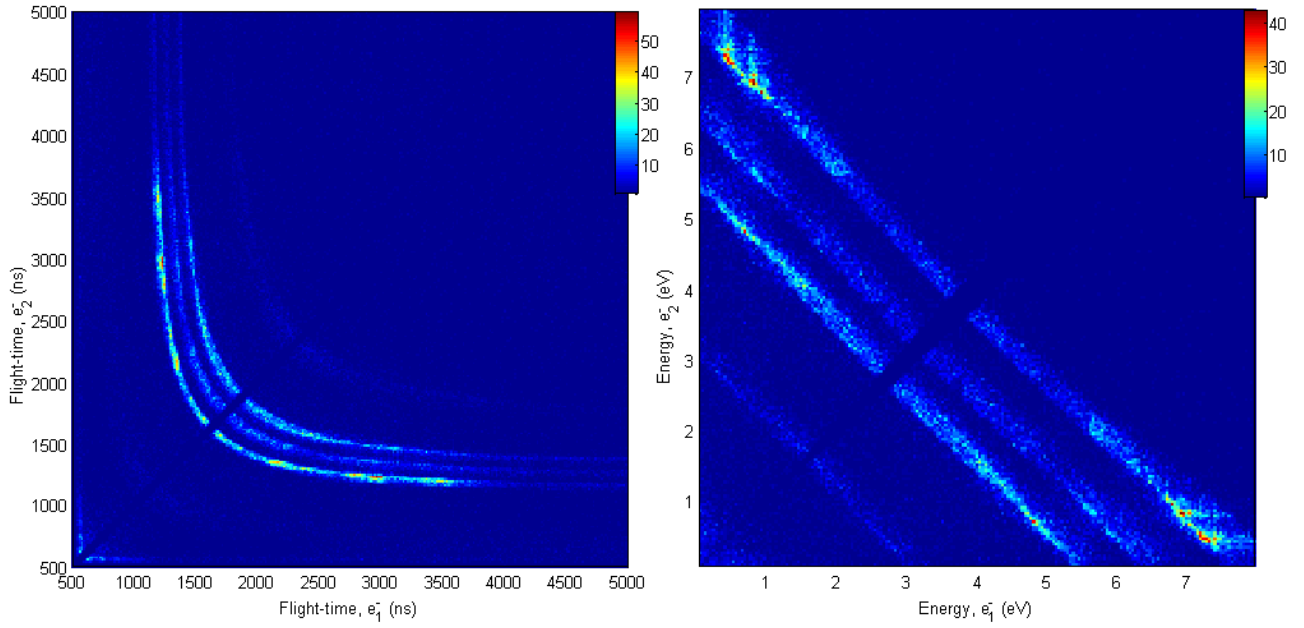


Figure 5.4: Xe^{2+} e_1^- vs e_2^- correlation density map: Time-of-flight axes (left); kinetic energy axes (right). The diagonal dark region where $e_1 \sim e_2$ is due to the dead-time of the TDC card (and also due to configurable software filtering of events, which discards all hits that are closer in time than 50 ns). Note that both graphs are mirrored along the line $e_1 = e_2$.

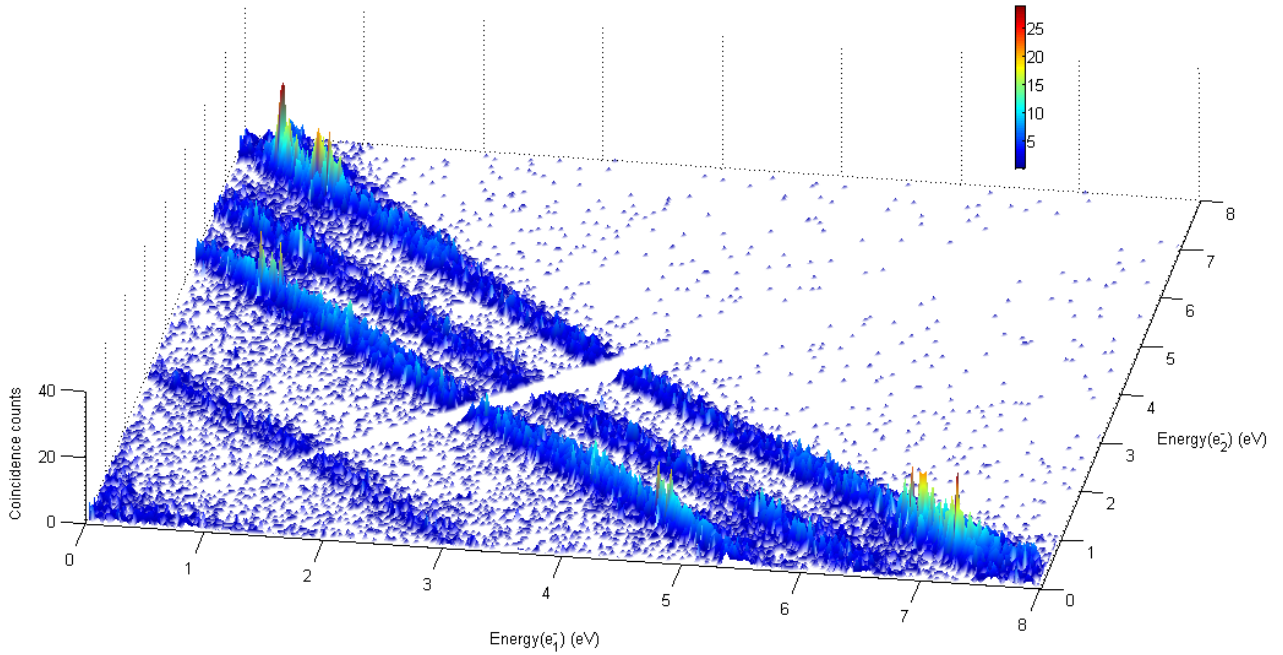


Figure 5.5: Xe^{2+} energy correlation map in 3D view.

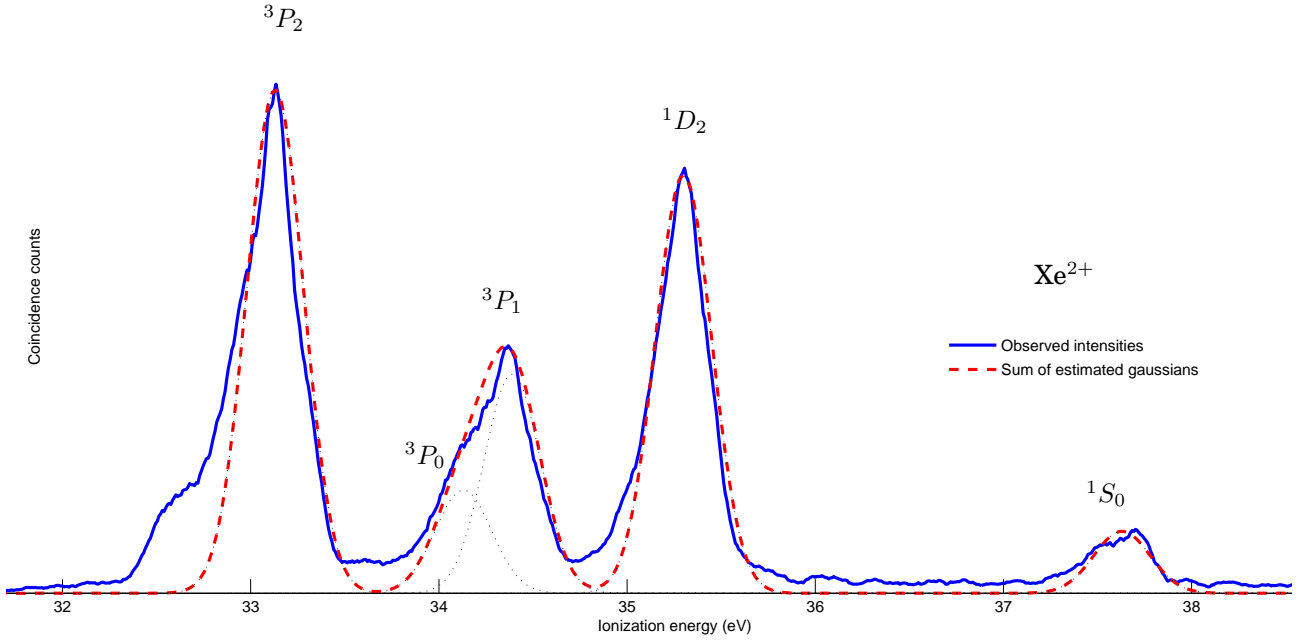


Figure 5.6: Double photoionization spectrum of Xe at 40.814 eV, including assignments for the lines observed.

The solid blue curve shows the double ionization spectrum of Xe. The resolved states are: 3P_2 at 33.1 eV, 3P_1 at 34.4 eV, 1D_2 at 35.3 eV, and 1S_0 at 37.6 eV, with FWHM resolutions of approximately 0.4 eV, 0.3 eV, 0.5 eV, and 0.4 eV respectively. The comparatively low resolution is mainly due to not having optimized the spectrometer much before recording the spectrum, as well as due to a wider monochromator bandwidth at 40.814 eV compared to 21.218 eV.

Note: The 3P_0 and 3P_1 states are not resolved in the present spectrum, but one can see that it is still possible to estimate their positions by estimating two Gaussian curves with FWHM 0.3 eV (the black dotted curves), and summing them: The dotted red curve is the sum of such estimated Gaussians for all states: 3P_2 at 33.13 eV, 3P_0 at 34.13 eV, 3P_1 at 34.39 eV, 1D_2 at 35.30 eV, and 1S_0 at 37.63 eV, with 0.3 eV FWHM resolution.

Table 5.2: Double ionization energies (DIP) and intensities for the Xe^{2+} states observed; present study (pres.) compared to literature values (lit.) previously reported by Eland et al. [9]. The 3P_0 state is not clearly resolved in the present study, and its estimated values are listed here for completeness.

State	DIP (eV)		Intensity	
	pres.	lit.	pres.	lit.
3P_2	33.1	33.1	6.0	6.5
$(^3P_0$	34.1	34.2	1.2	1.1)
3P_1	34.4	34.4	2.6	2.3
1D_2	35.3	35.3	5	5
1S_0	37.6	37.6	0.7	0.8

Chapter 6

Conclusions and discussion

The spectra acquired and the analysis of them as presented in Chapter 5 show vibrational energy levels of various electronic states of O_2^+ as well as electronic states of Xe^{2+} in accordance with previous literature values. This demonstrates that the software which I have developed can indeed be used for its intended purpose: acquisition of TOF-PEPECO spectra.¹

6.1 Improvements

There are several ways to improve the TDC module and the DAQ software further.

