Towards Velocity Map Imaging: Implementation of a Self-Centering Inverse Abel Transform Script

Richard James Bentley-Moyse
Supervisor: Duncan Wild

Masters Dissertation submitted as part of the M.Sc degree
in the School of Physics, University of Western Australia

Date of submission: 31/Oct/14

Declaration

This is to certify that:

(i) this dissertation comprises of my own original work,

(ii) due acknowledgement has been made in the text to all other materials used,

(iii) the dissertation is less than 60 pages in length, exclusive of tables, equations, references, appendices and footnotes.

I authorise the Head of the School of Physics and the Head of the School of Chemistry to pass a copy of this dissertation to any person judged to have an acceptable reason for access to the information.

Richard J. Bentley-Moyse

Duncan A. Wild
Supervisor

Acknowledgments

If I have seen further it is by standing on ye sholders of Giants.
Sir Isaac Newton, 1676

To dwell on the past is to forget the future, but the path traversed shapes you, moulds you and defines you. Not a day goes by that doesn't end in quiet reflection upon the triumph of those before me, those who pushed the boundaries and continue to drive me to always be the better man, scientist and partner. This year, and indeed the year before, have tested my faith in myself, in my resolve to finish what I'd started and to push past the illness dogging my every advance.

First and foremost, my thanks go to my research group. I couldn't have asked for more patience or understanding; you were the Mystery Incorporated to my Ghost of the Haunted Lab. Shanee and Kim, our chats will always be the highlight of my procrastination. Marcus, I will always thank you for introducing me to To Øl and chatting to me about programming and scientific writing. Little Duncan, I will always look up to you; thanks for bearing the brunt of the club work this year! Chia-Yang, ever a smile and always positive. Big Duncan, I couldn't ask for a more down-to-Earth and welcoming supervisor; your patience and idea-bouncing has been the most appreciated throughout this project.

My second thanks go to my friends, both from UWA and those beyond. Working through the hardship wouldn't have been possible without your support, comfort, laughs and understanding - I hope I see more of you all! I especially commend the UWA gang for their ceaseless tolerance of my ranting and tyrades...!

For helping me learn to analyse C++ scripts and clicking that final piece into place, I thank A/Prof. Dylan Jayatilaka. For providing me their inverse Abel transform to work with and answering my novice questions relating to it, I extend my deep gratitude to the Gascooke group, Flinders University. For developing an open-source C++ RANSAC circle detection implementation and allowing me to use it as the skeleton for an ellipse detecting scheme, I give both thanks and respect to Kevin Hughes, as well as an appreciation for sharing his implementation in the open-source community.

No man is an island. You are my rock Deanna; you're love and guidance helped me deal with the illness, the struggles and the stress. I look forward to the future, Our future together; I know I can overcome anything with you by my side.

Abstract

Velocity map imaging (VMI) is an ion imaging technique used in photoelectron spectroscopy to give the photoelectron spectrum and the photoelectron angular distribution (PAD) for an anionic species, with quicker image acquisition times and often better resolution than many comparable techniques. The current laboratory apparatus, a time-of-flight mass spectrometer coupled with a photoelectron spectrometer (TOF-PES), is being extended to include a VMI photoelectron spectrometer. After photodetachment, a carefully chosen electric field maps all photoelectrons with the same initial velocity vector to the same position on a phosphor screen-CCD camera assembly.

Whilst powerful in terms of maintenance and resolution, a significant issue with VMI exists with the experimental images themselves. Due to the geometry of the VMI apparatus, the images captured are a 2D projection of the 3D photoelectron initial velocity distribution. When the distribution is cylindrically symmetrical parallel with the image, this projection is known as the Abel transform of the distribution. By applying the inverse Abel transform (IAT) to experimental images, it is possible to recover a slice through the axis of symmetry of the 3D distribution (which is equivalent to the entire distribution).

A C++ implementation of the IAT for use with FITS images was kindly provided by the Gascooke group, Flinders University which was subsequently ported from a *Windows* compiler-specific form to generic *Linux*-compatible form for use on the laboratory computers. The original script relies on the user knowledge of the center coordinate of the distribution, however it difficult to calculate the centre both without computer assistance and in an experimental setting. Further to this, the IAT script was found to be extremely sensitive (within 1pixel error) of the center coordinate of the test image by inspection of the radial spectrum of the image.

After detailed analysis of the IAT script and rigorous mathematical justifcation, work began on implementing an auto-centering scheme for the script. A random sample consensus (RANSAC) scheme was found to be most suitable due to the low number of 'active' pixels in the images. RANSAC works by first detecting all 'edge' points, where sharp colour changes occur, and then fitting a random sample of points to a set of initial criteria - if these criteria are not met, the script resets and chooses another random sample set until the initial criteria are satisfied. Building upon a previous open-source C++ RANSAC circle detecting script provided by Kevin Hughes, a script for detecting ellipses was produced. Circular images are expected due to the electric field mapping so misalignment can easily be determined from the ellipse parameters allowing for straightforward calibration. The script is extremely successful in detecting rings in simple experimental images but is unreliable for general photographic images due to interference with the edge detecting stage of the script. The IAT and ellipse detection scripts were successfully combined to produce a precise, self-centring tool for treatment of images from the soon-to-be-operational camera.

Summary of Student Achievement

Richard J. Bentley-Moyse 20773252

During this Master of Physical Science research project, I have managed to achieve the following as part of my research:

- I have found my niche in the laboratory as being the only member specialising in image analysis, and I have rigorously learnt and justified the theory behind both the image processing and the camera operation itself. I am currently responsible for image treatment and will continue to develop more efficient computation methods during the course of my PhD.
- I ported the provided *Windows XP, Borland* C++ compiler-specific IAT script for use in *Linux* with a generic compiler (eg. Command line)
- By building on the structure of Kevin Hughes' existing open-source RANSAC circle detection script, I was able to overhaul the selection criteria and devise my own method for detecting ellipses using his existing RANSAC scheme. Using this ellipse detecting script, I was able to modify the ported IAT script to automatically detect the input image centre, eliminating any user-end errors resulting from incorrect centre calculation. The ellipse detecting script will continue to be developed and improved, and will eventually be made available online and open-source.
- I have partially completed the next step towards extracting information from experimental images, a *Python* script to recover the radial spectrum from the inverse Abel transformed image. From this radial spectrum, using a Jacobian transformation, a kinetic energy spectrum can be recovered. This will only be possible once the camera and electrostatic lens are finished being configured, and the resulting velocity-to-radius electrostatic mapping is known.
- In order to achieve the above, I have learnt to read and write the C++ and *Python* programming languages with only minor programming experience from *Mathematica* over the course of the last two years work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Physical chemistry is largely concerned with the study of physical matter and how chemical reactions occur in terms of the principles of physics, specifically quantum mechanics. Spectroscopy is the study of the interaction of matter and light and is frequently used for a multitude of different experimental scenarios.[1–3] In this thesis, the apparatus and operational procedures currently used by the research group are introduced, as well as the groundwork for a slow-electron velocity map imaging spectrometer that is in development for the apparatus. The focus of this thesis is to develop the tools necessary to treat experimental images from the new camera and to work towards the extraction of useful information from these images.

This chapter is structured in the following way. In Section 1.1, an overview of the techniques and the laboratory apparatus is given. Section 1.2 introduces velocity map imaging, the basis of the new technique and equipment, with Section 1.2.1 further discussing the new imaging technique and issues related to image aquisition and data extraction.

1.1 Time-of-Flight Photelectron Spectrometer

The experimental apparatus consists of a time-of-flight mass spectrometer coupled with a photoelectron spectrometer (TOF-PES), allowing for mass-selective photoelectron spectroscopy. The apparatus is a large vacuum chamber assembly and allows for the addition of new chambers and components, as well as the modification of existing parts. Current research in the laboratory is focused on the study of clusters (two or more species interacting with one another) and the solvation of anions.[4,5]

1.1.1 Anion Photoelectron Spectroscopy

Photoelectron spectroscopy (PES), also known as photodetachment spectroscopy, is a spectroscopic technique used to probe the binding (or ionisation) energies of different atomic, molecular and cluster species. The technique is based on the photoelectric effect in which a photon of sufficient energy interacts with an atom or molecule to eject an electron (namely, a photoelectron). Anion PES focuses on the photelectrons detached from an anionic species in order to provide information about both the neutral species. For a species M excited by a photon of frequency ν:

$$M^- + \mathrm{h}\nu \rightarrow M + \mathrm{e}^-$$

The kinetic energy of the electrons E_K will be that excess energy from overcoming the binding energy.

$$E_\mathrm{K} = E_\gamma - E_\mathrm{B} \tag{1.1}$$

As the frequency of the incident photon is known and the velocity of the photoelectrons can be measured, the electron binding energy for the species can be determined:

$$E_\mathrm{B} = E_\gamma - E_\mathrm{K} = \mathrm{h}\nu - \frac{1}{2}m_e v^2$$

The ejected electrons will have kinetic energies characteristic of the neutral state accessed and the energy level of the target anion upon which photodetachment took place. That is,

$$E_\mathrm{B} = E(\text{neutral}) - E(\text{anion})$$

The photodetached electrons are directed down a flight tube of known length s. If the time-of-flight τ is measured, the kinetic energy of an electron can be found using

$$E_\mathrm{K} = \frac{1}{2}m_\mathrm{e}v^2 = \frac{1}{2}m_\mathrm{e}\left(\frac{s}{\tau}\right)^2 \tag{1.2}$$

If both the resulting neutral and the target anion are in the lowest vibrational state, the energy difference will correspond to what is known as the adiabatic electron affinity EA_a, which is defined to be the energy released when an electron attaches to a gas-phase atom.[6] The electron binding energy and electron affinity are equal if the geometry of the anion and neutral are sufficiently similar, and as such can give use an indepth analysis of the structure of the neutral due to the vibrational wavefunction overlap of the anion and neutral product.[7]

A crucial principle in anion spectrometry is the Frank-Condon principle[6] which states that when a molecule undergoes an electronic transition, the electronic transition takes place much faster than the nuclei can respond as the target nuclei are much more massive than their associated electrons. Before any photon absorption takes place, the molecule are considered to be in the lowest vibrational energy state of its electronic energy state, with the most probable location of the nuclei being at some equilibrium separation. As a result of this, the configuration of the nucleus or nuclei can be considered to remain constant during the electronic transition. Hence, when the species is excited the most probable transition is that vibrational level of the upper energy level that has the closest internuclear separation to the initial equilibrium separation. This is demonstrated

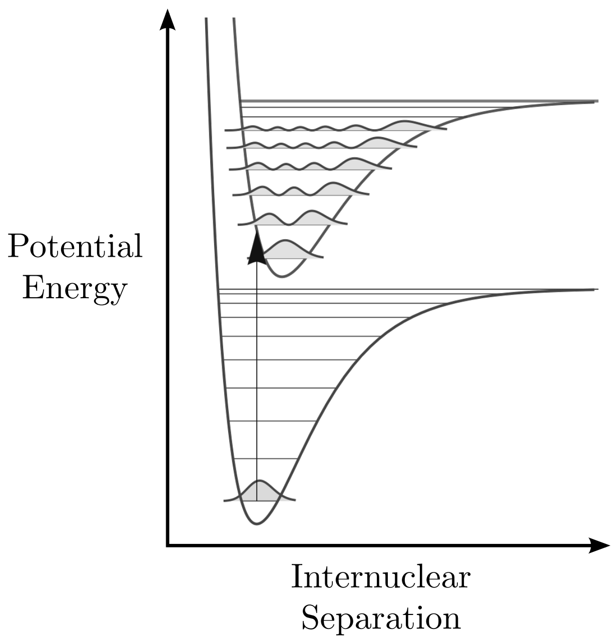


Figure 1.1: Photodetachment scheme demonstrating the Frank-Condon principle

graphically in Figure 1.1.

1.1.2 Time-of-Flight Mass Spectrometry

Time-of-Flight mass spectrometry measures the mass-to-charge number ratio of analyte ions. A simplified look, assuming ions begin at rest, involves accelerating ions of mass m and charge $q = ez$ over a short distance d using an electric field of strength E, then a measurment of the time t it takes for the ions to traverse a drift tube of length l. The electric potential energy of the particles traversing the field is then

$$E_{\mathrm{P}} = ezEd$$

This potential energy is converted into kinetic energy, so that

$$E_{\mathrm{K}} = \frac{1}{2}mv^2 = ezEd$$

Note that the velocity will be $\frac{l}{t}$, so that the mass-to-charge ratio is

$$\frac{1}{2}m\left(\frac{l}{t}\right)^2 = ezEd \implies \boxed{\frac{m}{z} = 2eEd\left(\frac{t}{l}\right)^2}$$

1.1.3 Apparatus Overview

The construction of the TOF-PES was overseen and documented by LaMacchia[8] with further modifications by Quak[9].

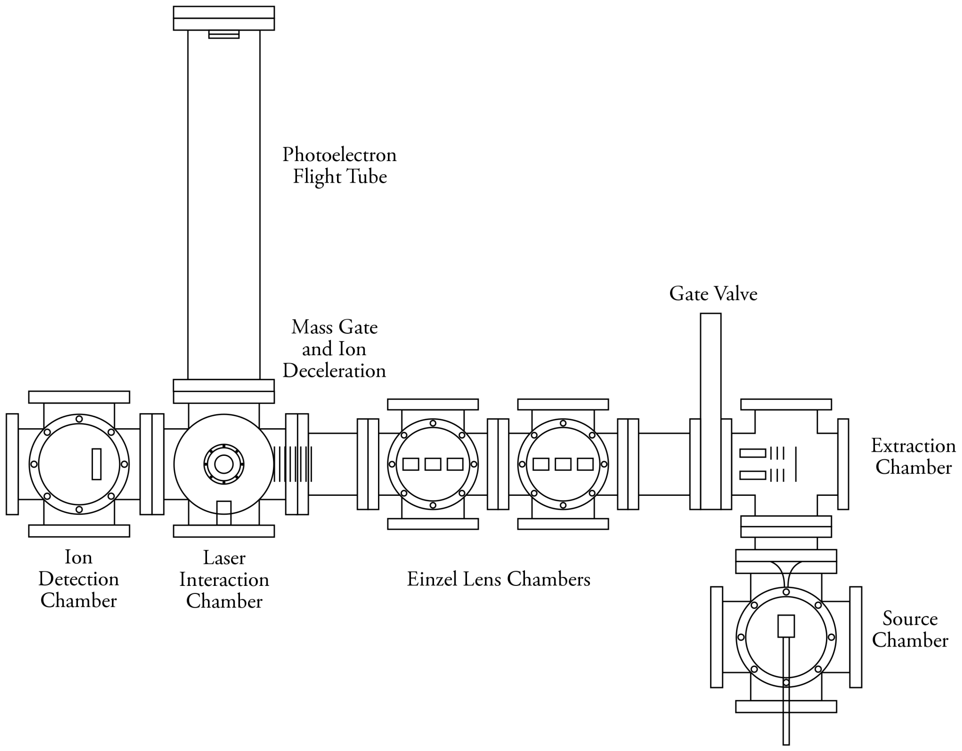


Figure 1.2: Schematic of the Wild laboratory TOF-PES without SEVI extension

Use of the TOF-PES first begins at the gas mixing station, where gaseous and liquid samples not readily in the gas phase, as well as buffer gases, are mixed. To form clusters, NF_3 is used as a source of F^- ions and similarly CCl_4 is used for Cl^- ions (but is not readily gaseous). The second sample gas species is mixed with the halide ion source. Argon is used both as a cooling buffer gas to decrease thermal spread of the molecular beam in later chambers of the apparatus and also as a source of slow electrons as documented by LaMacchia[8] in the basic reaction scheme, with M the central molecule of the cluster and using F^- ions as a an example:

$$\begin{aligned} Ar + e^-_{fast} &\rightarrow Ar^+ + e^-_{slow} + e^-_{fast} \\ NF_3 + e^-_{slow} &\rightarrow [NF_3]^- \rightarrow NF_2 + F^- \\ F^- + M &\rightarrow [F\cdots M]^- \end{aligned} \tag{1.3}$$

The gas mixture is injected into the source chamber (see Figure 1.2) through a pulsed nozzle driver which acts both as a way to control gas flow and also as a means to selectively give the gas pulse more energy. The source chamber houses twin rhenium filaments that act as a fast electron source through thermionic emission[8] producing a plasma in a scheme analogous to that shown in (1.3). The gas then passes through a conical skimmer that acts as a collimator with a thin (1mm or 3mm) orifice that reduces the spacial cross-section of the molecular beam.

Upon entering the extraction chamber, the beam is redirected down the flight tube by a series of five stainless steel plates separated by ceramic rods, with 3 of the plates having annuli to allow the passage of anions, as first demonstrated by Wiley and McLaren[10]. A set of $X - Y$ deflection plates are then used to adjust the direction of the resulting anion beam, as shown in Figure 1.3

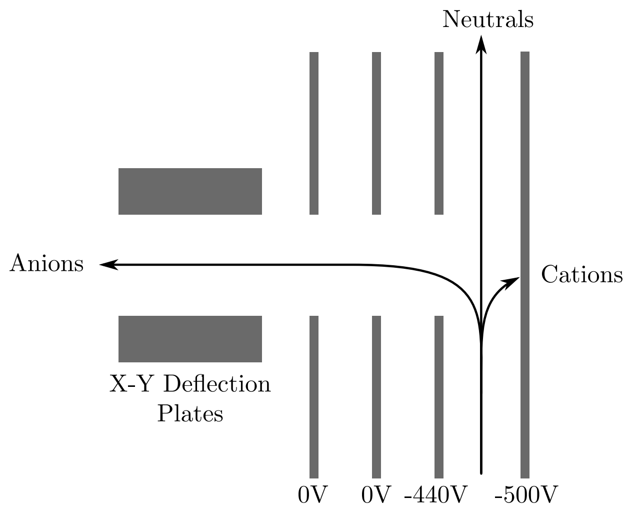


Figure 1.3: The extraction chamber plate array and resulting ions paths

The apparatus is separated into two sections by a gate valve placed between the extraction chamber and the time-of-flight tube to allow for frequent extraction chamber cleaning, with both sections kept at different pressures during operation and standby to promote movement of the molecular beam in a single direction through the apparatus.[2] Two Einzel lenses (see Figure 1.4) located in the time-of-flight tube refocus the beam to counteract beam spreading due to Coulomb repulsion. An Einzel lens is comparable to an optical lens in that it focuses the molecular beam without changing the energies of the particles upon exiting[11].

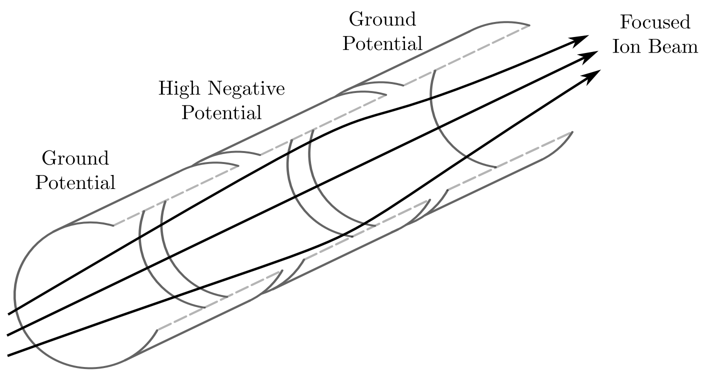


Figure 1.4: Einzel lens diagram with cut-out showing ion path

The molecular beam then passes into the laser interaction chamber for photoelectron spectrum measurements or continues onwards to an ion detector for mass spectrum measurements. The laser is a fixed wavelength 1064nm Nd:YAG (Neodynium : Yttrium

Aluminium Garnate $Y_3Al_5O_{12}$) pulsed laser. The frequency is then doubled to give 532nm, and doubled again to give 266nm, using an optics array. The laboratory is also in possession of a tuneable dye laser for frequency scanning that is currently not in use.

1.2 Velocity Map Imaging (VMI)

Velocity map imaging is a technique that was first developed by Eppink and Parker[12]. Previous ion imaging techniques suffered from distortions in experimental images due to use of conventional grid electrodes. Eppink and Parker introduced the use of a simple cylindrical three-plate electrostatic lens instead of the grid to both combat these distortions, enlarge the image and allow the use of an electrostatic field to map ions onto the 2D imaging plane. With a sagacious choice of electrostatic lens configuration, an electric field can be found such that charged particles with the same initial velocity can be mapped to the same point on the detector, independent of the individual particles distance from the electrostatic lens cylindrical axis[12].

A series of images are collected and combined to produce experimental images consisting of rings corresponding to the kinetic energies of the photoelectrons. As all particles originate from the same interaction region, and the field maps all anions regardless of initial position, the detected rings share a common centre.

1.2.1 VMI Camera

A typical VMI camera setup consists of an electrostatic lens and a twin micro-channel plate (MCP) array coupled with a phosphor screen (PS) and charge-coupled devide (CCD) camera as shown in Figure 1.5. The analyte is injected as a mixed molecular beam that is excited by perpendicular laser radiation linearly polarised in a mutually perpendicular orientation. The general geometry of a VMI experiment is shown in Figure 1.6.

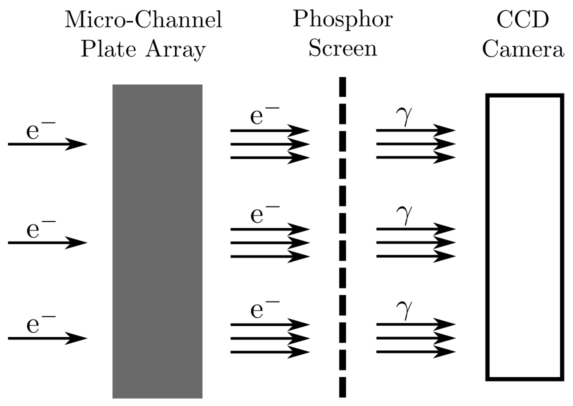


Figure 1.5: VMI camera arrangement

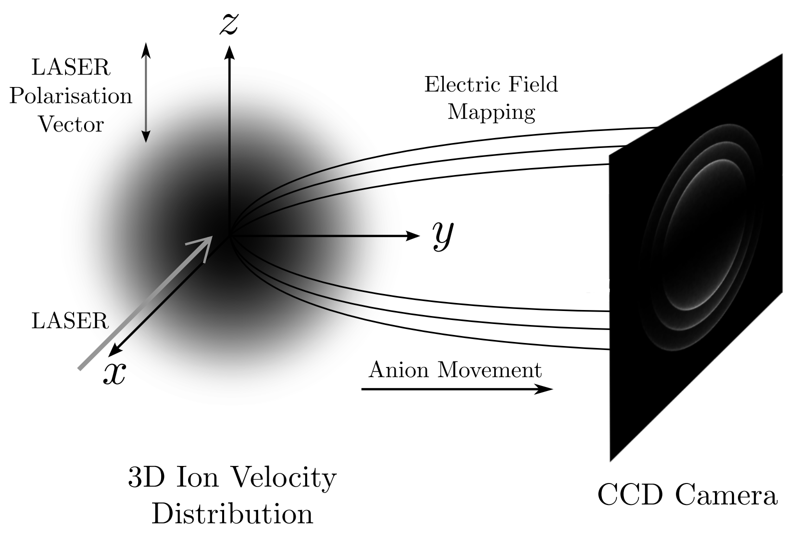


Figure 1.6: Geometry of VMI

Micro-Channel Plate Array (MCP)

Utilising the electrostatic lens, the photodetached electrons are guided towards the MCP setup which acts as an electron multiplier, amplifying the experimental signal. The basic structure of a MCP involves a large array (of the order 10^4 and above) of parallel electron multipliers, each acting as an independent channel. The gain of an MCP setup is primarily determined by the length to diammeter ratio of the MCP, allowing for almost any size reduction of the MCP without effecting the gain threshold[13]. This property is useful when using the channels in measuring spatial information, as a greater density of channels will give greater spacial resolution.

