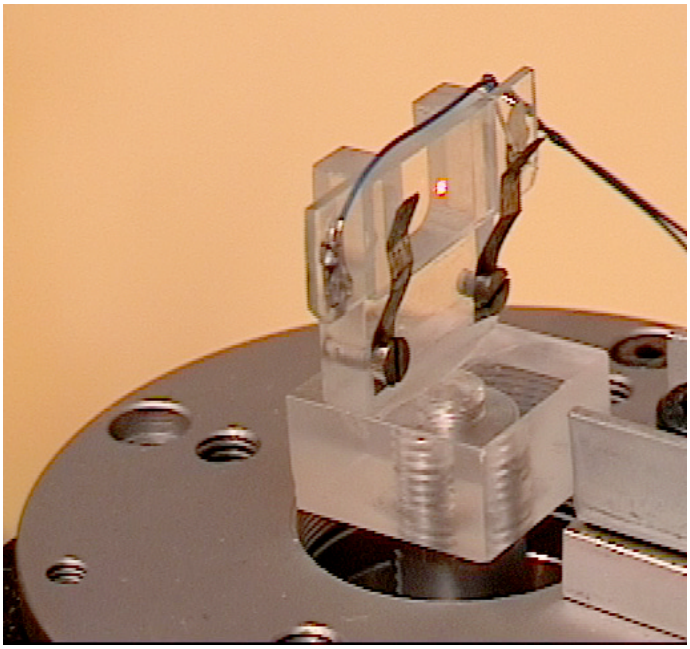
FOI-R--0247--SE
November 2001
ISSN 1650-1942
**Scientific report**

Hans Kariis, Per Rudquist, Koen D'havé and Ulf Ekström

# Manufacture and Evaluation of Polymer Dispersed Liquid Crystal Test Components



Sensor Technology
SE-581 11 Linköping

SWEDISH DEFENCE RESEARCH AGENCY

Sensor Technology
P.O. Box 1165
SE-581 11 Linköping

FOI-R--0247--SE

November 2001

ISSN 1650-1942

**Scientific report**

Hans Kariis, Per Rudquist, Koen D'havé and Ulf Ekström

# Manufacture and Evaluation of Polymer Dispersed Liquid Crystal Test Components

**Issuing organization**
FOI – Swedish Defence Research Agency
Sensor Technology
P.O. Box 1165
SE-581 11 Linköping

**Report number, ISRN**
FOI-R--0247--SE

**Report type**
Scientific report

**Research area code**
6. Electronic Warfare

**Month year**
November 2001

**Project no.**
E3021, E3826

**Customers code**
5. Contracted Research

**Sub area code**
61 Electronic Warfare, Electromagnetic Weapons

**Author/s (editor/s)**
Hans Kariis
Per Rudquist
Koen D'havé
Ulf Ekström

**Project manager**
Sören Svensson

**Approved by**

**Sponsoring agency**
FM

**Scientifically and technically responsible**
Hans Kariis

**Report title**
Manufacture and Evaluation of Polymer Dispersed Liquid Crystal Test Components

**Abstract (not more than 200 words)**
Components for protection against laser radiation have been manufactured using orthoconic antiferroelectric liquid crystals. LC droplets were dispersed in an index matched polymer matrix forming a transparent component. When a voltage is applied across the thin component the rodlike molecules are tilted, destroying the index matching. The lack of index matching results in scattering of incoming laser light. The scattering was measured as a function of angle, voltage and temperature. The dynamic properties of the components were analysed as well as the spectral distribution of the transmitted light in both open and closed states. The results are presented together with suggestions for improvements based on the results and a theoretical model presented.