6.1.1 TDC module

It is possible to make a minor modification to the TDC card so that it sends interrupt signals on hit detection. If hardware interrupts is to be used in the future, then the timer callback function should be modified slightly and become the hardware interrupt callback function. This would optimize the processor utilization even further, but it is not really necessary for the spectrometers when the lamp pulse rate is essentially constant.

6.1.2 Server

The server could be rewritten in C and run as a daemon or service, instead of running through the quite slow Ruby interpreter. Until now, the response time of the server when a command is given

¹In fact it can be used to acquire TOF-PE(n)CO spectra, where n is any positive integer ($n \leq 5$ in this version of the software, but it can be increased up to the TDC card's maximum number of hits per channel).

by the client has not been an issue, but for certain scenarios on the laser system, measurements will be performed in periods of one second each, and the delay when restarting the measurement is approximately 50 ms (plus an additional 50 ms before the client receives confirmation), which means that 5–10% of the total acquisition time in an experiment is wasted. The protocol can be improved as well, e.g. the commands for reading data from the server has error checking routines that can be removed to reduce the time spent on communication between the server and client.²

6.1.3 Client

The client does not contain all necessary functionality to completely replace the old software in all experiments yet. It needs e.g. some code for analyzing the ion channel(s) in PEPEPICO measurements. Furthermore, certain features would be useful to add for the graphical spectra:

- The ability to study reference spectra in a series of measurements;
- Automatic calculation of the area below a peak, as well as the FWHM;
- Automatic calibration of the energy and time offsets, where the user only needs to mark peaks in a reference spectrum and the program performs the calculations required;

²The TCP/IP protocol already corrects communication errors on the network, and the other possible sources of error comes from incorrect client behavior, which should not exist in the final version.

Initially, the client made use of a simple OpenGL spectrum plotting module which I wrote myself. This module was not as advanced as the Gnuplot library which later on was included instead. The OpenGL plotting module however does not occupy the CPU as much as Gnuplot does, especially if the computer has a GPU³ with OpenGL acceleration. It could be improved with functionality for zooming and panning in the spectra, like Gnuplot already has, and then become the default plotting module in the client.

The data format in coincidence data files saved by the client are twice as large as they need to be, as mentioned in Section 4.3. If one does not care about backward compatibility, this can be fixed with a change of a format specification in the routine that saves the files, as well as a similar change in the routines for loading coincidence files.

6.1.4 Position sensitivity

J. H. D. Eland, who designed the electron detector used in the spectrometers, implemented from the beginning the possibility of obtaining not only a read-out signal, but also position sensitive signals (X, Y) by using the principle of delay line wires. I.e. depending on where on the detector a hit occurs, signal pulses in the X- and Y-channels will have different delays, indicating the (X, Y)-coordinate of the hit. Since this particular delay detector had never been tested in its position sensitivity mode at Oxford, parts of the original plan of my thesis were to explore it in collaboration with Fabian Österdahl. Test experiments revealed that these position sensitive pulses are in fact very weak, and that specially designed electronics would be needed for analyzing them.⁴ Knowing where the

electrons hit the detector would facilitate optimization of the spectrometer; electrons should hit the detector's center for best performance. To further examine the detector and enable the position sensitivity might improve the resolution of the spectrometer. Only minor modifications to the DAQ software are required to make use of the hits' position data.

6.2 Alternatives

Instead of relying on the TDC8 ISA model of the TDC card, one could replace it with newer models, such as the PCI versions. Since this gives no real improvement except for the possibility of using a less expensive motherboard in the computer, the TDC8 ISA is however sufficient at present. There are certain disadvantages with the present TDC card however, e.g. the double-hit resolution is approximately up to 20 ns, which means that if two electrons reach the detector with approximately the same flight-time, they will be registered by the TDC card as just one electron hit.

Instead of relying on CFDs and a TDC card for timing, one can make use of fast analog-to-digital converters (ADCs) for sampling the (pre-amplified) signals from the electron detector, and e.g. perform pulse detection algorithmically in a fast field-programmable gate array (FPGA) or similar circuit. Double hits occurring very closely in time can be detected more easily in this way. Such a solution would require a lot of work though,⁵ and the improvements to the system would only be marginal for present experiments, since the electron pairs released in a double ionization share the energy randomly between them, and thus the flight-times will most often be unequal.

³GPU = Graphics Processing Unit.

⁴The electronics for analyzing the position of electron hits were originally supposed to be built in cooperation with F. Österdahl as a related project, but the focus was changed, due to his decease. Prototype software has however been written by myself with basic functionality for calculating and showing electron hit positions without the need for these electronics (instead relying on the use of existing electronic equipment in the laboratory). This software has not been able to be tested so far, due to problems which the detector's position-sensitivity mode revealed after the software was written. The detector needs to be checked more thoroughly for errors by an expert, and at present time it

is unknown when it will be functioning well enough to test the software.

⁵There are similar products available on the market, e.g. the UHAB – *Ultra High-Speed Acquisition Board* from BitSim AB [23].

Appendix A

Details of the spectrometer calibration procedure

Known binding energies of certain states in a sample gas (in this case the vibrational energy levels of various electronic states in O_2^+) are used to calculate expected kinetic energies E_{kin} of the electrons released through photoionization, and their associated time-of-flight spectrum. The calculation of the values of E_{kin} for the various states used in the following calculations is discussed more rigorously in Appendix B.

Slow electrons are best suited for determining the energy offset,¹ and the B -state vibrations are therefore used for calculation of E_0 . Fast electrons are more sensitive to the time offset than slow electrons, and the X -state vibrations are therefore used to calculate the t_0 -parameter.

The values of E_0 , t_0 and D – and their uncertainties – are all correlated. The value of D has previously been determined in other calibrations of the spectrometer; it is here assumed to be an exact constant² (given by R. Feifel): $3438 \text{ ns} \cdot (\text{eV})^{-1/2}$. The numerical values of E_0 and t_0 are calculated in the following way (here using data from the first run of the molecular oxygen spectrum in Section 5.1 – the same procedure has been done for the second run):

1. Initially, t_0 is assumed to be exactly 0 ns ($\sigma_t = 0$ ns).
2. A first estimate of the numerical value of E_0 is calculated from equation (3.1), using the t -values of the spectral lines in Figure 5.3 on page 26 corresponding to the B -state vibrational energy levels:

$$E_0 = D^2/(t + t_0)^2 - E_{kin}, \quad (\text{A.1})$$

and the uncertainty of the E_0 -value (σ_{E_0}) is given by the error propagation formula for equation (A.1):

$$\sigma_{E_0}^2 = \left(\left| \frac{\partial E_0}{\partial t} \right| \cdot \sigma_t \right)^2 + \left(\left| \frac{\partial E_0}{\partial t_0} \right| \cdot \sigma_{t_0} \right)^2 + \left(\left| \frac{\partial E_0}{\partial E_{kin}} \right| \cdot \sigma_{E_{kin}} \right)^2, \quad (\text{A.2})$$

where

$$\left| \frac{\partial E_0}{\partial t} \right| = \left| \frac{\partial E_0}{\partial t_0} \right| = \frac{2D^2}{(t + t_0)^3} \text{ and } \left| \frac{\partial E_0}{\partial E_{kin}} \right| = 1.$$

¹From (A.1) we have $\partial E_0/\partial t_0 = -2D^2/(t + t_0)^3$. For slow electrons, $t \gg t_0$, so $t + t_0 \approx t$, and the correlation between E_0 and t_0 is thus less noticeable than for fast electrons.

²One could add terms for σ_D (uncertainty in D) to equations (A.2) and (A.4). There would also have to be terms for the correlation between the uncertainties of D and E_0 as well as D and t_0 . However, we here assume that the uncertainty in D is negligible.

The values of σ_t are estimated by reflecting on how accurately the peaks in the spectrum can be located graphically, and are set here to 2–5 ns. The uncertainty σ_{t_0} is initially set to 0 in the calculation.³ The values of E_{kin} are calculated using equation (B.8), and $\sigma_{E_{kin}}$ using equation (B.9).⁴ The values of $2D^2/(t+t_0)^3$ are less than $0.001 \text{ eV} \cdot (\text{ns})^{-1}$ for all t -values. The individual terms in equation (A.2) have magnitudes in the range of 5×10^{-7} to $3 \times 10^{-5} (\text{eV})^2$, and the major contributions come from the uncertainties $\sigma_{E_{kin}}$, which are taken from the literature, and therefore difficult to optimize further.

3. A weighted average⁵ and its uncertainty is calculated for the values of E_0 obtained in the previous step. These weighted averages are used as numerical values in the following calculations. The first such values are $E_0 = 0.556 \text{ eV}$, and $\sigma_{E_0} = 0.006 \text{ eV}$ (more decimals are used in the actual calculations).
4. The next estimate of the numerical t_0 -value is calculated from equation (3.1), now using the t -values of the spectral lines in Figure 5.2 on page 26 corresponding to the X -state vibrational energy levels (and the corresponding theoretical values of E_{kin}):

$$t_0 = D / (E_{kin} + E_0)^{1/2} - t. \quad (\text{A.3})$$

The uncertainty σ_{t_0} is for the following iterations calculated with the formula:

$$\sigma_{t_0}^2 = \left(\left| \frac{\partial t_0}{\partial t} \right| \cdot \sigma_t \right)^2 + \left(\left| \frac{\partial t_0}{\partial E_0} \right| \cdot \sigma_{E_0} \right)^2 + \left(\left| \frac{\partial t_0}{\partial E_{kin}} \right| \cdot \sigma_{E_{kin}} \right)^2, \quad (\text{A.4})$$

where

$$\left| \frac{\partial t_0}{\partial E_0} \right| = \left| \frac{\partial t_0}{\partial E_{kin}} \right| = \frac{D}{2(E_{kin} + E_0)^{3/2}} \text{ and } \left| \frac{\partial t_0}{\partial t} \right| = 1.$$

The values of σ_t are set to 1 ns since the X -states' peaks are much narrower than the B -states' peaks, so it is easier to locate the centroids. The numerical values of E_0 and σ_{E_0} from step 3 are used here. The values of $D/2(E_{kin} + E_0)^{3/2}$ are less than $67 (\text{eV})^{-1} \cdot \text{ns}$ for all t -values. The individual terms in equation (A.2) have magnitudes in the range 0.09 to 0.17 ns, except for the terms $(\left| \frac{\partial E_0}{\partial t} \right| \cdot \sigma_t)^2$, all of which are equal to 1 ns and completely dominated by the uncertainties σ_t of the spectral lines.⁶

5. The values of t_0 are combined into a weighted average like E_0 were in step 3. The first such values are $t_0 = 42.2 \text{ ns}$, and $\sigma_{t_0} = 1.1 \text{ ns}$, which are for the following iterations used in the formulas.

