Scanning knife-edge method for laser plasma size measurements

Diploma paper by

Johan Anderberg

LRAP-165
Lund, September 1994

Abstract

In this Diploma paper the knife edge method is developed to measure the size of a small laser-plasma x-ray source. The method is theoretically discussed, showing that very high spatial resolution (~2 μm) is feasible. Accurate computer control of a DC motor for knife edge positioning and simultaneous data acquisition has been implemented. Experimental measurements on small (20-25 μm) water-window laser plasma x-ray sources demonstrate the method. Compared to other x-ray size measurement methods, the simplicity and high resolution of the knife edge method make it attractive for fast 1-D measurements of plasma size. 2-D imaging can be performed using tomographic methods.

Table of contents

1 Introduction

In this Diploma paper, the knife edge method for measurements of the size of a small soft laser-plasma x-ray sources, is investigated. Sect. 1 provides a background to x-ray emission from laser-produced plasmas. In Sect. 2 the knife edge method is theoretically described and compared to other size measurement methods. In Sect. 3 the equipment of the knife edge method is described and in Sect. 4 the experiment and results are discussed.

1.1 Overview

The source of a soft x-ray microscope must have a high brightness. Since brightness is defined as the number of photons emitted per $sr \cdot \mu m^2 \cdot BW$, the size of the source is important. Thus, exact size measurements are important. Many methods are available, and each have their advantages, but the knife edge method has a considerable high resolution and is very simple.

A brief introduction of laser produced plasma and the soft x-ray microscope source is made in this section as well as size measurements.

1.2 Laser Produced Plasma (LPP)

Plasma is regarded as the fourth state of matter and is characterised as an assembly of ions and electrons. In its simplest form it is a state of ionisation, either partial or complete, but it retains its overall neutrality. There are five main atomic processes in plasmas. In collisional excitation or ionisation an electron either excitates an ion or neutral atom or has enough kinetic energy to remove another electron from an atom. Photo- excitation and ionisation work both in the same way but they use photons instead of electrons. Bremsstrahlung involves an encounter between an electron and an ion and may be regarded as in a continuum state of a charged atomic system. In the inverse process a photon is absorbed by the electron-ion system and the electron is lifted from a lower level in the continuum to a higher one. Both Bremsstrahlung and the inverse one are of great importance in laser produced plasmas.

The basic features of LPP[1] are high density, very high temperature and that it lasts mainly the same time as the length of the laser pulse. In Fig. 1.1 (a) a cross-section of a plasma is shown. Figure 1.1 (b-d) displays important plasma parameters. The laser radiation field penetrates a very short depth in a metal but the strength of the field is sufficient to result in a powerful interaction with conducting electrons so that heating, evaporation and ionisation occur rapidly. This is true for a irradiance of 10^{16}-10^{20} W/m^2. Isolators that are transparent at optical wavelengths have a much higher ionisation potential. Nevertheless free electrons are produced. When the initial electrons

are formed, the laser energy starts being absorbed and a thin sheet of plasma is created near the surface. Laser radiation is now absorbed by inverse Bremsstrahlung. At the beginning the plasma frequency is much smaller than the laser frequency, $\omega_p << \omega$, and the rate of absorption depends on the square of the electron density, n_e^2. The absorbed energy causes an increase in electron temperature which produces further ionisation and a consequent increase in n_e until n_e reaches the critical density n_c. The surface then becomes opaque to incoming radiation and in the layers just in front of the surface the absorption becomes very large. This does not stop the plasma growth. Because of the heating which follows the absorption of energy by inverse Bremsstrahlung the plasma is driven away from the target surface and n_e decreases resulting in a laser absorption closer to the target. These two absorption processes are self regulating. Altogether it results in a heating and expansion of plasma throughout the pulse.

1.3 X-ray emission

There are three processes that result in a radiative emission from a plasma[2]. The two which involve free electrons, Bremsstrahlung from hot electrons colliding with ions and recombination radiation, which give a continuous radiation spectrum. The third is a bound-bound process, radiative decay of excited ions, that gives line radiation. The relative contributions of these processes depend on the target material, power density of the laser pulse and the spectral region to be studied. Some general aspects can be said about the total spectrum. Low atomic number target with moderate plasma temperatures (T_e= 10-100 eV) is dominated by line radiation. With the increase of temperature ions are stripped of their electrons. This increases the Bremsstrahlung. For higher atomic numbers the dominating process is recombination. The emission is then quasi-blackbody with spectral structure resulting from the free-bound and bound-bound transitions.

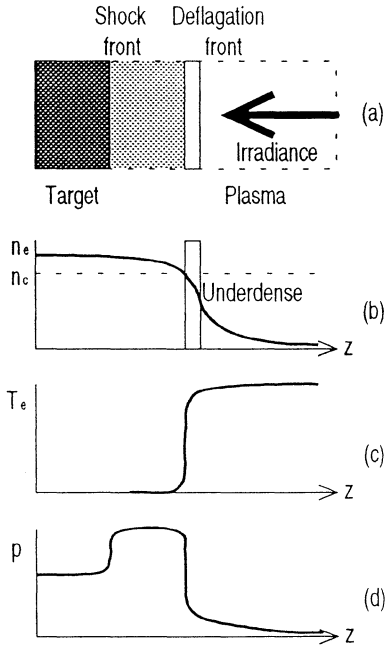


Figure 1.1 Some plasma features along the irradiation axis.
a) A cross-section of a plasma.
b) The electron density.
c) The electron temperature.
d) The plasma density.

Different regions of the X-ray spectra originate predominately from specific zones of the plasma. Figure 1.2 show a typical x-ray spectrum from a plasma where the different zones are marked. Starting from low energies, X-rays below 1 keV are

emitted primarily from the over dense region of the plasma. The subkilovolt spectrum exhibits sharp structure by the final state shell designation. X-rays in the kilovolt region are mostly emitted from the corneal plasma zone. Bound- bound lines and bound-free continuum from highly charged ions are the principal emissions. There are K, L and M-shell spectra depending on where the electron vacancy is. The X-ray spectrum above about 10 keV is generally dominated by Bremsstrahlung from superthermal electrons.

1.4 X-ray microscope source - droplet target

Biological cell investigation may be the most important field for the X-ray microscope. With visible light, structures below 0.2 μm can not be observed due to diffraction. An electron microscope has a resolution of 0.1-10 nm, but the cells have to be dehydrated, thinly sliced and stained with contrast elements. With a soft x-ray microscope[3], a resolution of ~10-20 nm is possible and preparations are not needed. Of course the cell will not live for a very long time depending on how long it is exposed to radiation, but the advantages are dominating.

The soft x-ray region is roughly defined as the spectral range between 0.2 nm and 30 nm. The most important region for microscopy is called the 'water window'. Between the oxygen absorption edge at 2.3 nm and the carbon absorption edge at 4.4 nm the absorption coefficient of water is 10 times smaller than that of protein. This gives a natural contrast for biological samples.

Today, the most common x-ray source is synchrotron radiation which uses a bending magnet or a undurlater at a synchrotron that emits broadband radiation in the soft x-ray regime with great brightness. However, to get a tabletop soft x-ray microscope one needs a much smaller machine. A laser produced plasma source is compact, has high brightness and high spatial stability. However, most targets used for the source are solids, which create a big problem dealing with debris. The majority of the debris is hot ions and larger

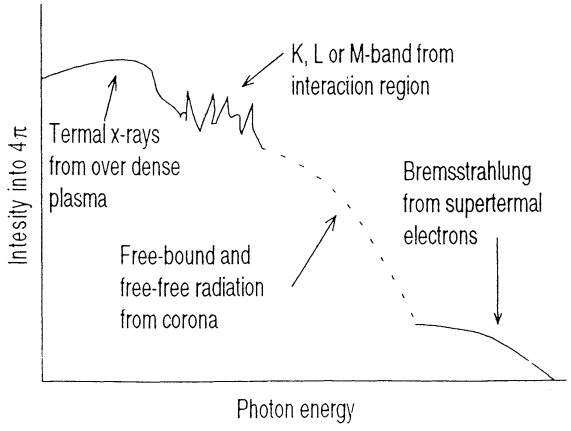


Figure 1.2 Typical x-ray spectrum from a plasma disk reaching from 100 eV to 100 keV. Different energies emit from different zones in the plasma. (Adapted from Ref. 2)

particles. These particles most certainly will damage any optical or x-ray component that is close to the plasma. There are ways of reducing the effect of the debris but it also often reduces the emitted x-ray intensity. A better way is to reduce the production of debris which can be done with a droplet target[4].

In this experiment a target of small ethanol droplets is used, so small that the focused laser beam will efficiently evaporate and ionise them. Because of the low debris production and that the source is in free space, all x-ray components may be placed much closer to the source and radiation is available in almost any direction in space. Two conceivable fields where improvements can be made are within high resolution imaging of wet biological objects and narrow-linewidth semiconductor processing.

1.5 Size measurement - small sources

To determine how efficient and useful the source is, one needs to know the extension of the x-ray emitting part of the plasma. In this case we used the knife edge method[5] due to its simplicity and very high resolution. Like the x-ray components the knife-edge may be positioned close to the source and it will not interfere with the laser beam. This gives a high resolution and the alignment is easier when small distances to the plasma are necessary. In the experiment the distance between plasma and knife-edge was 4 mm and between knife-edge and monochromator 180 mm resulting in a theoretical resolution of ~2 μm. The method is one dimensional but can be extended to two dimensions with tomographic techniques[6]. A disadvantage is that the measurement takes quite a long time.

Other methods available are the hole camera and the zone-plate coded imaging method[7]. The hole camera is of course two dimensional and fast but the resolution is poor if the geometrical arrangements are the same as other methods. It is hard to position the pinhole close enough to the plasma to get a good resolution. The zone-plate coded imaging method (ZPCI) is attractive in a few ways. The resolution can be up to ~8 μm and the method is two dimensional. The signal to noise ratio is excellent because of the zone-plate camera. Problems arise with the object distance. The zones-plates are often destroyed near the object. Another disadvantage is difficulties in the absolute calibration of the data reconstruction.

2 Theory

The scanning knife edge method will be described in detail for measuring the size of the source. Other methods will be mentioned in comparison to the first.

2.1 Scanning knife edge method

The most common applications of the knife edge method is for measurements of Gaussian laser beam. This application is important in any experiment where the focal point intensity of a laser beam needs to be determined.

The basics of the theory[8] is the Fresnel diffraction because we are dealing with a near-field region, which cancels some approximations within the Fraunhofer diffraction. Therefore we must go back to the Huygens-Fresnel principle. The Huygens's principle is:
Every point on a primary wave front serves as the source of spherical secondary wavelets, such that the primary wavefront at some late time is the envelope of these wavelets. Moreover, the wavelets advance with a speed and frequency equal to those of the primary wave at each point in space.
Fresnel continues and says:
The amplitude of the optical field at any point beyond is the superposition of all these wavelets if we consider both amplitude and relative phase.
With this in consideration we shall derive the Fresnel integrals and use these first on a rectangular aperture and then on a semi-infinite opaque screen, which is the knife-edge. Observe Fig. 2.1 where S is the monochromatic point source and dS is an area element at some arbitrary point A. The contribution to the optical disturbance at P

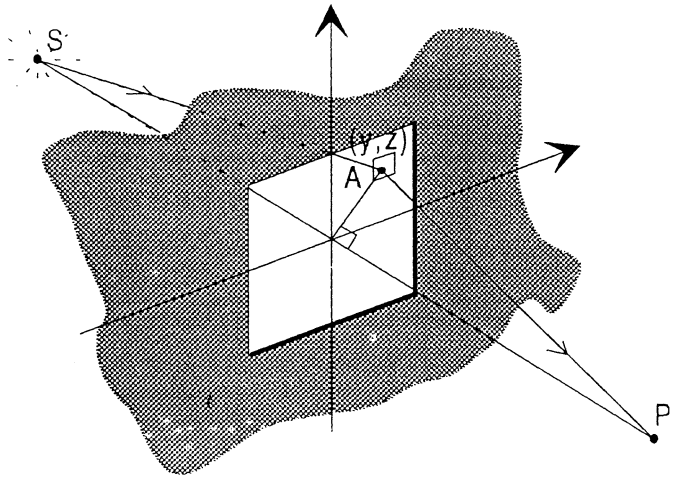


Figure 2.1 A monochromatic source and a picture plane. In between there is a rectangular aperture with some arbitrary point A.

from the secondary sources on dS has the form given by

$$dE_p = \frac{K(\theta)\varepsilon_0}{\rho r \lambda}\cos[k(\rho + r) - \omega t]dS, \qquad (2.1)$$

where ρ is distance between the source and the aperture, r is the distance between the aperture and picture plane, and λ is the wavelength.