**Keywords**
Protection. Laser, liquid crystal, voltage, scattering, PDLC

**Further bibliographic information**

**Language** English

**ISSN** 1650-1942

**Pages** 52 p.

**Price acc. to pricelist**

**Security classification**

| | | |
|---|---|---|
| **Utgivare**<br>Totalförsvarets Forskningsinstitut - FOI<br>Sensorteknik<br>Box 1165<br>581 11 Linköping | **Rapportnummer, ISRN**<br>FOI-R--0247--SE | **Klassificering**<br>Vetenskaplig rapport |
| | **Forskningsområde**<br>6. Telekrig | |
| | **Månad, år**<br>November 2001 | **Projektnummer**<br>E3021, E3826 |
| | **Verksamhetsgren**<br>5. Uppdragsfinansierad verksamhet | |
| | **Delområde**<br>61 Telekrigföring med EM-vapen och skydd | |
| **Författare/redaktör**<br>Hans Kariis<br>Per Rudquist<br>Koen D'havé<br>Ulf Ekström | **Projektledare**<br>Sören Svensson | |
| | **Godkänd av** | |
| | **Uppdragsgivare/kundbeteckning**<br>FM | |
| | **Tekniskt och/eller vetenskapligt ansvarig**<br>Hans Kariis | |
| **Rapportens titel (i översättning)**<br>Tillverkning och undersökning av PDLC testkomponenter | | |



**Sammanfattning (högst 200 ord)**

Komponenter för skydd mot laser har tillverkats med hjälp av ortokoniska antiferroelektriska vätskekristaller. En transparent komponent erhölls genom att vätskekristalldroppar dispergerades i en indexmatchad polymermatris. När en spänning lades över komponten kom de stavformiga molekylerna att vinklas, vilket förstörde indexmatchningen. Utan indexmatchning sprids inkommande laserljus. Denna spridning mättes som funktion av infallsvinkel, spänning och temperatur. Komponenternas dynamiska egenskaper undersöktes liksom den spektrala fördelningen hos transmitterat ljus, både i öppet och stängt tillstånd. Resultaten presenteras tillsammans med förslag till förbättringar baserade på erhållna resultat och på en teoretisk modell, som också presenteras.



| | |
|---|---|
| **Nyckelord**<br>Vätskekristall, laser, PDLC, antiferroelektrisk, spänning, skydd | |
| **Övriga bibliografiska uppgifter** | **Språk** Engelska |
| **ISSN** 1650-1942 | **Antal sidor:** 52 s. |
| **Distribution enligt missiv** | **Pris: Enligt prislista**<br>**Sekretess** |

# Contents

# Introduction

Human eyes, as well as other optical sensors, might be damaged by intentional or accidental laser exposure[i]. To protect eyes from harmful laser radiation a number of technologies might be utilised, non-linear optical absorbers are among the most promising[ii]. These materials absorb high intensity light while they are transparent to low intensity light. The intensity needed to activate the materials is, for currently available materials, very high. Therefore non-linear absorbers cannot protect against interference from low intensity lasers. The non-linear absorbers themselves also have to be protected, as their damage threshold is not high enough to withstand laser exposure for a long time.

To address these two problems, polymer dispersed liquid crystal components might be used. Within the project "Photonics in defence applications" such components have been developed and produced at Chalmers University of Technology in Gothenburg. In the first phase 1 of the "Photonics" program (1998-2000) liquid crystal (LC) shutters and spatial light modulators (SLMs), based on field-controlled polarisation between polarisers were investigated[iii]. This is a well-known technique since the electrooptic geometry is the basis for generally all commercial LC displays today. The main results were that the switching speed and extinction in the closed state were satisfactory while the transmission in the open state (limited to below 50% due to the crossed polariser geometry) was too low.

In the present project we are working with liquid crystal electrooptic techniques that do not need polarisers and we are concentrating our efforts to polymer dispersed liquid crystals based on a new type of antiferroelectric liquid crystals with unique optical and electrooptic properties. The components have been analysed at FOI in Linköping and the results obtained are reported here.

# Polymer Dispersed Liquid Crystals, PDLCs

Conventional polymer dispersed liquid crystals (see figure 1a) are composite materials consisting of micron-sized droplets of nematic liquid crystal dispersed in an optically transparent (and non-birefringent) polymer matrix. The nematic liquid crystal has positive dielectric anisotropy, $\Delta\varepsilon > 0$, (which makes the elongated molecules orient along an applied electric field) and the ordinary index of refraction $n_\perp$ (measured perpendicular to the average direction of the long molecular axis) is matched to the one of the polymer, $n_p$. There is no correlation between the droplets and the director configurations of the droplets are randomly distributed in the PDLC film. In the field-off state there is then a mismatch of the effective refractive indices of the droplets and the polymer, i.e. incident light experiences different effective refractive indices in the droplets and the polymer. The PDLC film therefore strongly scatters light, the film is not transparent and has a white opaque or translucent appearance. On application of an electric field across the PDLC the liquid crystal molecules reorient into the field direction due to the dielectric coupling. Light from an angle of incident normal to the surface now experiences nearly the same refractive index in the nematic droplets and in the polymer and there is almost no scattering. Upon removal of the voltage, the droplets return to their original scattering orientation. Thus, we have a field-controlled scattering device, being scattering in the off state and transparent in the on state. As there are no polarizers involved, the transmission in the open state can be more than 80%. However, the physical properties of nematic liquid crystals make the switching from the transparent to the scattering state (the interesting process for laser protection) relatively slow (typically tens of milliseconds). Therefore we are concentrating our efforts onto a much faster PDLC system based on so-called orthoconic antiferroelectric liquid crystals (OAFLCs).