The steps 2–5 are repeated over and over again, each time inserting the previously calculated values in the formulas. The Matlab code used for the actual calculation can be studied in Listing F.2. From the second iteration, we get: $E_0 = 0.520 \pm 0.006 \text{ eV}$; $t_0 = 44.5 \pm 1.1 \text{ ns}$. After a few iterations, the numerical values stabilize:⁷ $E_0 = 0.518 \pm 0.006 \text{ eV}$ and $t_0 = 44.6 \pm 1.1 \text{ ns}$.

³The value will increase in the calculation for the next iteration, and it converges to an upper limit after a few iterations.

⁴In which $\sigma_{h\nu} = 0.001 \text{ eV}$ is used.

⁵Weighted average of values x_i with individual uncertainty σ_i is given by $x_{avg} = \sum_i (w_i x_i) / \sum_i (w_i)$, where $w_i = 1/\sigma_i^2$. The uncertainty of the weighted average is given by $\sigma_{avg} = (\sum_i w_i)^{-1/2}$ [24, ch.7].

⁶If we could locate the lines with down to 1/10 of the present uncertainty (better resolution than this would be unnecessary due to the magnitude of the other terms), the t_0 -value could be determined with slightly higher certainty. One can make the recorded spectrum sharper e.g. by carefully adjusting the position of the permanent magnet or the gas inlet needle relative to the incoming photon beam, so that the interaction between the gas and the light occurs at an optimal location in the magnetic field. Another analytical method of increasing the resolution has been thought of, but not attempted for this spectrometer: to deconvolve the spectrum with the shape of the lamp pulse. The error propagation formula for the time-to-energy conversion formula (3.1) is very similar to equation (A.2), and an uncertain value of t_0 will highly influence the result for electrons with high energy.

⁷All visible decimals stop changing between the iterations after a while, when studying the numbers with 15 visible digits.

Appendix B

Spectral lines of O_2^+ photoelectron spectra

Here we shall examine in some detail the conventional photoelectron spectrum of molecular oxygen which is well-known and which was found to be well suited for energy calibration of the spectrometer. A diatomic molecule has more degrees of freedom (where energy can be “stored”) than a single atom: The nuclei can vibrate back and forth relative to their equilibrium distance, and the molecule can rotate about its center of mass. Electron spectra of diatomic molecules can therefore be expected to reflect spectral lines corresponding to a superposition of the energy levels given by the electronic, vibrational and rotational states [7, p.204]. The rotations are not resolved in the present study (and are in fact even difficult to resolve in conventional photoelectron spectrometers as e.g. developed by K. Siegbahn and coworkers in Uppsala, since their magnitudes (μeV) are much less compared to the energies of the electronic and vibrational levels).

The spectral lines in the O_2^+ photoelectron spectra are expected to correspond to the possible values of

$$T = T_e + G, \quad (\text{B.1})$$

where T_e denotes the *electronic energy* [7, p.204], the energy at the bottom of the potential curve in Figure B.1, and G corresponds to the lines associated with vibrational energy levels which we shall discuss in what follows [7, p.204].

The potential energy V , which is a function of the distance r between the nuclei in a diatomic molecule, can be quite accurately approximated by the so-called *Morse potential* [7, pp.186–187] (see Figure B.1):

$$V(r) = D_e \left(1 - e^{-\alpha(r-r_e)} \right)^2, \quad (\text{B.2})$$

where r_e is the equilibrium distance, D_e is the dissociation energy (i.e. the work required to separate the atoms), and α is a shape-parameter. Since the vibrations of the nuclei relative to each other will perform anharmonic oscillations, the potential vibrational energy can be expressed as a Taylor expansion:

$$E(x) = E_0 + x \left. \frac{dE}{dx} \right|_{r_e} + \frac{x^2}{2!} \left. \frac{d^2E}{dx^2} \right|_{r_e} + \frac{x^3}{3!} \left. \frac{d^3E}{dx^3} \right|_{r_e} + \dots, \quad (\text{B.3})$$

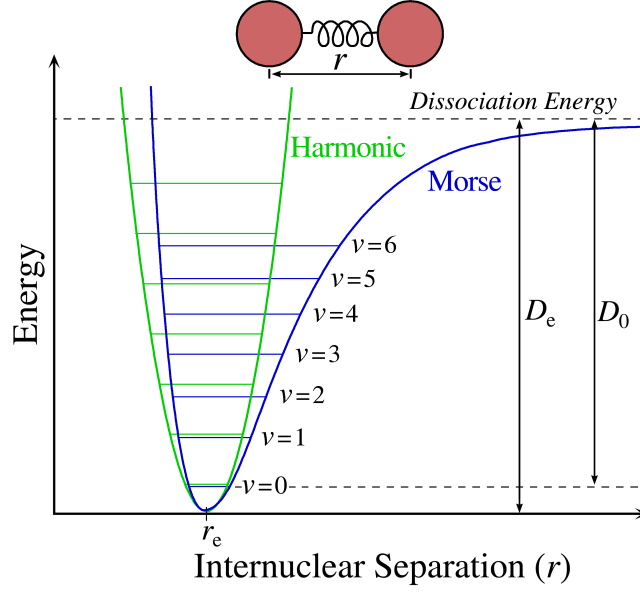


Figure B.1: The Morse potential.

The potential energy E as a function of the distance r between the nuclei in a diatomic molecule. The function for the Harmonic potential is: $E(r) = k(r - r_e)^2/2$, where k is a shape parameter. The Morse potential is given by formula (B.2). The illustration is taken from [25].

which, for the Morse function, reads as:

$$E(x) = \frac{1}{2}kx^2 + \frac{7}{12}\alpha^4 D_e x^3.$$

The energy eigenvalues E_{vib} for the potential vibrational energies are:

$$E_{vib} = h\nu \left(\nu + \frac{1}{2} \right) - h\nu x \left(\nu + \frac{1}{2} \right)^2, \quad (\text{B.4})$$

where, $h\nu$ is called the *harmonicity constant* and $h\nu x$ is called the *anharmonicity constant*, both of which can be experimentally determined. Within spectroscopy however, term values are normally given by the expression:

$$G(\nu) = \omega_e \left(\nu + \frac{1}{2} \right) - \omega_e x_e \left(\nu + \frac{1}{2} \right)^2, \quad (\text{B.5})$$

where, ω_e and $\omega_e x_e$ are vibrational constants expressed in units¹ cm^{-1} relative to the potential curve's minimum, T_e [7, pp.190–192].

Often T_e , ω_e and $\omega_e x_e$ can be found in the literature for various electronic states as determined by other spectroscopy methods (e.g. high-resolution optical spectroscopy). Therefore we can make use of the following equation in order to describe an electronic state with vibrational sublevels in a molecule:

$$T(\nu) = T_e + \omega_e \left(\nu + \frac{1}{2} \right) - \omega_e x_e \left(\nu + \frac{1}{2} \right)^2. \quad (\text{B.6})$$

Theoretical kinetic energies for the possible values of the term value T when varying ν for the different vibrations of the X - and B -state correspond to flight-time peaks observed in the spectra in Figure 5.1 on

¹1 eV = 8065.6 cm^{-1}

Table B.1: *Electron configuration of O_2^+ . Only the states referred to in Listing F.2 are displayed here [26, 27].*

State	E_I (eV)	ω_e (cm ⁻¹)	$\omega_e x_e$ (cm ⁻¹)
$X^2\Pi_g$	12.071 ± 0.005	1904.7 ± 0.7	16.25 ± 0.09
$b^4\Sigma_g^-$	18.171 ± 0.003	1193 ± 4	16.9 ± 0.5
$B^2\Sigma_g^-$	20.296 ± 0.004	1155 ± 8	22.2 ± 0.5

page 26.²

The calculation is performed with numerical values from Table B.1, and a Matlab code as presented in Listing F.2. The values of T_e are calculated with the formula:

$$T_e = E_I - \frac{1}{2}\omega_e + \frac{1}{4}\omega_e x_e, \quad (\text{B.7})$$

which in fact means that the expression for T in the Matlab code is:

$$T(\nu) = E_I + \omega_e \cdot \nu - \omega_e x_e (\nu^2 + 1),$$

and the uncertainty is given by:

$$\sigma_T(\nu) = \left\{ (\sigma_{E_I})^2 + (\sigma_{\omega_e} \cdot \nu)^2 + (\sigma_{\omega_e x_e} [\nu^2 + 1])^2 \right\}^{1/2}.$$

The numerical values of E_I , ω_e and $\omega_e x_e$ used in the calculation are presented in Table B.1.

In photoelectron spectra the expected lines for kinetic energies of the photoelectrons are hence given by:

$$E_{kin} = h\nu - T(\nu), \quad (\text{B.8})$$

with uncertainties:

$$\sigma_{E_{kin}}(\nu) = \left\{ (\sigma_{h\nu})^2 + [\sigma_T(\nu)]^2 \right\}^{1/2}. \quad (\text{B.9})$$

These expression are used to calculate the values of E_{kin} and $\sigma_{E_{kin}}$ used in in formula (A.1) for determining the value of E_0 and formula (A.3) for determining the value of t_0 , as well as the formulas for determining their uncertainties. The calibration is also discussed in Section 5.1.

²Note that $h\nu$ is the photon energy and ν in (B.6) is a quantum number.

Appendix C

TDC8 ISA and alternative timing cards

The TDC card¹ is a prototype which was developed by J. H. D. Eland, Oxford University, during the 1990s. It utilizes the Time-to-Digital Converter chip *LeCroy MTD133B*, which has a time resolution of 500 ps/channel, and a maximum time range of 32 μ s [28]. It supports both *common start* and *common stop* mode of operation.²

Modern derivatives of the TDC card are sold by the company *RoentDek*³ *Handels GmbH*, Germany: *TDC8 (ISA)*, *TDC8 PCI* and *TDC8 PCI2*. The latter two use the PCI bus instead of the ISA bus [30]. There is also a card called *TDC8 HP* (also for the PCI bus). Comparisons of the specifications given by RoentDek are listed in Table C.1 below.