The obliquity factor, $K(\theta)$, is set to one. Let $1/\rho r = 1/\rho_o r_o$, because the dimensions of the aperture are small in comparison to distances SO and OP. Doing a more sensitive approximation than that used in the Fraunhofer, analysis the disturbance at P in the complex representation is

$$E_p = \frac{\varepsilon_0 e^{-i\omega t}}{\rho_0 r_0 \lambda}\int_{y_1}^{y_2}\int_{z_1}^{z_2} e^{ik(\rho+r)}dydz. \qquad (2.2)$$

We now introduce the dimensionless variables u and v defined by

$$u \equiv y\sqrt{\frac{2(\rho_0 + r_0)}{\lambda\rho_0 r_0}}, \qquad v \equiv z\sqrt{\frac{2(\rho_0 + r_0)}{\lambda\rho_0 r_0}}. \qquad (2.3)$$

Utilising the new variables, we arrive at

$$E_p = \frac{\varepsilon_0}{2(\rho_0 + r_0)}e^{i[k(\rho_0+r_0)-\omega t]}\int_{u_1}^{u_2} e^{i\pi u^2/2}du\int_{v_1}^{v_2} e^{i\pi v^2/2}dv. \qquad (2.4)$$

The integral can be evaluated using the Fresnel integrals, which are defined as

$$\Phi(w) \equiv \int_0^w \cos(\pi w'^2/2)dw', \qquad \Psi(w) \equiv \int_0^w \sin(\pi w'^2/2)dw', \qquad (2.5)$$

where w can either u or v. The term in front of the integral represents the unobstructed disturbance at P divided by two, $E_u/2$. If we change the exponential function to its form of *cos* and *sin*, the Fresnel integrals can be used, and be entered in Eq. (2.4). The disturbance at P is then written as

$$E_p = \frac{E_u}{2}[\Phi(u) + i\Psi(u)]_{u_1}^{u_2}[\Phi(v) + i\Psi(v)]_{v_1}^{v_2} \qquad (2.6)$$

An easier way to demonstrate the function within physics to calculate the irradiance, which is

$$I_p = \frac{I_0}{4}\left\{\left[\Phi(u_2)-\Phi(u_1)\right]^2+\left[\Psi(u_2)-\Psi(u_1)\right]^2\right\} \times\left\{\left[\Phi(v_2)-\Phi(v_1)\right]^2+\left[\Psi(v_2)-\Psi(v_1)\right]^2\right\} \tag{2.7}$$

It is now very simple to change the rectangular aperture into a semi infinite opaque screen. We just set $z_2 = y_1 = y_2 = \infty$ and the irradiance becomes

$$I_p = \frac{I_0}{2}\left\{\left[\tfrac{1}{2}-\Phi(v_1)\right]^2+\left[\tfrac{1}{2}-\Psi(v_1)\right]^2\right\} \tag{2.8}$$

This is also the irradiance distribution of a knife edge with a point source theoretically. In fact neither source or detector is small enough to be represented as a point but they may be simulated as many points in a row. The slit of the detector has a certain extension. This slit width is not a serious deviation because the irradiance distribution is often much larger than the slit. More troublesome is the source extension as it turns out to be a incoherent light source instead of a monochromatic one. Incoherence makes the fringes of the irradiation distribution to smooth out and the falloff becomes more gradual, as seen in Fig. 2.2.

To calculate some kind of resolution of the method we shall consider the two-point resolution. The comparison criterion rests on the ability of the respective systems to resolve two closely spaced point sources. This has been done by summarising two irradiance distributions with different displacements. In Fig. 2.3 a characteristic extra knee can be seen and if we differentiate the function, two peaks evolve. These two peaks correspond to the two sources. As we move the two sources closer to each other, the knee will gradually move to the left and

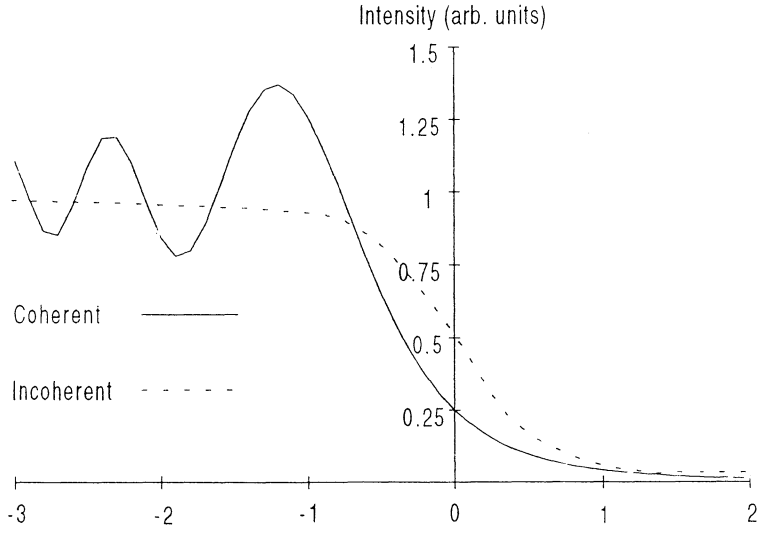


Figure 2.2 Theoretical intensity responses of coherent and incoherent systems to a knife-edge object.

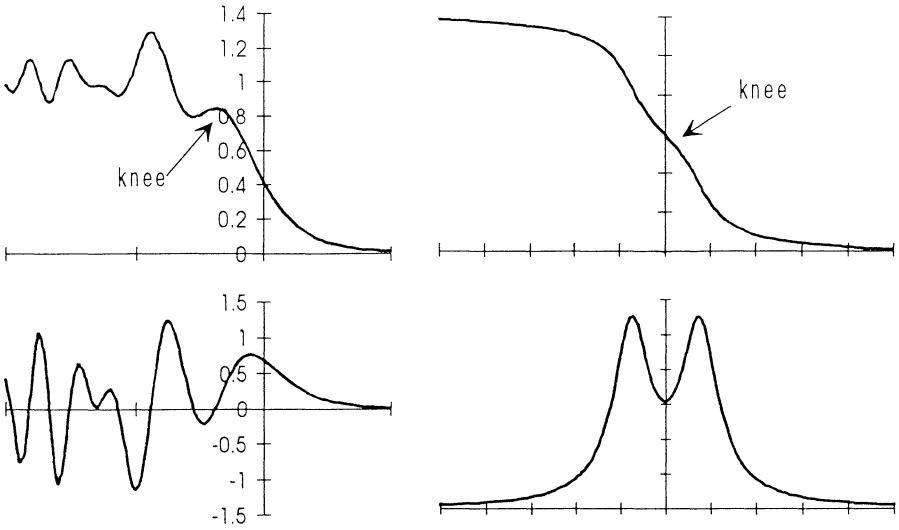


Figure 2.3 Two summarised irradiance distributions with different displacements and the differentiated function. Coherent and incoherent light.

become smaller. The peaks will then be lower too. When one can't detect the knee nor the peaks, the limit of the resolution is reached. This has been done for a numerous geometrical arrangements. For our experimental arrangement (cf. Sect. 4.1), where the knife-edge to plasma distance was 4 mm, the resolution was theoretically determined to ~2 μm.

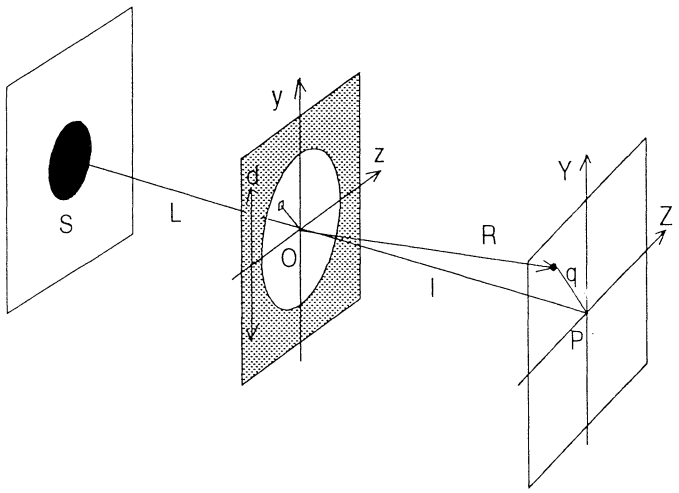


Figure 2.4 The hole camera imaging system. A monochromatic source and a picture plane. Between there is a pinhole aperture. Distance L is much longer than distance l.

2.2 Other methods

Hole camera method

This very simple imaging system builds its theory on Fraunhofer diffraction. Figure 2.4 shows the geometry of the system. Under Fraunhofer conditions the hole diameter, *d*, is much smaller than the distances SO and OP. This means that the optical waves are plane with good approximation when they arrive to the hole. The irradiance distribution is described as

$$I = \frac{\varepsilon_A^2 A^2}{2R^2}\left[\frac{2J_1(kdq/R)}{kdq/R}\right]^2, \tag{2.9}$$

where $J_1(kdq/R)$ is a Bessel function and A is the aperture area. R is assumed to be essentially constant over the pattern. Because of the axial symmetry, the towering central maximum corresponds to a high irradiance circular spot known as the *Airy disk*. With good approximation one can get an optimal hole diameter by assuming that *L* is much larger than *l*. The diameter of the spot is

$$d = \sqrt{\frac{2.44\lambda Ll}{L+l}}. \tag{2.10}$$

We can use the two-point resolution here too, or the so-called *Rayleigh criterion* of resolution. Using this, the minimum resolvable separation of the image is

$$\delta = 1.22\frac{\lambda l}{d}. \tag{2.11}$$

This means that two incoherent point sources are just resolved if the centre of one Airy disk is separated from the other by the distance to

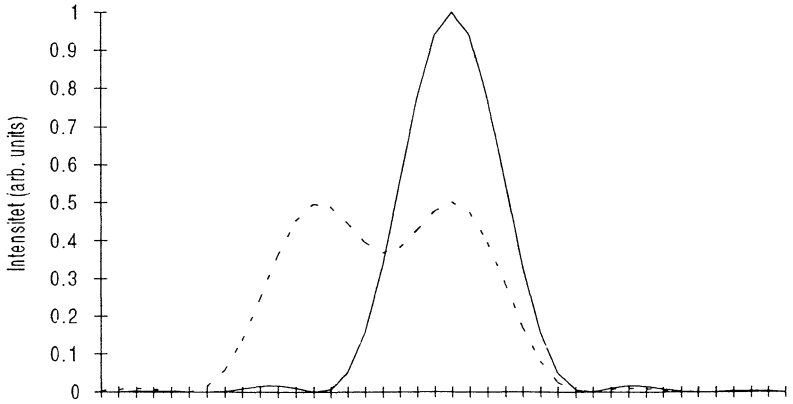


Figure 2.5 The solid line shows the intensity distribution of a pin hole. The dashed line show the Rayleigh criterion of resolution.