Phosphor Plate

The electrons exit the MCP and are incident upon a phosphor screen which acts to 'convert' the electrons into photons. A phosphor screen works by utilising cathodoluminescence, wherein the impact of an electron upon a material results in the emission of photons.[14] This conversion from electrons to photons does not result in loss of information for the experiment as by this point the velocity of the electrons, and hence the relative position on the experimental image, has already been mapped by the electrostatic lens.

Charge-Couple Device Camera (CCD)

The photons produced by the phosphor plate are detected by a CCD camera placed directly behind the screen as shown in Figure 1.5. For the new camera, images will be timed to be taken just after the beginning to just after the end of every pulse interval of the laser. Only a small number of electrons make it to the detector over this interval, so a large number of images are required. After a certain number of images are collected, they will be summed and averaged as the camera proceeds to take more images. After a length of time, enough images will be taken for rings corresponding to the electron energy levels to be formed.

Slow-Photoelectron Velocity-Map Imaging Spectroscopy (SEVI)

For a VMI experiment, a DC field large enough to direct all photoelectrons from the photodetachment region is used. In slow-electron velocity map imaging spectroscopy (SEVI), the same setup is used as in VMI, except a much lower DC field is used in order to only extract low kinetic energy photoelectrons[15]. This allows for detection over a small range of kinetic energies and the enlarging of the resultant image on the detector by manipulating the electrostatic lens. By scanning through different frequencies using a tunable laser, multiple high resolution spectra can be produced over a short measurement period and the spectra combined[16]. After configuration of the VMI camera is complete, the aim is to use SEVI spectroscopy to act as a counterpoint to the currently active photoelectron spectrometer.

1.2.2 Experimental Difficulties in VMI

By using a cylindrically symmetrical electric field for the VMI camera mapping, the resultant images with be that of circular rings corresponding the kinetic energy of the photoelectrons. Hence, a major concern is that of centering the images when calculating the photelectron spectrum or the PAD, which depend greatly upon the center of the distribution. A scheme for detecting ellipses is presented in detail in Chapter 4 that allows for automatically centering the images in later treatment.

Further to this, due to the geometry of the experiments, the images taken are 2D projections of the system under examination (in this case the 3D photoelectron initial velocity distribution). The difficulty lies in extracting the original distribution whilst taking into account experimental noise and the effects of pixelation. The use of a linearly polarised laser imposes cyclindrical symmetry due to the relationship between the photoelectron angular distribution and the laser polarisation[17]

When this system is cyclindrically symmetrical along an axis parallel to the image taken, the particular projection is called the Abel transform.[18] As such, a popular method for calculating the original distribution, or equivalently a slice though the axis of symmetry of the original 3D distribution, is the so-called inverse Abel transform. Both methods are explained in greater detail in Chapter 3.

Chapter 2

Mathematical Techniques

Linear systems arise in a plethora of different fields of study and as such are of great relevance to mathematicians, engineers and scientists alike. Furthermore, many nonlinear systems can be approximated as linear over small ranges. Many physical systems can be modelled well by linear differential equations, which are much easier to work with than their more general, nonlinear, cousins. Signal and image processing frequently make use of linear constructions, or at least the properties of linear systems, to calculate transformed images. An issue that often arises with image treatment is that the images are discrete, meaning that any continuous linear system needs to be discretised for handling images. As images can be defined as matrices, the ability to write a linear differential equation in a discretised matrix form is of particular relevance to image processing.

In this chapter, focus is restricted to linear *ordinary* differential equations (linear ODE's) with a single input and a single output with regards to later applications in Chapter 3. It should be noted that in general, all linear physical systems will be described by a linear *partial* differential equation, except in the case where the system input and output are functions of only one independent variable. It is also possible in this setup for systems to have multiple input and output functions. With respect to later applications, such systems with multiple inputs and outputs are outside the scope of this work. In section 2.1, the mathematical principles behind linear systems are briefly explained. Further to this, a subset of linear systems called linear time-invariant (LTI) systems are introduced and some useful properties and formulations presented. In section 2.2, the state-space representation of linear systems is introduced, in which an n-th order linear ODE is decomposed into n-vector array of 1^{st}-order linear ODEs.

2.1 Linear Systems Theory

2.1.1 Linear Systems and Notation

In a mathematical sense, a system is a collection of components where the interactions between the individual components and the dynamics of the system are determined by a mathematical equation. A linear system is simply that in which this mathematical equation is linear, that is it follows the *superposition principle* which is discussed in further detail below.

The general form of a linear n-th order ODE with a single input and output is

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}y(t) + \cdots + P_2(t)\frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) + P_1(t)\frac{\mathrm{d}}{\mathrm{d}t}y(t) + P_0(t)y(t) = Q\big(u(t)\big) \tag{2.1}$$

When considering such systems, it is convenient to introduce a 'system' notation that symbolically describes the system. For a system $\mathcal{H} = \mathcal{H}\{\cdot\}$ with input $u(t)$ and output $y(t)$, it is written

$$y(t) := \mathcal{H}\left\{u(t), t\right\} \tag{2.2}$$

This can simply be read as 'for system $\mathcal{H}$ with input $u(t)$, the corresponding output is $y(t)$. The benefit of this notation is that it vastly simplifies discussion of systems described by complicated equations whilst still allowing the programming of desired properties.

2.1.2 The Superposition Principle

A system is said to be linear if it is described by a mathematical equation that follows the *superposition principle*. Such an equation is also described as linear. The superposition principle is an amalgamation of two different mathematical conditions, namely the additivity condition and the homogeneity condition, as opposed to a separate mathematical construct.[19]

Principle 1 (The Superposition Principle)**.** Any system described by a linear ODE has the property that if $\alpha y_1(t)$ is the system output for system input $\alpha u_1(t)$ and if $\beta y_2(t)$ is the system output for system input $\beta u_2(t)$ with α, β a constant then $\alpha y_1(t) + \beta y_2(t)$ is the system output from system input $\alpha u_1(t) + \beta u_2(t)$.
In system notation, the superposition principle is demonstrated as

$$\boxed{\mathcal{H}\left\{\alpha u_1(t) + \beta u_2(t), t\right\} = \alpha\mathcal{H}\left\{u_1(t), t\right\} + \beta\mathcal{H}\left\{u_2(t), t\right\}}$$

The Additivity Condition

The *additivity condition* is simply the Superposition Principle without scaling. In system notation, the additivity condition is demonstrated as

$$\mathcal{H}\left\{u_1(t) + u_2(t), t\right\} = \mathcal{H}\left\{u_1(t), t\right\} + \mathcal{H}\left\{u_2(t), t\right\}$$

The Homogeneity Condition

The *homogeneity condition* requires that if the system input $u(t)$ is scaled by a constant, then the system output $y(t)$ is scaled by the same constant. That is,

$$\mathcal{H}\left\{\alpha u(t), t\right\} = \alpha\mathcal{H}\left\{u(t), t\right\}$$

2.1.3 The Time-Invariance Principle

If a further requirement is imposed upon the system, namely the *time-invariance principle*, the system will belong to a subset of linear systems known as linear time-invariant (LTI) systems. The choice of time as the invariant quantity is largely a matter of convention as LTI systems commonly appear in signal processing and control theory. LTI systems (and all other time-invariant systems) have the property that if $y(t_1)$ is the system output for a system input $u(t_1)$, then for the system input $u(t_2)$ the output will be $y(t_2)$. In simpler words, the system output does not depend explicitly upon time.[19]

In considering Expression (2.1), note that if the functions $P_n(t)$ are set as constants, system output $u(t)$ is no longer explicitely dependent upon time and the ODE will be time-invariant as required. This can be demonstrated as follows, by considering the general linear ODE with time-shifted system input $u(t+\tau)$ and corresponding output denoted $\hat{y}(t)$, noting that the system itself is *not* time-shifted.

i.e

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}\hat{y}(t) + \cdots + P_2(t)\frac{\mathrm{d}^2}{\mathrm{d}t^2}\hat{y}(t) + P_1(t)\frac{\mathrm{d}}{\mathrm{d}t}\hat{y}(t) + P_0(t)\hat{y}(t) = Q\big(u(t+\tau)\big) \tag{2.3a}$$

Consider now the unshifted ODE (with t' the independent variable denoting time),

$$\frac{\mathrm{d}^n}{\mathrm{d}t'^n}y(t') + \cdots + P_2(t')\frac{\mathrm{d}^2}{\mathrm{d}t'^2}y(t') + P_1(t')\frac{\mathrm{d}}{\mathrm{d}t'}y(t') + P_0(t')y(t') = Q\big(u(t')\big) \tag{2.3b}$$

If the change of variable $t' = t+\tau$ is made, with τ a constant, the change of variable gives $\mathrm{d}t = \mathrm{d}t'$. Hence, via the chain rule,

$$\frac{\mathrm{d}^n}{\mathrm{d}t'^n}y(t') = \frac{\mathrm{d}^n}{\mathrm{d}t'^n}y(t+\tau) = \frac{\mathrm{d}^n}{\mathrm{d}t^n}y(t+\tau)\cdot\left(\frac{\mathrm{d}t}{\mathrm{d}t'}\right)^n = \frac{\mathrm{d}^n}{\mathrm{d}t^n}y(t+\tau) \tag{2.4}$$

Making use of this substitution and the results from (2.4) it is apparent that

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}y(t+\tau) + \cdots + P_1(t+\tau)\frac{\mathrm{d}}{\mathrm{d}t}y(t+\tau) + P_0(t+\tau)y(t+\tau) = Q\big(u(t+\tau)\big) \tag{2.5}$$

Comparison of (2.5) with (2.3a) (excluding the n-th term as it has no coefficient) yields

$$P_m(t)\frac{\mathrm{d}^m}{\mathrm{d}t^m}\hat{y}(t) = P_m(t+\tau)\frac{\mathrm{d}^m}{\mathrm{d}t^m}y(t+\tau) \tag{2.6}$$

with $m < n$ for $m, n \in \mathbb{N}^0$.

Firstly, this implies that

$$P_m(t) = P_m(t+\tau) \tag{2.7a}$$

In general, this will not be true for arbitrary functions $P_m(t)$, which implies that P_m is *constant*. Further to this and coupled by consideration of the n-th terms of each expression, it is also implied that

$$\hat{y}(t) = y(t+\tau) \tag{2.7b}$$

Using the results from (2.7a) and (2.7b) with (2.3a) gives

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}y(t+\tau) + \cdots + P_1\frac{\mathrm{d}}{\mathrm{d}t}y(t+\tau) + P_0 y(t+\tau) = Q\big(u(t+\tau)\big) \tag{2.8}$$

So that the output $\hat{y}(t)$ is the basic output $y(t)$ with equivalent time-shifting to the input $u(t+\tau)$.
i.e.

$$u(t) \mapsto y(t) \implies u(t+\tau) \mapsto y(t+\tau) \text{ as required.}$$

Therefore, the general form of a LTI ordinary differential equation (with single input) is

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}y(t) + \cdots + P_1\frac{\mathrm{d}}{\mathrm{d}t}y(t) + P_0 y(t) = u(t) \tag{2.9}$$

Principle 2 (The Time-Invariance Principle)**.** Any system described by a linear ODE with constant coefficients will have the property that if the system input $u(t)$ yields a system output of $y(t)$, then the time-shifted system input $u(t+\tau)$ yields a system output of $y(t+\tau)$ shifted by the same quantity.
Mathematically, In system notation, the time-invariance principle is demonstrated as

$$\boxed{\mathcal{H}\left\{u(t+\tau), t\right\} = y(t+\tau)}$$

2.1.4 The Dirac Delta Function

The Dirac delta function $\delta(\tau)$, named for Paul Dirac, is a non-physical function with a singularity given an argument of zero.
i.e

$$\delta(\tau) := \begin{cases} 0 & : \tau \neq 0 \\ \infty & : \tau = 0 \end{cases} \tag{2.10}$$

The Dirac delta function is defined to have unit area. That is,

$$\int_{-\infty}^{\infty} \delta(\tau)\,\mathrm{d}\tau := 1 \tag{2.11}$$

By further definition,

$$\delta(\tau) := \frac{\mathrm{d}}{\mathrm{d}\tau}H(\tau) \tag{2.12}$$

where

$$H(\tau) := \begin{cases} 1 & : \tau > 0 \\ 0 & : \tau \leqslant 0 \end{cases} \tag{2.13}$$

is the Heaviside Step function. Another useful definition of the Dirac delta function is

$$\int_{-\infty}^{\infty} f(\tau)\delta(\tau)\,\mathrm{d}t = f(0) \tag{2.14}$$

This can be verified using integration by parts and (2.12), assuming $\lim_{\tau\to\infty} f(\tau) = 0$:

$$\begin{aligned}\int_{-\infty}^{\infty} f(\tau)\delta(\tau)\,\mathrm{d}\tau &= [f(\tau)H(\tau)]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} f'(\tau)H(\tau)\,\mathrm{d}\tau \\ &= -\int_{-\infty}^{\infty} f'(\tau)H(\tau)\,\mathrm{d}\tau \\ &= -\int_{0}^{\infty} f'(\tau)\,\mathrm{d}\tau \\ &= f(0) \text{ as required.}\end{aligned}$$

In fact, the more general equation (with the Dirac delta function centered on $\tau = t$) is[19]

$$\int_{-\infty}^{\infty} f(\tau)\delta(t-\tau)\,\mathrm{d}\tau = f(t) \tag{2.15}$$

Expression (2.15) can be verified by making the substitution $\tau \to \tau + t$ in the integrand of (2.14) and by noting that $\delta(\alpha) = \delta(-\alpha)$:

$$\begin{aligned} \int_{-\infty}^{\infty} f(\tau)\delta(\tau)\,\mathrm{d}\tau &= f(0) \\ \rightarrow \int_{-\infty}^{\infty} f(\tau+t)\delta(\tau)\,\mathrm{d}\tau &= f(t) \\ \int_{-\infty}^{\infty} f(\tau)\delta(\tau-t)\,\mathrm{d}\tau &= f(t) \\ \int_{-\infty}^{\infty} f(\tau)\delta(t-\tau)\,\mathrm{d}\tau &= f(t) \end{aligned}$$

The Dirac delta function is used extensively in the treatment of linear systems as, by using expression (2.15), any system input can be rewritten in the form

$$u(t) = \int_{-\infty}^{\infty} u(\tau)\delta(t-\tau)\,\mathrm{d}\tau \tag{2.16}$$

Consider the output y(t) of some system LTI system $\mathcal{G}$. Then, using the superposition principle (specifically, the homogeneity condition)

$$\begin{aligned} y(t) &= \mathcal{G}\left\{u(t), t\right\} \\ &= \mathcal{G}\left\{\int_{-\infty}^{\infty} u(\tau)\delta(t-\tau)\,\mathrm{d}\tau, t\right\} \\ &= \int_{-\infty}^{\infty} u(\tau).\mathcal{G}\left\{\delta(t-\tau), t\right\}\,\mathrm{d}\tau \end{aligned}$$

The term $\mathcal{G}\left\{\delta(t-\tau), t\right\}$, labeled as $h(t-\tau)$, is simply the output from system $\mathcal{G}$ resulting from the Dirac delta function as an input.

i.e

$$h(t) = \mathcal{G}\left\{\delta(t), t\right\} \tag{2.17}$$

Therefore,

$$y(t) = \int_{-\infty}^{\infty} u(\tau)h(t-\tau)\,\mathrm{d}\tau \tag{2.18}$$

The function $h(\cdot)$, known as the *impulse response function*, fully characterises an LTI system in that only $h(t)$ and the input $u(t)$ are required to calculate the output $y(t)$.

2.1.5 Convolution Integrals

The form of the argument in both $\delta(t-\tau)$ and $h(t-\tau)$, while peculiar at first glance, has been chosen with judicious care. The notation

$$f(t) \ast g(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,\mathrm{d}\tau \tag{2.19}$$

is used to represent the *convolution* of two continious functions $f(t)$ and $g(t)$. Therefore, expression (2.18) becomes

$$y(t) = u(t) * h(t) \tag{2.20}$$

2.1.6 The Causality Principle

As shown, the impulse response function $h(t)$ of an LTI system characterises the behaviour of the system. Looking back at the general form for a LTI ordinary differential equation (2.9), the homogenous LTI ODE is

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}y_{\mathrm{H}}(t) + \cdots + P_1\frac{\mathrm{d}}{\mathrm{d}t}y_{\mathrm{H}}(t) + P_0 y_{\mathrm{H}}(t) = 0 \tag{2.21}$$

with homogenous solution $y_{\mathrm{H}}(t)$. In linear systems theory, the solution $y_{\mathrm{H}}(t)$ is known as the *natural* or *transient* response function of a system; that is, the response of the system to no input.[19] Therefore, if $h_1(t)$ is considered the particular solution (or output) to the LTI ordinary differential equation with input $\delta(t)$ and $y_{\mathrm{H}}(t)$ the system output resulting from an input of 0, then by the superposition principle $h_2(t) = h_1(t) + y_{\mathrm{H}}(t)$ is also a valid impulse response function for the system with system input $\delta(t)$. This is easily shown by:

$$\begin{aligned}\left[\frac{\mathrm{d}^n}{\mathrm{d}t^n}+\cdots+P_1\frac{\mathrm{d}}{\mathrm{d}t}+P_0\right]h_2(t) &= \left[\frac{\mathrm{d}^n}{\mathrm{d}t^n}+\cdots+P_1\frac{\mathrm{d}}{\mathrm{d}t}+P_0\right]\big(h_1(t)+y_{\mathrm{H}}(t)\big)\\ &= \left[\frac{\mathrm{d}^n}{\mathrm{d}t^n}+\cdots+P_1\frac{\mathrm{d}}{\mathrm{d}t}+P_0\right]h_1(t)\\ &\quad+\left[\frac{\mathrm{d}^n}{\mathrm{d}t^n}+\cdots+P_1\frac{\mathrm{d}}{\mathrm{d}t}+P_0\right]y_{\mathrm{H}}(t)\\ &= \delta(t)+0\\ &= \delta(t)\end{aligned}$$

Therefore, $h(t)$ is not uniquely determined by the system. As LTI systems theory is most often applied to the modelling of physical systems, it is fitting to invoke the causality principle and hence impose physical constraints upon the system. In simple terms, the causality principle states that the cause cannot precede the effect. That is, the output is zero until the system experiences an input.

Principle 3 (The Causality Principle)**.** The output $y(t-\tau)$ of any LTI system must vanish prior to the input start time τ. Moreover, the impulse response function for any LTI system must vanish for all $t<\tau$.
In system notation

$$\boxed{h(t-\tau)=\mathcal{H}\left\{\delta(t-\tau),t\right\}=0,\text{ for all } t<\tau}$$

The causality principle forces selection of the $h(t)$ that vanishes for all time less than τ. If we construct a new impulse response function as we have previously $h_{\text{new}}(t) = h(t)+y_H(t)$ as previously, $h_{\text{new}}(t)$ is only valid if it satisfies the causality principle. As $h(t)$ satisfies the causality principle, for $h_{\text{new}}(t)$ to also satisfy the causality principle requires that $y_H(t)=0$, which is only true in the trivial case but not in general. Hence, $h(t)$ is the unique impulse function for the LTI system.

2.2 State-Space Representation

The *state-space* representation is a representation of an n^{th} order ODE as an n-vector array of 1^{st}-order linear ODEs. The main purpose of the state-space representation of ODEs is to simplify calculations that would otherwise involve tedious integral transforms and to allow for easier treatment of multiple input and output systems.

The general structure of a continuous-time linear dynamical system in state-space representation is[19]

State Equation

$$\dot{\mathbf{x}}(t)=A(t)\mathbf{x}(t)+B(t)\mathbf{u}(t) \tag{2.22}$$

Output Equation

$$\mathbf{y}(t) = C(t)\mathbf{x}(t) + D(t)\mathbf{u}(t) \tag{2.23}$$

Impulse Response Function

$$h(t) = C(t)\mathrm{e}^{tA(t)}B(t) \tag{2.24}$$

where

- t usually denotes time (in signal processing uses) but can be a spatial variable
- $\mathbf{x}(t)$ is called the system state (vector)
- $\mathbf{u}(t)$ is called the system input (vector). Written $u(t)$ in the single input case
- $\mathbf{y}(t)$ is called the system output vector). Written $y(t)$ in the single output case
- $A(t)$ is called the system or dynamics matrix, and determines the affect the current state $\mathbf{x}(t)$ has on the state change $\mathbf{x}'(t)$
- $B(t)$ is called the control or input matrix, and determines the affect that the system input $\mathbf{u}(t)$ has upon the state change $\mathbf{x}'(t)$
- $C(t)$ is called the output or sensor matrix, and determines the relationship between the system state $\mathbf{x}(t)$ and system output $\mathbf{y}(t)$
- $D(t)$ is called the feed-forward or feedthrough matrix, and determines the affect that system input $\mathbf{u}(t)$ will have on system output $\mathbf{y}(t)$

In the case of time-invariance, these become

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{2.25}$$

$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t) \tag{2.26}$$

$$h(t) = C\mathrm{e}^{tA}B \tag{2.27}$$

The explicit solution to state equation is given by[19]

$$\mathbf{x}(\rho_{i+1}, z) = \Phi(\rho_{i+1}, \rho_i)\mathbf{x}(\rho_i, z) - \frac{1}{\pi}\int_{\rho_i}^{\rho_{i+1}} \frac{\Phi(\rho_{i+1}, r)\tilde{B}}{r}\frac{\partial F(r,z)}{\partial r}\,\mathrm{d}r \tag{2.28}$$

where

$$\Phi(\rho, \rho_0) = \mathrm{e}^{A\ln\left(\frac{\rho_0}{\rho}\right)} = \begin{bmatrix} \left(\frac{\rho_0}{\rho}\right)^{\lambda_1} & 0 & \cdots & 0 \\ 0 & \left(\frac{\rho_0}{\rho}\right)^{\lambda_2} & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\rho_0}{\rho}\right)^{\lambda_k} \end{bmatrix} \tag{2.29}$$

2.2.1 Example 1: Hooke's Law with Damping

A good introductory example to state-space representation is the consideration of a familiar linear second order mass-spring system, Hooke's law with a damping term:

$$m\ddot{z}(t) + d\dot{z}(t) + kz(t) = F$$

where $z(t)$ is the displacement of the mass, $\dot{z}(t)$ is the velocity of the mass, $\ddot{z}(t)$ is the acceleration of the mass with m as the value of the mass, d is the damping coefficient and k is Hooke's constant.