There are three different prototypes for the TDC8 ISA card available at AlbaNova, two of which are used in the old computers for the two spectrometers, and the third is now in use in a new computer set-up, which was built by myself.

¹Actually, there are three cards at AlbaNova. There are two used by the old computers – one for each spectrometer. The third card has been used by myself during the work for this thesis, on a different computer. There are minor differences between the three cards, but they are more or less the same.

²For a description of the differences between *common start* and *common stop*, see [29].

³Website: <http://www.roentdek.com>

Table C.1: Comparison of TDC cards sold by RoentDek Handels GmbH [30, 31].

	TDC8 ISA(PCI)	TDC8 HP
<i>Resolution</i>	500 ps	< 100 ps
<i>Double hit resolution / dead-time</i>	< 20 ns (typ. 10 ns)	< 10 ns (typ. < 5 ns)
<i>Max hits / trigger</i>	16	unlimited
<i>Data rate</i>	18(30) k events/s/ch/hit	4 Mhits/s

Appendix D

Technical details for the TDC module

D.1 Source code overview

In the file `tdc.c`, the following statements in the kernel module tell the kernel which functions to call when the module is loaded/unloaded:

```
module_init(tdc_init_module);  
module_exit(tdc_cleanup_module);
```

tdc_init_module checks the system timer resolution to verify that a real-time kernel is running; registers a *character device*¹ for the driver, with a major² and minor device number, which uniquely identifies the device in the kernel, and gets a major and minor device number from the kernel (unless a module parameter is given which specifies a major number); allocates memory for buffers and variables needed by the module; initializes queues and mutual exclusion variables used in the module; sets up a virtual node at `/proc/tdc_measurement` in the file system.³

tdc_cleanup_module Releases all memory that was allocated in `tdc_init_module`; deletes character device; unregisters the character device region (major and minor numbers); deletes the node at `/proc/tdc_measurement`.

The character device registered in `tdc_init_module` is defined as a structure which is required for the kernel to know which module functions to call when certain events occur in the system:

```
struct file_operations tdc_fops = {  
    .owner = THIS_MODULE,  
    .read = tdc_read,  
    .write = tdc_write,  
    .open = tdc_open,  
    .release = tdc_release  
    // optionally, additional functions can be declared, for i.e. polling, ioctl, etc.  
};
```

¹Fundamentally, there are three types of devices in Linux: *char*, *block*, and *network* devices. Character device drivers are the most common type of device driver in the Linux kernel. They can transfer data directly to and from user processes byte by byte, and most often implement the *open*, *close*, *read*, and *write* system calls [19, pp.5–7].

²Determined automatically by the kernel unless specified in a module parameter. Some numbers are reserved for the most common devices, and cannot be used for the TDC card [19, p.46].

³When the user or a program opens this virtual file and reads as a text-file, the module's function `tdc_proc_measurement` is called, which displays a lot of detailed information about the current status of the TDC card and the kernel module, as well as statistics about any ongoing data acquisition.

The driver can in principle be thought of as a special file (from the perspective of a user process). When the TDC device is opened, the `tdc_open` function is thus called in a special way by the Linux kernel. When it is closed, the `tdc_release` function is called, and so on. The following tasks are performed by these functions (outlined briefly):

tdc_open Called when a process opens the device, for instance by the Ruby command:

```
tdc = File.open("/dev/tdc", "rw")
```

If the device is opened in read mode by more than two processes at once, the error code `-EBUSY`, “Device is busy”⁴ will be returned – only one process can be allowed to read the FIFO buffer: data read is immediately removed from the buffer. (Multiple writing processes can open the device at the same time though.)

tdc_read Called when a process tries to read data from the device. For instance when the following Ruby code is executed:

```
data = tdc.read() # read out all events from FIFO buffer as they arrive
```

The function reads data from the FIFO-buffer if there is any data there, otherwise tries to put the reading process to sleep until data are available in the buffer, or until the acquisition is ended.

tdc_write Called when a process tries to write data to the device; i.e. for each of the following Ruby commands when they are executed:

```
tdc.write("set_time_range_0_10000") # setup time window (units are half nanoseconds)
tdc.write("clear") # clear FIFO buffer etc
tdc.write("start") # start data acquisition
# ...
tdc.write("stop") # end data acquisition
```

This is how to control the TDC module from user programs.⁵ Special commands are available for changing TDC card settings, starting/stopping data acquisition, etc. See Section D.5 for a detailed list of the commands and how to use them.

tdc_release Called when a process closes the device after it has finished reading or writing from/to it. For example, when the following Ruby command is executed:

```
tdc.close()
```

Most of the code for controlling the TDC card has been abstracted into descriptive functions defined in the file `TDC_Device.c`. In particular, the function `tdc_timer_callback` is noteworthy. This function is called on each⁶ timer event – starting when the command `start` has been given to the device driver – and it checks if the TDC card has received any events. If so, then the events are decoded (by the function `tdc_decode_events`) and stored in a FIFO buffer (by the function `tdc_add_hits_to_fifo`) in the computer’s memory. Data in the FIFO buffer is stored in a way which is described in Section D.3.

⁴`EBUSY = 16`; from definitions of system errors in header file `errno.h` in source for Linux 2.6.

⁵Another popular method is to use *IOCTL* codes and functions for controlling the device [19, ch.6]. This is not supported in the present version of the driver, but can be implemented easily in the code. *IOCTL* do not make use of strings – instead so-called *magic numbers* represent each command – and thus the technique requires slightly less processing, potentially improving the response time of the TDC module with a few clock-cycles.

⁶I.e. 4000 times/s if the command `set_trigger_rate_hz 4000` has been given.

D.2 Module parameters

There are several module parameters specified in `tdc.h` and `tdc.c`:

tdc_major (integer) The major device number, defaults to 0, which results in the device number being automatically set to an available major device number when the module is loaded.

tdc_nr_devs (integer) The number of TDC cards installed in the computer. Currently defaults to 1 and supports no more in the present version of the code. The code can in principle be extended to support multiple TDC cards operating at the same time, which would allow for even more channels to be used than what is available on one card.

tdc_base_address (integer) The base address/port of the TDC card in the computer's device space. Defaults to `0x320`, as defined by `TDC_BASEPORT` in `tdc_common.h`. The address can be changed on the TDC cards with switches, and must match this setting in the driver.

tdc_buffer_size (integer) The size (in bytes) of the FIFO-buffer, which stores the detected hits in (the computer's) memory until a process reads out data from the device file. Defaults to 1 megabytes, as defined by `TDC_BUFFER_SIZE` in `tdc_common.h`. A large buffer guarantees that no hits will be missed as a result of buffer overflow.⁷ If the computer has very little available RAM-memory, the buffer size can be decreased. As a minimum, 1 kilobyte is recommended. No more than 1 megabyte should ever be necessary with the present software, as empirical tests indicate that even under heavy system load, the buffer has never been filled up more than a few percent.

The module parameters are easily modified without requiring recompilation of the module source code. In order to change a parameter, the module needs only to be loaded with a special argument appended to the command, as described in 4.2.2.1 [32].

D.3 The FIFO-buffer

The data in the FIFO buffer is stored in a binary format, which can be decoded in the way described in Table D.1 on page x.

At byte `3X`, the decoding starts over for the next COM signal's hits: Byte `3X` gives the number of hits for the second COM signal, byte `3X + 1` gives the channel number for the first hit belonging to the second COM signal, etc.

D.4 Scripts for loading / unloading the LKM

The Bash script `tdc_load` performs the following steps to insert the LKM into the Linux kernel in a useful way:

1. Find out which `$group` is supposed⁸ to own the file node for step 5:

```
if grep -q '^staff:' /etc/group; then group="staff" else group="wheel" fi
```

⁷If the process which acquires data from the device file does not read out the buffer at least as fast as the buffer is filled with new data from the TDC card, the buffer will eventually become full, at which point data will be lost. This can occur only if the hit-rate on the TDC card is extremely high, and if the computer or data acquisition software is really slow, which is not the case.

⁸This varies among Linux distributions [19, ch.3].

Table D.1: Binary format of the FIFO-buffer

Byte	Data
0	X = Number of hits detected in conjunction with the first COM signal (1–16); if no hits were detected, no data are saved in the FIFO-buffer.
FOR $n = 0, 1, \dots, X - 1$, DO:	
$3n + 1$	The channel number for the n th hit. Channels are numbered from 0 to 7.
$3n + 2$ & $3n + 3$	The 16-bit delay value for the n th detected hit of the first COM signal; the time difference between the COM signal and the hit; in units of 0.5 ns. The format is <i>little-endian</i> : First byte gives the lowest 8 bits, and second byte gives the highest 8 bits.
LOOP (UNTIL X HITS HAS BEEN DECODED)	

2. Load the module ($\$*$ is replaced with any arguments passed to the file `tdc_load`):

```
insmod tdcmod.ko $*
```

3. Retrieve the `$major` number of the device (automatically set by the kernel unless the parameter `tdc_major` is changed), by reading information about `tdcmod` in the virtual file `/proc/devices`.
4. Create a node for a character device in the file system at `/dev/tdc`: `mknod /dev/tdc c $major 0`, where `$major` is the number retrieved in step 3.
5. Give the node relevant ownership and permissions for read/write access:

```
chgrp $group /dev/tdc
chmod 666 /dev/tdc
```

The Bash script `tdc_unload` unloads the LKM from the Linux kernel. It reverts the processes performed in `tdc_load`:

1. Execute the command `rmmod $module $*` to unload the kernel module.
2. Remove the node `/dev/tdc` from the file system.

D.5 Commanding the TDC module

The TDC module must be loaded into the kernel, and a node representing it must be available at `/dev/tdc`, as described in 4.2.2.1. Commands are written to `/dev/tdc` as if writing them to a regular text file. Some of the commands take one or two parameters. If a parameter is to be given, a single space character separates it from the command, and parameters are also separated by a single space.