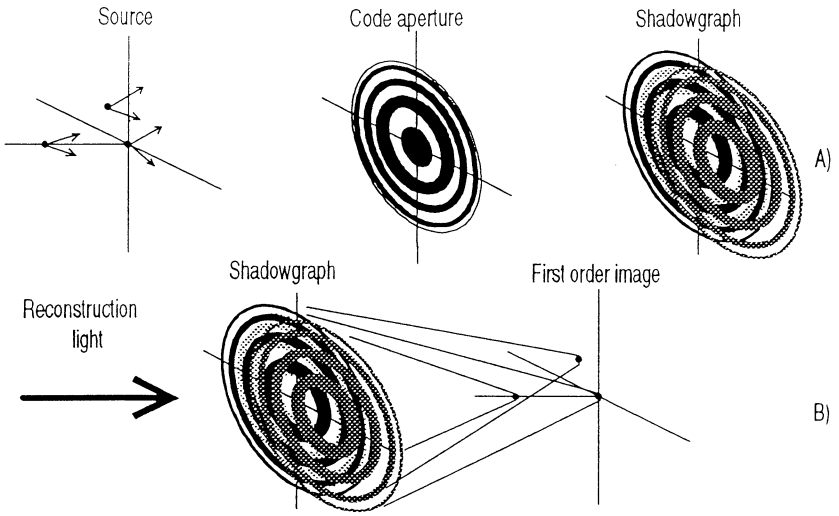


Figure 2.6 The principles of Fresnel zone-plate coded imaging.
A) Each zone plate shadow uniquely characterises the position of its associated source point.
B) Original three dimensional source distribution is reconstructed from the shadowgraph.

the first zero of the Airy disk. See Fig. 2.5.

The advantages of this method is that we get a 2-D image directly and that there is no chromatic aberration. However, the resolution is poor and different wavelengths are hard to separate. Even if the hole is positioned very close to the source (e.g. 5 mm), resolution better than ~6 μm can't be obtained for waterwindow wavelengths and ten times magnification.

Zone-plate coded imaging (ZPCI)

The method uses a shadow camera with a Fresnel zone-plate coded aperture[7]. As shown in Fig. 2.6, shadow graphs are first recorded photographically. The source pattern is then reconstructed optically. Some methods have a package of multi-layer filter film to enable many different spectra. The wavelength must be sufficiently short so that a ray optics approach is satisfied and sufficiently long for appreciable attenuation in the "opaque" zones of the coded aperture. ZPCI has a much better radiation collection efficiency compared to the hole camera of the same resolution, resulting in a better signal to noise ratio (S/N). Although the ZPCI imaging technique as such deteriorate the S/N, for small sources it is still better due to higher light collection efficiency. A resolution of ~8 μm has been recorded. A great disadvantage is the damage to the fragile zone-plates due to the close distance to the source. The two-step process and the variability in the reconstruction process of zone-plate images make it very difficult to calibrate in absolute numbers.

2.3 Conclusions

A hole camera gives a 2-D image directly but since it is impossible to get close to the source the resolution is limited to ~6 μm, which is an optimistic calculation.

The ZPCI method also gives a 2-D image directly and has a better radiation collection efficiency but sometimes suffers from chromatic aberration. This is unfavourable when the source has many spectral lines. The Fresnel zone plate is very fragile and can't be close to the source. Absolute calibration of the data is difficult but the resolution can be up to 8 μm.

The knife-edge method is slow and 1-D. 2-D imaging is possible using tomographic techniques[6]. However, the method gets you close to the source and has a high resolution. The equipment is also small and compact. For our measurements these criterion are the most important.

3 Computer controlled positioning of knife edge

There are three important parts dealing with the source measurement. First we want to position the knife edge accurately and then we want to make the data acquisition. A computer program, the third part, will control this. We will put some demands on positioning of the knife edge to guarantee the best result of the measurements. The positioning resolution should be 0.1 - 0.2 μm and the total range where the knife edge movement should be a few centimetres. Long movements are therefor important too. The knife edge positioning was implemented with a computer controlled DC-motor. Section 3.1 describes the hardware and Sect. 3.2 outlines the software. In Sect. 3.3 an description of the measurements and operation modes is made.

3.1 Hardware

There are of course many ways to implementate this construction. We have chosen five essential components. As shown in Fig. 3.1, for the motor controlled part they are: a PC, a PC multi-lab card, a home-made electronic unit, a DC motor. For data acquisition a boxcar integrator with a detector which is displayed in Fig. 3.2.

The PC can be of any kind with a 386 processor or better. It is important that it has an extra serial port and that there are slots for additional cards.

The PC multi-lab card (PCL-812) is a multifunctional analogue and digital I/O card with five different measurements: A/D conversion, D/A conversion, digital out- and input and counting/timing. For our purpose we needed an analogue output to control the speed and voltage of the motor and a digital out- and input connection to count distance pulses from the motor and control the motor direction. Of course, the card is too powerful but it gives the opportunity to add a few control function in the future, for example an additional motor or a more complex control.

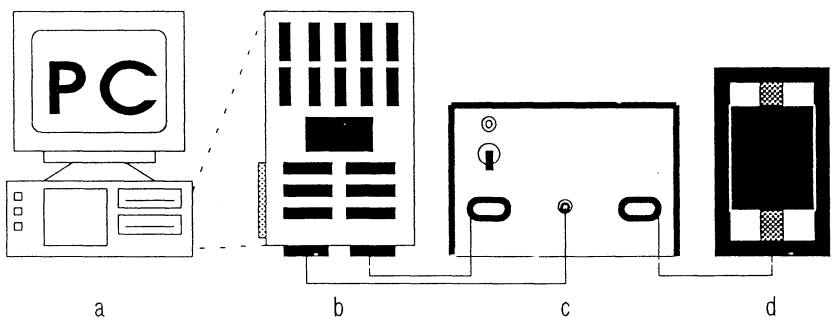


Figure 3.1 The PC to motor control line. a) The PC (386). b) The PC multi-lab card inside the PC. c) The motor control unit with amplifier. d) The Encoder Mike DC- motor.

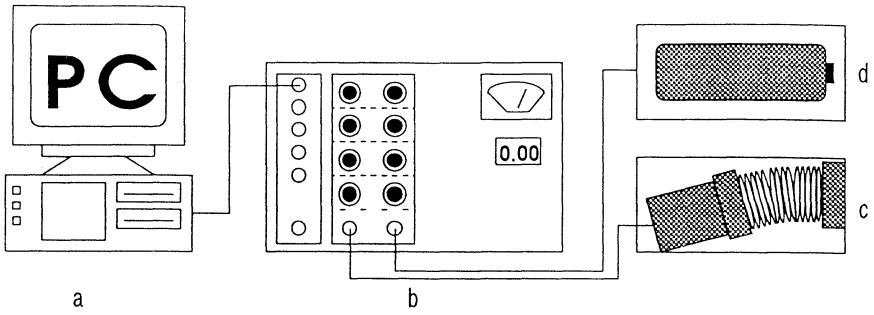


Figure 3.2 The data acquisition control line. a) The PC (386). b) The boxcar integrator and computer interface. c) The detector with x-ray monochromater. d) The Nd:YAG laser with trigger.

The DC motor is a miniature motorised translator called Encoder Mike™ (Oriel). It has a travel length of 25 mm and a <0.1 μm resolution just as required. An optical shaft encoder provides information with a resolution of 0.1 μm. Two optical sensors each provide ten signal pulses per micron of spindle travel. The motor may be operated at pressures around 10^{-6} torr which is important for our experiments.

The home-made electronic unit between the PC and the motor has been built because the PC can't support the motor with enough power. The analogue signal from the PC is amplified from the range 0-5 V to 0-12 V with the DC motor control unit. To switch the direction of the motor a relay is used which is controlled by the direction signal. The signal which indicates each 0.1 μm step is a short pulse. It is sent through a Schmith-trigger to sharpen it and make easier for the PC to read it. For details see Appendix D.

The data acquisition requires a boxcar integrator (Stanford Research System) with a computer interface module (SRS 245). Signals are sent by an x-ray detector to the boxcar integrator which treats the signals and sends them over to the interface. The integrator is externally trigged at the same rate as the laser, that is ~10 Hz. Signals are often averaged from 3 up to 30 signals. The analogue input to the interface can then be read by the PC via a serial port. To communicate with the SR245 we use a high level language with ASCII characters. The most useful feature is its ability to independently read the input ports and store a series of data points just by sending a command. Once a scan has been completed you may read the stored values with another command. Commands and data are sent between the interface and the PC serial port with a rate of 19.2 Kbaud and everything is easily guided by a computer program.

3.2 Software

The computer program is written in Borland C++ for two reasons. First of all C++ is a object oriented language and secondly because C++ is the most common language in the industry at present time. C++ is a development from the C language, in that way a C program may work in a C++ environment. The different objects in C++ are

called classes and they work as independent programs. The difference and the big advantage is in the program structure. In C or Pascal you write the program from top to bottom but in C++ you write different classes. In a class you describe the object, what it can do and how it communicates with other classes. Another big advantage is that you may use the same name for a function or a variable in different classes without any danger of name conflicts.

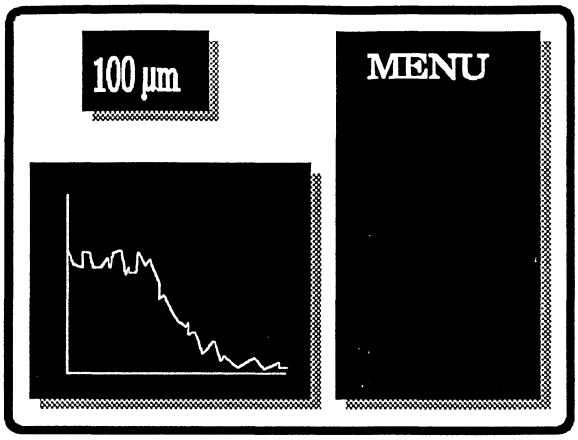


Figure 3.3 The screen with menu, position- and diagram-window.

When I started writing the software it was too much work making it in a Window environment compared to today's version. That is why it is written in DOS but I tried to make the screen similar to a window system as seen in figure 3.3.

The program contains three classes: Menu, Motor and COMPort. Both Motor and COMPort may work separately but the Menu class is dependent of the other two. Here follows a short description of each class. For details see Appendix A.

COMPort is basically a control program for a serial port. With it we can of course open and close a certain port and set parameters like baud rate, parity and stop bits. The main functions are writing and reading ASCII characters through the port and both single characters and strings are allowed. It is convenient to check if the communication works with another program (e.g. Terminal) before using COMPort.

Motor is created to control the DC motor by using the Multi-Lab Card (PCL-812). There are a few different ways making the motor go back and forth and the easiest way is just to press a button. By doing so you send a signal to rise the motor voltage as long as the button is pressed. This method has no precision what so ever. For that you need a scale to show where the motor is. There is a relative scale where you set the zero by yourself, but an absolute scale is not functional because there is no good absolute zero. You can reset the scale whenever you want. For a more precise movement you can either write the position you want to go to or just write a distance. Naturally it works both forward and backward. The signals sent to the motor are formed as pulses where you can alter voltage and pulse length. This is especially used when the motor move its shortest step. Both voltage and pulse length is increased slowly until the program detects a movement via the encoder signals.

Menu is the class that unifies everything. It opens a motor object and uses its facilities. Most important however, Menu describes the border between PC and user, the appearance of the screen, the interaction with the operator, how data should be presented ect. There are three different windows; menu window, position window and a diagram window. The purpose of the program, the different modes of operation, are implemented here and will be described in the next section.

3.3 How to measure

There are four modes of operation that deals with the motor movement and one that makes a complete measurement. The first four are essentially what the motor can do. Move it by pressing a button and change the direction with another. Type a position in microns where you want the motor to move to or type the distance and then decide in what direction. Finally there is a reset operation of the scale. However, most important is the automatic scanning operation when you want a real measurement. Choosing this you will be asked to type a few parameters that describes the scan. First, the full length of the scan and then how long each step between the data acquisitions should be. The step can be from 1 to 1000 microns or you can choose *'shortest step'* which is 0.1-0.3 microns. There is a scale factor to assure that the graph won't reach outside the diagram (maximum intensity) and finally you may want to make an average of many data acquisitions.

The loop procedure starts with a motor movement, defined by the operator, and a data acquisition from the x-ray detector. Both motor position and intensity data are stored in two separate vectors and they are then displayed in the position window and in the diagram. A new motor movement starts and the loop continues until the end of the scan. The vectors may be stored on data files.

How correct are the motor movements? The answer to that is displayed in Fig. 3.4. The movements have been checked with a Heidenhain MT 12B micrometer which has a resolution of 50 nm. The worst results were made with many 'shortest step' operations in a row as shown in (b). This is because the error originate from starts and stops where a encoder pulse from the motor may be miscounted, i.e. bit errors. Never the less the error is <1%. As seen in Fig. 3.4 (c-d) the results are much better for longer distances with few stops. In Appendix A the exact routines for movements are explained.