Choose the components of $x_1(t) = z(t)$ and $x_2(t) = \dot{z}(t) = \dot{x}_1(t)$

$$\dot{x}_2(t) = \ddot{z}(t) = \frac{F - d\dot{z}(t) - kz(t)}{m} = \frac{F - dx_2(t) - kx_1(t)}{m}$$

$$\Rightarrow \begin{bmatrix} \dot{z}(t) \\ \ddot{z}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} z(t) \\ \dot{z}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F$$

So, if we take the system input to be the external forces $u = F$ and with system state

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} z(t) \\ \dot{z}(t) \end{bmatrix}$$

we have

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{d}{m} \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t) = A\mathbf{x}(t) + Bu(t)$$

If the output is the position of the mass then

$$\mathbf{y}(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} z(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = C\mathbf{x}(t)$$

2.2.2 Example 2: 3rd-order Ordinary Differential Equation

Take the following general 3rd-order ODE

$$\frac{\mathrm{d}^3}{\mathrm{d}t^3}y(t) + \alpha(t)\frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) + \beta(t)\frac{\mathrm{d}}{\mathrm{d}t}y(t) + \gamma(t)y(t) = u(t)$$

Assemble the state vector $\mathbf{x}(t)$ as follows:

$$\begin{aligned} x_1(t) &= y(t) \\ x_2(t) &= \frac{\mathrm{d}}{\mathrm{d}t}y(t) \\ x_3(t) &= \frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) \end{aligned}$$

Then

$$\dot{x_1}(t) = x_2(t) = \frac{\mathrm{d}}{\mathrm{d}t}y(t)$$
$$\dot{x_2}(t) = x_3(t) = \frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t)$$
$$\dot{x_3}(t) = \frac{\mathrm{d}^3}{\mathrm{d}t^3}y(t)$$

The state vector and its derivative are then

$$\mathbf{x}(t) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y(t) \\ \frac{\mathrm{d}}{\mathrm{d}t}y(t) \\ \frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) \end{bmatrix}$$

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t}y(t) \\ \frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) \\ \frac{\mathrm{d}^3}{\mathrm{d}t^3}y(t) \end{bmatrix}$$

Therefore, the state-space representation is

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\alpha(t) & -\beta(t) & -\gamma(t) \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \mathbf{x}(t)$$

This result can easily be generalised for an n-th order linear ODE by using expression (2.1). As previously demonstrated, the dynamics behind an LTI system are governed by a differential equation of the form given in expression (2.9). Therefore, any (single input-single output) n-th order LTI system can be represented by a set of state-space equations:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & & I_{n-2} \\ \vdots & & \\ -P_{n-1} & \cdots & -P_1 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ \vdots \\ b_1 \end{bmatrix} u(t) \tag{2.30}$$

$$y(t) = \begin{bmatrix} 0 & \cdots & 1 \end{bmatrix} \mathbf{x}(t) \tag{2.31}$$

where I_m is the identity matrix and P_m constants.

Chapter 3

The Abel & Inverse Abel Transforms

From henceforth, the particular naming practice of ‘time’ invariant is unfortunate - this section is primarily concerned with *space*-invariance as our functions act over the space-domain. As such, the invariance in following sections will also be referred to as *shift* invariance, but follows the same properties as previously presented.

In Section 3.1, the forward Abel transform (FAT) is presented with a brief description and physical interpretation. Section 3.2 introduces the inverse Abel transform (IAT) and a modified version with shift-invariant properties. Section 3.3 introduces the state-space representation of LTI ODEs, with the result being discretised in Section 3.4. In Section 3.5, a C++ implementation of the scheme is discussed.

3.1 The Abel Transform

Any projection of a cylindrically symmetrical distribution onto a plane parallel to the axis of symmetry is described as an Abel transform of the distribution. Lasers are commonly used in ion imaging technique for excitation and photodetachment, and as such can impart cylindrical symmetry upon a system under examination. The geometry of most ion imaging techniques and subsequent projection image of the target distribution is a common issue for direct imaging techniques. The Abel transform, as well as the inverse Abel transform, are shown graphically in Figure 3.1.

Mathematically, for a system in which the z-axis is the axis of symmetry, the Abel transform of a 3D distribution function $f(\rho, z)$ is given by[18]

$$F(x,z) = 2\int_{\rho}^{\infty} \frac{f(\rho,z)\rho}{\sqrt{\rho^2 - x^2}}\,\mathrm{d}\rho \tag{3.1}$$

where

$$\rho^2 = x^2 + y^2$$

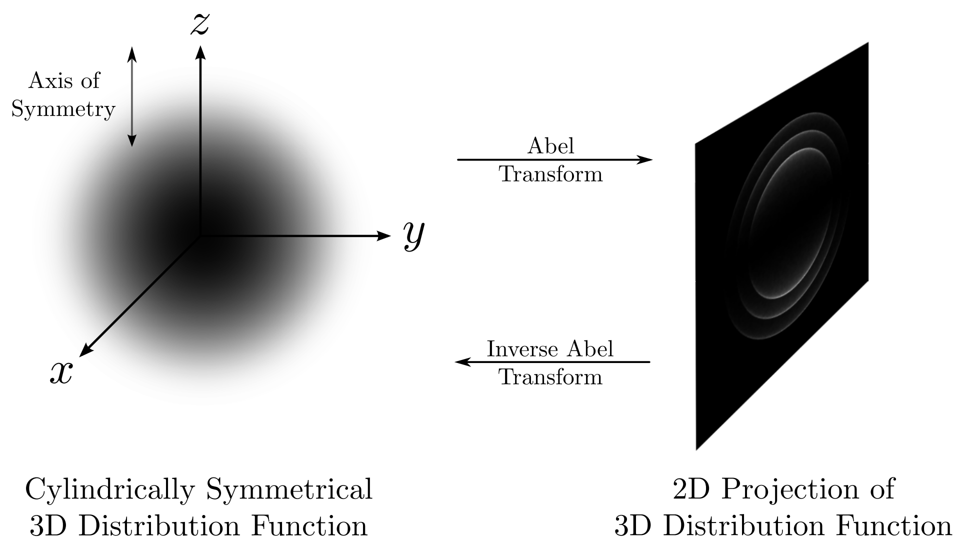

z
Axis of Symmetry
y
x
Abel Transform
Inverse Abel Transform
Cylindrically Symmetrical 3D Distribution Function
2D Projection of 3D Distribution Function

Figure 3.1: Visual representation of the Abel and inverse Abel transforms

3.2 The Inverse Abel Transform

Due to the projection issue in VMI and other ion imaging techniques, the inverse Abel transform (IAT) is used extensively for the treatment of experimental images. The IAT offers a means to recover a slice of a cylindrically symmetrical 3D system from the 2D projection of said system onto a plane perpendicular to the axis of symmetry.

Mathematically, for a system in which the z-axis is the axis of symmetry, the inverse Abel transform of a 2D projection $F(x,z)$ is given by[18]

$$f(\rho,z) = -\frac{1}{\pi}\int_{\rho}^{\infty}\frac{\partial F(x,z)}{\partial x}\frac{\mathrm{d}x}{\sqrt{x^2-\rho^2}} \tag{3.2}$$

where ρ is the cylindrical radius ($\rho^2 = x^2 + y^2$) and it is assumed that $F(x,z)$ approaches zero faster than x^{-1}. Now, with the use of the Heaviside step function and a minor rearrangement, the bounds can be altered to give

$$f(\rho,z) = \int_{0}^{\infty}-\frac{1}{\pi}\frac{\partial F(x,z)}{\partial x}\frac{H\left(1-\frac{\rho}{x}\right)}{x\sqrt{1-\left(\frac{\rho}{x}\right)^2}}\mathrm{d}x \tag{3.3}$$

with $H(1-\frac{\rho}{x})$ being the Heaviside step function as defined as in expression (2.13). Now, a point transformation can be applied to ρ and x to alter the form of the transformation: Let

$$x = \mathrm{e}^{-t} \implies t = -\ln(x) \quad ; \quad \rho = \mathrm{e}^{-\tau} \implies \tau = -\ln(\rho)$$

and

$$\frac{\mathrm{d}x}{dt} = -\mathrm{e}^{-t} \implies \mathrm{d}x = -\mathrm{e}^{-t}\mathrm{d}t$$

Note that as $x \to \infty$, $t \to -\infty$ and as $x \to 0$, $t \to \infty$. Use tilde to differentiate new modified functions:

$$f(\rho = \mathrm{e}^{-\tau}, z) \to \tilde{f}(\tau, z) \quad ; \quad F(x = \mathrm{e}^{-t}, z) \to \tilde{F}(t, z)$$

By the chain rule,

$$\frac{\mathrm{d}\tilde{F}}{\mathrm{d}t} = \frac{\mathrm{d}F}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}t} = -\frac{\mathrm{d}F}{\mathrm{d}x}\mathrm{e}^{-t}$$

So

$$\begin{aligned}\tilde{f}(\tau, z) = f(\mathrm{e}^{-\tau}, z) &= \int_{\infty}^{-\infty} -\frac{1}{\pi}\frac{1}{(-\mathrm{e}^{-t})}\frac{\partial \tilde{F}(t)}{\partial t}\frac{H\left(1 - \frac{\mathrm{e}^{-\tau}}{\mathrm{e}^{-t}}\right)}{\mathrm{e}^{-t}\sqrt{1 - \left(\frac{\mathrm{e}^{-\tau}}{\mathrm{e}^{-t}}\right)^2}}\left(-\mathrm{e}^{-t}\right)\mathrm{d}t \\ &= \int_{-\infty}^{\infty}\frac{1}{\pi}\frac{\partial \tilde{F}(t)}{\partial t}\frac{H\left(1 - \mathrm{e}^{-(\tau - t)}\right)}{\mathrm{e}^{-t}\sqrt{1 - \mathrm{e}^{-2(\tau - t)}}}\mathrm{d}t\end{aligned}$$

The Heaviside integrand will only be unity if

$$\mathrm{e}^{-(\tau - t)} < 1 \implies -(\tau - t) < 0 \implies \tau > t$$

Hence, the Heaviside can be rewritten to be $H(\tau - t) = H(t - \tau)$:

$$\tilde{f}(\tau, z) = \int_{-\infty}^{\infty}\frac{1}{\pi}\frac{\partial \tilde{F}(t)}{\partial t}\frac{H(\tau - t)}{\mathrm{e}^{-t}\sqrt{1 - \mathrm{e}^{-2(\tau - t)}}}dt \tag{3.4}$$

With a slight rearrangement,

$$\tilde{f}(\tau, z) = \int_{-\infty}^{\infty}\frac{1}{\pi\mathrm{e}^{-t}}\frac{\partial \tilde{F}(t)}{\partial t}\frac{H(\tau - t)}{\sqrt{1 - \mathrm{e}^{-2(\tau - t)}}}dt \tag{3.5}$$

Defining the (modified) input as

$$\tilde{u}(\tau) = \frac{1}{\pi\mathrm{e}^{-\tau}}\frac{\partial \tilde{F}(\tau)}{\partial \tau} \tag{3.6}$$

and the (modified) impulse response function as

$$\tilde{h}(\tau) = \frac{1}{\sqrt{1 - \mathrm{e}^{-2\tau}}} \quad ; \quad \tau \geqslant 0 \tag{3.7}$$

it is apparent that this modified IAT is in the form of a convolution. Hence, the modified

IAT can be written in convolution notation as

$$\tilde{f} = \tilde{u} \ast \tilde{h} \tag{3.8}$$

3.3 The IAT as a State-Space system

As shown, convolution is a linear space-invariant operation. Hence, the IAT can be represented as a LTI system with the state-space representation

$$\begin{aligned}
\tilde{\mathbf{x}}'(\tau, z) &= \tilde{A}\tilde{\mathbf{x}}(\tau, z) + \tilde{B}\left(\frac{1}{\pi \mathrm{e}^{-\tau}} \frac{\partial \tilde{F}(\tau, z)}{\partial \tau}\right) = \tilde{A}\tilde{\mathbf{x}}(\tau, z) + \tilde{B}\tilde{u}(\tau, z) \\
\tilde{f}(\tau, z) &= \tilde{C}\tilde{\mathbf{x}}(\tau, z) + \tilde{D}\left(\frac{1}{\pi \mathrm{e}^{-\tau}} \frac{\partial \tilde{F}(\tau, z)}{\partial \tau}\right) = \tilde{C}\tilde{\mathbf{x}}(\tau, z) + \tilde{D}\tilde{u}(\tau, z)
\end{aligned} \tag{3.9}$$

As $\tilde{A}$ is a square matrix, it is convenient to choose a diagonal matrix and the simple case of no direct-feedthrough, giving

$$\tilde{A} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_k \end{bmatrix} \tag{3.10}$$

$$\tilde{B} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \tag{3.11}$$

$$\tilde{C} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \tag{3.12}$$

Recognise that the exponential of any square matrix X is defined (via Taylor series) to be

$$\mathrm{e}^{X} = 1 + X + \frac{1}{2!}X^2 + \frac{1}{3!}X^3 + \ldots = \sum_{n=0}^{\infty} \frac{1}{n!}X^n$$

Note also that

$$\tilde{A}^n = \begin{bmatrix} \lambda_1^n & 0 & \cdots & 0 \\ 0 & \lambda_2^n & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_k^n \end{bmatrix} \implies \mathrm{e}^{A\tau} = \begin{bmatrix} \mathrm{e}^{\lambda_1 \tau} & 0 & \cdots & 0 \\ 0 & \mathrm{e}^{\lambda_2 \tau} & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathrm{e}^{\lambda_k \tau} \end{bmatrix}$$

Hence,

$$\tilde{h}(\tau) = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \mathrm{e}^{\lambda_1 \tau} & 0 & \cdots & 0 \\ 0 & \mathrm{e}^{\lambda_2 \tau} & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathrm{e}^{\lambda_k \tau} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} = \sum_{n=1}^{k} b_n \mathrm{e}^{\lambda_n \tau} \tag{3.13}$$

i.e

$$\sum_{n=1}^{k} b_n \mathrm{e}^{\lambda_n \tau} = \frac{1}{\sqrt{1 - \mathrm{e}^{-2\tau}}} \tag{3.14}$$

Parameters to this expression have been calculated to 9th order by Hansen[18] with an rms error of 0.001. These values are given in Table 3.1.

k	b_k	λ_k
1	0·318	0
2	0·19	−2·1
3	0·35	−6·2
4	0·82	−22·4
5	1·8	−92·5
6	3·9	−414·5
7	8·3	−1889·4
8	19·6	−8990·9
9	48·3	−47391·1

Table 3.1: 9th order parameter fit to Expression (3.14)

Utilising the inverse coordinate transform

$$\tau = -\ln(\rho) \implies \mathrm{d}\tau = -\frac{1}{\rho}\,\mathrm{d}\rho$$

and expression (3.9) gives

$$-\rho \mathbf{x}'(\rho, z) = \tilde{A}\mathbf{x}(\rho, z) + \tilde{B}\left(\frac{1}{\pi\rho}\left(-\rho\,\frac{\partial F(\rho, z)}{\partial \rho}\right)\right)$$

$$f(\rho, z) = \tilde{C}\mathbf{x}(\rho, z)$$

Therefore

$$\mathbf{x}'(\rho, z) = -\frac{1}{\rho}\tilde{A}\mathbf{x}(\rho, z) + \frac{1}{\pi\rho}\tilde{B}\,\frac{\partial F(\rho, z)}{\partial \rho}$$

$$f(\rho, z) = \tilde{C}\mathbf{x}(\rho, z) \tag{3.15}$$

3.4 Discretising the Inverse Abel Transform

Previous working has only dealt with the continous case but it is far more practical, with regards to the nature of image treatment, that current results be adjusted for the continous case.[20] First, the equation is written as

$$\mathbf{x}(\rho_{i+1}, z) = \Phi(\rho_{i+1}, \rho_i)\mathbf{x}(\rho_i, z) - \frac{1}{\pi}\int_{\rho_i}^{\rho_{i+1}} \frac{\Phi(\rho_{i+1}, r)\tilde{B}}{r}\frac{\partial F(r,z)}{\partial r}\,\mathrm{d}r$$

Let

$$\begin{aligned}\rho_i &= (N-i)\,\Delta\\ \rho_{i+1} &= (N-i-1)\,\Delta\end{aligned}$$

with

$$\Delta = \frac{\rho_{max}}{N-1}$$

where N is the total number of data points, Δ is the data stepsize and $i[0, N-2]$. Then, using the previous definitions for $\Phi(\rho_{i+1}, \rho_i)$ and $\tilde{B}$ and assuming that $\partial_r F(r,z)$ is constant on $[\rho_i, \rho_{i+1}]$ (staircase / zero-order hold approximation) gives

$$\begin{aligned}\begin{bmatrix} x_1(\rho_{i+1}) \\ x_2(\rho_{i+1}) \\ \vdots \\ x_k(\rho_{i+1}) \end{bmatrix} &= \begin{bmatrix} \left(\frac{N-i}{N-i-1}\right)^{\lambda_1} & 0 & \cdots & 0 \\ 0 & \left(\frac{N-i}{N-i-1}\right)^{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{N-i}{N-i-1}\right)^{\lambda_k} \end{bmatrix} \begin{bmatrix} x_1(\rho_i) \\ x_2(\rho_i) \\ \vdots \\ x_k(\rho_i) \end{bmatrix} \\ &\quad - \frac{1}{\pi}\frac{\partial F(\rho_i, z)}{\partial \rho_i}\int_{\rho_i}^{\rho_{i+1}} \begin{bmatrix} \left(\frac{r}{\rho_{i+1}}\right)^{\lambda_1} & 0 & \cdots & 0 \\ 0 & \left(\frac{r}{\rho_{i+1}}\right)^{\lambda_1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{r}{\rho_{i+1}}\right)^{\lambda_k} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \frac{\mathrm{d}r}{r}\end{aligned}$$

Focusing on the integral,

$$\int_{\rho_i}^{\rho_{i+1}} \begin{bmatrix} \left(\frac{r}{\rho_{i+1}}\right)^{\lambda_1} & 0 & \cdots & 0 \\ 0 & \left(\frac{r}{\rho_{i+1}}\right)^{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{r}{\rho_{i+1}}\right)^{\lambda_k} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \frac{\mathrm{d}r}{r} = \begin{bmatrix} b_1 \int_{\rho_i}^{\rho_{i+1}} \frac{r^{\lambda_1 - 1}}{(\rho_{i+1})^{\lambda_1}}\,\mathrm{d}r \\ b_2 \int_{\rho_i}^{\rho_{i+1}} \frac{r^{\lambda_2 - 1}}{(\rho_{i+1})^{\lambda_2}}\,\mathrm{d}r \\ \vdots \\ b_k \int_{\rho_i}^{\rho_{i+1}} \frac{r^{\lambda_k - 1}}{(\rho_{i+1})^{\lambda_k}}\,\mathrm{d}r \end{bmatrix}$$

Now,

$$
\begin{aligned}
b_k \int_{\rho_i}^{\rho_{i+1}} \frac{r^{\lambda_k - 1}}{(\rho_{i+1})^{\lambda_k}} \,\mathrm{d}r &= \begin{cases} b_k \ln\left({}^{\rho_{i+1}}/_{\rho_i}\right) & \lambda_k = 0 \\ \frac{b_k}{\lambda_k r^{\lambda_k}} \left((\rho_{i+1})^{\lambda_k} - (\rho_i)^{\lambda_k}\right) & \lambda_k \neq 0 \end{cases} \\
&= \begin{cases} b_k \ln\left({}^{\rho_{i+1}}/_{\rho_i}\right) & \lambda_k = 0 \\ \frac{b_k}{\lambda_k} \left(1 - ({}^{\rho_i}/_{\rho_{i+1}})^{\lambda_k}\right) & \lambda_k \neq 0 \end{cases} \\
&= \begin{cases} -b_k \ln\left(\frac{N-i}{N-i-1}\right) & \lambda_k = 0 \\ \frac{b_k}{\lambda_k} \left(1 - \left(\frac{N-i}{N-i-1}\right)^{\lambda_k}\right) & \lambda_k \neq 0 \end{cases}
\end{aligned}
$$

Note that $\rho_i = \rho_{i+1} + \Delta$, so that ρ_i is the point *before* ρ_{i+1}. If the components are labelled by iteration as opposed to radius (that is, i instead of ρ) then this order is reversed and the final discrete solution state-equation expression (with $\partial F_i = \frac{\partial F(\rho_i, z)}{\partial \rho_i}$) is found to be

$$
\mathbf{x}_{i+1} = \Phi_i \, \mathbf{x}_i + \partial F_i \, \Gamma_i \tag{3.16}
$$

with

$$
\Gamma_i = \begin{bmatrix} b_1 \gamma_i(\lambda_1) \\ b_2 \gamma_i(\lambda_2) \\ \vdots \\ b_k \gamma_i(\lambda_k) \end{bmatrix}
$$

where

$$
\gamma_i(\lambda_k) = \begin{cases} -b_k \ln\left(\frac{N-i}{N-i-1}\right) & \lambda_k = 0 \\ \frac{b_k}{\lambda_k} \left(1 - \left(\frac{N-i}{N-i-1}\right)^{\lambda_k}\right) & \lambda_k \neq 0 \end{cases}
$$

3.5 Realisation of an IAT Script

This formulation works by progressively evaluating inwards towards the origin of $F(x, z)$ based on the previously calculated value. It should be noted that when $i = N - 1$, the system is undefined; that is when $\rho = 0$. It should also be noted that due to this singularity, the IAT actually amplifies noise close to the origin and is a significant issue with the technique. Due to derivatives being calculated numerically, noise is also amplified and some degree of interpolation is employed. These drawbacks are perhaps outweighed by the benefit of faster computation, as for N data points and k state variables, computing the IAT this way has a computational burden of $\mathcal{O}(2kN)$ whereas direct numerical integration is $\mathcal{O}(N^2)$[18].

The IAT is also very sensitive to noise near the axis of symmetry. Considering the original 3D distribution, all points located at some distance r from cyclindrical axis only contribute to points with $x \leqslant r$ in the projection (see Expression (3.2)). Hence, the projection of the distribution actually contains *less* information about the image regions closer to the symmetry axis in comparison to those further away. Therefore, only $x = 0$ points contain information about the centreline of the original distribution, and every point in the projection contains information about points with $r = r_{\max}$. Experimentally, noise is evenly distributed across the experimental image; as such, the reconstructed image has increased noise towards the centreline.

A C++ implementation of the state-space approach to the IAT was provided by the Gascooke group, Flinders University. A 401 pixel by 401 pixel test image centred on (201,201) (see Figure 3.2) was also provided, consisting of 3 equidistant rings representative of the type of image that will be collected by the camera. A distorted version of the image was produced by scaling the horizontal axis with an increase of 50 pixels in total width, giving the distorted image dimensions of 451 pixels by 401 pixels centred on (226,201). A pre-compiled *Windows* executable was also included that was the most recently updated version of the script, allowing for initial testing before the porting was complete. All computers in the laboratory, excluding one low hardware-specification *Windows XP* machine, currently run on the *Linux Mint* operating system due to free-licensing and software flexibility. The original script was compiled using a *Borland* C++ compiler on *Windows XP* and as such employed commands specific to both *Borland* and *Windows* C++ programming. Many of these headers do not have direct analogues from *Windows* to *Linux* and as such an intimate knowledge of the script was required as well as a sufficient familiarity with the C++ programming language.