For example, if the trigger rate needs to be changed quickly, the following Bash shell command can be executed:

```
echo "set_trigger_rate_hz_4000" > /dev/tdc
```

Here, the command is `set_trigger_rate_hz` and the parameter is 4000. A single command can be given at a time; if multiple commands are needed, they must be executed in separate write operations.⁹

D.5.1 Command list

Parameters in brackets are optional – if they are omitted the default value is used. Parameters must be natural numbers. If there is a failure in performing any of the commands, the write operation will return with error code `-EFAULT`, “Bad address”. If any parameter passed with the command does not pass validation, the write operation will fail with error code `-EINVAL`, “Invalid argument”.¹⁰

More details can be found in the file `tdc.c` in the `tdc_write` function, and also in the file `TDC_Device.c` where most of the code is located for the functions mentioned below.

start [*N*] Start data acquisition (measurement), i.e. start the timer, which periodically checks the TDC card for new events and saves them to the FIFO-buffer in memory as they arrive – asynchronously and in parallel with other processes that are running. The parameter $N \in [0, 4294967295]$ gives an upper limit to the number of trigger signals to check, and if a number greater than 0 is entered, the measurement will automatically stop when the limit is reached. If the measurement has been started and is stopped, it will be reset.¹¹ The measurement will fail to start if it has already been started, and is neither paused nor stopped. More details can be found in the source of function `tdc_start_measurement`.

pause Pause data acquisition. During a pause, no events will be detected in the TDC card, and no events will be read into the FIFO-buffer. The timer stops. When paused, the acquisition can be started again with a start command. The counters and information in `/proc/tdc_measurement` will not be reset when the measurement is started again. If no measurement is running, pause will fail.

stop End data acquisition. When acquisition is ended, data will still be available in the FIFO-buffer until it is cleared, or read by another process. The stop command cannot fail, unless using non-blocking writing, in which case it might fail.¹²

clear Clear all data. This empties the FIFO buffer and all measurement information in `/proc/tdc_measurement`. It also terminates any running timers. It can fail only if the timer callback function is executing, and the timer therefore fails to stop.¹³

set_trigger_rate_hz *M* Set timer callback rate – the parameter $M \in [1, 100000]$ – in units of Hz.¹⁴ This

⁹One possible modification to the TDC module is allowing multiple commands in a single write operation. This can be done i.e. by separating them with a newline character, and modifying the `tdc_write` function to split the command string accordingly. However, since there is no need for this functionality in the software developed, the TDC module checks only for one single command on a write operation at present.

¹⁰`EFAULT = 14, EINVAL = 22`; from definitions of system errors in header file `errno.h` in source for Linux 2.6.

¹¹A `clear` command must be explicitly given if the FIFO-buffer should also be emptied. The FIFO-buffer will NOT by default be cleared when the measurement is reset.

¹²If the timer callback function is executed precisely when the stop command is given, the timer will fail to terminate. In this case, the write operation will be put to sleep until the callback function has finished, and the timer is able to terminate. However, if the write operation is performed in non-blocking mode, the error code `-EAGAIN` will be returned if the timer callback is currently executing, since a process must not be put to sleep if it is writing to the device in non-blocking mode [19, ch.6].

¹³This issue has been solved in the stop command, and a similar approach could be used for the clear command. However, the clear command is executed by the DAQ software only upon creation of a new measurement – which cannot be created if a measurement is already running. Thereby this is not really an issue.

¹⁴Empirical tests on the computer set-up indicate that a timer callback rate above 100 kHz occasionally freezes the computer, which is the reason for the artificial limit of the callback rate. This also means that in the data acquisition software, the highest trigger rate possible is 25 kHz at the moment. However, the TDC card which I have tested seems to be reliable only when the trigger rate is lower than approximately 6 kHz; otherwise some signals are missed, and the delay times become inaccurate. The exact limit can be better determined with the other TDC cards in the lab and a good pulse generator, since there are some minor differences between each of the TDC cards.

is the rate at which the timer callback function should be called. This function periodically checks the TDC card for new events, and therefore should be a multiple of the trigger rate (COM signal pulse rate) – approximately 1–4 times the trigger rate is a good value. If an illegal value is given for M , `set_trigger_rate_hz` will fail.

set_trigger_period_ns N Set the period/interval, in units of ns, for the timer callback function. This command is equivalent to `set_trigger_rate_hz M` if $N = 10^9/M$.

set_com_mode N Configure the TDC card for use with *common stop* (if N is 0) or *common start* (if N is 1) trigger mode. If N is not 0 or 1, the `set_com_mode` will fail.

set_time_range M N Set the time window (in units of 0.5 ns) for TDC card. The first parameter $M \in [0, N]$ sets the minimum time to allow for an event, and the second parameter $N \in [M, 65535]$ sets the maximum time to allow for an event. Detected hits outside this time window will not be registered by the TDC card as events. Thus, the maximum delay from a COM signal to a hit that can be measured is 32767.5 ns. If M or N is missing, `set_time_range` will fail.

set_config N [M] Configure the TDC card.¹⁵ The first parameter, $N \in [0, 65535]$ is the maximum time (default 65535) in units of 0.5 ns, and is equivalent to `t_max` in the command `set_time_range`. The second parameter, $M \in [0, 8]$ is the number of channels to use (default 8).

¹⁵This command is obsolete in the present version of the module and data acquisition software, since newer commands have been introduced with more functionality.

Appendix E

Protocol for Client–Server communication

The protocol is modeled in the data acquisition software in the file `DAQ-suite/model/protocol.rb`, which is “self-documenting” – if the file is executed by the Ruby interpreter, it will describe all the commands and how to use them, and write this text into the file `DAQ-suite/doc/protocol_0.7.txt` (if the current protocol version is 0.7) as well as to the console. Future changes to the protocol can therefore automatically be reflected in the documentation easily. The special protocol command `HELP` can also be used to learn more about the different commands available on the server, if one is developing a special client software on a different machine.

E.1 The protocol at present

Note: The binary data transmitted in some commands have the same format as the data in the FIFO-buffer of the TDC module (see Section D.3).

```
ToF Client/Server protocol version 0.7
Development version, revision 553
-----
```

Description of symbols in this text:

```
$VAR           = some variable, substitute with appropriate text
(Action)       = Some action, Action, should happen
Server: text1  = Server sends the string "text1" to client
Client: text2  = Client sends the string "text2" to server
```

Note: After the string, there is a newline to indicate end of message.
This should be ignored by client, and also by server.
However, when sending binary data, newline characters might be part of the data stream. In this case, the server will first tell the client how many bytes it is going to send, so the client can parse the data correctly.

The connection should be made in Telnet mode, and most of the commands and responses are possible to easily understand just by reading them from a console.

For testing purposes, you can therefore connect to the server using a telnet client, and type commands and read responses. (Note that binary responses will be more difficult to understand.)

Typical chain of events:

1. (Client connects to server)
2. Server: HELLO, Welcome to ToF server
3. (Client disconnects if greeting not correct)
4. Server: #Protocol version: 0.7.553
5. (Client verifies protocol major version and disconnects if incorrect.)

Note:

The major version is 0.7, and the server is at revision 553.
 (If the server is running on a computer that doesn't have
 subversion installed, the version will be shown as: 0.7.X.)
 Protocol should not change between major versions, even if revisions differ.

However, because of human error, the protocol might accidentally be changed
 between revisions, so if you see a higher revision number than expected,
 and the protocol doesn't work as you expected, something has probably changed,
 and needs to be reversed, or major version should be incremented.

6. Client: \$COMMAND \$PARAMETER_1 \$PARAMETER_2 ... \$PARAMETER_N
- 7a) Server: \$COMMAND ACCEPTED
- 7b) Server: IMPOSSIBLE
- 7c) Server: \$COMMAND RESTRICTED

Note:

The client sends a command, \$COMMAND, to the server, where \$COMMAND
 must be one of the accepted commands described below. The server
 replies according to (7a) if the command is valid.

If the command is invalid, server sleeps for 1 second, and
 then replies according to (7b). (The short sleep is a security
 so the client doesn't just flood the server with invalid commands
 without realizing there is an error.)

If the command is restricted, and the client is not a master client,
 the server replies according to (7c) and does not execute it.

The parameters, which are optional, are separated by space,
 and depend on which \$COMMAND is given. More details are available
 in the section Command reference below.

In this version of the protocol, the client must be certain to
 give valid parameters, since the server does not check them.

Example1: A valid request

 (Client connected to server while no other clients were connected,
 and is therefore master client.)
 Client: START
 Server: START ACCEPTED

Example2: An invalid request

 Client: SELFDESTRUCT
 (approximately 1 sec delay)
 Server: IMPOSSIBLE

Example3: A restricted request

 (Client connected to server while other client(s) were connected,
 and therefore has limited privileges.)
 Client: START
 Server: START RESTRICTED

Note: Listen timeout is 20 seconds. If server waits
 for more than this amount of time, client will lose it's connection.

Command reference:

[*\$PARAMETER*] means that *\$PARAMETER* is not required, and if omitted, its default value will be used instead.

'#' Means that the rest of the line is a comment.

QUIT: Terminate connection with server.

Command procedure:

1. Client: QUIT
2. Server: QUIT ACCEPTED
- # Server awaits next command from client...

CLEAR: Clears the Measurement details. Should be sent before a new measurement is started.

This command is only available to master clients.

Command procedure:

1. Client: CLEAR
2. Server: CLEAR ACCEPTED
3. Server: \$STREAM
- # Server awaits next command from client...

Possible values of \$STREAM (responses):

CLEARED - on success.

TDC_INFO: Show information about the TDC module, details about the data from card, driver, etc.

Command procedure:

1. Client: TDC_INFO
2. Server: TDC_INFO ACCEPTED
- # Server tells stream size (bytes):
3. Server: SEND \$NUM_BYTES_TO_SEND
- # Client acknowledges:
4. Client: GET \$NUM_BYTES_TO_GET
- # Server verifies client response
- # if \$NUM_BYTES_TO_SEND equals \$NUM_BYTES_TO_GET:
5. Server: ACK
6. Server: \$TEXT_STREAM
- # Client should read data from server until
- # it has received \$NUM_BYTES_TO_GET bytes.
7. Client: GOT \$NUM_BYTES_RECEIVED
- # Server verifies client response
- # if \$NUM_BYTES_TO_SEND equals \$NUM_BYTES_RECEIVED:
8. Server: DONE
- # else:
8. Server: FAILED
- # else:
5. Server: ERR
- # Server awaits next command from client...

STATUS: Returns the status of the server. \$STREAM = {the status of the server}.

Command procedure:

1. Client: STATUS
2. Server: STATUS ACCEPTED
3. Server: \$STREAM

Server awaits next command from client...