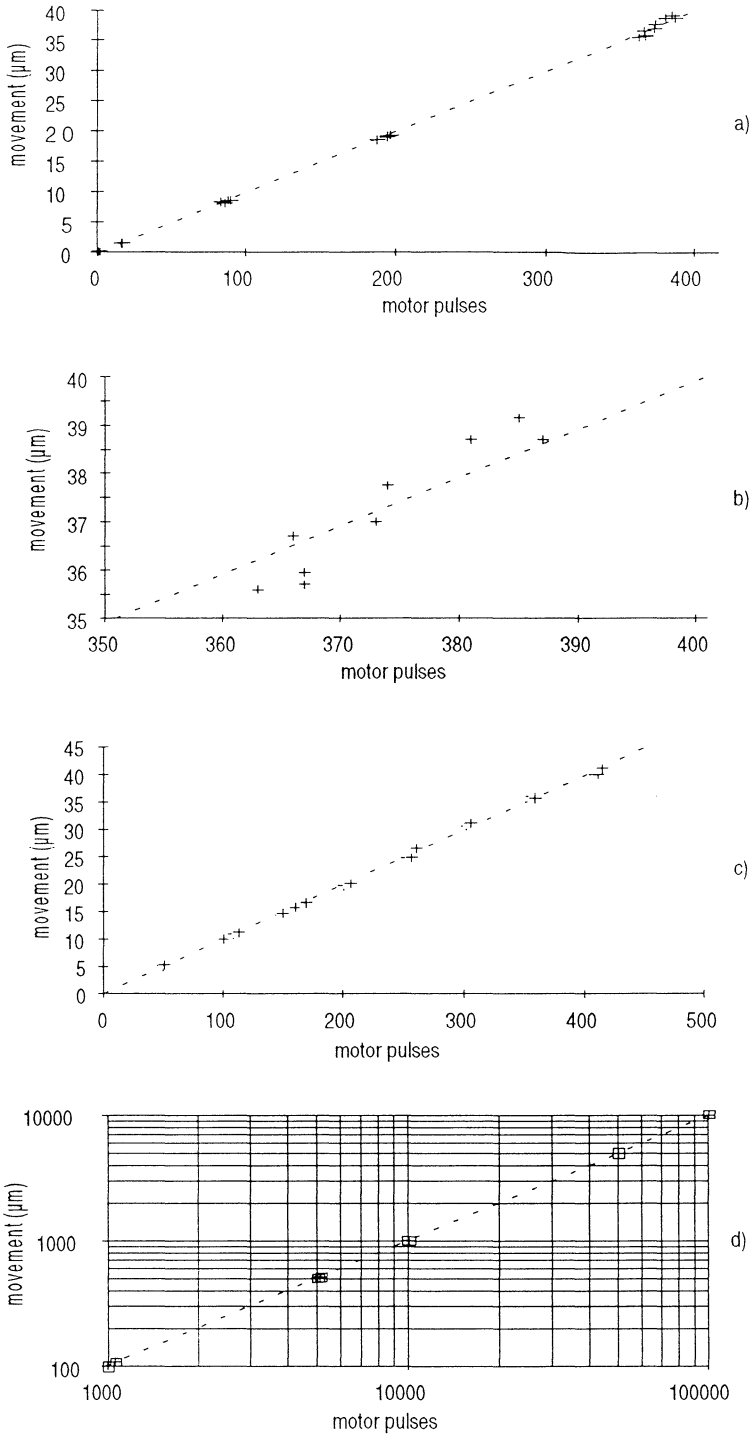


Figure 3.4 Diagrams of motor pulses (one pulse each 0.1 µm) versus the actual movement of the motor in microns. a) Different number of 'shortest step' from 1 to 200 steps. b) A section of the first diagram where all points are from 200 'shortest step'. c) Longer movements between 5 and 40 microns. d) Even longer movements up to one centimetre.

4 Experiments

4.1 Experimental arrangements for LPP

The experimental arrangements for the laser produced plasma soft x-ray droplet source[4] is shown in Fig. 4.1. Ethanol droplets come from a 10 μm diameter vibrating capillary ink jet printing nozzle with a frequency of ~1 MHz. The speed of the droplets is ~50 m/s and the separated distance is ~50 μm. The laser is a frequency-doubled active/active/passive modelocked 10 Hz Nd:YAG laser which produces ~140 ps pulses with 70 mJ/pulse at λ=532 nm. With a 50 mm lens the intensity in the focus reaches ~$4 \cdot 10^{14}$ W/cm^2. The x-ray source is spectrally characterised with a grazing incidence monochromator (Minuteman Lab. Inc.) at 90 degrees angle to the incident laser beam. Figure 4.2 shows a spectrum of a droplet source. The vacuum chamber operates at ~10^{-4} mbar pressure and is continuously pumped because a part of the ethanol droplet evaporates before entering the liquid nitrogen trap.

The knife edge method was used to determine the size of the plasma source with the DC motor assembled as displayed in Fig. 4.3. On the motor an arm and a razor blade was mounted so that they did not interfere with the laser beam. The razor blade was levelled with the 20 μm slit on the monochromator. A second 1 mm slit was positioned perpendicular and just in front of the entrance slit, to maximise

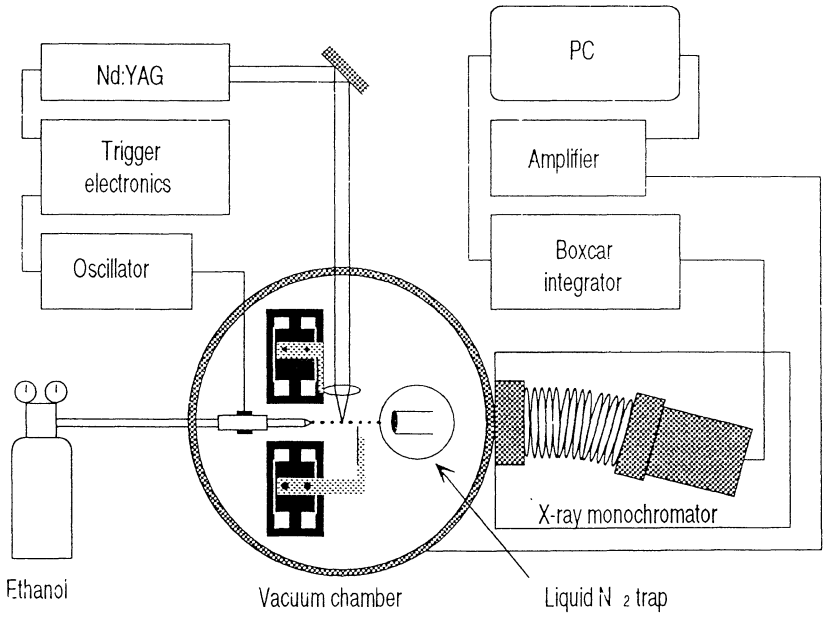


Figure 4.1 The experimental arrangements for the LPP soft x-ray droplet source. There are two motors inside the chamber. One that orientates the laser focus and one to move the knife-edge.

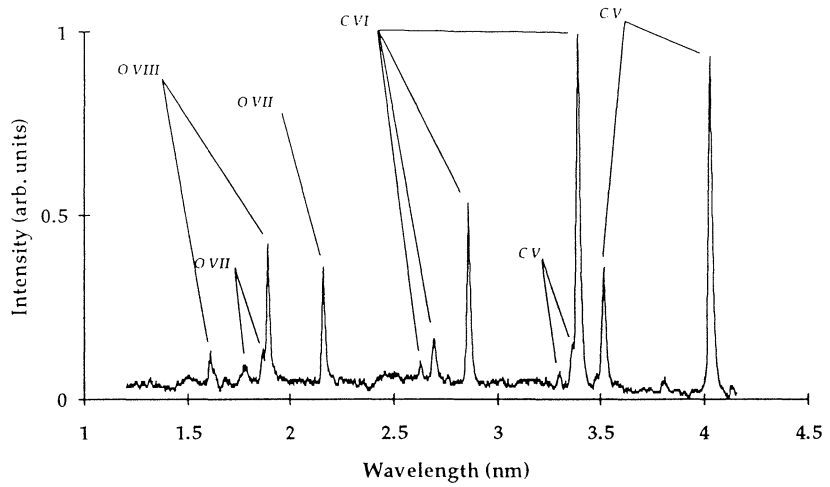


Figure 4.2 The spectrum of the droplet source within the soft x-ray region. (from Ref. 4)

resolution. Should the knife edge have an angle compared to the entrance slit, the resolution would deteriorate. The distance between the plasma source and the knife edge was measured to be 4 mm and 180 mm between knife edge and slit. Detection of x-ray was made with the monochromator adjusted to x-ray flux at 3.374 nm (C VI: $^2P_{1/2}$, 1s-2p). To optimise the intensity on this line we had to synchronise a few parameters to get a stable source. Another motor was used to move the laser focus so that the plasma stays within focus. The nozzle must be in perfect a lineament of the monochromator. The size of the x-ray plasma source could now be measured.

The software program was started and the source was found by moving the motor until the intensity dropped on the box-car integrator. Numerous scans were made with different resolutions and the results of the experiments are shown in the next section.

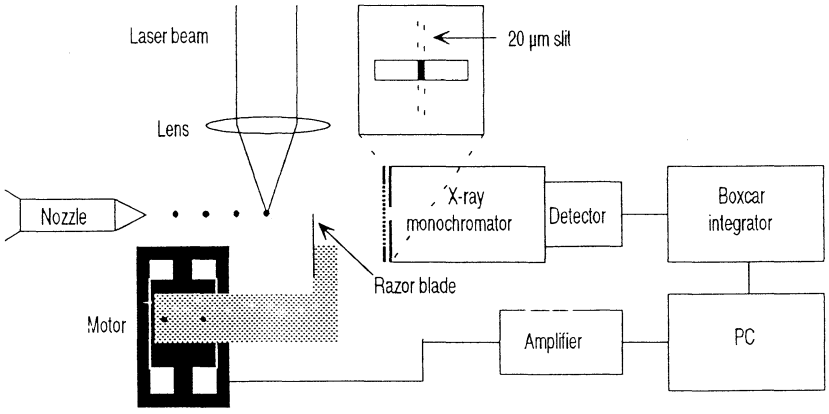


Figure 4.3 The special arrangements for the knife-edge positioning.

4.2 Results

The first series of measurements were made without the 1 mm horizontal slit which results in lower resolution but gives a stronger signal. The distance between the nozzle and the source was 6 mm. Both the boxcar and the program was set to average ten shots per data point and each knife edge movement was 10 µm. In this way the effects of fluctuation of the source were lowered. In Fig. 4.4 (a), three measurements have been averaged together to get an even smoother graph. A fast way to calculate the diameter of the plasma is to take the 50% range with the steepest slope. This gives the size of the central core plasma. The size was ~24 µm.

The last series were made with the 1 mm slit and the distance between the nozzle and the source was 12.5 mm. Each step was set to be 2 µm and each data point consisted of 15 averaged shots. Tree graphs have been averaged together but because of the higher resolution the fluctuations are greater, as shown in Fig. 4.4 (b). At the end of the graph there is a 'tail' due to excitation of a gas surrounding the core. The central core plasma size was ~25 µm. A more accurate measurement of the x-ray core size require Abel inversion[4]. This results in slightly smaller diameters for the kernel which emits 50% of the photons.