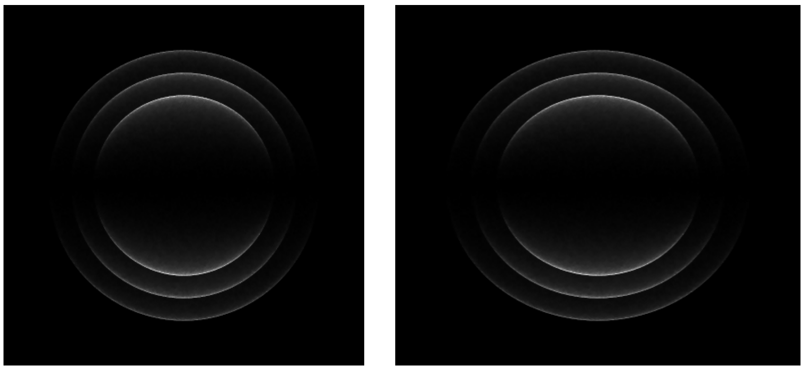
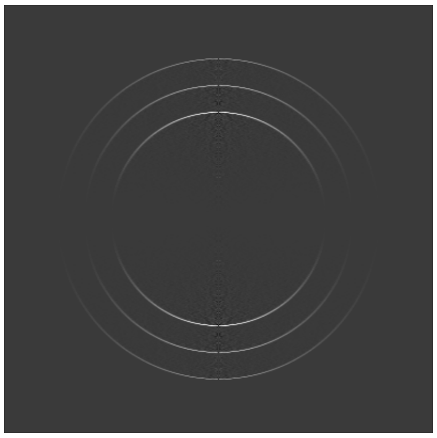
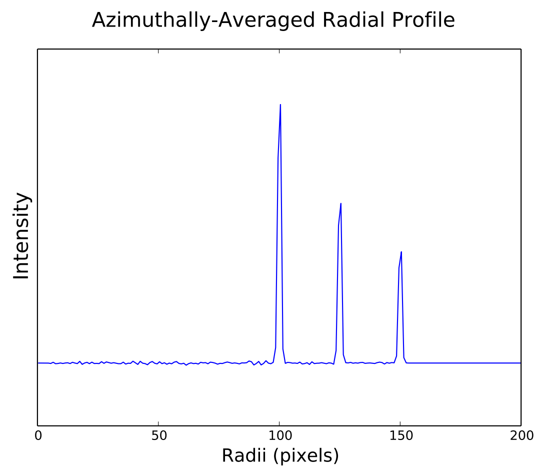

(a) Original test image (b) Distorted test image

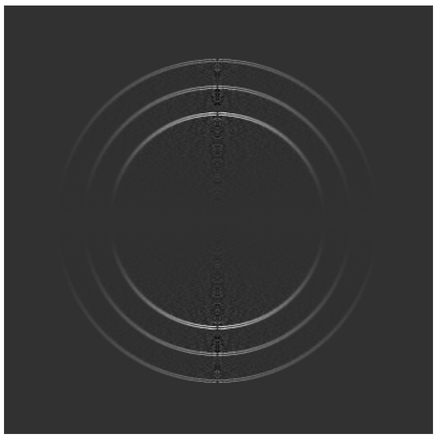
Figure 3.2: Test images

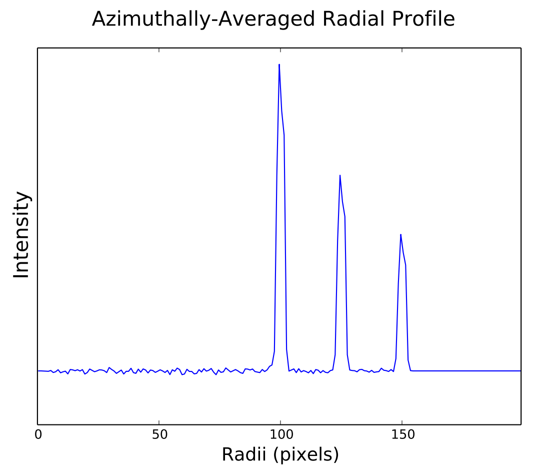
The script takes the file directory and the centre coordinates of the distribution in the image as inputs. As shown in Figure 3.3, as the distance between the input centre and the actual centre increases, the peaks begin to broaden which results in loss of resolution along the independent axis. As this distance approaches 2 pixels, splitting of the rings in the transformed image is readily apparent and verifiable upon referring to its radial profile. As such, the performance and precision of the IAT script is heavily dependent upon the centre detection procedure.


(a) IAT about (201,201)

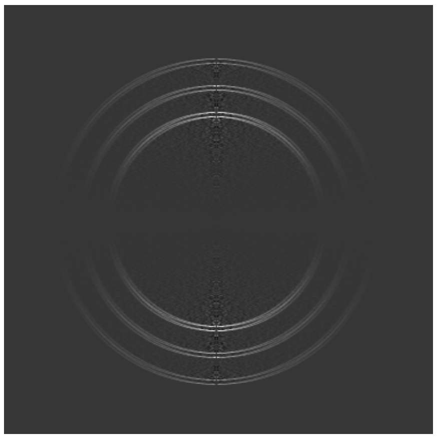


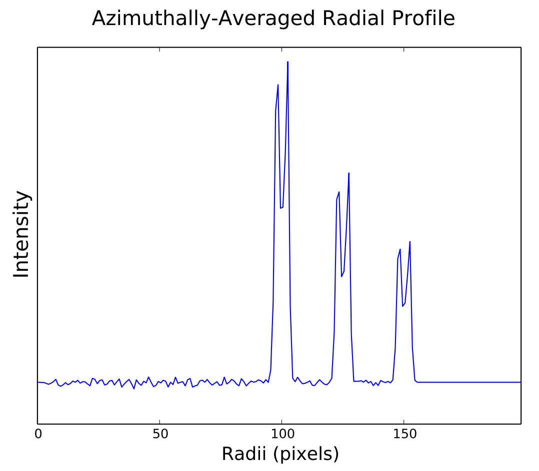
(b) Radial profile of (a)


(c) IAT about (200,200)



(d) Radial profile of (c)


(e) IAT about (199,199)



(f) Radial profile of (e)

Figure 3.3: Output images from IAT about three different centre coordinates with associated radial profile (*TesseracT*)

Chapter 4

Centre Detection in Experimental Images

By considering the cylindrical nature of the electrostatic mapping field, the form of the images can be exploited for use with detecting centres. In most experimental situations, images containing multiple circular rings will be detected; the rings will share a common centre due to the geometry of apparatus setup. Using a more general detecting method for ellipses as opposed to circles, it is possible to implement a script that both automatically centers the images even when misaligned. Such a script would allow for greater precision to maximise to resolution of the SEVI spectrometer, as well as allow for easy calibration using the ellipse parameters.

In Section 4.1, the RANSAC algorithm is introduced as well as advantages and disadvantages to the method. A RANSAC-based circle detection scheme is given in Section 4.2 along with the basic theory of operation. In Section 4.3 RANSAC-based ellipse detection scheme was implemented, building upon the circle detecting script, with the parameters investigated in Section 4.4. The new self-centering IAT script is introduced in Section 4.5, which continues on to discuss the testing results and current issues with the SC-IAT. The chapter concludes with a discussion in Section 4.6 on limitations and issues with the ellipse detection scheme, as well as future direction and improvements.

4.1 Random Sample Consensus (RANSAC)

Random sample consensus (RANSAC) is a sampling algorithm first introduced by Fischler and Bolles[21] for the purposes of fitting a model to experimental data. The RANSAC procedure begins by randomly selecting as small a data set as possible to fit a chosen model and then adds subsequent data points that fit the model criteria. If a certain threshold determined by the model is not met the algorithm will restart, choosing a different random set of data points. Once this threshold is met, the points fitting the criteria will be removed from active use and the algorithm will continue scanning samples of the remaining points for further model fitting. Once a certain minimum number of active points are left, the algorithm terminates, giving the parameters for the successful model fits. The power of RANSAC in comparison to other similar fitting techniques lies with the decrease in computation speed (due to a reduced number of calculations and the flexibility of model choice). With that said, a significant set back of the RANSAC scheme is that the sample points will only be used for one model, so models with overlap will suffer a slight drop in precision due to the loss of points.

4.2 Circle Detection Scheme

An open-source C++ implementation of a RANSAC circle detection script was provided by Kevin Hughes.[22] The overview given below is brief, but the original script has been included in Appendix A for further reference. The script relies heavily upon the open-source C++ computer vision library *OpenCV*[23]. Along with many features for the treatment of images (such as read, write etc), the library is indispensible for visualising and manipulating experimental images on-the-fly during computation. The overall procedure is summarised in Figure 4.3, highlighting the consistency checks and the recursive nature of the script.

4.2.1 Canny Edge Detection Algorithm

The circle detecting scheme first detects all edge pixel points in the experimental image by using the Canny edge detection algorithm implemented in *OpenCV* based on the work by Canny.[24] The algorithm was optimised to maximise the signal-to-noise ratio of the gradient, minimise multiple detection responses to a single edge and to localisation of the edges. All three optimisation aims work towards improving computation time and precision. The set of edge points is used for model fitting as the Canny edge detector significantly diminishes background noise which decreases the total number of points in active use, leading to a better model fit and faster computation time.

The algorithm first smoothes the image to reduce background noise appearing in the edge set of points. In the second step, the gradients at each pixel on the image are calculated. Thirdly, a technique called *non-maximum suppression* is emplyed to thin the edges, by checking if the point is a local maximum. The final step, a technique known as *edge tracking by hysteresis* involves confirmation that the detected edges are indeed actual edges in the image by designating an upper and lower Canny threshold.[25] All detected edges above the upper Canny threshold $C_{\max}$ are considered ‘definite’ edges, all detected edges below the lower threshold are discarded and all detected edges laying between the thresholds that are not part of a ‘definite’ edge are discarded. The final step has the added benefit of removing a substantial amount of small pixel noise, and is implemented in *OpenCV* such that the lower threshold is set to half of the value of the upper threshold; Canny recommends an upper threshold:lower threshold ratio of between 2:1 and 3:1.[24]

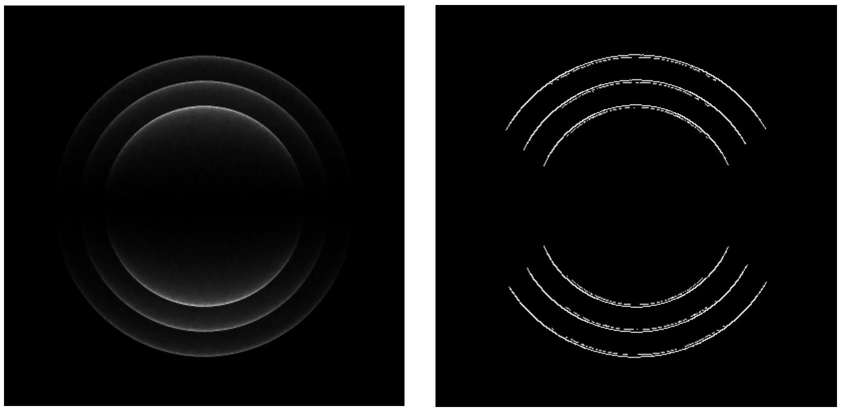

(a) Input image (b) Canny edge detection output

Figure 4.1: Canny edge detection output with $C_{\max} = 190$

Figure 4.1 shows a comparison between an input image and the associated Canny edge detection output.

4.2.2 RANSAC Step

A set of four points A, B, C, D are then randomly selected from the edge pixel set using the *OpenCV* random number generator. A consistency check then ensures the points are separated beyond a set minimum separation to ensure a good fit. From these, a line intersecting the first and second point (AB) and another line intersecting the second and third random point (BC) are calculated. Another consistency check then ensures that the four points are not collinear within another set tolerance. For AB and BC, perpendicular bisectors through the midpoints (P_{AB} and P_{BC}) are calculated and the intersection point of these is determined (as shown in Figure 4.2). This is taken as the center of the circle X[26], with a third consistency check determining if D lies upon the circle.

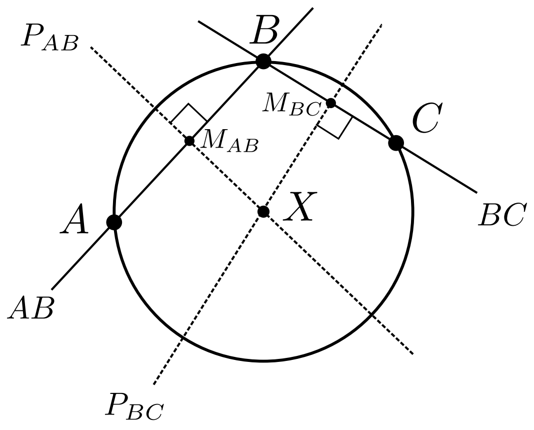


Figure 4.2: Geometry of test centre determination

A voting procedure then determines the proportion of points lying on the circle. By exploiting the symmetry of the circle, test radii are calculated for each edge point and test centre. If the test radius is within a certain tolerance, the edge point counts as a vote; otherwise it counts as a non-vote. After all points are tested, the ratio of the number of points lying on the circle to the pixel circumference of the test circle is calculated. If above a certain threshold, a circle is detected and the points that voted are removed from further iterations of the script; the non-vote points are then used for the next iteration of the script until a minimum number of edge points remains. Once the detection criteria are met and the number of active points below the minimum point threshold, the script terminates after returning the centre and radius of each determined circle.

4.3 Ellipse Detection Scheme

The ellipse detection scheme builds upon the basic structure of the circle detection script. Until the RANSAC step, both scripts share the same procedure however, a significant drawback of ellipse detection is that the script can no longer rely on the rotational symmetry of a circle to calculate parameters, as a more general model is required for fitting. The setup of the RANSAC is similar to the circle detector script, with a sample of 4 randomly selected points used to find a test centre utilising the bisector method as in 4.2.

The general equation for an ellipse is

$$\alpha_0 x^2 + \alpha_1 xy + \alpha_2 y^2 + \alpha_3 x + \alpha_4 y = 1$$

The initial stages of the script calculate the ellipse test centre in the same way as with a circle. The set of edge points is then centred on the test centre by subtracting it from all edge points. This centring is equivalent to translation of the ellipse to the test centre, and as such does not affect the *physical* parameters such as rotation or axis size. Although the physical parameters remain the same, does introduce modified constants different to the general model for an ellipse, i.e $A_n \neq \alpha_n$. Hence, the model becomes that of an ellipse centred on the origin:[27]

$$A_0 x^2 + A_1 xy + A_2 y^2 = 1 \tag{4.1}$$

Using the first three of the four initial sample points, namely $p_1 = (a_x, a_y)$, $p_2 = (b_x, b_y)$ and $p_3 = (c_x, c_y)$, a matrix equation can be derived and solved to give the ellipse equation parameters:

$$\begin{aligned} A_0 a_x^2 + A_1 a_x a_y + A_2 a_y^2 &= 1 \\ A_0 b_x^2 + A_1 b_x b_y + A_2 b_y^2 &= 1 \implies A = X^{-1}B \\ A_0 c_x^2 + A_1 c_x c_y + A_2 c_y^2 &= 1 \end{aligned} \tag{4.2}$$

where

$$A = \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad X = \begin{bmatrix} a_x^2 & a_x a_y & a_y^2 \\ b_x^2 & b_x b_y & b_y^2 \\ c_x^2 & c_x c_y & c_y^2 \end{bmatrix}$$

The inverse of X is calculated using the *OpenCV* matrix inverse function. The script then

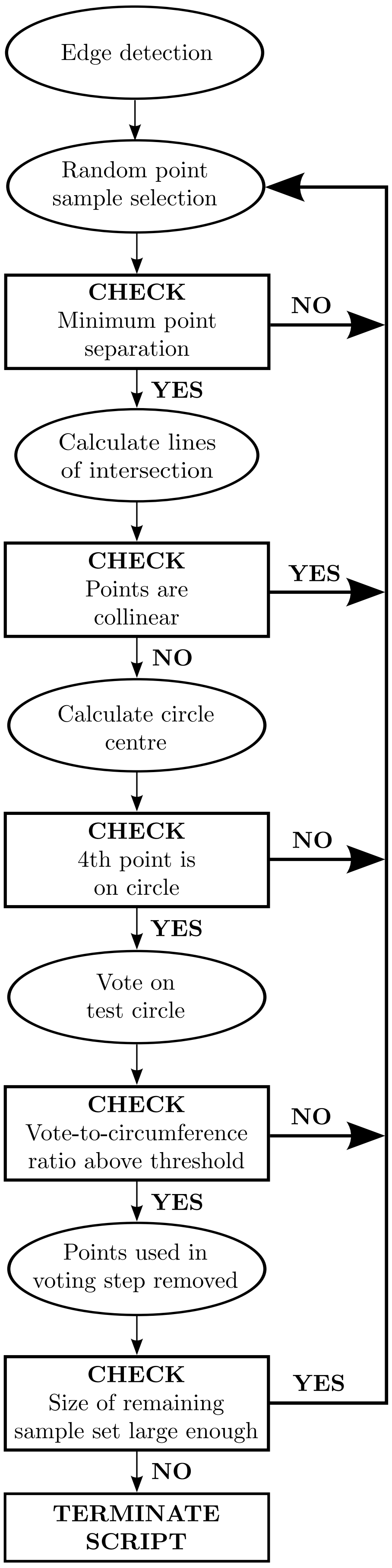

Edge detection
Random point sample selection
CHECK Minimum point separation
NO
YES
Calculate lines of intersection
CHECK Points are collinear
YES
NO
Calculate circle centre
CHECK 4th point is on circle
NO
YES
Vote on test circle
CHECK Vote-to-circumference ratio above threshold
NO
YES
Points used in voting step removed
CHECK Size of remaining sample set large enough
YES
NO
TERMINATE SCRIPT

Figure 4.3: Flowchart Schematic of RANSAC Circle Detection Script

checks to see if the fourth point lays on the ellipse; if not, the script returns to the sample selection step. If the point lies on the circle, parameters corresponding to the semi-major axis, the semi-minor axis and the rotation (α, β and θ respectively) are calculated from the general A matrix constants. The expressions relating the physical components and the A components are briefly derived below:

Begin with the equation for an ellipse centred on the origin with no rotation

$$\frac{x'^2}{\alpha^2} + \frac{y'^2}{\beta^2} = 1$$

Rotation of the ellipse can be considered as the following coordinate transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

with positive rotation θ in the anti-clockwise direction. So, the equation for an ellipse rotated θ radians about the origin is

$$\frac{\big(x\cos(\theta) - y\sin(\theta)\big)^2}{\alpha^2} + \frac{\big(x\sin(\theta) + y\cos(\theta)\big)^2}{\beta^2} = 1$$

Upon rearranging, this gives

$$\left(\tfrac{\cos^2(\theta)}{\alpha^2} + \tfrac{\sin^2(\theta)}{\beta^2}\right) x^2 + 2\cos(\theta)\sin(\theta)\left(\tfrac{1}{\beta^2} - \tfrac{1}{\alpha^2}\right) xy + \left(\tfrac{\cos^2(\theta)}{\beta^2} + \tfrac{\sin^2(\theta)}{\alpha^2}\right) y^2 = 1 \qquad (4.3)$$

Comparison of Expression (4.1) and Expression (4.3) yields

$$\begin{aligned} A_0 &= \left(\frac{\cos^2(\theta)}{\alpha^2} + \frac{\sin^2(\theta)}{\beta^2}\right) \\ A_1 &= 2\cos(\theta)\sin(\theta)\left(\frac{1}{\beta^2} - \frac{1}{\alpha^2}\right) \\ A_2 &= \left(\frac{\cos^2(\theta)}{\beta^2} + \frac{\sin^2(\theta)}{\alpha^2}\right) \end{aligned}$$

The system of equations was solved for α, β and θ giving

$$\alpha = \frac{1}{\sqrt{A_0 - \frac{A_1\sin(\theta)}{2\cos(\theta)}}} \qquad (4.4)$$

$$\beta = \frac{1}{\sqrt{A_0 - \frac{A_1\sin(\theta)}{2\cos(\theta)} + \frac{A_1}{\cos(\theta)\sin(\theta)}}} \qquad (4.5)$$

$$\theta = \tan^{-1}\left(\frac{A_0 - A_2 - \sqrt{(A_2 - A_0)^2 + A_1^2}}{A_1}\right) \qquad (4.6)$$

The voting procedure then substitutes each edge point into the left-hand side of the ellipse equation in expression (4.1), and calculates the difference from 1. This can be used as a measure of the deviation of each point from the ellipse model. If this difference is less than a tolerance value, the point counts as a vote; otherwise it counts as a non-vote.

Another difficulty arises with ellipse detection in comparison to circle detection; there is no exact explicit expression for the circumference of an ellipse[28]. An exact infinite series exists, however two popular approximations proposed by Ramanujan are popular due to a much faster computation time and small associated error. Ramanujan's second, more precise, approximation was used to calculate the circumference of the ellipses

$$C \approx \pi(a+b)\left(1 + \frac{3h}{10 + \sqrt{4-3h}}\right) \tag{4.7}$$

where

$$h = \frac{(\alpha-\beta)^2}{(\alpha+\beta)^2}$$

This approximation has error $\mathcal{O}(h^5)$ with Ramanujan's other approximation have error $\mathcal{O}(h^3)$.[29] In these experiments, only slight eccentricities in the circle radius are expected, that is $\alpha - \beta \ll 1$, so that the error in the approximation is *very* small.

The vote count-to-circumference ratio is calculated and compared to a set threshold; if the threshold is met, an ellipse is detected. The remaining set of non-vote points are used for sampling in order to detect more ellipses. The script terminates after a minimum number of active points are left.

The script outputs the parameters of each detected ellipse in both the console and a separate text file, and displays the detected ellipses in the input image, allowing for a quick visual confirmation of the fit.

4.4 Configuration of Parameters

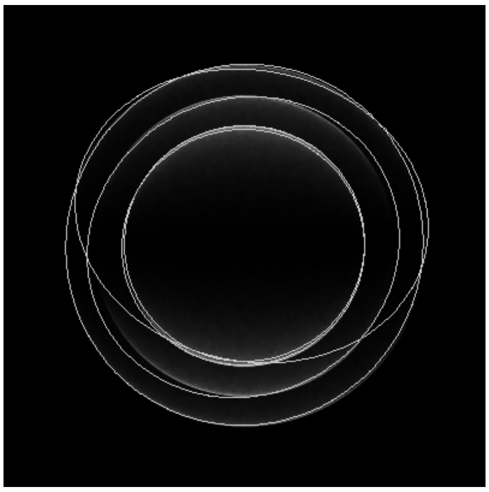
The ellipse detection script has a number of user-specified input variables, specifically the upper threshold on the Canny edge detector, the ellipse detection threshold (vote-to-circumference ratio) and the number of iterations respectively. Internally, the script also has a minimum point separation for the initial sample points, a collinearity tolerance, a tolerance for the fourth sample point to lay on the ellipse and a minimum points threshold. The internal parameters were largely untouched from the original circle detecting script with an initial point minimum separation of 10 pixels, a collinearity tolerance of 1 pixel, a tolerance on the fourth point of 0.03 pixels and a minimum of 10 active points.