CONFIGURE: Configure settings for TDC card; time ranges, etc.

This command is only available to master clients.

Command procedure:

```
1. Client: CONFIGURE [$T_MIN] [$T_MAX] [$TRIGGER_RATE]
[$COM_MODE] [$MAX_HITS_PER_CHANNEL]
2. Server: CONFIGURE ACCEPTED
3. Server: $STREAM
# Server awaits next command from client...
```

Possible values of \$STREAM (responses):

```
CONFIGURED          - on success.
FAILED_TO_CONFIGURE - on failure.
```

Parameters for CONFIGURE:

\$PARAMETER_1: [\$T_MIN] (default: 0 (Set by driver))

\$T_MIN can be any integer between 0 and 65535 (unit: 0.5 ns). It configures the minimum valid delay for a hit. The TDC card will discard hits with shorter delay than \$T_MIN.

\$PARAMETER_2: [\$T_MAX] (default: 65535 (Set by driver))

\$T_MAX can be any positive integer between \$T_MIN and 65535 (unit: 0.5 ns). It configures the maximum valid delay for a hit. The TDC card will discard hits with longer delay than \$T_MAX.

\$PARAMETER_3: [\$TRIGGER_RATE] (default: ? (Set by driver))

\$TRIGGER_RATE can be any integer between 1 and 100000 (unit: Hz). It configures how often the driver will check the TDC card for new events. Ideally, it should be a multiple of the lamp pulse frequency (COM pulse rate). Suggestion: 4000 Hz is good if using a laser with pulse rate of 1000 Hz. Too high value will reduce performance. Too low value will increase number of missed events.

\$PARAMETER_4: [\$COM_MODE] (default: COMMON_START (Set by driver))

\$COM_MODE can be either COMMON_STOP or COMMON_START. It configures if the TDC card will use common stop or common start mode for triggering.

\$PARAMETER_5: [\$MAX_HITS_PER_CHANNEL] (default: 16 (Set by driver))

\$MAX_HITS_PER_CHANNEL can be any integer between 1 and 16. It configures how many hits the TDC card will accept per channel for each COM pulse.

START: Starts the data gathering from TDC card.

This command is only available to master clients.

Command procedure:

```
1. Client: START [$NUM_COM_EVENTS]
2. Server: START ACCEPTED
3. Server: $STREAM
# Server awaits next command from client...
```

Possible values of \$STREAM (responses):

ALREADY_STARTED	- on already_started.
STARTED	- on success.
FAILED_TO_START	- on failure.

Parameters for START:

\$PARAMETER_1: [\$NUM_COM_EVENTS] (default: 0)

\$NUM_COM_EVENTS can be any positive integer, or 0 for unlimited. If a limit is set, then the measurement will automatically stop when the number of COM signals reach the limit.

PAUSE: Pauses data gathering from TDC card.

This command is only available to master clients.

Command procedure:

```
1. Client: PAUSE
2. Server: PAUSE ACCEPTED
3. Server: $STREAM
# Server awaits next command from client...
```

Possible values of \$STREAM (responses):

PAUSED	- on success.
--------	---------------

HELP: Show help. HELP \$COMMAND gives help on how to use the command \$COMMAND, if it is a valid command.

Command procedure:

```
1. Client: HELP [$COMMAND]
2. Server: HELP ACCEPTED
# Server tells stream size (bytes):
3. Server: SEND $NUM_BYTES_TO_SEND
# Client acknowledges:
4. Client: GET $NUM_BYTES_TO_GET
# Server verifies client response
# if $NUM_BYTES_TO_SEND equals $NUM_BYTES_TO_GET:
5. Server: ACK
6. Server: $TEXT_STREAM
# Client should read data from server until
# it has received $NUM_BYTES_TO_GET bytes.
7. Client: GOT $NUM_BYTES_RECEIVED
# Server verifies client response
# if $NUM_BYTES_TO_SEND equals $NUM_BYTES_RECEIVED:
8. Server: DONE
# else:
8. Server: FAILED
# else:
5. Server: ERR
# Server awaits next command from client...
```

Parameters for HELP:

\$PARAMETER_1: [\$COMMAND] (default:)

If \$COMMAND is a valid command, HELP \$COMMAND shows detailed help on how to use that command. If \$COMMAND is '*', help for all commands are given. If \$COMMAND is invalid, then HELP \$COMMAND informs that \$COMMAND is not a command.

STOP: Stops data gathering from TDC card.

This command is only available to master clients.

Command procedure:

```
1. Client: STOP
2. Server: STOP ACCEPTED
3. Server: $STREAM
# Server awaits next command from client...
```

Possible values of \$STREAM (responses):
 STOPPED - on success.

READ_BUF: Read data from buffer.

Command procedure:

```
1. Client: READ_BUF [$BUFFER_TYPE] [$OFFSET]
[$NUM_LIMIT]
2. Server: READ_BUF ACCEPTED
# Server tells stream size (bytes):
3. Server: SEND $NUM_BYTES_TO_SEND
# Client acknowledges:
4. Client: GET $NUM_BYTES_TO_GET
# Server verifies client response
# if $NUM_BYTES_TO_SEND equals $NUM_BYTES_TO_GET:
5. Server: ACK
6. Server: $TEXT_STREAM
# Client should read data from server until
# it has received $NUM_BYTES_TO_GET bytes.
7. Client: GOT $NUM_BYTES_RECEIVED
# Server verifies client response
# if $NUM_BYTES_TO_SEND equals $NUM_BYTES_RECEIVED:
8. Server: DONE
# else:
8. Server: FAILED
# else:
5. Server: ERR
# Server awaits next command from client...
```

Parameters for READ_BUF:

\$PARAMETER_1: [\$BUFFER_TYPE] (default: ALL)
 \$BUFFER_TYPE can be any of
 (SINGLES|DOUBLES|TRIPLES|TOF_HISTOGRAM|RAW)

\$PARAMETER_2: [\$OFFSET] (default: 0)
 \$OFFSET tells where to start in buffer. 0 = first
 byte.

\$PARAMETER_3: [\$NUM_LIMIT] (default: {num bytes
 remaining})
 \$NUM_LIMIT tells how many bytes in \$BUFFER_TYPE to
 read from the server, and should be a positive integer if
 you want to read at most \$NUM_LIMIT number of bytes. Note
 that the server might give you less bytes than you asked
 for, if the buffer is not filled up. If you don't give an
 upper limit, the server will send the whole buffer.

TEST: Tests the communication between server and client.

Command procedure:

```
1. Client: TEST
2. Server: TEST ACCEPTED
# Server awaits next command from client...
```

Appendix F

Analysis of spectra with Matlab

Code is presented here, demonstrating how the binary files saved from the data acquisition software can be read into Matlab for analysis and drawing graphs.

Listing F.1: *O₂ state-plot, 1st run*

```
clear all
close all

% parse the binary data from single electron spectrum:
spec_file = './syre-xenon-kalibrering/2008-05-27_193022_single_electrons-elToF.tofspc';
f = fopen(spec_file, 'r');
histo = zeros(1,6000);
dummy = 1;
t_min = 0;
t_max = 0;
while dummy == 1
    [t, dummy] = fread(f, 1, 'uint16', 0, 'ieee-le');
    [y, dummy] = fread(f, 1, 'uint32', 0, 'ieee-le');
    histo(t) = y;
    if min(size(t)) == 0
        break
    end
    t_min = min([t_min t]);
    t_max = max([t_max t]);
end
fclose(f);

t = 500:6000;
y = histo(500:6000);

% assuming poisson distriubution, calculate standard deviation
y_std = sqrt(y);

fig = figure
plot(t,y)
title('O_2,_single_photoelectrons_time-of-flight')
xlabel('Time-of-flight_(ns)')
ylabel('Intensity_(arb._units)')
axis([800 4000 0 2000])
set(fig, 'Position', [0,900,1000,400])
bg_color = [0.7, 0.7, 0.7];
```

```

% X-state
figure
hold on
ax = [1050 1130 100 2000]
% these times are obtained by eye-measure from the figure
X_centroids = [1060 1075 1088.5 1103 1118.5]
jbbfill(t, y+y_std, y-y_std, bg_color, bg_color, 1, 1)
hold on
% Moving Average Filter
y_f = runmean(y, 1, 'zero');
h = plot(t, y_f, 'b-')
set(h, 'LineWidth', 1.5);
xlabel('Time-of-flight_(ns)')
ylabel('Intensity_(arb._units)')
legend('Uncertainty_in_observed_intensity', ...
       'Slightly_noise-filtered_spectrum')
axis(ax)
for i=1:length(X_centroids)
    mark_time(X_centroids(i), sprintf('X_%d:', i-1))
end

% B-state
figure
hold on
ax = [2700 3550 0 250]
axis(ax)
% these times are obtained by eye-measure from the figure
B_centroids = [2795 2970 3161 3357]
jbbfill(t, y+y_std, y-y_std, bg_color, bg_color, 1, 1)
hold on
% Moving Average Filter
y_f = runmean(y, 11, 'zero');
h = plot(t, y_f, 'b-')
set(h, 'LineWidth', 1.5);
xlabel('Time-of-flight_(ns)')
ylabel('Intensity_(arb._units)')
legend('Uncertainty_in_observed_intensity', ...
       'Highly_noise-filtered_spectrum')
axis(ax)
for i=1:length(B_centroids)
    mark_time(B_centroids(i), sprintf('B_%d:', i-1))
end
end

```

Listing F.2: O₂ calibration, 1st run

```

clc, clear all, close all, format long, format compact
%Weighted average:
w_avg = inline('sum(_w.*x_)/_sum(_w_)','x','w');
%Kinetic energy calculated from Time-of-Flight:
E = inline('(D_/_(t+t0)).^2-_E0','t','t0','E0','D');

% -----
% Empirical values:
% -----

% 1 eV = 8065.6 cm^-1 (for conversion from the Table in Phys. Scr. 1)

```

```

% Errors *_err are the estimated uncertainty in graphical determination
% of the centroid positions of the peaks in the O2 spectrum.
% We need to quadratically add these errors to the propagated errors
% in the theoretical peak positions.