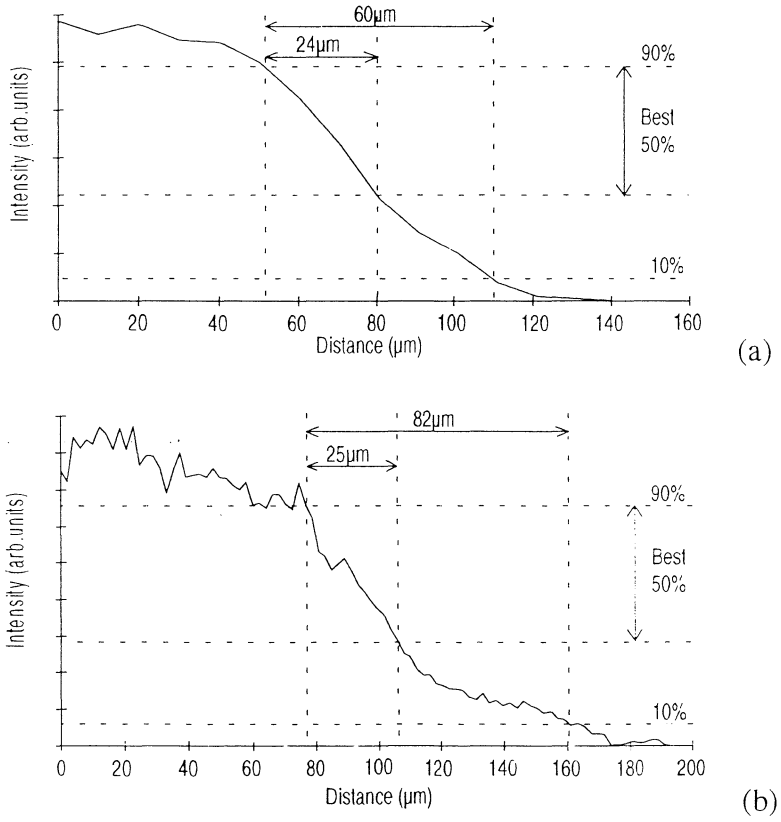


Figure 4.4 Knife edge measurements. The central core plasma size is taken as the 50 % intensity range where the slop is steepest.

5 Conclusion/Discussion

The knife-edge method was used to determine the size of a laser plasma soft x-ray droplet source. A series of measurements has been performed, resulting in a ~25 μm x-ray source size. The method has been shown to have a theoretical resolution of ~2 μm for the geometry in this case. The knife-edge was controlled by a computer via an amplifier. To assure a good mechanical resolution, up to 0.1 μm, of the motor a special software was created. The shortest possible movement of the motor was 0.1-0.3 μm. The software also took care of the data acquisition from an x-ray detector. A graph was displayed showing the movement of the motor versus the intensity from the monochromator.

The simplicity, high resolution and the ability to get very close to the source makes this method favourable. The hole camera yields lower resolution although it is fast and 2-D. The zone plate coded imaging method has a fairly good resolution but the zone plate is fragile and the extension in space does not fit into the experimental arrangements.

Further work will be performed to produce a tomographic image of the source. Using two different motors with a knife edge on each will do the job, and the software is easily extended to control two motors. A simpler and cheaper way is to use the same motor twice in different directions.

6 Acknowledgements

I am very grateful for all the help that I received from the people working at the Division of Atomic Physics, Lund Institute of Technology. Especially I would like to mention the following:

Hans Hertz, my instructor and inventor of the project, who supported and helped me all through the project. He had a special ability of detecting and giving possible solutions to all the problems that I was stuck into. The project has been very varied, extending over many technical fields, and in that way been very educating.

Lars Rymell, my co-instructor, who helped me install the knife edge method into the laser-plasma vacuum tank system and made the experiments possible.

Åke Bergquist, the electronic guru, who built the Motor Control Unit and helped me with connections between PC and motor.

Bertil Hermansson, the computer guru, who made the PC work at all times.

Lars Malmqvist, **Jörgen Larsson**, **Anders Persson** and **Roger Berg** who helped me with software problems.

Appendix A

A.1 Class Motor

The motor class controls the motor and make use of every feature the motor has. You may position the motor in many ways with different demands on accuracy. The positioning is done relative a scale with the same resolution as the pulses given by the motor. A description of all class variables and functions here follows.

Class variables

base is the base address on the PC-LabCard to which you add 0-15 depending what function you want.
volt is the voltage output on the analogue connector.
MaxVolt is the maximum voltage allowed.
direction can either be **forward** or **backward** and sets the direction of the motor.

Class functions

Motor is the constructor of the object. Here the class variables are defined and initialised. The direction and its address is set to forward.
In **SetVoltage** the voltage is set. Decimals are allowed but any input over maximum will be set to zero.
In **SetDirection** the direction is set. Input is either **forward** / **backward** or 1 / 0. A digital outport is then set high or low which will effect a relay in the amplifier box. Because of backlash the motor is moved a few microns without adding on the scale. The backlash is not constant all over the movement range so a more sophisticated routine where you check for pulses might be necessary.
ResetRelPos sets class variable **RelPos** to zero.
GetRelPos returns the value of the class variable **RelPos** in microns.
Pulse generates a voltage pulse with a specified amplitude and pulselength. When the motor moves it sends back short pulses which are detected and summarised. After the generated pulse the motor might move a bit further why a loop counts remaining position pulses. The function returns the sum of short motor pulses.
ShortestStep makes the motor move its absolutely shortest movement. In a loop voltage and pulselength are gradually increased. First the pulse length increases with a fixed voltage until a maximum pulse length or until a motor encoder pulse is detected. If maximum pulse length is reached the voltage is enhanced one step and so on until a motor pulse detection. Unfortunately the motor sometimes moves more than one pulse due to mechanical irregularities in the motor mechanism, resulting in movements of 0.1 - 0.3 µm. The amount of pulses are returned and are also added to the position variable **RelPos**.

MoveDistance tells the motor to move a specified distance in microns. Depending on how long the movement is, different pulses are used. Longer than 10 µm the motor runs continuously but the voltage is decreased gradually. When there are 10 microns left to the destination shorter pulses are used and the final approach, 0.6 µm, is made by **ShortestStep** to assure a correct movement independent of how long it is. The distance is then added or subtracted to **RelPos** depending on in what direction the movement was made.
GotoPos is just a variation of **MoveDistance**. Here you give the relative position you want to go to and the routine check if it is forward or backward.

A.2 Class COMPort

The class COMPort is basically an already existing class with very few adjustments. This class controls and uses all features of a serial port or a so called communication port. There are a lot of definitions which are all gathered in a header file called COMPORT.H where good explanations are written.

Class functions (private)

SetPort sets the port number to be used.
SetSpeed sets speed of communication, the baud rate.
SetOthers sets the rest of the communicating parameters like parity, bits and stopbit.
ReadCOM returns a character from the serial port.
InitCOMPort turns on communicative interrupts.

Class functions (public)

COMPort is the constructor where parameters port, baud rate, parity, wordlength and stopbit are entered. It is then initialised.
ReadString reads a string from the serial port.
WriteString outputs a string to the serial port.
WriteChar outputs a character to the serial port.
MakeScan sends operations to the boxcar integrator and receives a given amount of samples. The routine tells the box car integrator to collect a sample from the detector and to send the result. The samples are then averaged.
~COMPort is the destructor.

A.3 Class Menu

Class Menu is the class that unites class Motor and Scan. It is also the interface between the PC and the user where the screen and typing is controlled. To be able to write and draw on the same screen everything is done in graphic mode which results in a "quasi-window system". The difficult part is the entering of variables.

Class functions (private)

ActMenuWindow defines the boarders and colours of the menu window and activates it.
ActPosWindow defines the boarders and colours of the position window and activates it.
ActDiagramWindow defines the boarders and colours of the diagram window and activates it.
WriteMenu activates the menu window and writes the options of functions.
InitDiagram activates the diagram window and draws axis and labels of the diagram.
DrawInDiagram is given the former and the present point in the diagram and draws a line between those. The spacing is dependent of the scan length.

Class functions (public)

Menu is the constructor where the graphic mode is initiated. Here the fonts are chosen by a file path and name and the different windows are activated.
DoMenu activates the menu window and writes the options. It then asks you to type your choice and activates that choice.
WritePosition activates the position window and checks in the motor class for the position which is written.
ResetRelPos resets the position scale to zero.
MoveByButton clears the menu window and writes instructions. By pressing 'a' the motor moves and the direction is changed with 'd'. ESC returns you to the menu.
MoveTo clears the menu window and asks for a position to make the motor move to. The procedure uses the motor class functions and then writes the new position.
MoveDistance clears the menu window and asks for a distance and a direction to make the motor move. The procedure uses the motor class functions and then writes the new position.
Scan clears the menu window and initiates the communication port class object. It asks for parameters for the scan such as scan length, spacing between steps, intensity scale factor and the amount of averaging samples. Before the scan loop starts the diagram is initiated, the direction set to forward and the scale is reset. The loop stores motor position and intensity on two vectors and they are displayed in the diagram. The motor is moved to new position which is written in the position window. After the loop the procedure asks if a file is to be created. If yes, the vectors are stored on a data file, xxxx.dat, with a space between every element.

Appendix B

B.1 Header file 'knivmeny.h'

```
#include <dos.h>
#include <ctype.h>
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip.h>
#include <conio.h>
#include <string.h>
#include "COMPORT.H"
#include "COMPORT.CPP"
#include "MOTOR.CPP"

class Menu
{
  Motor TheMotor;
        void WriteMenu();
        void InitDiagram(float);
        void DrawInDiagram(float,float,float,float,float);
        void ActMenuWindow();
        void ActPosWindow();
        void ActDiagramWindow();

        public:

        Menu();
        int  DoMenu();
        void WritePosition();
        void ResetRelPos();
        int  MoveByButton();
        int MoveTo();
        int MoveDistance();
        int  Scan();
};
```