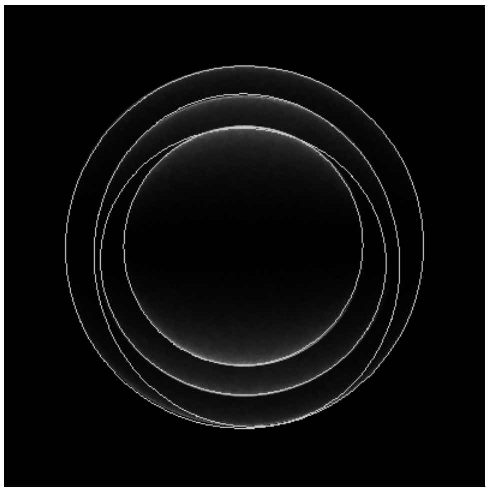
Two different computers were used for the experiment, namely *TesseracT* in the laboratory and *Albatross* at home. In terms of hardware, the two computers are separated by almost a decade of technological development, with *TesseracT* the younger computer. The most significant hardware difference between the computers with respect to computation power is the CPU, for *Albatross* utilises a *dual core* CPU versus *TesseracT* which utilises a hyperthreaded *quad core* processor. With such a large amount of time between the hardware, *Albatross* is ideal for determining how the script runs on an 'everyday' personal computer as opposed to the in-laboratory workstation, for the run-times on both computers will be drastically different. In addition to this, two computers with identical hardware may also give differing results due to differing software installations and other minor differences. To avoid confusion, tabulated results will specify which computer was used.

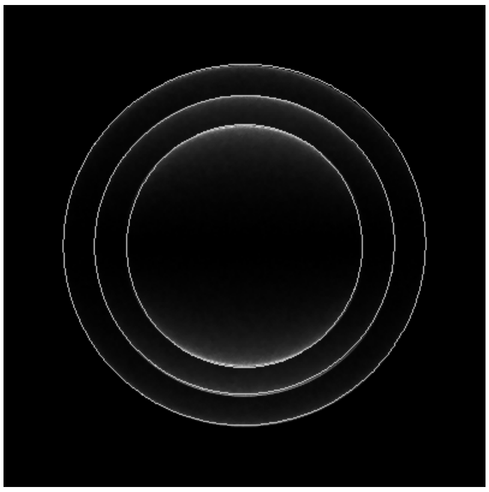
The ellipse detection script was tested over three ellipse detection thresholds, namely 0.80, 0.90 and 0.95, for both the original and the test image. Thresholds below 0.6 were deemed unreliable due to initial testing and often detected extra poorly fit, ellipses as demonstrated in Figures 4.5(a) and 4.5(b). For each ellipse detection threshold, the upper Canny threshold was set between 150 and 230, incrementing by 10. For each ellipse threshold and upper Canny threshold, the number of iterations was set to 10 and then incremented by a factor of 10 until the all three rings in the test image were detected. For each measurement, the number of detected ellipses was recorded and a visual judgement of the ellipse fit to the image made, as well as the script run-time and the detected image centre.

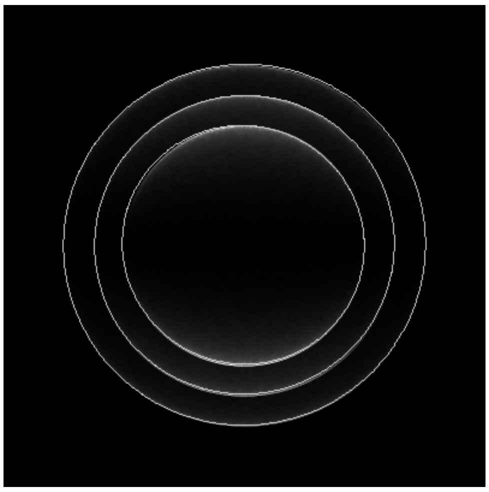
The original test image was first used to gather values, with Tables 4.2 to 4.4 giving the data collected for the three ellipse detection thresholds. It is apparent that the script fits the model visually well for most results, with greatest distance between the actual centre and detected centre found to be 1.05 pixels. An interesting outcome is that the run-time fluctuates with a changing upper Canny threshold, without clearly increasing or decreasing. As well as this, it is useful to note that a maximum iteration count of 10^4 succeeded in detecting all 3 ellipses for almost all measurements. In experimental applications with the VMI camera, this means that once a number of suitable iterations is found for a particular measurement the number will not need to be adjusted later in the experiment. For larger upper Canny Threshold values (> 220), only two ellipses were detected, even with a larger number of maximum iterations. This is likely due to the fact that a large upper Canny threshold results in a smaller amount of detected edge points, therefore making it less likely for the script to find a set of test points that not only lie on an ellipse, but also that the associated test ellipse has a sufficient number of votes to breach the ellipse detection threshold.

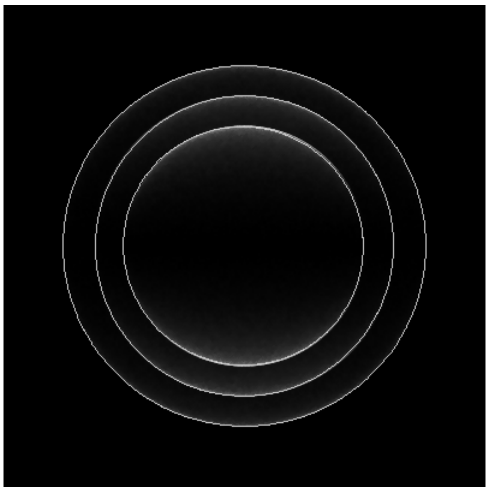
By focusing attention on a fixed upper Canny threshold for original image, the dependence of the model fit on the ellipse detection threshold can be both quantitatively and qualitatively determined. Table 4.1 shows that increasing the ellipse detection threshold results in a better fit, both visually and in terms of the detected centre, but at the cost of a higher run-time. The increase in run-time is expected, for as the ellipse detection threshold increases, the probability of the script randomly choosing three points that mutually lie on an ellipse in the image decreases. Further to this, Figure 4.4 clearly demonstrates that as the ellipse detection threshold increases, the visual quality of the fit improves.


(a) Ellipse Threshold = 0.5


(b) Ellipse Threshold = 0.6


(c) Ellipse Threshold = 0.7


(d) Ellipse Threshold = 0.8


(e) Ellipse Threshold = 0.9

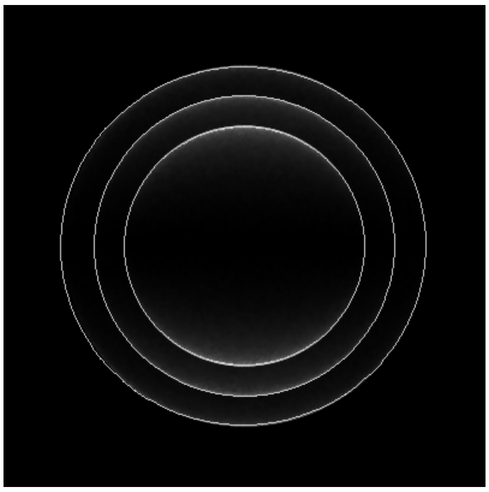

(f) Ellipse Threshold = 0.95

Figure 4.4: Original image fits with $C_{\max} = 190$, 10^4 iterations (*TesseracT*)

Table 4.1: Values for original test image with $C_{\max} = 190$ (*TesseracT*)

Ellipse Threshold	Ellipses	run-time (s)	Centre Coordinates	Visual Fit
0.5	5	0·52	201.8, 195.8	Poor
0.6	4	0·6	201, 207.5	Poor
0.7	3	0·62	201, $200.\overline{3}$	Good
0.8	3	0·66	$201.\overline{3}$, $200.\overline{3}$	Average
0.9	3	0·77	$200.\overline{6}$, 201	Very good
0.95	3	1·13	$200.\overline{6}$, 201	Very good
0.99	3	1·25	201, 201	Very good

Table 4.2: Values for original test image with ellipse threshold = 0.80 (*Albatross*)

$C_{\max}$	$i_{\max}$	Ellipses	run-time (s)	Centre Coordinates	Visual Fit
150	10^2	3	0·0715	$201.\overline{3}$, $200.\overline{3}$	Good
150	10^3	3	0·231264	$201.\overline{3}$, $200.\overline{3}$	Good
150	10^4	3	1·81497	$201.\overline{3}$, $200.\overline{3}$	Good
160	10^3	3	0·375548	$200.\overline{6}$, 201	Very good
170	10^3	3	0·345236	$200.\overline{6}$, 201	Good
180	10^2	3	0·1056	$200.\overline{6}$, 201	Very good
190	10^2	2	0·090889	201, 200	Very good
190	10^3	3	0·237445	$200.\overline{6}$, 200	Good
200	10^2	1	0·105034	201, 201	Very good
200	10^3	3	0·305034	$200.\overline{6}$, $200.\overline{6}$	Very good
210	10^3	3	0·506631	201, $200.\overline{6}$	Good
220	10^3	3	0·349171	$200.\overline{6}$, 201	Good

Table 4.3: Values for original test image with ellipse threshold = 0.90 (*Albatross*)

$C_{\max}$	$i_{\max}$	Ellipses	run-time (s)	Centre Coordinates	Visual Fit
150	10^2	3	0·071272	201, $200.\overline{6}$	Very good
160	10^3	2	0·793007	201, 200.5	Very good
160	10^4	3	2·30477	201, $200.\overline{6}$	Very good
170	10^3	3	0·501911	201, $200.\overline{6}$	Very good
180	10^3	3	0·409688	$200.\overline{6}$, 201	Very good
190	10^3	1	0·75723	201, 201	Very good
190	10^4	3	1·90887	$200.\overline{6}$, 201	Very good
200	10^3	3	0·521413	201, $200.\overline{6}$	Very good
210	10^4	3	2·76855	$200.\overline{3}$, 201	Very good
220	10^4	3	3·40758	201, 201	Very good
230	10^4	2	4·38534	200.5, 201	Very good
230	10^5	2	30·562	200.5, 201	Very good
230	10^6	2	292·724	200.5, 201	Very good

Table 4.4: Values for original test image with ellipse threshold = 0.95 (*Albatross*)

$C_{\max}$	$i_{\max}$	Ellipses	run-time (s)	Centre Coordinates	Visual Fit
150	10^3	3	0·611174	$200.\overline{6}$, 201	Very good
160	10^3	1	0·865904	201, 201	Very good
160	10^4	3	2·58226	201, $200.\overline{6}$	Very good
170	10^3	1	0·714246	202, 201	Very good
170	10^4	3	1·70509	$201.\overline{3}$, $200.\overline{6}$	Very good
180	10^3	2	0·445061	200.5, 201	Very good
180	10^4	3	1·46772	$200.\overline{6}$, 201	Very good
190	10^4	3	3·16462	$200.\overline{6}$, 201	Very good
200	10^4	3	2·63942	$200.\overline{6}$, 201	Very good
210	10^4	3	2·65606	202, 201	Very good
220	10^4	2	5·35316	201, 201	Very good
220	10^5	2	30·2266	201, 201	Very good
230	10^5	2	31·2485	200.5, 201	Very good

With a general idea for parameter value ranges, the distorted image received similar treatment to the original image. With the upper Canny threshold fixed to 190, the ellipse detection threshold was varied over the same values as the circle detecting script. From Table 4.5, it becomes apparent that the model fits improve until the ellipse detection threshold is above 0.9. Figure 4.5 also demonstrates this with poor fits for low ellipse detection thresholds. Above this value, it can be seen that a larger maximum number of iterations is required as few or no ellipses are detected. The run-time values plateau, meaning that the script has run through the total number of iterations without discovering any ellipses. As such, the run-time doesn't increase for each 'plateau' value because the same number of iterations was used. Also noteworthy is that for most values of the upper Canny threshold, the runtime increases with increasing

As shown in Tables 4.6 to 4.8, for the distorted test image the script is unreliable for lower ellipse detection thresholds, evident both from the visual fits as well as the variation in detected centre coordinate. With increasing upper Canny thresholds, the fits improve both visually and in terms of stability of the centre coordinates; however, it is clear from Table 4.8 that there is a trade off between the quality of the model fit and the runtime, as for upper Canny Thresholds larger than 210 not all ellipses are detected. Table 4.8, the final ellipse detection threshold in particular, shows a significant increase in computation time following an increase in iterations but the same number of ellipses are detected.

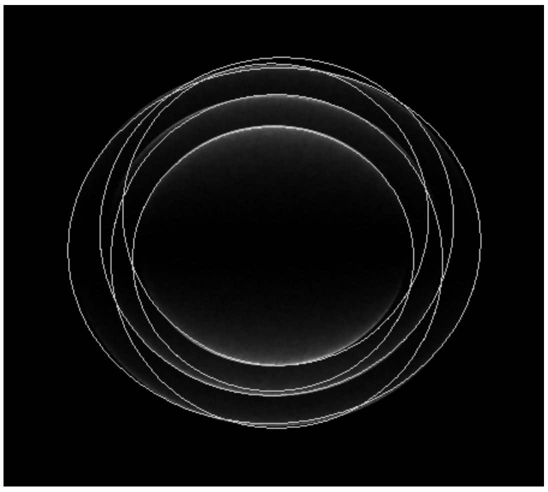
Looking at the results for both test images together, a qualitative deduction at how the eccentricity of the resultant rings effects the run-time of the ellipse detection script can be made. First, comparison of Tables 4.1 and 4.5 shows that run-times are off the same order, with a slight increase in the run-time for the distorted test image. The detected centres were much worse in the distorted image, with the closest detected centre 1 pixel from the actual centre versus the original image that detected the actual centre, which can also be verified by the visual fits and an inspection of Figures 4.4 and 4.5. As such, the script has some dependence upon the eccentricity. Rather than a direct dependence, it is likely that this is because the distorted test image was made by stretching the original test image without conserving the circumference. Hence, the distorted picture will actually have a larger number of edge points, decreasing the probability of choosing 3 random points that

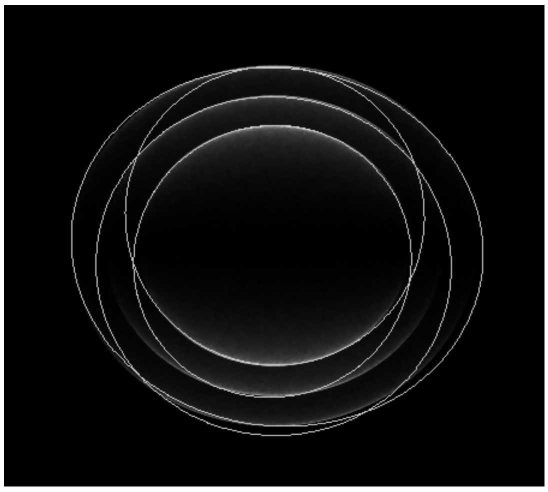
Table 4.5: Values for distorted test image with $C_{\max} = 190$ (*TesseracT*)

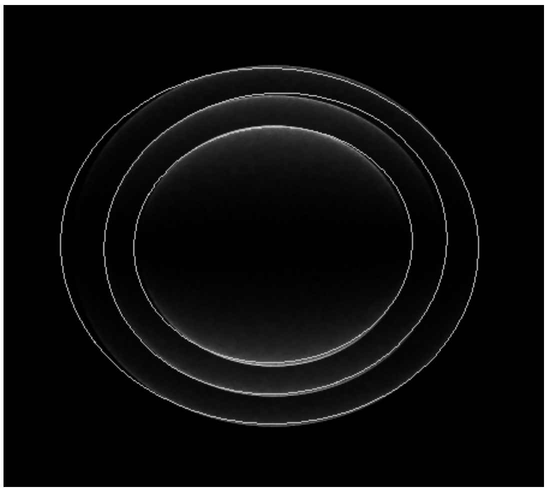
Ellipse Threshold	Ellipses	run-time (s)	Centre Coordinates	Visual Fit
0.5	5	0·82	225.8, 196.4	Poor
0.6	4	0·93	224.75, 201.5	Poor
0.7	3	1·97	$223.\overline{6}, 199$	Poor
0.8	1	2·79	227, 200	Good
0.9	0	3·33	-	-
0.95	0	3·37	-	-
0.99	0	3·31	-	-

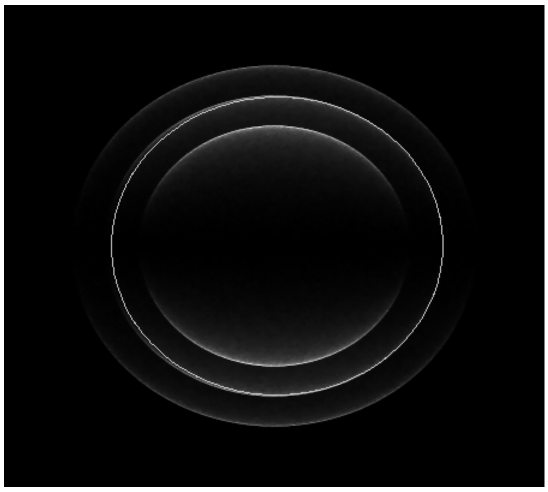
lie on an ellipse in the image, which increases the run-time. As more iterations would be required for satisfying the criteria, this explains why less ellipses are detected in the distorted image for higher upper Canny thresholds, for a greater number of iterations is required to detect the same number of ellipses. In fact, the eccentricity seems to have such an effect that in rough terms, the run-time for detection in the distorted image is about a factor of 10 greater than that for the original image.

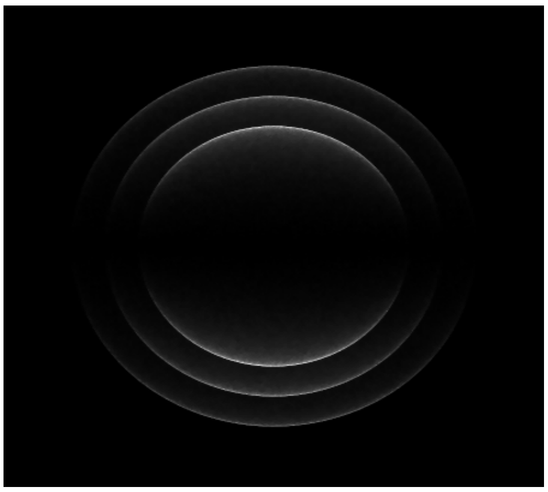
For both the original and distorted image, upper Canny thresholds above 180 seem to give the most stable centre results with minimum fluctuation. Similarly, ellipse detection thresholds above 0.8 give the best fits, both visually and in terms of the detected centre, as readily apparent on comparison between Figure 4.4 and 4.5 as well as Tables 4.1 and 4.5. The run-time for all calculations is roughly R/T $\approx \mathcal{O}(\text{Iterations}^{-4})$. For example, the run-time for 10^5 iterations is on the order of 10 seconds.


(a) Ellipse Threshold = 0.5


(b) Ellipse Threshold = 0.6


(c) Ellipse Threshold = 0.7


(d) Ellipse Threshold = 0.8


(e) Ellipse Threshold = 0.9

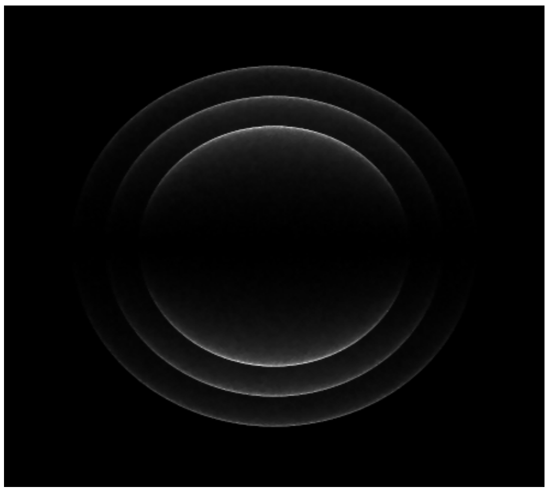

(f) Ellipse Threshold = 0.95

Figure 4.5: Distorted image fits with $C_{\text{max}} = 190$, 10^4 iterations (*TesseracT*)

Table 4.6: Values for distorted test image with ellipse threshold = 0.80 (*Albatross*)

$C_{\max}$	$i_{\max}$	Ellipses	run-time (s)	Centre Coordinates	Visual Fit
150	10^4	1	9·5792	224,202	Average
150	10^5	3	38·387	224, 201	Good
160	10^4	3	4·27671	226, 201	Average
170	10^4	3	4·69528	$226.\overline{3}$, 200.3	Average
180	10^4	2	6·57615	224.5, 202	Poor
180	10^5	3	30·4459	$225, 201.\overline{6}$	Average
190	10^4	1	7·53457	228, 201	Good
190	10^5	3	36·0599	$225.\overline{6}$, $201.\overline{3}$	Bad
200	10^4	3	4·85489	$225.\overline{3}$, 201	Good
210	10^5	3	49·0335	$225.\overline{6}$, 201	Good
220	10^4	2	5·48194	226, 201	Good
220	10^5	3	21·2245	$225.\overline{3}$, $200.\overline{6}$	Average
230	10^5	3	40·2546	$225.\overline{6}$, $200.\overline{6}$	Average

Table 4.7: Values for distorted test image with ellipse threshold = 0.90 (*Albatross*)

$C_{\max}$	$i_{\max}$	Ellipses	run-time (s)	Centre Coordinates	Visual Fit
150	10^5	3	46·4376	$225.\overline{3}$, 201	Poor
160	10^5	3	53·7546	$226.\overline{3}$, $200.\overline{6}$	Good
170	10^5	3	75·5592	$225.\overline{3}$, $200.\overline{3}$	Good
180	10^5	3	35·853	226, 201	Good
190	10^5	3	61·5295	$226.\overline{3}$, 201	Good
200	10^4	1	6·3723	225, 201	Good
200	10^5	3	24·6989	225, $200.\overline{6}$	Good
210	10^5	2	68·1097	225, 201	Good
210	10^6	3	171·458	225, 201	Good
220	10^5	3	60·8674	$225.\overline{6}$, 201	Good
230	10^6	3	438·011	226, 201	Good

Table 4.8: Values for distorted test image with ellipse threshold = 0.95 (*Albatross*)

$C_{\max}$	$i_{\max}$	Ellipses	run-time (s)	Centre Coordinates	Visual Fit
150	10^5	3	50·9535	$225.\overline{6}$, 201	Average
160	10^5	3	51·8082	$225.\overline{3}$, 201	Good
170	10^5	1	87·6729	225, 201	Very good
170	10^6	3	228·404	$225.\overline{6}$, 201	Very good
180	10^5	1	61·8048	225, 201	Very good
180	10^6	3	171·816	$225.\overline{6}$, 201	Very good
190	10^5	1	74·4372	227, 201	Very good
190	10^6	3	231·31	$225.\overline{6}$, 201	Very good
200	10^4	1	6·47164	225, 201	Very good
200	10^5	3	33·4996	$225.\overline{3}$, 201	Very good
210	10^6	3	248·115	$225.\overline{3}$, 201	Very good
220	10^5	2	71·1765	225.5, 201	Very good
220	10^6	2	385·115	225.5, 201	Very good

4.5 Self-Centering Inverse Abel Transform Script

The ellipse detection script developed in Section 4.3 was combined with the IAT script from Chapter 3. The self-centering IAT (SC-IAT) was developed in such a way that the input parameters from the original ellipse detecting script are present. In future developments, these input parameters will be initialised in a separate script, leaving only the file location up to the user - if the script and images are stored statically (i.e. in the same place all the time) then the script can be setup so that no user input is required, further streamlining the data extraction process. The SC-IAT was tested based on parameter values obtained from Section 4.4. The number of iterations was chosen to be 10^5, for based on the parameter analysis it is unlikely that there will be any undetected ellipses for such a relatively large number of iterations. Self-centred transforms were computed for two ellipse detection thresholds (0.8 and 0.9) with three different upper Canny thresholds (190, 200, 210) each. To determine the quality of the fit, radial profiles were calculated for all images.