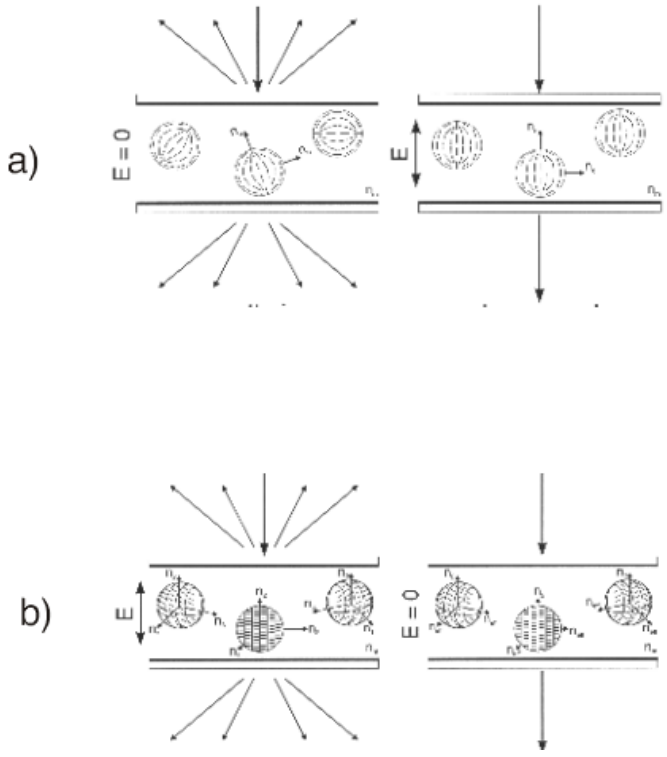



***Figure 1 Illustration of conventional nematic PDLC (a) and polymer dispersed otrhoconic AFLC (b). In (a) the field off state is scattering and the field on state is transparent. In (b) the situation is reversed and the swithing from transparent to scattering is about 1-2 orders faster than in (a). From reference vii.***

## Orthoconic AFLCs

In antiferroelectric liquid crystals (AFLC) the rodlike molecules are oriented in layers (smectic order) , one molecule thick. The molecules are tilted the angle $\theta$ with respect to the layer normal and the tilt order is anticlinic, i.e the molecules tilt in oposite directions $\pm\theta$ in adjacent layers. Due to the chirality (lack of mirror symmetry) there is a local electric polarisation **P** connected to each layer. **P** is directed perpendicular to the molecular long axis and in the plane of the smectic layer. Moreover, **P** is coupled to the tilt direction and the anticlinic order makes the material antipolar and antiferroelectric. An electric field, applied perpendicular to the layer normal, can switch the molecules to its synclinic (same tilt direction in all layers) states. In the special case of $\theta = 45°$ the (AFLC) becomes locally uniaxial with its optic axis perpendicular to the so-called tilt plane. The field induced synclinic states, on the other hand, have its effective optic axis along the molecules, thus tilted away ±90° from the layer normal, depending on the sign of the applied field. In a ”surface-stabilised geometry” where the orthoconic AFLC material has its tilt-plane strictly parallel to the two bounding surfaces of a cell the effective optic axis can be switched between three mutually orthogonal directions by means of application of –E, 0, and +E.