B.2 Menu file 'menu.cpp'

```
#include "KNIVMENY.H"

// Activate Menu window and write options
void Menu::WriteMenu()
{
        ActMenuWindow();
        outtextxy(2,2,"MENU");
        outtextxy(2,15,"1)  Manuel move of motor.");
        outtextxy(2,25,"2)  Move to a relativ position.");
        outtextxy(2,35,"3)  Move a distance.");
        outtextxy(2,45,"4)  Reset relativ position.");
        outtextxy(2,55,"5)  Scan.");
        outtextxy(2,65,"6)  Quit.");
};

/* Activate Diagram window and draw x-y-axis and lables */
void Menu::InitDiagram(float scanlength)
{
        char str[30];
        float x;
        float sl=scanlength;
        ActDiagramWindow();
        settextstyle(2,0,6);
```

```
        outtextxy(100,10,"Scan diagram");
        settextstyle(2,0,0);
        outtextxy(55,255,"Knife edge displacement (microns)");
        settextstyle(2,1,0);
        outtextxy(5,60,"Intensity (arb. units)");
        settextstyle(2,0,0);
        line(50,30,50,230);
        line(50,230,310,230);
        settextjustify(2,1);
        for (float i=0; i<=1; i=i+.25)   // y-axis
        {
                putpixel(49,230-200*i,15);
                gcvt(i,2,str);
                outtextxy(47,230-200*i,str);
        };
        settextjustify(1,2);
        for (float k=0; k<=sl; k=k+sl/10)    //x-axis
        {
                x=260*k/sl;
                putpixel(x+50,231,15);
                itoa(k,str,10);
                outtextxy(x+50,235,str);
        };
        settextjustify(0,2);
};

// Draw the graph in the diagram //
void Menu::DrawInDiagram(float oldx, float oldy, float newx, float newy,
                                                                        float scanlength)
{
        setviewport(0,150,340,460,1);
        setcolor(15);
        oldx=50+260*oldx/scanlength;
        newx=50+260*newx/scanlength;
        oldy=230-200*oldy;
        newy=230-200*newy;
        line (oldx,oldy,newx,newy);
        setcolor(14);
};

void Menu::ActMenuWindow()   // Activate the menu window //
{
        setviewport(350,50,630,460,1);
        setcolor(8);
        setfillstyle(1,8);
        bar(10,10,270,410);
        setfillstyle(1,1);
        bar(0,0,260,400);
        setcolor(14);
};

void Menu::ActPosWindow()   // Activate the position window //
{
        setviewport(50,50,300,140,1);
        setcolor(1);
        setfillstyle(1,8);
        bar(10,10,250,80);
        setfillstyle(1,1);
        bar(0,0,240,70);
        setcolor(14);
};

void Menu::ActDiagramWindow()  // Activate the diagram window //
{
        setviewport(0,150,340,460,1);
        setcolor(8);
        setfillstyle(1,8);
        bar(10,10,340,310);
        setfillstyle(1,9);
```

```
        bar(0,0,330,300);
        setcolor(14);
};

/* Constructor. Activate graphic mode. Draw the different windows.
  Write the menu. */
void Menu::Menu()
{
        int gdriver=DETECT, gmode, errorcode;
        initgraph(&gdriver, &gmode, "D:\BORLANDC\BGI\EGAVGA.BGI");
        errorcode=graphresult();
        if (errorcode!=grOk)
        {
                cprintf("Graphics error: %s\n", grapherrormsg(errorcode));
                cprintf("Press any key to halt:");
                getch();
                exit(1);
        };
        setbkcolor(0);
        settextstyle(2,0,0);
        setusercharsize(1,1,1,1);
        ActMenuWindow();
        WritePosition();
        ActDiagramWindow();
};

/* Activate and write the menu. Ask for choice and call it*/
int Menu::DoMenu()
{
        ActMenuWindow();
        WriteMenu();
        // Ask for alternativs and then do the selected alternativ //
        outtextxy(2,100,"Type your choise.");
        int choise;
        int incorrect=1;
        while (incorrect)
        {
                incorrect=0;
                choise=getch();
                switch (choise)
                {
                        case 49: MoveByButton();
                        break;
                        case 50: MoveTo();
                        break;
                        case 51: MoveDistance();
                        break;
                        case 52: ResetRelPos();
                        break;
                        case 53: Scan();
                        break;
                        case 54: return -1;
                        default: incorrect=1;
                };
        };
        return 0;
};

/* Activate position window write the position of the motor */
void Menu::WritePosition()
{
        float pos;
        char  str[20];
        ActPosWindow();
        settextstyle(2,0,5);
        outtextxy(20,10,"Relativ position (microns)");
        settextstyle(2,0,7);
        settextjustify(2,2);
        pos=TheMotor.GetRelPos();
```

```
        sprintf(str, "%4.1f",pos);
        outtextxy(150,30,str);
        settextstyle(2,0,0);
        settextjustify(0,2);
};

// Move motor by pressing 'a' or change direction and return to menu
int Menu::MoveByButton()
{
        ActMenuWindow();
        outtextxy(2, 2,"MANUEL MOVE OF MOTOR.");
        outtextxy(2,20,"Press <a> to move the motor.");
        outtextxy(2,35,"Press <d> to change direction..");
        outtextxy(2,50,"Press <Esc> to return to MENU.");
        int button;
        int forward=1, backward=0;
        int dir=forward;
        TheMotor.SetDirection(dir);
        button=getch();
        while (button!=ESC)
        {
                if (button=='a')
                {
                        TheMotor.SetVoltage(9);
                        delay(30);
                }else
                if (button=='d')
                {
                        dir=dir^1;
                        TheMotor.SetDirection(dir);
                        bar(2,70,200,90);
                        if (dir==forward) outtextxy(2,70,"Direction: forward   ");
                        if (dir==backward) outtextxy(2,70,"Direction: backward   ");
                }else return 0;
                while (kbhit());
                TheMotor.SetVoltage(0);
                button=getch();
        };
        delay(100);
        return 0;
};

// Move motor to a relative position, write the position
// and return to menu
int Menu::MoveTo()
{
        ActMenuWindow();
        outtextxy(2, 2,"MOVE TO A RELATIV POSITION.");
        outtextxy(2,20,"Type the position you want the");
        outtextxy(2,30,"motor to go to:");
        char    ch;
        char    str[2],string[5];
        int             i=0;
        int             negativ=FALSE;
        for (int k=0;k<5;++k) string[k]=0;
        outtextxy(2,50,"Your input:");
        while ( (ch=getch()) != CR )
        {
                if (isdigit(ch))
                {
                        itoa(ch-48,str,10);
                        strcat(string,str);
                        outtextxy(80+i*8,50,str);
                        ++i;
                } else if (ch=='-')
                {
                        outtextxy(80,50,"-");
                        ++i;
                        negativ=TRUE;
```

```
                } else      return 0;
        };
        float position=atof(string);
        if (position > 10000) return 0;
        if (negativ) position=-position;
        TheMotor.GotoPos(position);
        WritePosition();
        return  0;
};

// Move the motor a distance, write the new position
// and return to menu
int Menu::MoveDistance()
{
        char ch, str[2];
        int forward=1, backward=0;
        char string[5];
        for (int k=0;k<5;++k) string[k]=0;
        ActMenuWindow();
        outtextxy(2, 2,"MOVE A DISTANCE.");
        outtextxy(2,20,"Type the distance in microns you want the");
        outtextxy(2,30,"motor to move.");
        int i=0;
        outtextxy(2,50,"Your input:");
        while ( (ch=getch()) != CR )
        {
                if (isdigit(ch))
                {
                        itoa(ch-48,str,10);
                        strcat(string,str);
                        outtextxy(80+i*8,50,str);
                        ++i;
                }else return 0;
        };
        float distance=atof(string);
        outtextxy(2,70,"Choose forward or backward motion.");
        outtextxy(2,80,"<f>=forward, <b>=backward.");
        int choise=0;
        while ((choise!='f') && (choise!='b'))
        {
                choise=getch();
                if (choise=='f') TheMotor.SetDirection(forward);
                if (choise=='b') TheMotor.SetDirection(backward);
        };
        if (distance >10000) return 0;
        TheMotor.MoveDistance(distance);
        WritePosition();
        return 0;
};

// Reset the relativ position, write it and return to menu
void Menu::ResetRelPos()
{
        TheMotor.ResetRelPos();
        WritePosition();
};

/* Define the serial port. Ask for scan parameters. Control the motor
  and the boxcar integrator. Draw graph in the diagram. Create a file
  and write the file. */
int Menu::Scan()
{
        int             port    =COM1;    // Communication parameters //
        int             speed   =19200;
        int             parity  =NO_PARITY;
        int   bits  =8;
        int             stopbits=2;

        COMPort TheCOMPort(port,speed,parity,bits,stopbits);
```

```
        ctrlbrk(My_break);
        int                     nrofscan=0;
        float       motorvector[1000];    // Vectors for saving on file //
        float       scannervector[1000];
        float       newx, newy;           // Graph parameters //
        float       oldx, oldy;
        char        ch, str[2],string[5];
        for (int k=0;k<5;++k) string[k]=0;
        int                     i;

        ActMenuWindow();
// Ask for scan parameters //
        outtextxy(2, 2,"SCAN.");
        outtextxy(2,20,"Type the length of the scan in microns.");
        i=0;
        outtextxy(2,40,"Your input:");
        while ( (ch=getch()) != CR )
        {
                if (isdigit(ch))
                {
                        itoa(ch-48,str,10);
                        outtextxy(80+i*8,40,str);
                        strcat(string,str);
                        ++i;
                } else return 0;
        };
        float scanlength=atof(string);
        if (scanlength > 10000) return 0;
        outtextxy(2,70,"Type the length of each scan-step.");
        outtextxy(2,80,"If you want shortest possible step");
        outtextxy(2,90,"(0.1-0.2 microns) type <0>.");
        i=0;
        outtextxy(2,110,"Your input:");
        for (k=0;k<5;++k) string[k]=0;
        while ( (ch=getch()) != CR )
        {
                if (isdigit(ch))
                {
                        itoa(ch-48,str,10);
                        outtextxy(80+i*8,110,str);
                        strcat(string,str);
                        ++i;
                } else return 0;
        };
        float scanstep=atof(string);
        if (scanstep > 1000) return 0;

        outtextxy(2,130,"Type maximum intensity scalefactor.");
        i=0;
        outtextxy(2,150,"Your input:");
        for (k=0;k<5;++k) string[k]=0;
        while ( (ch=getch()) != CR )
        {
                if (isdigit(ch))
                {
                        itoa(ch-48,str,10);
                        outtextxy(80+i*8,150,str);
                        strcat(string,str);
                        ++i;
                } else return 0;
        };
        float intscale=atof(string);
        if (intscale > 10000) return 0;
        outtextxy(2,170,"Type amount of samples for average.");
        outtextxy(2,180," (Max 100)");
        i=0;
        outtextxy(2,200,"Your input:");
        for (k=0;k<5;++k) string[k]=0;
        while ( (ch=getch()) != CR )
```

```
        {
                if (isdigit(ch))
                {
                        itoa(ch-48,str,10);
                        outtextxy(80+i*8,200,str);
                        strcat(string,str);
                        ++i;
} else return 0;
        };
        int samples=atoi(string);
        if (samples > 100) return 0;
        InitDiagram(scanlength);
        TheMotor.SetDirection(1);
        ResetRelPos();
        while (TheMotor.GetRelPos() <= scanlength+1)  // Scan loop
        {
                ++nrofscan;
                sound(220);
                delay(20);
                nosound();
                scannervector[nrofscan] = TheCOMPort.MakeScan(samples);
                motorvector[nrofscan] = TheMotor.GetRelPos();
                if (nrofscan==1)
                {
                        oldy=scannervector[nrofscan]*intscale/10;
                        newy=oldy;
                        oldx=0;
                        newx=motorvector[nrofscan];
                } else {
                        oldy=scannervector[nrofscan-1]*intscale/10;
                        newy=scannervector[nrofscan]*intscale/10;
                        oldx=motorvector[nrofscan-1];
                        newx=motorvector[nrofscan];
                };
                DrawInDiagram(oldx,oldy,newx,newy,scanlength);
                if (scanstep==0) {TheMotor.ShortestStep();}
                else        TheMotor.MoveDistance(scanstep);
                WritePosition();
        };

// Create and write a file
        unsigned count=0;
        int handle;
        char filename[25], stri[1];
        int correct=0;
        FILE *datafile;
        ActMenuWindow();
        outtextxy(2, 2,"CREATE A FILE.");
        outtextxy(2,20,"Do you want to save the graph on");
        outtextxy(2,30,"a file? (y/n)");
        char choise=getch();
        if (choise!='n')
        {
                while (correct==0)
                {
                        outtextxy(2,50,"Type the filename.");
                        count=0;
                        filename[0]='\0';
                        while ( (ch=getch()) != 13 && count<8)
                        {
                                sprintf(stri,"%c",ch);
                                strcat(filename,stri);
                                outtextxy(120+count*8,50,stri);
                                ++count;
                        };
                        strcat(filename,".dat");
                        outtextxy(2,70,"The file is:");
                        outtextxy(100,70,filename);
```

```
                    outtextxy(2,90,"Is it correct? (y/n)");
                    int answer=getch();
                    if (answer=='y') {correct=1;}
                    else                                bar(2,50,200,110);
                };
                if ( (datafile=fopen(filename,"wt"))==NULL )
                {
                    printf("Cannot open output file.\n");
                    exit(0);
                };
                outtextxy(2,110,"The file is created.");
                delay(3000);

                for (i=1; i<=(nrofscan); ++i)
                    fprintf(datafile,"%.1f %.3f\n",motorvector[i],scannervector[i]);
                fclose(datafile);
                outtextxy(2,130,"The file is written.");
                delay(5000);
        };
        TheCOMPort.~COMPort();      // Close TheCOMPort
  return 0;
};

int main()
{
        Menu TheMenu;
        while (TheMenu.DoMenu() != -1);
        getch();
        closegraph();
        return (0);
}
```