```

```

% photon energy (eV):
hnu = 21.2182; u_hnu = 0.001; % acc. to R. Feifel

```

```

% X-state
% Format: Value; Uncertainty; (units: eV)
X.E_0 = 12.071; X.u_E_0 = 0.005;
X.omega_e = 1904.7 / 8065.1; X.u_omega_e = 0.7 / 8065.1;
X.omega_e_x_e = 16.25 / 8065.1; X.u_omega_e_x_e = 0.09 / 8065.1;
% units: ns
X.centroids.t = [1060 1075 1088.5 1103 1118.5]';
X.centroids.t_err = [1 1 1 1 1]';
X.nu = [0:4]';
% b-state
% Format: Value; Uncertainty; (units: eV)
b.E_0 = 18.171; b.u_E_0 = 0.003;
b.omega_e = 1193 / 8065.1; b.u_omega_e = 4 / 8065.1;
b.omega_e_x_e = 16.9 / 8065.1; b.u_omega_e_x_e = 0.5 / 8065.1;
% units: ns
b.centroids.t = [1775 1811 1850 1889 1931]';
b.centroids.t_err = [3 10 3 5 10]';
b.nu = [0:4]';

```

```

% B-state
% Format: Value; Uncertainty; (units: eV)
B.E_0 = 20.296; B.u_E_0 = 0.004;
B.omega_e = 1155 / 8065.1; B.u_omega_e = 8 / 8065.1;
B.omega_e_x_e = 22.2 / 8065.1; B.u_omega_e_x_e = 0.5 / 8065.1;
% units: ns
B.centroids.t = [2795 2970 3161 3357]';
B.centroids.t_err = [2 5 5 5]';
B.nu = [0:3]';

```

```

D = 3438 % length constant.      %short spectrometer => 3438.0
                                %long spectrometer  => 9265.0
u_D = 0 % assume D is 100% certain

```

```

% initial guess
E0_wavg = 0; u_E0_wavg = 0;
t0_wavg = 0; u_t0_wavg = 0;

```

```

values = []
for iteration=1:50

```

```

    % Determine E0
    disp('Determining_E0');
    E_kin = []; u_E_kin = [];
    E_kin_empiric = []; u_E_kin_empiric = [];
    t = []; u_t = [];
    for STATE=[B]
        % -----

```

```

% Theoretical values
% -----
% see Report, eq. ?, derived from expression of T_e
T = STATE.E_0 + STATE.omega_e * STATE.nu - STATE.omega_e_x_e * (STATE.nu.^2 + 1);
u_T= sqrt( STATE.u_E_0^2 + (STATE.u_omega_e*STATE.nu).^2 + (STATE.u_omega_e_x_e*(STATE.nu
.^2+1)).^2 );
E_kin = [E_kin; hnu - T];
u_E_kin = [u_E_kin; sqrt( u_hnu^2 + u_T.^2)];
% -----
% Observered values
% -----
t = [t; STATE.centroids.t];
u_t = [u_t; STATE.centroids.t_err];
end

E0 = (D ./ (t+t0_wavg)).^2 - E_kin
dE0dt = -2*D^2 ./ (t+t0_wavg).^3
dE0dt0 = -2*D^2 ./ (t+t0_wavg).^3
dE0dE = -1*ones(size(t))
dE0dD = 2*D./(t+t0_wavg).^2
u_E0 = sqrt( (dE0dt.*u_t).^2 + (dE0dt0.*u_t0_wavg).^2 + ...
(dE0dE.*u_E_kin).^2 + (dE0dD.*u_D).^2)
disp('Bidrag:')

(dE0dt.*u_t).^2
(dE0dt0.*u_t0_wavg).^2
(dE0dE.*u_E_kin).^2
(dE0dD.*u_D).^2

E0_mean = mean(E0)
E0_wavg = w_avg(E0,1./u_E0.^2)
u_E0_wavg = 1/sqrt(sum(1/u_E0.^2));

% Determine t0
disp('Determining_t0');
E_kin = []; u_E_kin = [];
E_kin_empiric = []; u_E_kin_empiric = [];
t = []; u_t = [];
%for STATE=[X b]
for STATE=[X]
% -----
% Theoretical values
% -----
% see Report, eq. ?, derived from expression of T_e
T = STATE.E_0 + STATE.omega_e * STATE.nu - STATE.omega_e_x_e * (STATE.nu.^2 + 1);
u_T= sqrt( STATE.u_E_0^2 + (STATE.u_omega_e*STATE.nu).^2 + (STATE.u_omega_e_x_e*(STATE.nu
.^2+1)).^2 );
E_kin = [E_kin; hnu - T];
u_E_kin = [u_E_kin; sqrt( u_hnu^2 + u_T.^2)];
% -----
% Observered values
% -----
t = [t; STATE.centroids.t];
u_t = [u_t; STATE.centroids.t_err];
end

```

```

t0 = D ./ sqrt(E_kin+E0_wavg) - t;
dt0dD = 1./sqrt(E_kin + E0_wavg)
dt0dt = -1*ones(size(t))
dt0dE0 = -D ./ (2*(E_kin+E0_wavg).^(3/2))
dt0dE = -D ./ (2*(E_kin+E0_wavg).^(3/2))
u_t0 = sqrt( (dt0dt.*u_t).^2 + (dt0dE0.*u_E0_wavg).^2 + ...
    (dt0dE.*u_E_kin).^2 + (dt0dD.*u_D).^2);

disp('Bidrag:')
(dt0dt.*u_t).^2
(dt0dE0.*u_E0_wavg).^2
(dt0dE.*u_E_kin).^2
(dt0dD.*u_D).^2

t0_mean = mean(t0);
t0_wavg = w_avg(t0,1./u_t0.^2);
u_t0_wavg = 1/sqrt(sum(1/u_t0.^2));

disp(sprintf('Iteration_#%d,_E0:_%e_t0:_%e', iteration, E0_wavg, t0_wavg));
values = [values; E0_wavg u_E0_wavg t0_wavg u_t0_wavg];
end
values

disp('E0_and_uncertainty_(eV)')
disp([E0_wavg u_E0_wavg])
disp('t0_and_uncertainty_(ns)')
disp([t0_wavg u_t0_wavg])
E_kin = []; u_E_kin = [];
t = []; u_t = [];
for STATE=[X, b, B]
    % -----
    % Theoretical values
    % -----
    % see Report, eq. ?, derived from expression of T_e
    T = STATE.E_0 + STATE.omega_e * STATE.nu - STATE.omega_e_x_e * (STATE.nu.^2 + 1);
    u_T = sqrt( STATE.u_E_0^2 + (STATE.u_omega_e*STATE.nu).^2 + (STATE.u_omega_e_x_e*(STATE.nu.^2+1))
        .^2 );
    E_kin = [E_kin; hnu - T];
    u_E_kin = [u_E_kin; sqrt( u_hnu^2 + u_T.^2)];
    % -----
    % Observered values
    % -----
    t = [t; STATE.centroids.t];
    u_t = [u_t; STATE.centroids.t_err];
end
E_kin_empiric = E(t, t0_wavg, E0_wavg, D);
%u_E_kin_empiric = Edt(t, t0_wavg, E0_wavg, D) .* u_t;

disp('TheoreticUncertaintyObservedDeviation_(eV)')
disp([E_kin u_E_kin E_kin_empiric E_kin_empiric-E_kin])

disp('sum_of_deviations:')
disp(sum([E_kin_empiric-E_kin]))
disp('standard_deviation:')
std([E_kin_empiric-E_kin])
disp('variance:')

```



```
var([E_kin_empiric-E_kin])

stem(E_kin_empiric-E_kin)
```

Listing F.3: Xe code

```
clear all, close all
%Weighted average:
w_avg = inline('sum(_w.*x_)/_sum(_w_)','x','w');
%E0 and uncertainty (eV) for first run ; second run
E0 = [0.51800565581733; 0.51838068606310];
u_E0 = [0.00612925762669; 0.00637491588721];
%t0 and uncertainty (ns)
t0 = [44.58068229060260; 45.25219972239302];
u_t0 = [1.13164491482258; 1.14338036283470];
% Combine using weighted average:
E0 = w_avg(E0, 1./u_E0.^2), u_E0 = 1/sqrt(sum(1./u_E0.^2))
t0 = w_avg(t0, 1./u_t0.^2), u_t0 = 1/sqrt(sum(1./u_t0.^2))
% compensate for higher photon energy which affects t0 slightly
t0 = t0 - 5
D = 3438 % length constant.
u_D = 0
hnu = 40.814; % eV
u_hnu = 0.001

t_to_E = inline('(D./(t+t0)).^2-_E0','t','D','t0','E0');

% Load the data from the binary coincidence files:
f = fopen('./syre-xenon-kalibrering/2008-05-27_200542_double_electrons-e2ToF.e2c.txt','r');
[val, count] = fread(f, [2, Inf], 'uint32', 0, 'ieee-le');
fclose(f);

e1.t = val(1,:); e2.t = val(2,:);
u_t = 0.5; % assume 0.5 ns t uncertainty

e1.E_kin = t_to_E(e1.t, D, t0, E0);
e2.E_kin = t_to_E(e2.t, D, t0, E0);
% uncertainties given by error propagation:
for i=1:2
    if i==1; e = e1;
    else e = e2; end
    dEdt0 = -2*D^2 ./ (e.t + t0).^3; dEdt = dEdt0;
    dEdD = 2*D ./ (e.t + t0).^2; dEdE0 = -1;
    u_E_kin = sqrt((dEdt0*u_t0).^2 + (dEdt0*u_t0).^2 + ...
        (dEdt*u_t).^2 + (dEdD*u_D).^2 + (dEdE0*u_E0).^2 );
    if i==1; e1.u_E_kin = u_E_kin;
    else e2.u_E_kin = u_E_kin; end
end

figure
% Make a 2d histogram of flight times
bins = 300;
[H,X,Y] = hist2d([e1.t, e2.t], bins, bins, [0,5000], [0,5000]);
% mirror it in the line y = x
for i=1:bins; for j=i:bins
    H(i,j) = H(j,i);
end; end;
```

```

subplot(1,2,1), pcolor(X,Y,H); colorbar
shading flat; axis square tight; axis([500 5000 500 5000]);
xlabel('Flight-time,  $\tau_{e1}$  (ns)'), ylabel('Flight-time,  $\tau_{e2}$  (ns)')
% Make a 2d histogram of kinetic energies
bins = 200, E_min = 0, E_max = 8
[H,X,Y]=hist2d([e1.E_kin, e2.E_kin],bins,bins,[E_min,E_max],[E_min,E_max]);
H=H(2:end-1,2:end-1); X=X(2:end-1); Y=Y(2:end-1);
% mirror it in the line y = x
for i=1:bins-2; for j=i:bins-2
    H(j,i) = H(i,j);
end; end
subplot(1,2,2), pcolor(X,Y,H); colorbar
shading flat; axis square tight; axis([0 8 0 8]);
xlabel('Energy,  $\tau_{e1}$  (eV)'), ylabel('Energy,  $\tau_{e2}$  (eV)')

figure % 3D histogram
surf(X,Y,H), colorbar, shading flat, shading interp
title('Xenon,  $\tau_{e1}$  vs  $\tau_{e2}$  Energy, density')
xlabel('Energy,  $\tau_{e1}$  (eV)'), ylabel('Energy,  $\tau_{e2}$  (eV)')
zlabel('intensity (arbit. units)')

E_kin_sum = e1.E_kin + e2.E_kin;
u_E_kin_sum = sqrt(e1.u_E_kin.^2 + e2.u_E_kin.^2)

% Histogram of sum of kinetic energies
E_min = 1, E_max = 10, N_bins = 1000;
[E_counts,E_bins] = hist(E_kin_sum, linspace(E_min,E_max,N_bins));
E_counts = E_counts(2:end-1); E_bins = E_bins(2:end-1);

figure % how does uncertainties depend on energy?
[H,X,Y] = hist2d([E_kin_sum, u_E_kin_sum], 40, 40, [0,10], [0,0.05]);
pcolor(X,Y,H); colorbar
title('Uncertainties of  $E_{\{kin,1\}} + E_{\{kin,2\}}$ ')
xlabel('Kinetic energy (eV)'), ylabel('Uncertainty in kinetic energy (eV)')

figure % how does uncertainty propagate to DIP?
DIP = hnu - E_kin_sum;
u_DIP = sqrt(u_hnu.^2 + u_E_kin_sum.^2);
[H,X,Y] = hist2d([DIP, u_DIP], 100, 20, [32,38.5], [0.005,0.025]);
H=H(2:end-1,2:end-1); X=X(2:end-1); Y=Y(2:end-1);
pcolor(X,Y,H); colorbar, shading flat;
title('How the uncertainties in DIE varies with the DIE')
xlabel('Double Ionization Energy (eV)'), ylabel('Uncertainty of DIE (eV)')
% From the graph it is clear that the uncertainty is approx. 0.015 eV,
% in most of the region, varying as follows between the states:
% 3P2: 0.012 to 0.020 eV uncertainty
% 3P0 and 3P1: 0.009 to 0.016 eV uncertainty
% 1D2: 0.009 to 0.015 eV uncertainty
% 1S0: 0.007 to 0.0009 eV uncertainty

figure % create histogram of DIP etc
E_counts = flipud(E_counts); E_bins = hnu - E_bins
h = bar(E_bins, E_counts, 'histc'); hold on
set(h,'LineWidth',0.1,'FaceColor','r','EdgeColor','r');
% Moving Average Filter
E_f = runmean(E_counts, 5, 'zero');