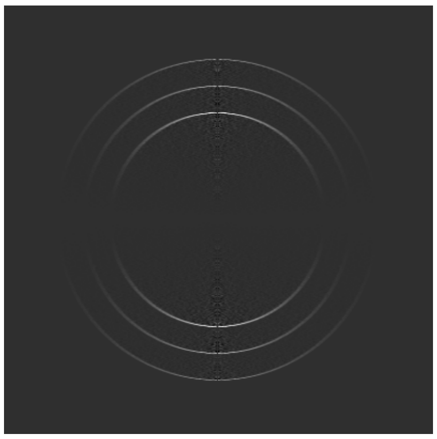
Upon examination of Figures 4.6 and 4.7, it is first apparent that there is no splitting in the profiles and that the full-width-at-half-maximum (FWHM) is very small, giving a visual indication of a good fit for all images. All detected centres were within 0.471 pixels of the actual centre, which corresponds to an estimated centre detection uncertainty of 0.235%.

Analysis of the distorted image transforms was not performed, for while the test image is a good means of testing the ellipse detecting ability of the script, it is not representative of the experimental images, even if misaligned. Also, it was not possible to perform a radial spectrum due to the elliptical nature of images. The SC-IAT script is many pages long across many separate documents; for this reason, the script has not been included with this dissertation but can be transmitted electronically on request.

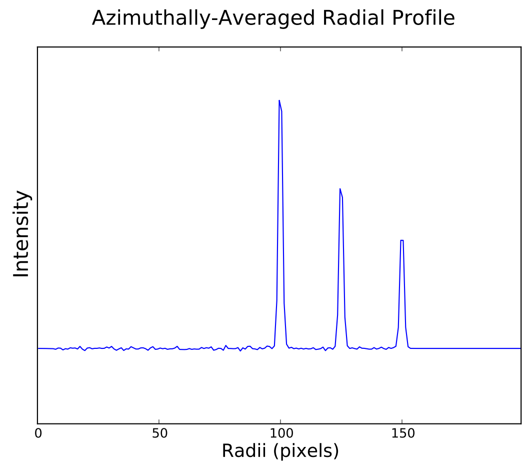
4.6 Further Work on Centre Detection and Image Reconstruction

As it stands, the ellipse detecting portion of the SCIAT is useable and producing qualitatively good results, with some quantitative justification. That said, the ellipse detection is not sufficiently reliable enough for more precise applications than current testing, for as demonstrated the script rarely finds the exact centre but often finds a centre close by. Further testing aims to discover if this variation is due simplfy to noise in the image or due to inherrant instability in the randomised selection of sample points. Future work with the ellipse detection scheme will involve adapting the script to detect circles over a range of upper Canny thresholds with a set ellipse detection threshold, and then averaging all centres. It is posited that if the noise is due to random variations in the picking procedure, then the averaging of sufficient ellipse centres will overcome this.

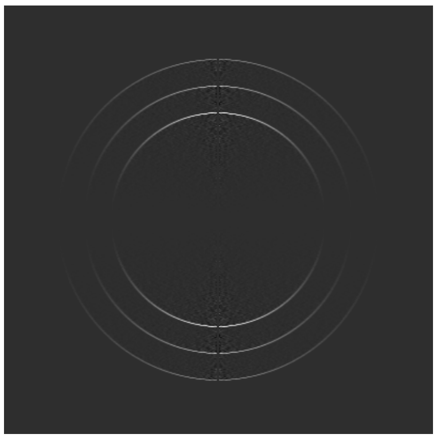
As previously discussed, both the SCIAT and original IAT implementation have sig-



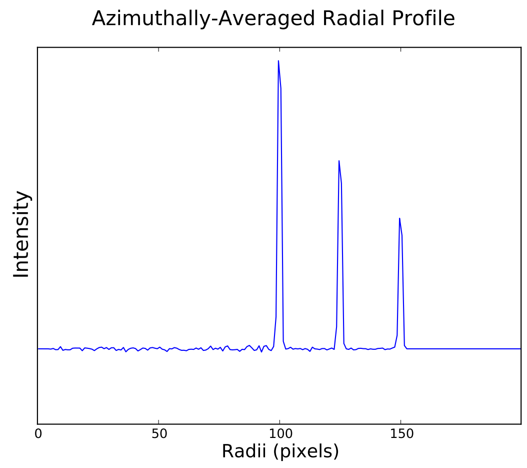
(a) $C_{\max} = 190$, centre (201, 200.$\overline{6}$)



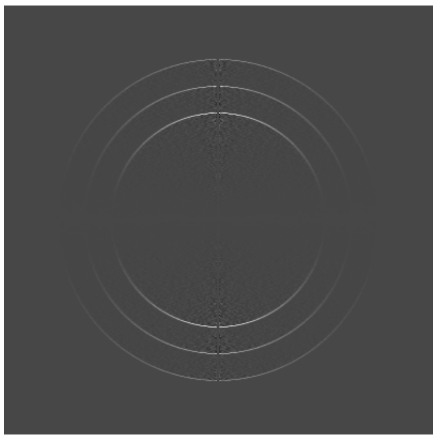
(b) Radial profile of (a)



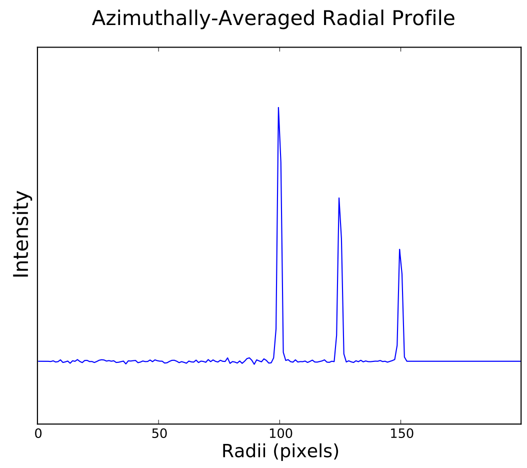
(c) $C_{\max} = 200$, centre (200.$\overline{6}$, 201)



(d) Radial profile of (c)

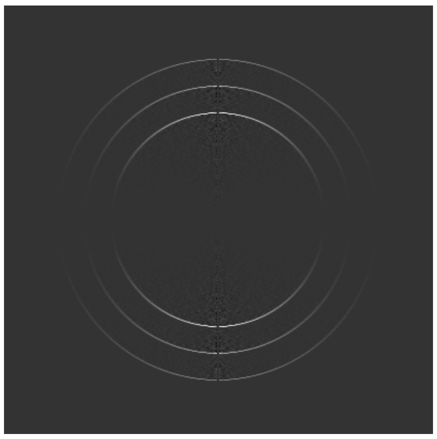


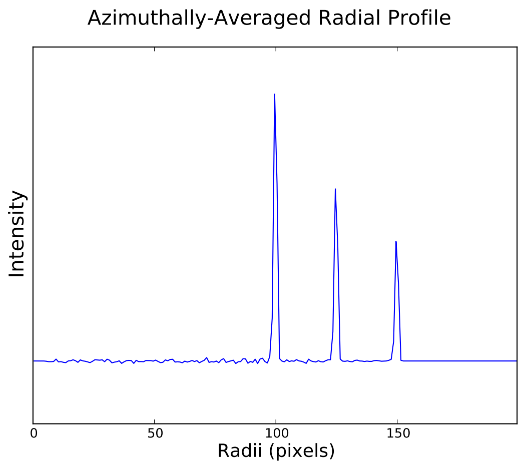
(e) $C_{\max} = 210$, centre (200.$\overline{6}$, 201.$\overline{3}$)



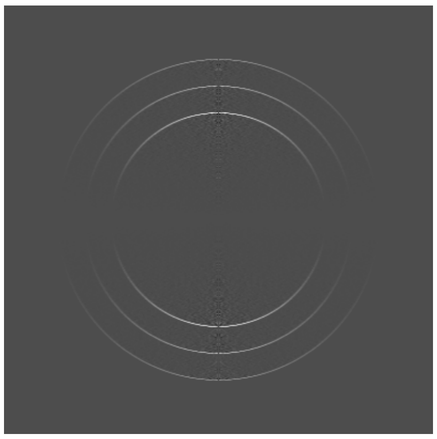
(f) Radial profile of (e)

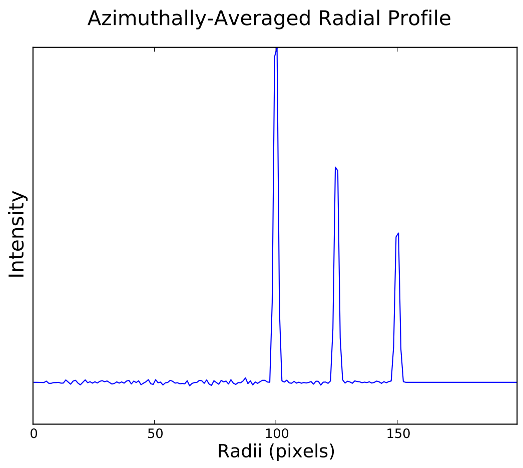
Figure 4.6: Results for ellipse detection threshold = 0.8, 10^5 iterations (*Albatross*)


(a) $C_{\max} = 190$, centre $(200.\overline{3}, 201)$

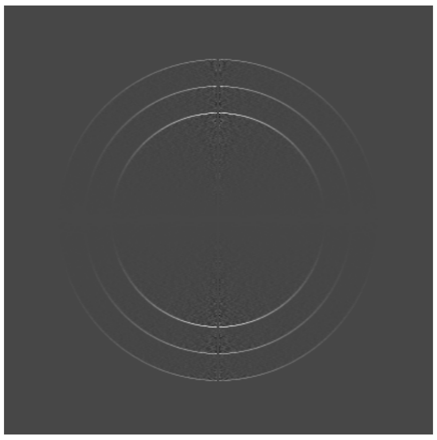


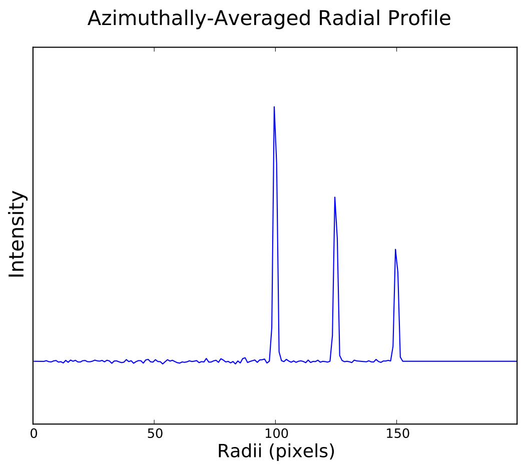
(b) Radial profile of (a)


(c) $C_{\max} = 200$, centre $(201, 200.\overline{3})$



(d) Radial profile of (c)


(e) $C_{\max} = 210$, centre $(200.\overline{6}, 200.\overline{6})$



(f) Radial profile of (e)

Figure 4.7: Results for ellipse detection threshold = 0.9, 10^5 iterations (*Albatross*)

nificant issues that need to be overcome. Of these, the major issues are associated with the IAT itself, namely the reduction of signal-to-noise ratio close to the centreline, the extreme centre dependence and the singularity at $\rho = 0$. Further to these issues, there is currently no way of quantifying the quality of the reconstruction. Future work will focus on techniques to assess the quality of the reconstructed images, as well as a way to include an uncertainty in the transformation, so that extracted information is rigorously justified and more statistically viable.

Aside from the IAT, other methods exist for finding the initial velocity distribution that improve upon issues presented with the recursive implementation as presented in Chapter 3. A prominent method in VMI applications is the Fourier-Hankel method as developed by Smith[30] which reformulates the inverse Abel transform $\mathcal{A}^{-1}$ as the inverse Hankel transform $\mathcal{H}^{-1}$ of the Fourier transform $\mathcal{F}$ of the projected function:

$$\begin{aligned}\mathcal{A}^{-1}[F(x,z)] &= \mathcal{H}^{-1}\mathcal{F}[F(x,z)] \\ &= 2\pi \int_0^\infty qJ_0(2\pi\rho q) \int_{-\infty}^\infty F(x,z)\mathrm{e}^{-2\mathrm{i}\pi xq}\,\mathrm{d}x\,\mathrm{d}q\end{aligned}$$

where $J_0(\cdot)$ is the zero-order Bessel function of the first kind. The benefit of this method is that it avoids the singularity at of the lower limit of the integral, but it introduces issues associated with symmetry about the x-axis and the resultant non-physical imaginary component calculated. As well as this, all transformation techniques have the same drawback of centreline noise amplification like the IAT.

Beyond the existing scripts, a recent technique has been developed for reconstructing images called maximum entropy velocity image reconstruction (MEVIR)[31]. MEVIR inverts the image without smoothing the data or invoking the IAT, avoiding the centreline noise associated with it as well as the necessity of knowing the image centre. As MEVIR is probability based, the approach finds the most likely 3D distribution to produce the 2D experimental image. Furthermore, the technique uses all information contained with the image and only this information. A comparison of the images in terms of noise, computation time as well as an calculating the difference of the images may well prove interesting in guiding work beyond this.

Chapter 5

Conclusions

The operation of the new VMI camera depends entirely upon the ability to collect useable data from experimental images. In this thesis, a range of topics were covered, each crucial to the understanding of the following chapter. In Chapter 1, the basic theory behind anion photoelectron spectroscopy was introduced, as well as the basic operation of the TOF-PES apparatus. The theory and basic operation behind the VMI camera being configured was then introduced, as well as the experimental image projection issue associated with the geometry of direct ion imaging technqiues.

Chapter 2 covered the mathematical principles and properties linear systems. In particular, a special class of linear system known as a linear time-invariant (LTI) system was introduced as well as some simple properties. Following this, the state-space representation of linear systems was presented that allows the decomposition of an n^{th} order differential equation into an n-vector array of 1^{st} order equations. Two examples of conversion to state-space notation were presented, followed by the general form for an LTI system in state-space notation.

The Abel transform, and its more commonly used sibling the inverse Abel transform (IAT), were introduced in Chapter 3, along with the associated geometrical representation. A coordinate transform is applied to the IAT, yielding a modified IAT that is in the form of a convolution. In Chapter 2, it was shown that all LTI systems can be written in the form of a convolution; conversely, the modified IAT can then be considered as an LTI system and as such be written as a state-space system. After discretisation of the modified IAT state-space equations and the inverse coordinate transform applied, the system is in a discrete matrix form. The final expression for the IAT presented is a recursive matrix equation. An IAT C++ script was provided for use by the Gascooke group, Flinders University. The IAT script suffers increased noise towards the centreline, which occurs because only the centreline data in the projection contains information about the centreline of the original distribution.

In Chapter 4, an overview is given for a RANSAC circle detection script, as implemented open-source in C++ by Kevin Hughes[22]. From this, a RANSAC ellipse detection script was written utilising the geometric properties of an ellipse in place of the simple cyclindrical symmetry used in the circle detecting script. Analysis found that the script was reliable at determining centres for small eccentricities, with ellipse detection thresholds greater than 0.8 and upper Canny thesholds greater than 180 found to be the most stable initial values. The RANSAC ellipse detection script was combined with the IAT script to produce a self-centring IAT (SC-IAT) script capable of self-detecting the centre of the distribution in the image, eliminating possible source of uncertainty due to user-input. The script was found to be very reliable for the original test image,

for different initial parameter values was undertaken, with The script was found to be reliable with an upper Canny threshold of 180 to 210, and an ellipse detection threshold greater than 0.80. The number of iterations will need to be adjusted depending on the situation, for example higher noise images versus lower noise images.

References

(1) Neumark, D. M. *Physical Chemistry Chemical Physics* **2005**, *7*, 433.

(2) Lapere, K. An Investigation of Gas-phase Clusters of Atmospheric Interest Using Anion Photoelectron Spectroscopy and Ab Initio Techniques. Honours Dissertation, University of Western Australia, 2009.

(3) Lapere, K.; LaMacchia, R.; Quak, L.; McKinley, A.; Wild, D. *Chemical Physics Letters* **Feb. 2011**, *504*, 13–19.

(4) Wild, D.; Lenzer, T *Physical Chemistry Chemical Physics* **2005**, *7*, 3793–3804.

(5) Lapere, K. M.; LaMacchia, R. J.; Quak, L. H.; Kettner, M.; Dale, S. G.; McKinley, A. J.; Wild, D. A. *The Journal of Physical Chemistry. A* **Apr. 2012**, *116*, 3577–84.

(6) Atkins, P.; de Paula, J., *Physical Chemistry*, 8th ed.; Oxford University Press: New York, 2006; Chapter 10–14, p 1087.

(7) Simons, J. *The Journal of Physical Chemistry A* **July 2008**, *112*, 6401–511.

(8) LaMacchia, R. Towards Anion Photoelectron Spectroscopy of Complexes and Clusters. Honours Dissertation, University of Western Australia, 2008.

(9) Quak, L.H. Investigation of Gas-Phase Complexes and Clusters with Anion Photoelectron Spectroscopy. Honours Dissertation, University of Western Australia, 2009.

(10) Wiley, W. C.; McLaren, I. H. *Review of Scientific Instruments* **1955**, *26*, 1150.

(11) Cavanagh, S. S.; Gibson, S. S.; Gale, M.; Dedman, C.; Roberts, E.; Lewis, B. *Physical Review A* **Nov. 2007**, *76*, 052708.

(12) Eppink, A. T. J. B.; Parker, D. H. *Review of Scientific Instruments* **1997**, *68*, 3477.

(13) Ladislas Wiza, J. *Nuclear Instruments and Methods* **June 1979**, *162*, 587–601.

(14) J. Garlick, G *Proceedings of the IRE* **1955**, *43*, 1907–1911.

(15) Osterwalder, A.; Nee, M. J.; Zhou, J.; Neumark, D. M. *The Journal of Chemical Physics* **Oct. 2004**, *121*, 6317–22.

(16) Neumark, D. M. *The Journal of Physical Chemistry. A* **Dec. 2008**, *112*, 13287–301.

(17) Reid, K. L. *Annual Review of Physical Chemistry* **Jan. 2003**, *54*, 397–424.

(18) Hansen, E. W.; Law, P.-L. *Journal of the Optical Society of America A* **Apr. 1985**, *2*, 510.

(19) Antsaklis, P. J.; Michel, A. N., *A Linear Systems Primer*; Birkhäuser Boston: Boston, MA, 2007.

(20) Mattheij, R.; Molenaar, J., *Ordinary Differential Equations in Theory and Practice*; Society for Industrial and Applied Mathematics: Jan. 2002.

(21) Fischler, M. A.; Bolles, R. C. *Communications of the ACM* **June 1981**, *24*, 381–395.

(22) Hughes, K. *circleDetector.cpp*[Online], version 1.0; https://github.com/pickle27/circleDetector (accessed March 2014).

(23) Bradski, G. *The OpenCV Library*[Online], version 11.4; http://www.opencv.org (accessed March 2014).

(24) Canny, J. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **Nov. 1986**, *PAMI-8*, 679–698.

(25) Dhankhar, P.; Sahu, N. *International Journal of Computer Science and Mobile Computing* **2013**, *2*, 86–92.

(26) Chen, T.-C.; Chung, K.-L. *Computer Vision and Image Understanding* **Aug. 2001**, *83*, 172–191.

(27) Song, G.; Wang, H., *Computer Analysis of Images and Patterns*; Kropatsch, W. G., Kampel, M., Hanbury, A., Eds.; Lecture Notes in Computer Science, Vol. 4673; Springer Berlin Heidelberg: Berlin, Heidelberg, 2007, pp 669–676.

(28) Littlewoord, J. E. *Nature* **Apr. 1929**, *123*, 631–633.

(29) Villarino, M. B. **2006**, *7*.

(30) Montgomery Smith, L.; Keefer, D. R.; Sudharsanan, S. *Journal of Quantitative Spectroscopy and Radiative Transfer* **May 1988**, *39*, 367–373.

(31) Dick, B. *Physical Chemistry Chemical Physics : PCCP* **Jan. 2014**, *16*, 570–80.