B.3 Motor file 'motor.cpp

```
// The class Motor describes the motor and its uses.
#include <dos.h>
#include <stdio.h>

class Motor
{
                int     base;               // class variables
                float   RelPos;
                float   MaxVolt, volt;
                enum    dir {backward, forward} direction;
        public:
                        Motor();            // constructor
                int     Pulse(int, float);  // functions
                float   GetRelPos();
                void    ResetRelPos();
                void    MoveDistance(float);
                void    GotoPos(float);
                void    SetVoltage(float);
                void    SetDirection(dir);
                int     ShortestStep();
};

// Constructor. Initialize class variables.
        Motor::Motor()
        {
                MaxVolt=10;
                volt=0;
                base=0x220;         // CN2 on LabCard.
                RelPos=0;
                direction=forward;
                outport(base+14, 0);    // Direction forward
        };

// Set the voltage of motor via Ana-port on LabCard
        void Motor::SetVoltage(float voltage)
```

```
{
        int  highbyte, lowbyte;
        if (voltage>10) voltage=0;        // 10 is max of voltage
        int voltref=voltage*287;
        highbyte=voltref/256;
        lowbyte=voltref%256;
        outport(base+4,lowbyte);      // Set Ana-port out on LabCard
        outport(base+5,highbyte);
};

// Set the direction and correct for backlash.
    void Motor::SetDirection(dir d)
    {
        if (direction!=d)
        {
            direction=d;
            if (direction==backward)
            {
                outport(base+14, 128);   // Set D/O-port
            }
            else
            {
                outport(base+14, 0);
            };
            MoveDistance(5);
            if (direction==forward)
            {
                RelPos=RelPos-100;      // Correction because of bachlash
            }
            else
            {
                RelPos=RelPos+100;
            };
        };
    };

    void Motor::ResetRelPos() { RelPos=0; };

// Returns the class variable RelPos, relative position in microns.
    float Motor::GetRelPos() { return RelPos/20; };

/* Generate a voltage pulse with amplitude <voltage> and <pulselength>.
    Read the position pulses from the motor and summarize them.
    Continue to read after the pulse untill the motor has stoped.
    Return the sum of positions.
                                                    */
    int Motor::Pulse(int pulselength, float voltage)
    {                    // a certain pulselength in ms.
        int  position;
        long highbyte;
        long stop;
        int  newread, oldread;
        volt=voltage;
        position=0;
        highbyte=pulselength/1.22222+32;
        outport(base+3, 0x20);      // Interrupt on terminal count, R/W MSB.
        outport(base, highbyte);    // Set counter.
        outport(base+3, 0);
        int i=1;
        oldread=inport(base+6);
        oldread=oldread&1;
        stop=0xFF00+highbyte;
        SetVoltage(volt);
        while (stop>0xff20)
        {
            newread=inport(base+6);  // Read position pulse from motor.
            newread=newread&1;       // Mask bit.
            if (newread!=oldread)
            {
```

```
                                position=position+1;  // Summarize.
                                oldread=newread;
                        };
                        outport(base+3, 0);
                        stop=inport(base);
                        if (i<2) {stop=0xff00+highbyte; position=0; ++i;};
                };
                SetVoltage(0);
                for (long k=0; k<50000; ++k)
                {
                        newread=inport(base+6);   // Read position pulse from motor.
                        newread=newread&1;
                        if (newread!=oldread)
                        {
                                position=position+1;
                                oldread=newread;
                        };
                };

                return position;
        };

/* Make the shortest possible step by altering the voltage pulse.
        Set the relative position and return the amount of position pulses.   */
        int Motor::ShortestStep()
        {
                float  MinVolt=6;
                int    MinPulse=60;
                float  voltInc;
                int    pulselength, pulseInc;
                int    step;
                volt=MinVolt;
                voltInc=1;
                pulselength=MinPulse;
                pulseInc=10;
                step=0;
                while ((step<2)&&(volt<=10))        // loop until the motor
                {                                                // has steped.
                        pulselength=pulselength+pulseInc;
                        if (pulselength>100)
                        {
                                pulselength=MinPulse;
                                volt=volt+voltInc;
                        };
                        step=step+Pulse(pulselength, volt);
                };
                if (direction==forward)     // Set RelPos.
                {
                        RelPos=RelPos+step;
                }
                else
                {
                        RelPos=RelPos-step;
                };
                return step;
        };

/*      Move a distance in microns. Depending on how long distance alter voltage and
        pulses. When closing in on the right position slow down.
        Finaly use ShortestStep.
                */
        void Motor::MoveDistance(float distance)
        {
                float dist=distance*20;
                float pos=0;
                int newread, oldread;
                oldread=inport(base+6);
                oldread=oldread&1;
                while (pos < (dist-200) )   // Long pulses.
```

```
                {
                        if ( (pos >= (dist-200000)) && (pos < (dist-20000)) ) volt=6;
                        if ( (pos >= (dist-20000)) && (pos < (dist-2000)) ) volt=4;
                        if ( (pos >= (dist-2000)) && (pos < (dist-200)) ) volt=3;
                        SetVoltage(volt);
                        newread=inport(base+6);  // Read positionpulse from motor.
                        newread=newread&1;
                        if (newread!=oldread)
                        {
                                pos=pos+1;
                                oldread=newread;
                        };
                };
                SetVoltage(0);          // Stop.
                for (long k=0; k<100000; ++k)
                {
                        newread=inport(base+6);  // Read positionpulse from motor.
                        newread=newread&1;
                        if (newread!=oldread)
                        {
                                pos=pos+1;
                                oldread=newread;
                        };
                };
                while ( (pos >= (dist-200)) && (pos < (dist-12)) )
                {
                        if (volt != 6) volt=volt+1;   // Shorter pulses.
                        pos=pos+Pulse(120, volt);
                };
                if (direction==forward)     // Set RelPos.
                {
                        RelPos=RelPos+pos;
                }
                else
                {
                        RelPos=RelPos-pos;
                };
                while (pos < dist) pos=pos+ShortestStep();  // Final aproach.
        };

// Go to a specific position.
        void Motor::GotoPos(float position)
        {
                dir d;
                float pos=position*20;
                if (pos!=RelPos)
                {
                        float distance=pos-RelPos;
                        if (distance<0) {d=backward; distance=-distance;}
                                        else d=forward;
                        SetDirection(d);
                        distance=distance/20;
                        MoveDistance(distance);
    };
        };
```

B.4 COMPort header file 'comport.h'

```
/*--------------------------------------------------------------------*
  FILENAME:             COMPORT.H

                                        Some definitions used by COMPORT.CPP

 *--------------------------------------------------------------------*/

#define COM1        1
#define COM2        2
#define COM1BASE      0x3F8  /*  Base port address for COM1  */
```

```
#define COM2BASE      0x2F8  /* Base port address for COM2 */
/*
   The 8250 UART has 10 registers accessible through 7 port addresses.
   Here are their addresses relative to COM1BASE and COM2BASE. Note
   that the baud rate registers, (DLL) and (DLH) are active only when
   the Divisor-Latch Access-Bit (DLAB) is on. The (DLAB) is bit 7 of
   the (LCR).

          o TXR Output data to the serial port.
          o RXR Input data from the serial port.
          o LCR Initialize the serial port.
          o IER Controls interrupt generation.
          o IIR Identifies interrupts.
          o MCR Send contorl signals to the modem.
          o LSR Monitor the status of the serial port.
          o MSR Receive status of the modem.
          o DLL Low byte of baud rate divisor.
          o DHH High byte of baud rate divisor.
*/
#define TXR         0     /* Transmit register (WRITE) */
#define RXR         0     /* Receive register  (READ)  */
#define IER         1     /* Interrupt Enable          */
#define IIR         2     /* Interrupt ID              */
#define LCR         3     /* Line control              */
#define MCR         4     /* Modem control             */
#define LSR         5     /* Line Status               */
#define MSR         6     /* Modem Status              */
#define DLL         0     /* Divisor Latch Low         */
#define DLH         1     /* Divisor latch High        */

/*-------------------------------------------------------------------*
 Bit values held in the Line Control Register (LCR).
          bit                 meaning
          ---                 -------
          0-1                 00=5 bits, 01=6 bits, 10=7 bits, 11=8 bits.
          2                   Stop bits.
          3                   0=parity off, 1=parity on.
          4                   0=parity odd, 1=parity even.
          5                   Sticky parity.
          6                   Set break.
          7                   Toggle port addresses.
 *-------------------------------------------------------------------*/
#define NO_PARITY     0x00
#define EVEN_PARITY   0x18
#define ODD_PARITY    0x08

/*-------------------------------------------------------------------*
 Bit values held in the Line Status Register (LSR).
          bit                 meaning
          ---                 -------
          0                   Data ready.
          1                   Overrun error - Data register overwritten.
          2                   Parity error - bad transmission.
          3                   Framing error - No stop bit was found.
          4                   Break detect - End to transmission requested.
          5                   Transmitter holding register is empty.
          6                   Transmitter shift register is empty.
          7          Time out - off line.
 *-------------------------------------------------------------------*/
#define RCVRDY        0x01
#define OVRERR        0x02
#define PRTYERR       0x04
#define FRMERR        0x08
#define BRKERR        0x10
#define XMTRDY        0x20
#define XMTRSR        0x40
#define TIMEOUT             0x80

/*-------------------------------------------------------------------*
```

```
 Bit values held in the Modem Output Control Register (MCR).
          bit          meaning
          ---                    -------
          0                      Data Terminal Ready. Computer ready to go.
          1                      Request To Send. Computer wants to send data.
          2                      auxillary output #1.
          3                      auxillary output #2.(Note: This bit must be
                                 set to allow the communications card to send
                                 interrupts to the system)
          4                      UART ouput looped back as input.
          5-7                    not used.
 *-------------------------------------------------------------------*/
#define DTR         0x01
#define RTS         0x02
#define MC_INT                 0x08

/*-------------------------------------------------------------------*
 Bit values held in the Modem Input Status Register (MSR).
          bit                    meaning
          ---                    -------
          0                      delta Clear To Send.
          1                      delta Data Set Ready.
          2                      delta Ring Indicator.
          3                      delta Data Carrier Detect.
          4                      Clear To Send.
          5                      Data Set Ready.
          6                      Ring Indicator.
          7                      Data Carrier Detect.
 *-------------------------------------------------------------------*/
#define CTS         0x10
#define DSR         0x20

/*-------------------------------------------------------------------*
 Bit values held in the Interrupt Enable Register (IER).
          bit                    meaning
          ---                    -------
          0                      Interrupt when data received.
          1                      Interrupt when transmitter holding reg. empty.
          2                      Interrupt when data reception error.
          3                      Interrupt when change in modem status register.
          4-7                    Not used.
 *-------------------------------------------------------------------*/
#define RX_INT      0x01

/*-------------------------------------------------------------------*
 Bit values held in the Interrupt Identification Register (IIR).
          bit                    meaning
          ---                    -------
          0                      Interrupt pending
          1-2          Interrupt ID code
                                 00=Change in modem status register,
                                 01=Transmitter holding register empty,
                                 10=Data received,
                                 11=reception error, or break encountered.
          3-7                    Not used.
 *-------------------------------------------------------------------*/
#define RX_ID       0x04
#define RX_MASK     0x07

/*
   These are the port addresses of the 8259 Programmable Interrupt
   Controller (PIC).
*/
#define IMR         0x21  /* Interrupt Mask Register port */
#define ICR         0x20  /* Interrupt Control Port       */

/*
   An end of interrupt needs to be sent to the Control Port of
          the 8259 when a hardware interrupt ends.
```

```
*/
#define EOI         0x20  /* End Of Interrupt */

/*
        The (IMR) tells the (PIC) to service an interrupt only if it
        is not masked (FALSE).
*/
#define IRQ3        0xF7  /* COM2 */
#define IRQ4        0xEF  /* COM1 */

/*
  The (IMR) tells the (PIC) to service an interrupt only if it
  is not masked (FALSE).
*/
#define IRQ3        0xF7  /* COM2 */
#define IRQ4        0xEF  /* COM1 */

#include <bios.h>
#include <dos.h>
#include <string.h>

#define timeout  10000    /* Readln_com times out at 10000 milliseconds*/
#define max_buffer  1000  /* Circular buffer size */
#define near_full   900   /* When buffer_length exceeds near_full */
                  /* RTS line is disabled */
#define near_empty  100   /* When buffer drops below near_empty */
                                                        /* RTS line is enabled */
#define _CPPARGS ...      // It is a CPP program. //

#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "d:\borlandc\eget\comport.h"

#define VERSION 0x0101

#define FALSE       0
#define TRUE        (!FALSE)

#define CR                      13    // carriage return
#define ESC         0x1B   /* ASCII Escape character */
#define ASCII       0x007F /* Mask ASCII characters  */
```