```

```

h = plot(E_bins,E_f,'b-'), set(h,'LineWidth',1);
title('Xe_electron_binding_energies_sum_e1+_e2')
xlabel('Double_Ionization_Potential_(eV)'), ylabel('Coincidence_counts')
legend('Histogram', 'Moving_average')

figure % plot DIP and mark electronic states
set(gcf, 'name', 'DIP_WithRelevantPeaks', 'numbertitle', 'on');
h = plot(E_bins,E_f,'b-'), set(h,'LineWidth',2); hold on
axis([31.5 39 0 250])
xlabel('Ionization_energy_(eV)'), ylabel('Coincidence_counts')
x = linspace(E_bins(1),E_bins(end), 400);
peaks = [...
    195 * gaussmf(x, [0.15 33.13]); % 3P2
    40 * gaussmf(x, [0.15 34.13]); % 3P0
    85 * gaussmf(x, [0.15 34.39]); % 3P1
    162 * gaussmf(x, [0.15 35.30]); % 1D2
    24 * gaussmf(x, [0.15 37.63]); % 1S0
]; p = peaks';
sigma_3P2 = fwhm(x,peaks(1,:))
sigma_3P0 = fwhm(x,peaks(2,:))
sigma_3P1 = fwhm(x,peaks(3,:))
sigma_1D1 = fwhm(x,peaks(4,:))
sigma_1S0 = fwhm(x,peaks(5,:))

% plot sum of estimated theoretical peaks
h = plot(x, sum(peaks),'r--')
set(h, 'LineWidth', 2)
legend('Observed_intensities', 'Sum_of_estimated_gaussians')
% plot each of the estimated theoretical peaks
plot(x,peaks,'k:')
int_unit = 162/5; % 1D1 has intensity 5 in Eland et al 2003
intensities = [195 40 85 162 24]/int_unit

```

Bibliography

- [1] M. Karbo. PC Architecture. URL <http://www.karbosguide.com/books/pcarchitecture/start.htm>. [Online; accessed 19-May-2008].
- [2] R. Feifel. Electrons as individuals and members of a collective. Docent lecture, 2007.
- [3] R. Feifel and L. Karlsson. *Kosmos*, 2007:119, 2007.
- [4] Nobel Foundation. The Nobel Prize in Physics 1921, 2008. URL http://nobelprize.org/nobel_prizes/physics/laureates/1921/. [Online; accessed 20-June-2008].
- [5] H. Kragh. *Quantum Generations*. Princeton University Press, 1999.
- [6] L. Karlsson and J. Wall. A brief introduction to photoelectron spectroscopy. *Compendium*, 2007. pp.2–7.
- [7] S. Andersson, F. Bruhn, J. Hedman, L. Karlsson, S. Lunell, K. Nilson, and J. Wall. *Atom- och Molekylfysik*. Repro Fysikum, 2nd edition, 2005.
- [8] J. H. D. Eland. *Chem. Phys.*, 294(Issue 2):171, 2003.
- [9] J. H. D. Eland, O. Vieuxmaire, T. Kinugawa, P. Lablanquie, R. I. Hall, and F. Penent. *Phys. Rev. Lett.*, 90(5):053003, 2003.
- [10] Tiobe.com. TIOBE Programming Community Index for July 2008, 2008. URL <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. [Online; accessed 13-July-2008].
- [11] D. J. Griffiths. *Introduction to Quantum Mechanics*. Pearson Education, Inc., 2nd edition, 2005.
- [12] P. Andersson. Photoelectron - photoelectron coincidence studies of xenon and krypton. Master's thesis, Uppsala Universitet, April 2007.
- [13] R. D. Molloy, A. Danielsson, L. Karlsson, and J. H. D. Eland. *Chem. Phys.*, 335:49, 2007.
- [14] C. R. Brundle, M. B. Robin, H. Basch, M. Pinsky, and A. Bond. *J. Am. Chem. Soc.*, 92(13):3863, 1970.
- [15] J. H. D. Eland and R. Feifel. *Chem. Phys.*, 327:85, 2006.
- [16] M. Takahashi, J. P. Cave, and J. H. D. Eland. *Rev. Sci. Instrum.*, 71(3):1337, 2000.
- [17] P. Kruit and F. H. Read. *J. Phys. E*, 16:313, 1983.
- [18] ArsTech.com. ARS technologies - USB 2.0 to ISA card ROHS, 2008. URL <http://www.arstech.com/item-USB-2-0-to-ISA-card-usb2isa.html>. [Online; accessed 14-July-2008].

- [19] J. Corbet, A. Rubini, and G. Kroah-Hartman. *Linux Device Drivers*. O'Reilly Media Inc., 3rd edition, 2005.
- [20] R. Dunn-Krahn, J. Maple, and Y. Coady. The crisis in systems code maintenance: sourceforge, we have a problem. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, page 75. ACM, 2005.
- [21] P. J. Salzman. The linux kernel module programming guide, 2008. URL <http://tldp.org/LDP/lkmpg/2.4/html/index.html>. [Online; accessed 14-July-2008].
- [22] P. Linusson. Double photoionisation of alcohol molecules, thiophene and bromine-substituted thiophenes. Master's thesis, Uppsala Universitet, April 2008.
- [23] BitSim AB. UHAB – Ultra High-Speed Acquisition Board, 2008. URL <http://www.bitsim.com/uhab-high-speed-board.htm>. [Online; accessed 08-Aug-2008].
- [24] J. R. Taylor. *An introduction to error analysis*. University Science Books, 2nd edition, 1997.
- [25] Wikipedia. Morse potential — wikipedia, the free encyclopedia, 2008. URL http://en.wikipedia.org/w/index.php?title=Morse_potential&oldid=219238280. [Online; accessed 19-June-2008].
- [26] O. Edqvist, E. Lindholm, L. E. Selin, and L. Åsbrink. *Phys. Scr.*, 1:25, 1970.
- [27] K. P. Huber and G. Herzberg. *Molecular Spectra and Molecular Structure: Spectra of Diatomic Molecules*, volume 4, page 504. Litton Education Publishing, Inc., 1979.
- [28] LeCroy. LeCroy models MTD133B and MTD135 8-Channel Time-to-Digital Converter Monolithic ICs, 2001. URL <http://www.lecroy.com/lrs/dsheets/mtd.htm>. [Online; accessed 14-July-2008].
- [29] Cronologic.de. TDC8 description, 2008. URL http://www.cronologic.de/products/time_measurement/tdc8/TDC8_description.pdf. [Online; accessed 14-July-2008].
- [30] O. Jagutzki. *TDC8 ISA & PCI and TDC8PCI2 Manual*. RoentDek Handels GmbH. URL [http://www.roentdek.com/mitte/doppel/links/fuer_manu/daq_hardw/TDC8%20Manual%20\(6.2.90.6\).pdf](http://www.roentdek.com/mitte/doppel/links/fuer_manu/daq_hardw/TDC8%20Manual%20(6.2.90.6).pdf).
- [31] O. Jagutzki and T. Jahnke. *TDC8HP System Manual*. RoentDek Handels GmbH. URL [http://www.roentdek.com/mitte/doppel/links/fuer_manu/daq_hardw/TDC8HP%20Manual%20\(6.2.90.3\).pdf](http://www.roentdek.com/mitte/doppel/links/fuer_manu/daq_hardw/TDC8HP%20Manual%20(6.2.90.3).pdf).
- [32] B. Henderson. Linux loadable kernel module howto, 2006. URL <http://tldp.org/HOWTO/Module-HOWTO/index.html>. [Online; accessed 15-July-2008].