Appendix A

RANSAC Circle Detection Script

The following script is an implementation of work by Chen and Chung[26], was developed in C++ and built upon an existing open-source RANSAC circle detection scheme. The script depends heavily on the open-source computer vision library *OpenCV*[23] for manipulating images and displaying results.

```
// circleDetector.cpp
//
// Kevin Hughes
//
// 2012
//
// This is an implementation of the circle detction RANSAC
  algorithm
// described in "An efficient randomized algorithm for
  detecting circles"
// by Chen, T.C. and Chung, K.L.
//

#include "/usr/include/opencv2/highgui/highgui.hpp"
#include "/usr/include/opencv2/imgproc/imgproc.hpp"
#include "/usr/include/opencv2/opencv.hpp"

#include <iostream>
#include <vector>
#include <string>

#include <time.h>

using namespace cv;
using namespace std;

// circleRANSAC
//
// input:
//      image - either CV_8UC1 or CV_8UC3
//      circles - return vector of Vec3f (x,y,radius)
//      canny_threshold - higher canny threshold, lower is set to canny_threshold / 2
//      circle_threshold - value between 0 and 1 for the percentage of the circle that needs to vote for it to be accepted
//      numIterations - the number of RANSAC loops, the function will quit if there is no points left in the set
//
void circleRANSAC(Mat &image, vector<Vec3f> &circles, double canny_threshold, double circle_threshold, int numIterations);

int main(int argc, char *argv[])
{
    if(argc != 5)
    {
        cout << "Usage: " << argv[0] << "<image file> <canny threshold> <circle threshold> <iterations>" << endl;
        return -1;
    }

    // collect arguemtns
    string filename = argv[1];
    double canny_threshold = atof(argv[2]);
    double circle_threshold = atof(argv[3]);
    int iterations = atoi(argv[4]);

    Mat image = imread(filename,0);
    vector<Vec3f> circles;

    const clock_t start = clock();
    circleRANSAC(image, circles, canny_threshold,
        circle_threshold, iterations);
    clock_t end = clock();

    cout << "Found " << (int)circles.size() << " Circles."
        << endl;

    double time = ((double)(end - start)) / (double)
        CLOCKS_PER_SEC;
    std::cout << "RANSAC runtime: " << time << " seconds" <<
        std::endl;

    // Draw Circles
    cvtColor(image,image,CV_GRAY2RGB);
    for(int i = 0; i < (int)circles.size(); i++)
    {
        int x = circles[i][0];
        int y = circles[i][1];
        float rad = circles[i][2];

        circle(image, Point(x,y), rad, Scalar(0,255,0));
    }

    imshow("circles", image);
    for (size_t i = 0; i < circles.size(); i++ )
      {
          std::cout << circles[i][0] << ", " << circles[i
            ][1] << ", " << circles[i][2] << std::endl;
      }
    waitKey();

    return 0;
}

void circleRANSAC(Mat &image, std::vector<Vec3f> &circles,
    double canny_threshold, double circle_threshold, int
    numIterations)
{
    CV_Assert(image.type() == CV_8UC1 || image.type() ==
        CV_8UC3);
    circles.clear();

    // Edge Detection
    Mat edges;
    Canny(image, edges, MAX(canny_threshold/2,1),
     canny_threshold, 3);

    // Create point set from Canny Output
    std::vector<Point2d> points;
    for(int r = 0; r < edges.rows; r++)
    {
        for(int c = 0; c < edges.cols; c++)
        {
            if(edges.at<unsigned char>(r,c) == 255)
            {
                points.push_back(cv::Point2d(c,r));
            }
        }
    }

    // 4 point objects to hold the random samples
    Point2d pointA;
    Point2d pointB;
    Point2d pointC;
    Point2d pointD;

    // distances between points
    double AB;
    double BC;
    double CA;
    double DC;

    // varibales for line equations y = mx + b
    double m_AB;
    double b_AB;
    double m_BC;
    double b_BC;

    // varibles for line midpoints
    double XmidPoint_AB;
    double YmidPoint_AB;
    double XmidPoint_BC;
    double YmidPoint_BC;

    // variables for perpendicular bisectors
    double m2_AB;
    double m2_BC;
    double b2_AB;
    double b2_BC;

    // RANSAC
    cv::RNG rng;
    int min_point_separation = 10; // change to be relative
      to image size?
    int colinear_tolerance = 1; // make sure points are not
      on a line
    int radius_tolerance = 3; // change to be relative to
      image size?
    int points_threshold = 10; //should always be greater
      than 4
    //double min_circle_separation = 10; //reject a circle
      if it is too close to a previously found circle
    //double min_radius = 10.0; //minimum radius for a
      circle to not be rejected

    int x,y;
    Point2d center;
    double radius;

    // Iterate
    for(int iteration = 0; iteration < numIterations;
      iteration++)
    {
        //std::cout << "RANSAC iteration: " << iteration <<
          std::endl;

        // get 4 random points
        pointA = points[rng.uniform((int)0, (int)points.size
          ())];
        pointB = points[rng.uniform((int)0, (int)points.size
          ())];
        pointC = points[rng.uniform((int)0, (int)points.size
         ())];
        pointD = points[rng.uniform((int)0, (int)points.size
         ())];

        // calc lines
        AB = norm(pointA - pointB);
        BC = norm(pointB - pointC);
        CA = norm(pointC - pointA);
        DC = norm(pointD - pointC);

        // one or more random points are too close together
        if(AB < min_point_separation || BC <
         min_point_separation || CA < min_point_separation
         || DC < min_point_separation) continue;

        //find line equations for AB and BC
        //AB
        m_AB = (pointB.y - pointA.y) / (pointB.x - pointA.x
         + 0.000000001); //avoid divide by 0
        b_AB = pointB.y - m_AB*pointB.x;

        //BC
        m_BC = (pointC.y - pointB.y) / (pointC.x - pointB.x
         + 0.000000001); //avoid divide by 0
        b_BC = pointC.y - m_BC*pointC.x;


        //test colinearity (ie the points are not all on the
          same line)
        if(abs(pointC.y - (m_AB*pointC.x + b_AB +
         colinear_tolerance)) < colinear_tolerance)
         continue;

        //find perpendicular bisector
        //AB
        //midpoint
        XmidPoint_AB = (pointB.x + pointA.x) / 2.0;
        YmidPoint_AB = m_AB * XmidPoint_AB + b_AB;
        //perpendicular slope
        m2_AB = -1.0 / m_AB;
        //find b2
        b2_AB = YmidPoint_AB - m2_AB*XmidPoint_AB;

        //BC
        //midpoint
        XmidPoint_BC = (pointC.x + pointB.x) / 2.0;
        YmidPoint_BC = m_BC * XmidPoint_BC + b_BC;
        //perpendicular slope
        m2_BC = -1.0 / m_BC;
        //find b2
        b2_BC = YmidPoint_BC - m2_BC*XmidPoint_BC;

        //find intersection = circle center
        x = (b2_AB - b2_BC) / (m2_BC - m2_AB);
        y = m2_AB * x + b2_AB;
        center = Point2d(x,y);
        radius = cv::norm(center - pointB);

        /// geometry debug image
        if(false)
        {
            Mat debug_image = edges.clone();
            cvtColor(debug_image, debug_image, CV_GRAY2RGB);

            Scalar pink(255,0,255);
            Scalar blue(255,0,0);
            Scalar green(0,255,0);
            Scalar yellow(0,255,255);
            Scalar red(0,0,255);

            // the 3 points from which the circle is
              calculated in pink
            circle(debug_image, pointA, 3, pink);
            circle(debug_image, pointB, 3, pink);
            circle(debug_image, pointC, 3, pink);

            // the 2 lines (blue) and the perpendicular
              bisectors (green)
            line(debug_image,pointA,pointB,blue);
            line(debug_image,pointB,pointC,blue);
            line(debug_image,Point(XmidPoint_AB,YmidPoint_AB),center,green);
            line(debug_image,Point(XmidPoint_BC,YmidPoint_BC),center,green);

            circle(debug_image, center, 3, yellow); // center
            circle(debug_image, center, radius, yellow);// circle

            // 4th point check
            circle(debug_image, pointD, 3, red);

            imshow("ransac debug", debug_image);
            waitKey(0);
        }

        //check if the 4 point is on the circle
        if(abs(cv::norm(pointD - center) - radius) > radius_tolerance) continue;

        // vote
        std::vector<int> votes;
        std::vector<int> no_votes;
        for(int i = 0; i < (int)points.size(); i++)
        {
            double vote_radius = norm(points[i] - center);

            if(abs(vote_radius - radius) < radius_tolerance)
            {
                votes.push_back(i);
            }
            else
            {
                no_votes.push_back(i);
            }
        }

        // check votes vs circle_threshold
        if( (float)votes.size() / (2.0*CV_PI*radius) >= circle_threshold )
        {
            circles.push_back(Vec3f(x,y,radius));

            // voting debug image
            if(false)
            {
                Mat debug_image2 = edges.clone();
                cvtColor(debug_image2, debug_image2, CV_GRAY2RGB);

                Scalar yellow(0,255,255);
                Scalar green(0,255,0);

                circle(debug_image2, center, 3, yellow); // center
                circle(debug_image2, center, radius, yellow);// circle

                // draw points that voted
                for(int i = 0; i < (int)votes.size(); i++)
                {
                    circle(debug_image2, points[votes[i]], 1, green);
                }

                imshow("ransac debug", debug_image2);
                waitKey(0);
            }

            // remove points from the set so they can't vote on multiple circles
            std::vector<Point2d> new_points;
            for(int i = 0; i < (int)no_votes.size(); i++)
            {
                new_points.push_back(points[no_votes[i]]);
            }
            points.clear();
            points = new_points;
        }

        // stop RANSAC if there are few points left
        if((int)points.size() < points_threshold)
            break;
    }

    return;
}
```

Appendix B

RANSAC Ellipse Detection Script

The following script, based upon work by Chen and Chung[26], was developed in C++ and built upon an existing open-source RANSAC circle detection scheme. The script depends on the open-source computer vision library *OpenCV*[23]. Over emailed communication, personal explicit permission was given to adapt the script further, with the intention for the polished script to be uploaded online for open-source distribution. The circle specific elements of the original script were removed, and a scheme for detecting ellipses was implemented.

```
// ellipseDetector.cpp
//
//
// Base script: Kevin Hughes, circleDetector.cpp (2012)
// Modified for ellipse detection by Richard Bentley-Moyse,
   2014

#include "/usr/include/opencv2/highgui/highgui.hpp"
#include "/usr/include/opencv2/imgproc/imgproc.hpp"
#include "/usr/include/opencv2/opencv.hpp"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>


#include <time.h>

using namespace cv;
using namespace std;
```

```

typedef cv::Vec<float, 5> Vec5f;

// ellipseRANSAC
//
// input:
//      image - either CV_8UC1 or CV_8UC3
//      ellipses - return vector of Vec5f (x,y,  ,  ,  )
//      canny_threshold - higher canny threshold, lower is set to canny_threshold / 2
//      ellipse_threshold - value between 0 and 1 for the fraction of the ellipse that needs to vote for it to be accepted
//      numIterations - the number of RANSAC loops, the function will quit if there is no points left in the set
//
void ellipseRANSAC(Mat &image, vector<Vec5f> &ellipses, double canny_threshold, double ellipse_threshold, int numIterations);

int main(int argc, char *argv[])
{
    if(argc != 6)
    {
        cout << "Usage: " << argv[0] << "<image filename> <output filename> <canny threshold> <ellipse threshold> <iterations>" << endl;
        return -1;
    }

    // collect arguments
    string filename = argv[1];
    string output_filename = argv[2];
    double canny_threshold = atof(argv[3]);
    double ellipse_threshold = atof(argv[4]);
    int iterations = atoi(argv[5]);
    std::ofstream os("Output.txt");


    Mat image = imread(filename,0);
    vector <Vec5f> ellipses;

    const clock_t start = clock();
    ellipseRANSAC(image, ellipses, canny_threshold, ellipse_threshold, iterations);
    clock_t end = clock();

    cout << "Found " << (int)ellipses.size() << " ellipses." << endl;
    os << "Found " << (int)ellipses.size() << " ellipses." << endl;


    double time = ((double)(end - start)) / (double) CLOCKS_PER_SEC;
    std::cout << "RANSAC runtime: " << time << " seconds" << std::endl;
    os << "RANSAC runtime: " << time << " seconds" << endl;

//----------------------------------------------------------------------

    // Draw ellipses
    cvtColor(image,image,CV_GRAY2RGB);

    for(int i = 0; i < (int)ellipses.size(); i++)
    {
        int x = ellipses[i][0];
        int y = ellipses[i][1];
        double a = ellipses[i][2];
        double b = ellipses[i][3];
        double angle = ellipses[i][4];

        ellipse(image, Point(x,y), Size(a,b), angle*180/CV_PI, 0, 360, Scalar(200,200,200), 1, 8);
    }

    imshow("Detected Ellipses", image);
    imwrite(output_filename, image);

```

```
     std::cout << endl;

    for (size_t i = 0; i < ellipses.size(); i++ )
      {
          std::cout << "Ellipse "
                    << i+1
                    << " at ("
                    << ellipses[i][0]
                    << ", "
                    << ellipses[i][1]
                    << ") with    = "
                    << ellipses[i][2]
                    << ",    = "
                    << ellipses[i][3]
                    << " and    = "
                    << ellipses[i][4]* 180/CV_PI
                    << "  "
                    << std::endl;
          os << "Ellipse "
                    << i+1
                    << " at ("
                    << ellipses[i][0]
                    << ", "
                    << ellipses[i][1]
                    << ") with    = "
                    << ellipses[i][2]
                    << ",    = "
                    << ellipses[i][3]
                    << " and    = "
                    << ellipses[i][4]* 180/CV_PI
                    << "  "
                    << endl;

      }

    std::cout << endl;

    double centre_x_average = 0.0;
    double centre_y_average = 0.0;

    for (size_t i = 0; i < ellipses.size(); i++)
    {
        centre_x_average = centre_x_average + ellipses[i][0]/ellipses.size();
        centre_y_average = centre_y_average + ellipses[i][1]/ellipses.size();
    }

    if (ellipses.size() != 0)
    {
        cout << "Average center determined to be at (" << centre_x_average << ", " << centre_y_average << ") " << endl;
        os << "Average center determined to be at (" << centre_x_average << ", " << centre_y_average << ") ";
    }
    else
    {
        cout << "No centers determined" << endl;
        os << "No centers determined";
    }


    waitKey();
    return 0;
}
// ---------------------------------------------------------------
void ellipseRANSAC(Mat &image, vector<Vec5f> &ellipses, double canny_threshold, double ellipse_threshold, int numIterations)
{
    CV_Assert(image.type() == CV_8UC1 || image.type() == CV_8UC3);
    ellipses.clear();

    imshow("Source Image", image);

    // Edge Detection
    Mat edges;
    Canny(image, edges, MAX(canny_threshold/2,1),
      canny_threshold, 3);

    imshow("Canny Edge Detection Output", edges);
    cout << "Press any key to continue." << endl;
    waitKey();
    destroyWindow("Source Image");
    destroyWindow("Canny Edge Detection Output");

    cout << "Running..." << endl;

    // Create point set from Canny Output
    std::vector<Point2d> points;
    for(int r = 0; r < edges.rows; r++)
    {
        for(int c = 0; c < edges.cols; c++)
        {
            if(edges.at<unsigned char>(r,c) == 255)
            {
                points.push_back(cv::Point2d(c,r));
            }
        }
    }

    // 4 point objects to hold the random samples
    Point2d pointA;
    Point2d pointB;
    Point2d pointC;
    Point2d pointD;

    // 4 point objects to hold shifted random samples
    Point2d pointA2;
    Point2d pointB2;
    Point2d pointC2;
    Point2d pointD2;


    // distances between points
    double AB;
    double BC;
    double CA;
    double DC;

    // variables for line equations y = mx + b
    double m_AB;
    double b_AB;
    double m_BC;
    double b_BC;

    // variables for line midpoints
    double XmidPoint_AB;
    double YmidPoint_AB;
    double XmidPoint_BC;
    double YmidPoint_BC;

    // variables for perpendicular bisectors
    double m2_AB;
    double m2_BC;
    double b2_AB;
    double b2_BC;

    // RANSAC
    cv::RNG rng;
    int min_point_separation = 20; // minimum number of pixels between initial test points
    int colinear_tolerance = 1; // make sure points are not on a line
    float ellipse_tolerance = 0.03; // Tolerance test points
    int points_threshold = 10; //remaining number of unvoted points

    //--New_Variables-Not_Used -----------------------------------------------------------------

    //double test_centre_threshold = 3.0; // maximum allowed difference between initial centre and test centres
    //double axis_ratio_threshold = 0.97; // minimum ratio of axes of detected ellipse
    //
     ----------------------------------------------------------------

    int u,v;
    Point2d centre;

    // Iterate
    for(int iteration = 0; iteration < numIterations;
     iteration++)
    {
        std::cout << "RANSAC iteration: " << iteration <<
         std::endl;

        // get 4 random points
        pointA = points[rng.uniform((int)0, (int)points.size
         ())];
        pointB = points[rng.uniform((int)0, (int)points.size
         ())];
        pointC = points[rng.uniform((int)0, (int)points.size
         ())];
        pointD = points[rng.uniform((int)0, (int)points.size
         ())];

        // calc lines
        AB = norm(pointA - pointB);
        BC = norm(pointB - pointC);
        CA = norm(pointC - pointA);
        DC = norm(pointD - pointC);

        // one or more random points are too close together
        if(AB < min_point_separation || BC <
         min_point_separation || CA < min_point_separation
         || DC < min_point_separation) continue;

        //find line equations for AB and BC
        //AB
        m_AB = (pointB.y - pointA.y) / (pointB.x - pointA.x
         + 0.000000001); //avoid divide by 0
        b_AB = pointB.y - m_AB*pointB.x;

        //BC
        m_BC = (pointC.y - pointB.y) / (pointC.x - pointB.x
         + 0.000000001); //avoid divide by 0
        b_BC = pointC.y - m_BC*pointC.x;


        //test colinearity (ie the points are not all on the
          same line)
        if(abs(pointC.y - (m_AB*pointC.x + b_AB +
         colinear_tolerance)) < colinear_tolerance)
         continue;

        //find perpendicular bisector
        //AB
        //midpoint
        XmidPoint_AB = (pointB.x + pointA.x) / 2.0;
        YmidPoint_AB = m_AB * XmidPoint_AB + b_AB;
        //perpendicular slope
        m2_AB = -1.0 / m_AB;
        //find b2
        b2_AB = YmidPoint_AB - m2_AB*XmidPoint_AB;

        //BC
        //midpoint
        XmidPoint_BC = (pointC.x + pointB.x) / 2.0;
        YmidPoint_BC = m_BC * XmidPoint_BC + b_BC;
        //perpendicular slope
        m2_BC = -1.0 / m_BC;
        //find b2
        b2_BC = YmidPoint_BC - m2_BC*XmidPoint_BC;

        //find intersection = ellipse center
        u = (b2_AB - b2_BC) / (m2_BC - m2_AB); //x-
         coordinate
        v = m2_AB * u + b2_AB;                  //y-
         coordinate
        centre = Point2d(u,v);
//--NEW-CODE
  ---------------------------------------------------------------------
```

```

        //Set center coordinate as origin
        pointA2 = pointA - centre;
        pointB2 = pointB - centre;
        pointC2 = pointC - centre;
        pointD2 = pointD - centre;

        std::vector<Point2d> points2 = points;
        for(size_t i = 0; i < points.size(); i++)
        {
            points2[i] = points[i] - centre;
        }

        //Construct X matrix from 3 simultaneous origin-
          centered
        //  ellipse equations using points A2, B2 and C2
        Matx33d X(pow(pointA2.x,2), 2.0 * pointA2.x *
          pointA2.y, pow(pointA2.y,2),
                    pow(pointB2.x,2), 2.0 * pointB2.x *
                      pointB2.y, pow(pointB2.y,2),
                    pow(pointC2.x,2), 2.0 * pointC2.x *
                      pointC2.y, pow(pointC2.y,2) );
        Matx33d Xi = X.inv();

        Matx31d B(1.0,
                    1.0,
                    1.0);
        // Calculate the parameters for "a0*x^2 + 2*a1*x*y +
           a2*y^2 = 1"
        Matx31d A = Xi * B;



                /// geometry debug image
                        if(false) //set to true to analyse
                          each iteration visually
                        {
                            Mat debug_image = edges.clone();
                            cvtColor(debug_image,
                                debug_image, CV_GRAY2RGB);

                            Scalar pink(255,0,255);
                            Scalar blue(255,0,0);
                            Scalar green(0,255,0);
                            Scalar yellow(0,255,255);
                            Scalar red(0,0,255);

                            // the 3 points from which the
                                circle is calculated in pink
                            circle(debug_image, pointA, 3,
                                pink);
                            circle(debug_image, pointB, 3,
                                pink);
                            circle(debug_image, pointC, 3,
                                pink);

                            // the 2 lines (blue) and the
                                perpendicular bisectors (green
                                )
                            line(debug_image,pointA,pointB,
                                blue);
                            line(debug_image,pointB,pointC,
                                blue);
                            line(debug_image,Point(
                                XmidPoint_AB,YmidPoint_AB),
                                centre,green);
                            line(debug_image,Point(
                                XmidPoint_BC,YmidPoint_BC),
                                centre,green);

                            circle(debug_image, centre, 3,
                                yellow); // center
                            //circle(debug_image, centre,
                                radius, yellow);// circle

                            // 4th point check
                            circle(debug_image, pointD, 3,
                                red);

```

```
                            imshow("ransac debug", debug_image);
                            waitKey(0);
                        }

        //check if the 4th point is considered to be on the ellipse
        if(norm(A(0)*pow(pointD2.x,2) + 2*A(1)*pointD2.x*pointD2.y + A(2)*pow(pointD2.y,2) - 1) < ellipse_tolerance) continue;

        //Calculate 'physical' parameters from these
        double theta = atan2(A(2) - A(0) - sqrt( pow(A(0)-A(2),2) + 4*pow(A(1),2) ), 2*A(1)); //in radians

        //if(theta > CV_PI)
        //{
        //  theta -= 2*CV_PI;
        //}
        //else if(theta < -CV_PI)
        //{
        //  theta += 2*CV_PI;
        //}

        double beta = 1.0/sqrt( A(0)/2.0 + A(2)/2.0 - A(1)/(2*cos(theta)*sin(theta)) );

        double alpha = 1/sqrt( A(0)+A(2) - 1/pow(beta,2) );

        if(alpha < beta)
        {
            double token = alpha;
            alpha = beta;
            beta = token;
        }


        //check if eccentricity is under desired threshold
        //double eccentricity = sqrt(1 - pow(beta/alpha,2));
        //if(eccentricity > eccentricity_threshold) continue
        ;

        //Voting procedure
        std::vector<int> votes;
        std::vector<int> no_votes;
        for(int i=0; i<(int)points.size(); i++)
        {
            //Test if point is within ellipse tolerance
            float vote_difference = norm(A(0)*pow(points2[i
              ].x,2) +A(1)*points2[i].x*points2[i].y + A(2)*
              pow(points2[i].y,2) - 1);

            if(vote_difference < ellipse_tolerance)
            {
                votes.push_back(i);
                //cout << "i= " << i << endl;
            }
            else
            {
                no_votes.push_back(i);
            }
        }

        // Approximate the circumference of the ellipse
          using Ramanujan's approximation (see wiki for now
          until source found)
        //double circumference = CV_PI * (3*alpha + 3*beta -
           sqrt(10*alpha*beta + 3*pow(alpha,2) + 3*pow(beta
          ,2)));
        double h = pow(alpha - beta,2)/pow(alpha + beta,2);
        double circumference = CV_PI * (alpha+beta)*(1.0 +
          3.0*h/(10.0 + sqrt(4.0-3.0*h)));



        //if( (float)votes.size() > circumference) continue;
        if( (float)votes.size() / circumference >=
          ellipse_threshold)
        {
            ellipses.push_back(Vec5f(u,v,alpha,beta,theta));

            //Remove points from set so they can only vote
             on one ellipse
            std::vector<Point2d> new_points;
            for(int i = 0; i < (int)no_votes.size(); i++)
            {
                new_points.push_back(points[no_votes[i]]);
            }
            points.clear();
            points = new_points;
        }
//
 -------------------------------------------------------------------------

                // stop RANSAC if there are few points left
        if((int)points.size() < points_threshold)

        break;
    }

    return;
}
```

Appendix C

Radial Profile Script

An script for extracting the radial spectrum from experimental images was developed, using an open-source implementation of an azimuthal average function(ref). It was also useful in visually determining center dependence of the IAT script. In future applications, the script will be extended to produce the photoelectron kinetic energy spectrum for determination of

Listing C.1: Main Script

```
from radialprofile import azimuthalAverage

from scipy import constants
import numpy  as np
np.seterr(divide='ignore', invalid='ignore')
import pyfits
import matplotlib.pyplot as plt


#---Load the FITS image:------------------------------
image_location = raw_input('Input file location: ')
hdul = pyfits.open(image_location)
image = hdul[0].data

x_size = image.shape[1]
y_size = image.shape[0]

#note that shape parameter returns a tuple of the format (
  NAXISn, ... , NAXIS2, NAXIS1) so that
#NAXIS1 is number of columns (x-axis), NAXIS2 is the number
  of rows (y-axis) etc