B.5 COMPort file 'comport.cpp'

```
class COMPort
{
        int SetPort(int);
        int SetSpeed(int);
        int SetOthers(int,int,int);
        char ReadCOM();
        void InitCOMPort();
        public:
                COMPort(int,int,int,int,int);
                void ReadString(char*);
                void WriteString(char*);
                int WriteChar(char);
                float MakeScan(int);
                ~COMPort();
};

int     COM             =0;     // Port adress

void I_enable()     // Turn on communicatin interrupts //
{
        disable();
        outportb(COM+IER,RX_INT);
        enable();
};

void I_disable()
{           // Turn off communicatin interrupts //
        disable();
        outportb(COM+IER,0);
        enable();
};

int My_break()  // Control-Break interrupt handler //
{
        I_disable();
        printf("\nInterrupts disabled!!!\n");
        return (0);
};

/* Constructor. Set up the port */
COMPort::COMPort(int Port, int Speed, int Parity, int Bits, int StopBit)
{
        int flag = 0;

        if (SetPort(Port))
          flag = -1;
        if (SetSpeed(Speed))
          flag = -1;
        if (SetOthers(Parity, Bits, StopBit))
          flag = -1;

        if (flag!=-1) InitCOMPort();
};

void COMPort::InitCOMPort()  // Turn on communicatin interrupts //
{
        I_enable();
}

int COMPort::SetPort(int Port)     /* Set the port number to use */
{

  switch (Port)
        {
                case COM1 : COM=COM1BASE;
```

```
                                                    break;
                    case COM2 : COM=COM2BASE;
                                                    break;
                    default   : return (-1);
  }
        return (0);
}

/* This routine sets the speed; will accept funny baud rates. */
/* Setting the speed requires that the DLAB be set on.      */
int COMPort::SetSpeed(int Speed)
{
  char              c;
  int               divisor;

  if (Speed == 0)       /* Avoid divide by zero */
    return (-1);
  else
    divisor = (int) (115200L/Speed);
        if (COM == 0)
    return (-1);
disable();
  c = inportb(COM + LCR);
        outportb(COM + LCR, (c | 0x80));        /* Set DLAB */
        outportb(COM + DLL, (divisor & 0x00FF));
        outportb(COM + DLH, ((divisor >> 8) & 0x00FF));
        outportb(COM + LCR, c);                 /* Reset DLAB */
  enable();
return (0);
}

/* Set other communications parameters */
int COMPort::SetOthers(int Parity, int Bits, int StopBit)
{
  int           setting;

        if (COM == 0)
        return (-1);
  if (Bits < 5 || Bits > 8)                                     return (-1);
        if (StopBit != 1 && StopBit != 2)               return (-1);
  if (Parity != NO_PARITY && Parity != ODD_PARITY && Parity != EVEN_PARITY)
                                                                  return (-1);

  setting  = Bits-5;
  setting |= ((StopBit == 1) ? 0x00 : 0x04);
  setting |= Parity;

  disable();
  outportb(COM + LCR, setting);
  enable();

  return (0);
}

char COMPort::ReadCOM()       // Returns a character from serial port //
{
        while(!(inp(COM+LSR)&1));  // Wait till data is ready //
        return (inp(COM));
}

// Reads a string from serial port //
void COMPort::ReadString(char *StrPoint)
{
        outp(COM+LSR,inp(COM+LSR)&254);
        do
        {
                *StrPoint=ReadCOM();
        } while(CR!=*StrPoint++);
};
```

```
/* Output a character to the serial port */
int COMPort::WriteChar(char ch)
{
        long int timeout = 0x0000FFFFL;

  /* Wait for transmitter to clear */
        while ((inportb(COM + LSR) & XMTRDY) == 0)
    if (!(--timeout))
                                return (-1);
  disable();
  outportb(COM + TXR, ch);
  enable();
        return (0);
};

// Output a string to the serial port //
void COMPort::WriteString(char *StrPoint)
{
        while(*StrPoint)
        {
                WriteChar(*StrPoint);
                StrPoint++;
        };
};

COMPort::~COMPort()     // Destructor //
{
        I_disable();
};

/* Makes a scan with the boxcar integrator. Returns the average
of the samples  */
float COMPort::MakeScan(int samples)
{
        char*       string;
        float       data, average;
        float   datasum=0;

        for (int i=0; i<samples; ++i)
        {
                WriteString("SC1:1\r");
                WriteString("N\r");
                ReadString(string);
                data=atof(string);
                datasum=data+datasum;
//              delay(100);
//              printf("data %6.3f\n",data);
        };
        average=datasum/samples;
        return average;
};
```

Appendix C

In this appendix I will try to write down all the small problems that I came across while programming with Borland C++. Some problems might seem very trivial but they are not as long as you don't know the answer. Most mistakes are simple and only done once but are very time consuming. Therefore reading this might save you some time.

There are two programming modes, text and graphic mode. These two can't work at the same time. If you want to both write and draw on the same screen you'll have to work in graphic mode. Many useful text commands doesn't exist in graphic mode why you have to create them yourself. An example of this is when you want to input a number or text from the keyboard to a variable and the screen. It is shown in Menu::Scan().

To be able to use graphic mode you must include a graphic driver in the same directory as your program, e.g. EGAVGA.BGI. By default all text is displayed as bit-mapped fonts. To use stroked fonts you have to include a font file, e.g. LITT.CHR. This gives you a greater freedom of changing text styles.

The compiler needs to be told that you are using graphic mode. This is done in the compiling settings.

If you are not very familiar with C or C++ note that the compiler is not sensitive to different types of variables within the same expression. For example, 'number=var1·var2/var3'. If you mix int and float types the result is often not what you expected, it becomes sometimes char.

Appendix D

The pictures below show the essential electronic parts of the DC motor control unit. There is an amplifier which enhances the analogue motor signal from the PC. A relay is used to define the motor direction which is controlled by a TTL signal. Counter signals from the motor are sharpened by a Schmitt trigger. The power supply is not represented. The DC motor control unit was constructed by Åke Bergquist.

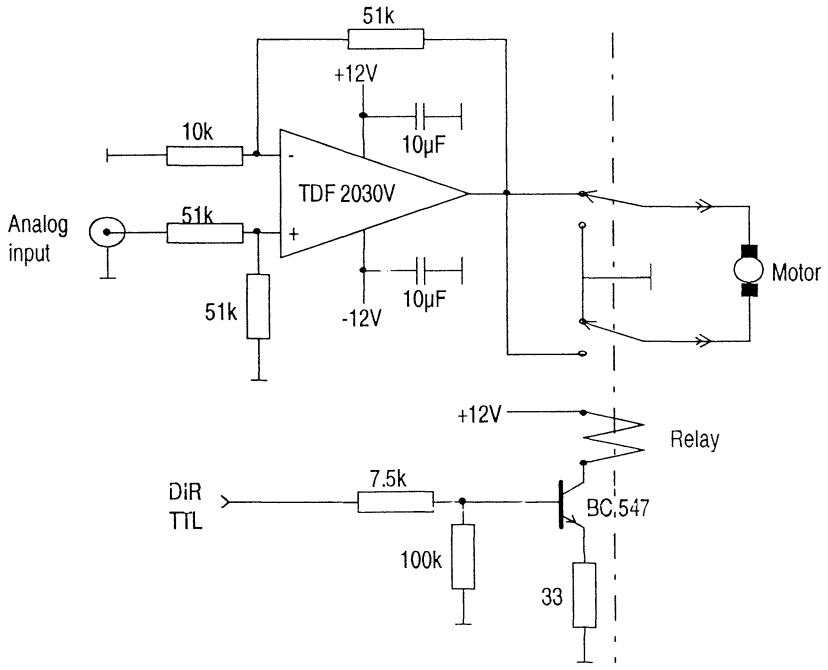
The tables show the connections to and from the unit.

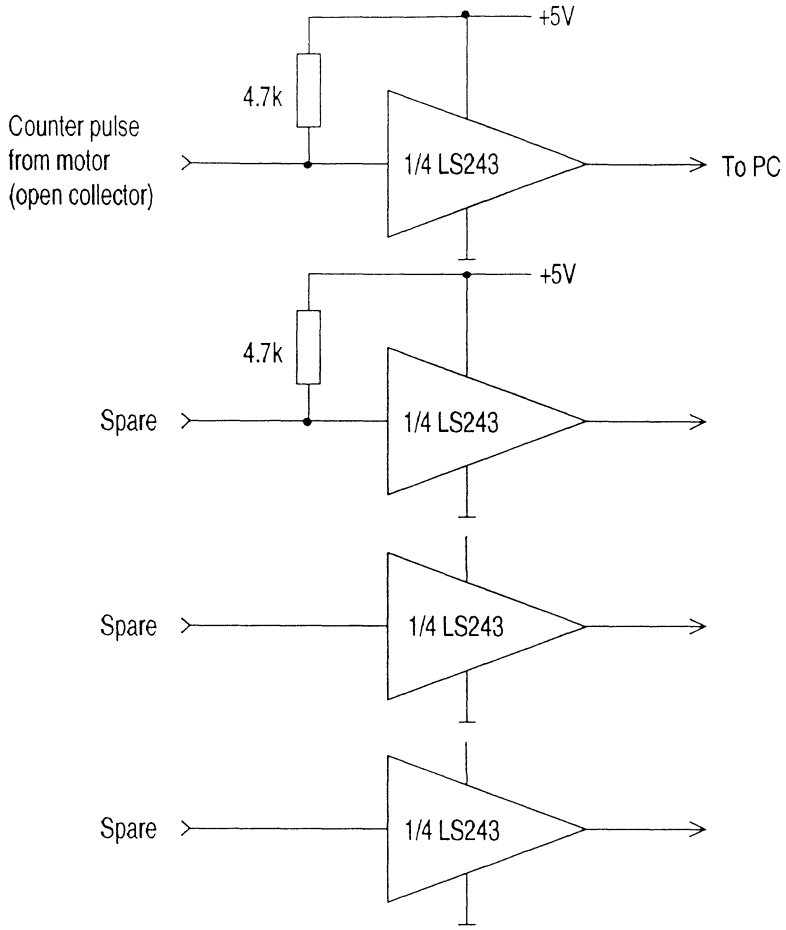
D-sub to DC motor

Cable colour	Pin	Function
yellow	1	motor winding
green	2	motor winding
grey	6	counter signal 1 from motor
brown	7	counter signal 2 from motor
shield	8	0 V chassis (common)
pink	9	+5 V from power supply

D-sub to PC

Cable colour	Pin	Function
blue	1	direction (TTL)
shield	2	0 V (common)
white	6	counter signal 1 to PC
N.C.	7	counter signal 2 to PC
N.C.	8	0 V chassis (common)




+5V
4.7k
Counter pulse
from motor
(open collector)
1/4 LS243
To PC
+5V
4.7k
Spare
1/4 LS243
Spare
1/4 LS243
Spare
1/4 LS243

References

[1] P. K. Carroll and E. T. Kennedy, "Laser-produced plasma", Contemp. Phys., **22** 61 (1981).

[2] R. Kauffman, "X-ray radiation from laser plasma", Handbook of plasma physics, Eds. M.N. Rosenbluth and R.Z. Sagdeev, Vol 3, Elsevier Science Pubishers B. V., (1991).

[3] A. G. Michette, "Laser-generated plasma: source requirements for X-ray microscopy", Journal of x-ray science and technology **2**, 1-16 (1990);
J. Kirz and H. Rarback, "Soft x-ray microscopes", Rev. Sci. Instrum., **56** 1-13 (1985);
A. G. Michette, "X-ray microscopy", Rep. Prog. Phys., **51** 1525 (1988);
J. A. Trail, "A compact scanning soft x-ray microscope", Ph. D. Dissertation, Edward L. Ginzton Laboratory, Stanford University (1989).

[4] L. Rymell and H. M. Hertz, "Droplet target for low-debris laser-plasma soft x-ray generation", Opt. Comm., **103** 105-110 (1993).

[5] Y. C. Kiang and R.W. Lang, "Measuring focused Gaussian beam spot sizes: a practical method", Appl. Opt., **22** 1296 (1983);
J. M. Khosrofian and B. A. Garetz, "Measurements of a Gaussian laser beam diameter through the direct inversion of knife-edge data", Appl. Opt., **22** 3406 (1983).

[6] H. M. Hertz and R. L. Byer, "Tomographic imaging of micrometer-sized optical and soft-x-ray beams", Opt. Lett., **15** 396 (1990).

[7] N. M. Celglio, D. T. Attwood and E. V.George, "Zone-plate coded imaging of laser-produced plasmas", J. Appl. Phys., **48** 1566 (1977);
"Fresnel zone plate coded imaging", Physics of laser fusion VII, Lawrence Livermore National Laboratory.

[8] E.H. Hecht, Optics, chap. 10 (Addison-Wesley Publishing Company, Inc, 1974).