```

```

#---Collecting input parameters:--------------------
center_x = float(raw_input("Center coordinate X-axis value
  is: "))
center_y = float(raw_input("Center coordinate Y-axis value
  is: "))
center = np.array([center_x, center_y])
bin = float(raw_input("Binsize is: "))

#---Fix graph limits to smallest viable radius (x, -x, y, -y
  ) from center:
if 2*center_y < y_size :
    ymax = center_y
else:
    ymax = y_size - center_y

if 2*center_x < x_size :
    xmax = center_x
else:
    xmax = x_size - center_x

if xmax < ymax:
    rmax_display = xmax
else:
    rmax_display = ymax

#---Extract Radial Profile:------------------------------
radial_prof_x, radial_prof_y = azimuthalAverage(image,
  center, binsize=bin, interpnan=True, returnradii=True)

#---Plotting the Radial Profile:------------------------
#ax = plt.subplot(111)
#ax.set_xlim(0.0,rmax_display,10.0)

fig = plt.figure()
plt.plot(radial_prof_x, radial_prof_y)
plt.subplot(111).set_xlim(0.0,rmax_display,10.0)
fig.suptitle('Azimuthally-Averaged Radial Profile', fontsize
  =19)
plt.xlabel('Radii (pixels)',fontsize=17)
plt.ylabel('Intensity',fontsize=19)
plt.subplot(111).get_yaxis().set_ticks([])
plt.show()

#---Debug commands--------------------------------
#print radial_prof

#print image.shape

#Image debug - displays input FITS image:
#plt.imshow(image,cmap=plt.get_cmap('gray'))
```

Listing C.2: azimuthalAverage radialAverage Script

```
import numpy as np

def azimuthalAverage(image, center=None, stddev=False, returnradii=False, return_nr=False,
        binsize=0.5, weights=None, steps=False, interpnan=False, left=None, right=None,
        mask=None ):
    """
    Calculate the azimuthally averaged radial profile.

    image - The 2D image
    center - The [x,y] pixel coordinates used as the center. The default is
             None, which then uses the center of the image (including
             fractional pixels).
    stddev - if specified, return the azimuthal standard deviation instead of the average
    returnradii - if specified, return (radii_array,radial_profile)
    return_nr   - if specified, return number of pixels per radius *and* radius
    binsize - size of the averaging bin.  Can lead to strange results if
        non-binsize factors are used to specify the center and the binsize is
        too large
```

```python
    weights - can do a weighted average instead of a simple average if this keyword parameter
        is set.  weights.shape must = image.shape.  weighted stddev is undefined, so don't
        set weights and stddev.
    steps - if specified, will return a double-length bin array and radial
        profile so you can plot a step-form radial profile (which more accurately
        represents what's going on)
    interpnan - Interpolate over NAN values, i.e. bins where there is no data?
        left,right - passed to interpnan; they set the extrapolated values
    mask - can supply a mask (boolean array same size as image with True for OK and False for not)
        to average over only select data.

    If a bin contains NO DATA, it will have a NAN value because of the
    divide-by-sum-of-weights component.  I think this is a useful way to denote
    lack of data, but users let me know if an alternative is prefered...

    """
    # Calculate the indices from the image
    y, x = np.indices(image.shape)

    if center is None:
        center = np.array([(x.max()-x.min())/2.0, (y.max()-y.min())/2.0])
    # Calculates hypotenuse
    r = np.hypot(x - center[0], y - center[1])

    if weights is None:
        weights = np.ones(image.shape)
    elif stddev:
        raise ValueError("Weighted standard deviation is not defined.")

    if mask is None:
        mask = np.ones(image.shape,dtype='bool')
    # obsolete elif len(mask.shape) > 1:
    # obsolete     mask = mask.ravel()

    # the 'bins' as initially defined are lower/upper bounds
      for each bin
    # so that values will be in [lower,upper)
    nbins = int(np.round(r.max() / binsize)+1)
    maxbin = nbins * binsize
    bins = np.linspace(0,maxbin,nbins+1)
    # but we're probably more interested in the bin centers
     than their left or right sides...
    bin_centers = (bins[1:]+bins[:-1])/2.0

    # how many per bin (i.e., histogram)?
    # there are never any in bin 0, because the lowest index
      returned by digitize is 1
    #nr = np.bincount(whichbin)[1:]
    nr = np.histogram(r,bins)[0]

    # recall that bins are from 1 to nbins (which is
     expressed in array terms by arange(nbins)+1 or xrange
     (1,nbins+1) )
    # radial_prof.shape = bin_centers.shape
    if stddev:
        # Find out which radial bin each point in the map
          belongs to
        whichbin = np.digitize(r.flat,bins)
        # This method is still very slow; is there a trick
          to do this with histograms?
        radial_prof = np.array([image.flat[mask.flat*(
          whichbin==b)].std() for b in xrange(1,nbins+1)])
    else:
        radial_prof = np.histogram(r, bins, weights=(image*
          weights*mask))[0] / np.histogram(r, bins, weights
          =(mask*weights))[0]

    if interpnan:
        radial_prof = np.interp(bin_centers,bin_centers[radial_prof==radial_prof],radial_prof[radial_prof==radial_prof],left=left,right=right)

    if steps:
        xarr = np.array(zip(bins[:-1],bins[1:])).ravel()
        yarr = np.array(zip(radial_prof,radial_prof)).ravel()
        return xarr,yarr
    elif returnradii:
        return bin_centers,radial_prof
    elif return_nr:
        return nr,bin_centers,radial_prof
    else:
        return radial_prof

def azimuthalAverageBins(image,azbins,symmetric=None, center=None, **kwargs):
    """ Compute the azimuthal average over a limited range of angles
    kwargs are passed to azimuthalAverage """
    y, x = np.indices(image.shape)
    if center is None:
        center = np.array([(x.max()-x.min())/2.0, (y.max()-y.min())/2.0])
    r = np.hypot(x - center[0], y - center[1])
    theta = np.arctan2(x - center[0], y - center[1])
    theta[theta < 0] += 2*np.pi
    theta_deg = theta*180.0/np.pi

    if isinstance(azbins,np.ndarray):
        pass
    elif isinstance(azbins,int):
        if symmetric == 2:
            azbins = np.linspace(0,90,azbins)
            theta_deg = theta_deg % 90
        elif symmetric == 1:
            azbins = np.linspace(0,180,azbins)
            theta_deg = theta_deg % 180
        elif azbins == 1:
            return azbins,azimuthalAverage(image,center=center,returnradii=True,**kwargs)
        else:
            azbins = np.linspace(0,359.9999999999999,azbins)
    else:
        raise ValueError("azbins must be an ndarray or an integer")

    azavlist = []
    for blow,bhigh in zip(azbins[:-1],azbins[1:]):
        mask = (theta_deg > (blow % 360)) * (theta_deg < (bhigh % 360))
        rr,zz = azimuthalAverage(image,center=center,mask=mask,returnradii=True,**kwargs)
        azavlist.append(zz)

    return azbins,rr,azavlist

def radialAverage(image, center=None, stddev=False, returnAz=False, return_naz=False,
        binsize=1.0, weights=None, steps=False, interpnan=False, left=None, right=None,
        mask=None, symmetric=None ):
    """
    Calculate the radially averaged azimuthal profile.
    (this code has not been optimized; it could be speed boosted by ~20x)

    image - The 2D image
    center - The [x,y] pixel coordinates used as the center. The default is
             None, which then uses the center of the image (including
             fractional pixels).
    stddev - if specified, return the radial standard deviation instead of the average
    returnAz - if specified, return (azimuthArray,azimuthal_profile)
    return_naz   - if specified, return number of pixels per azimuth *and* azimuth
    binsize - size of the averaging bin.  Can lead to strange results if
        non-binsize factors are used to specify the center and the binsize is
        too large
    weights - can do a weighted average instead of a simple average if this keyword parameter
        is set.  weights.shape must = image.shape.  weighted stddev is undefined, so don't
        set weights and stddev.
    steps - if specified, will return a double-length bin array and azimuthal
        profile so you can plot a step-form azimuthal profile (which more accurately
        represents what's going on)
    interpnan - Interpolate over NAN values, i.e. bins where there is no data?
        left,right - passed to interpnan; they set the extrapolated values
    mask - can supply a mask (boolean array same size as image with True for OK and False for not)
        to average over only select data.

    If a bin contains NO DATA, it will have a NAN value because of the
    divide-by-sum-of-weights component.  I think this is a useful way to denote
    lack of data, but users let me know if an alternative is prefered...

    """
    # Calculate the indices from the image
    y, x = np.indices(image.shape)

    if center is None:
        center = np.array([(x.max()-x.min())/2.0, (y.max()-y.min())/2.0])

    r = np.hypot(x - center[0], y - center[1])
    theta = np.arctan2(x - center[0], y - center[1])
    theta[theta < 0] += 2*np.pi
    theta_deg = theta*180.0/np.pi
    maxangle = 360

    if weights is None:
        weights = np.ones(image.shape)
    elif stddev:
        raise ValueError("Weighted standard deviation is not
          defined.")

    if mask is None:
        # mask is only used in a flat context
        mask = np.ones(image.shape,dtype='bool').ravel()
    elif len(mask.shape) > 1:
        mask = mask.ravel()

    # allow for symmetries
    if symmetric == 2:
        theta_deg = theta_deg % 90
        maxangle = 90
    elif symmetric == 1:
        theta_deg = theta_deg % 180
        maxangle = 180

    # the 'bins' as initially defined are lower/upper bounds
      for each bin
    # so that values will be in [lower,upper)
    nbins = int(np.round(maxangle / binsize))
    maxbin = nbins * binsize
    bins = np.linspace(0,maxbin,nbins+1)
    # but we're probably more interested in the bin centers
      than their left or right sides...
    bin_centers = (bins[1:]+bins[:-1])/2.0

    # Find out which azimuthal bin each point in the map
      belongs to
    whichbin = np.digitize(theta_deg.flat,bins)

    # how many per bin (i.e., histogram)?
    # there are never any in bin 0, because the lowest index
        returned by digitize is 1
    nr = np.bincount(whichbin)[1:]

    # recall that bins are from 1 to nbins (which is
        expressed in array terms by arange(nbins)+1 or xrange
        (1,nbins+1) )
    # azimuthal_prof.shape = bin_centers.shape
    if stddev:
        azimuthal_prof = np.array([image.flat[mask*(whichbin
            ==b)].std() for b in xrange(1,nbins+1)])
    else:
        azimuthal_prof = np.array([(image*weights).flat[mask
            *(whichbin==b)].sum() / weights.flat[mask*(
            whichbin==b)].sum() for b in xrange(1,nbins+1)])

    #import pdb; pdb.set_trace()

    if interpnan:
        azimuthal_prof = np.interp(bin_centers,
            bin_centers[azimuthal_prof==azimuthal_prof],
            azimuthal_prof[azimuthal_prof==azimuthal_prof],
            left=left,right=right)

    if steps:
        xarr = np.array(zip(bins[:-1],bins[1:])).ravel()
        yarr = np.array(zip(azimuthal_prof,azimuthal_prof)).
            ravel()
        return xarr,yarr
    elif returnAz:
        return bin_centers,azimuthal_prof
    elif return_naz:
        return nr,bin_centers,azimuthal_prof
    else:
        return azimuthal_prof

def radialAverageBins(image,radbins, corners=True, center=
    None, **kwargs):
    """ Compute the radial average over a limited range of
        radii """
    y, x = np.indices(image.shape)
    if center is None:
        center = np.array([(x.max()-x.min())/2.0, (y.max()-y.min())/2.0])
    r = np.hypot(x - center[0], y - center[1])

    if isinstance(radbins,np.ndarray):
        pass
    elif isinstance(radbins,int):
        if radbins == 1:
            return radbins,radialAverage(image,center=center,returnAz=True,**kwargs)
        elif corners:
            radbins = np.linspace(0,r.max(),radbins)
        else:
            radbins = np.linspace(0,np.max(np.abs(np.array([x-center[0],y-center[1]]))),radbins)
    else:
        raise ValueError("radbins must be an ndarray or an integer")

    radavlist = []
    for blow,bhigh in zip(radbins[:-1],radbins[1:]):
        mask = (r<bhigh)*(r>blow)
        az,zz = radialAverage(image,center=center,mask=mask,returnAz=True,**kwargs)
        radavlist.append(zz)

    return radbins,az,radavlist
```

Appendix D

Research Project Proposal

The project direction significantly deviated from what was initially intended. The laboratory apparatus is currently being upgraded to include the SEVI camera for use with photoelectron spectroscopy. The wavepacket simulations portion of the project is widely applicable to all techniques in the laboratory, so was put on hold until during my PhD in order to develop the software backing behind the new camera. This way, the camera will not only be operational but producing usable information in time for research during my PhD project.

An idealised flow chart of research direction is shown below:

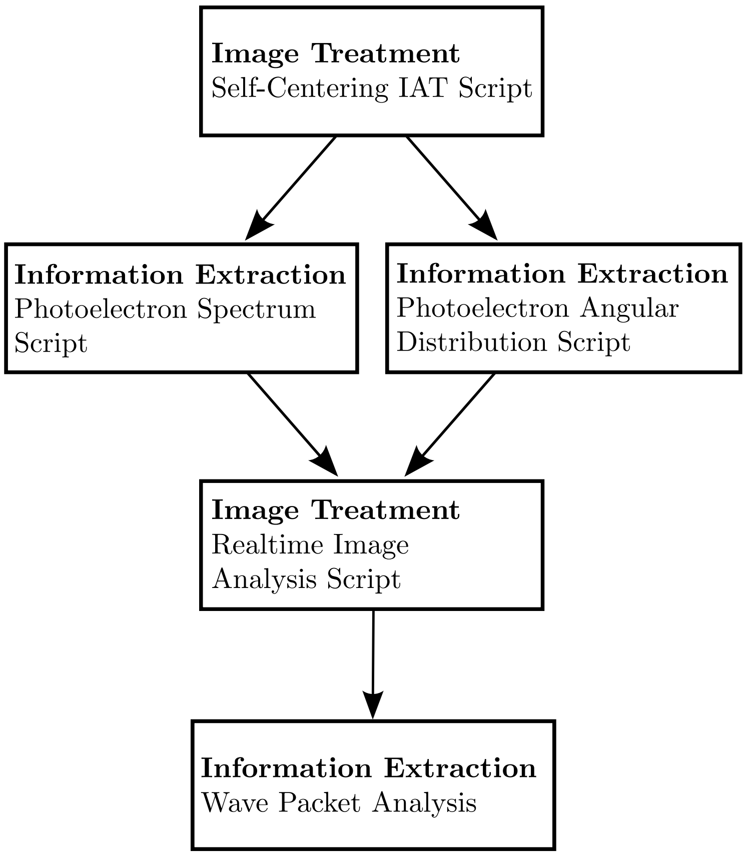


Image Treatment
Self-Centering IAT Script
Information Extraction
Photoelectron Spectrum Script
Information Extraction
Photoelectron Angular Distribution Script
Image Treatment
Realtime Image Analysis Script
Information Extraction
Wave Packet Analysis

M.Sc. Physics Project

Research Proposal

Richard Bentley-Moyse
School of Physics, University of Western Australia
April 2013

Wavepacket Dynamic Simulations for Photoelectron Spectroscopy

Keywords: photoelectron spectroscopy, *ab initio*, Time-of-Flight mass spectrometry, ion-molecule clusters, quantum chemistry
Supervisers: Asst Prof Duncan Wild (UWA-Chemistry), Marcus Kettner (UWA-Chemistry)

1 Research Plan

1.1 Aims

This project aims to explore the electronic and geometric structure of select gas-phase ion-molecule complexes and clusters via *ab initio* calculations and vibrational energy level modelling, followed by experimental verification and comparison. Emphasis for the project will lay in developing and using different techniques to model the dynamics of wavepackets and from these results provide an advanced theoretical method to predict experimental results for the research group. Comparison of the resulting methods with tried and tested experimental findings will allow for any adjustments and refinements of the theoretical models.

1.2 Significance

This project aims to develop advanced techniques to handle and model the dynamics of wavepackets for use in solving the Schrödinger equation for many-electron ion/molecule complexes and cluster systems. The outcomes will be used to explore the vibrational potential energy spectra of experimentally-tested anionic clusters and predict the vibrational potential energy spectra of upcoming experiments. The techniques will be used to verify current experimental findings and compare with past modelling techniques, as well as guide future spectroscopy of anionic clusters improving experimental efficiency.

As it stands, the Schrödinger equation can only be solved exactly for the Hydrogen atom. Analysis of a broader Schrödinger equation encompassing rovibrational effects within molecules, wavepackets and associated dynamics can give an alternative method for theoretical calculation to existing techniques, and hopefully such research can lead to a more complete quantum theory of matter.

1.3 Methods

Most work will be undertaken with the Wild Laser Spectroscopy Group located at the Bayliss Building in the University of Western Australia, with the potential for some use of the iVEC@UWA supercomputer located in iVEC at the University of Western Australia.

1.3.1 *Ab Initio* Calculations

Ab initio calculations in quantum chemistry refer to computational calculations from first-principles. In this project, *ab initio* calculations will be performed by utilising a variety of different modelling software packages, with the main focus in using the EasyWave and Gaussian software packages in conjunction with GAMESS and ORCA. The calculations will include multidimensional potential energy surface scans, vibrational frequency analysis, vibrational wavefunction generation and wave packet dynamics simulations. The latter technique involves following the dynamics and applying a Fourier transform to arrive at the excited state energy levels and hence allowing the prediction of experimental spectra.

1.3.2 Time-of-Flight Mass Spectrometer - Photoelectron Spectrometer

Experimental verification of the simulations will be conducted using a Time-of-Flight Mass Spectrometer coupled with a Photoelectron Spectrometer (TOF-PES). Gas phase ion-molecule complexes will be produced in a source chamber and extracted using an electric potential to separate the anions from the cations allowing us to analyse a particular ionic complex/cluster chosen using mass spectrometry(1). The specific sample ionic species will then be bombarded with ultra-violet laser radiation from a Nd:YAG laser and the resulting ejected photoelectrons collected by the photoelectron spectrometer. This gives not only a detailed analysis of the mass of the ionic species but also the energy spectrum of the ejected photoelectrons; from this the interactions of the electrons within the species can be determined and compared to *ab initio* calculations(2).

1.4 Status

The research group currently consists of three PhD. chemistry students, two Master of Physical Science physics students and one Honours chemistry student. The research group has been been utilising Time-of-Flight spectroscopy to analyse the energy spectra of different ion-molecule clusters, namely chloride-carbon monoxide complexes(6), fluoride-acetylene clusters(7), bromide-carbon monoxide complexes(8) and iodide-carbon monoxide clusters(9). From this analysis, detailed information about the potential energy surface and vibrational frequencies of ion-molecule clusters. All experimental findings are backed up by rigorous *ab initio* calculations.

1.5 Problems

Potential problems may arise in the learning of basic programming and computational techniques required for the project with no formal background. The use of and nomenclature of different software packages may prove troublesome, especially with transferring data between the various packages. As fluent understanding of different techniques to treat the Schrödinger equation for many-electron systems is required, some difficulty can be expected in grasping new topics without the aid of a Lecturer-Student learning setting. Other difficulties may include scheduling time with the experimental apparatus, but as only minimal experimentation will be required, it is expected that this will be easy to schedule around others using the apparatus. Marcus Kettner, a PhD. student in the group, is supervising the modelling and theory component of the project; his experience in programming and computing with be an invaluable aid in gaining a good background in data modelling. There is a good focus on physical and theoretical chemistry in the department and a firm emphasis on inter-group collaboration between the different groups exists, so various chemists in the department can be consulted on new theoretical techniques if assistance is required.

2 Benefits

In all facets of science, a greater understanding of atomic and molecular electronic structure allows us a greater opportunity to exploit the novel and 'exotic' properties of matter; properties that can in turn lead to significant technological advances. This research project hopes to lead to a deeper understanding of matter at a fundamental level and potentially provide a stepping stone to new technology and devices.

3 Publications

The measurement and interpretation of the vibrational energy spectrums of different ionic complexes and clusters has been investigated by the Wild group(6; 7; 8; 9) using the Gaussian and GAMESS software packages.

Professor Anna Krylov (University of South California) and her research group have also produced papers looking into theoretical methods using wavepacket dynamic analysis to provide an interface between spectroscopic results and electronic structure/dynamics(3; 4; 5). It is with use of Krylov's software package EasyWave that we hope to successfully model the research groups data using wavepacket calculations.

4 Costs

The research group already has current licenses for all software packages, has both it's own computational computers and allocated time at both the Fornax supercomputer (iVEC@UWA) located at iVEC in UWA and National Computational Infrastructure (NCI) under the National Computational Merit Allocation Scheme (NCMAS), so the only cost expected for the computation side of the project is a personal workstation with an expected cost of $1100.

In addition to this, it is expected that we will attend a conference in December 2013 with an expected airfare cost of $400 return and conference fees totalling $500.

References

[1] LaMacchia, RJ 2008, 'Towards anion photoelectron spectroscopy of complexes and clusters', *Honours Thesis*

[2] Lindle, DW and Hemmers, OA 2001, 'Time-of-flight photoelectron spectroscopy of atoms and molecules', *Proceedings of the 5th International School and Symposium on Synchrotron Radiation in Natural Science*, vol. 328, no. 1-2, p. 27-34

[3] Vanovschi, V; Krylov, AI; and Wenthold, PG 2008, 'Structure, vibrational frequencies, ionization energies, and photoelectron spectrum of the para-benzyne radical anion', *Theoretical Chemistry Accounts*, vol. 120, p. 45-58

[4] Koziol, L; Wang, Y; Braams, BJ; Bowman, JM; and Krylov, AI 2008, 'The theoretical prediction of infrared spectra of trans- and cis- hydroxycarbene calculated using full dimensional ab initio potential energy and dipole moment surfaces', *Journal of Chemical Physics*, vol. 128, n. 20

[5] Koziol, L; Mozhayskiy, VA; Braams, BJ; Bowman, JM; and Krylov, AI 2009, '*Ab initio* calculation of photoelectron spectra of the hydroxycarbene diradicals', *Journal of Physical Chemistry A*, vol. 113, p. 7802-7809

[6] Lapere, KM; LaMacchia, RJ; Quak, LH; McKinley, AJ; and Wild, DA 2011, 'Anion photoelectron spectroscopy and ab initio calculations of the gas phase chloride-carbon monoxide complex: Cl-...CO', *Chemical Physics Letters*, vol. 504, no. 1-3, p. 13-19

[7] Wild, DA; Loh, ZM; and Bieske, EJ 2011, 'Infrared spectra and ab initio calculations of fluoride-acetylene clusters: F-...(HCCH)n, n = 3-6', *Australian Journal of Chemistry*, vol. 64, p. 633-637

[8] Lapere, KM; LaMacchia, RJ; Quak, LH; Kettner, M; et al 2012, The bromide - carbon monoxide gas phase complex: anion photoelectron spectroscopy and ab initio calculations, *Australian Journal of Chemistry*, vol. 65, no. 5, p. 457-462

[9] Lapere, KM; LaMacchia, RJ; Quak, LH; Kettner, M; et al 2012, Anion photoelectron spectra and ab initio calculations of the iodide-carbon monoxide clusters: i(CO)n, n = 14, *The Journal of Physical Chemistry A*, vol. 116, no. 14, pp. 3577-3584