## Polymer dispersed orthoconic AFLCs

Consider an OAFLC in a PDLC structure where the tilt plane of the material inside the droplets is parallel to the plane of the film, cf figure 1b. This means that the optic axis of all AFLC droplets is perpendicular to the PDLC film. If the ordinary refractive index of the OAFLC is matched to the one of the polymer there will be no scattering of incident light and the PDOAFLC is transparent. However, when the material is switched into any of the ferroelectric states the index matching is gone and the film scatters light. The geometry with vertical smectic layers inside the droplets seems to be obtained by applying a strong ”aligning” electric field, which straightens out the layers in the direction of the field. Compared to conventional nematic PDLCs we notice some important differences as shown in Table 1.

| PDOAFLC | PDLC |
|---|---|
| Transparent in field free state | Scattering in field free state |
| Scattering in field-on state | Transparent in field-on state |
| Switching transparent to scattering is fast: 50-100μs switching times seem possible | Switching transparent to scattering is slow: typically 10-100ms |
| New system, previously not studied | Well known system. Commercialised |

***Table 1***

# Manufacturing of PDOAFLC

We have used two different methods for the preparation of PDOAFLC:
Polymerisation induced phase separation (PIPS)
Solvent induced phase separation (SIPS)

## PIPS

In PIPS the liquid crystal is dissolved in a reactive monomer. The solution is placed between two electroded glass plates separated at a well-defined distance by means of spacers mixed in the solution. Everything is then illuminated with ultraviolet (UV) light for about 10 minutes. The UV light induces the polymerisation of the reactive monomer molecules and in the process the liquid crystal material separates from the polymer and forms small droplets embedded in the created polymer matrix. In our case, the polymerisation (curing) was performed at different temperatures in the antiferroelectric phase, but also at elevated temperatures where the material is in its SmC*, SmA*, or isotropic phases. No significant importance of the curing temperature could be found from preliminary electrooptic evaluations. The maximum concentration of OAFLC which could be fully dissolved in the monomer material turned out to be about 20-25%.

Materials used:

Liquid crystal materials: W107, W107a, and W129, Orthoconic AFLC mixtures synthesised by Professor Dabrowski at the Military University of Technology, Warsaw, Poland.

Matrix material: Norland Adhesive 68 (NOA 68), UV curing glue, refractive index 1.56.

ITO-coated glass, 1.1 mm thickness

Spacer balls of diameter, 15-50μm

## SIPS

In SIPS the liquid crystal material is dissolved together with a polymer in a solvent, e.g. chloroform. The solution is put on an ITO coated glass plate. On evaporation of the solvent the liquid crystal-polymer system phase separates and liquid crystal droplets are formed in the polymer matrix. Note that in SIPS the cell must be assembled after the phase separation process of the PDLC in order to allow for the solvent to escape from the solution. The PDLC film is then heated until the PDLC softens and the second ITO plate is put on top and the cell is pressed together. After the cell assembling the PDLC can be heated to the isotropic phase of the polymer where the liquid crystal dissolves in the polymer. On subsequent cooling the phase separation once again occurs (temperature induced phase separation, TIPS). By tuning the cooling rate in this process the size of the liquid crystal droplets can be controlled to some extent. We found that somewhat higher concentrations of LC material could be used in SIPS compared to PIPS.

Materials used:

Liquid crystal materials: W107, W107a, and W129.

Matrix material: Polyvinyl butyral PVB, refractive index 1.48.
Solvent: Chloroform

The cells investigated were manufactured with the parameters in table 2.

| Namn | LC material | Polymer | % LC | Thickness [µm] | Method | Temp. [ºC] |
|---|---|---|---|---|---|---|
| PDAFLC:1 | W107 | NOA68 | 20 | 15 | PIPS | 100 |
| PDAFLC:2 | W107 | NOA68 | 20 | 15 | PIPS | 110 |
| PDAFLC:3 | W107 | NOA68 | 20 | 15 | PIPS | 120 |
| PDAFLC:4 | W107 | NOA68 | 20 | 15 | PIPS | 140 |
| PDAFLC:5 | W107 | NOA68 | 20 | 15 | PIPS | 80 |
| W129:1 | W129 | PVB | 40 | 23 | SIPS | - |
| W129:3 | W129 | PVB | 40 | 23 | SIPS | - |

***Table 2 Manufacture parameters for the liquid crystal components***

## Inverted System – Plastic spheres in AFLC matrix

In an attempt to increase the scattering efficiency by means of increasing the concentration of scattering objects we have also built the first cells of an inverted system. Here we mixed plastic spheres of 3µm diameter into the AFLC in a ratio of about 50/50 in volume. We also introduced a small amount of larger balls (15µm) to set the thickness of the cells. These inverted cells show a significantly stronger scattering in the closed state than the PDOAFLCs, but the transmission in the open state is also low due to residual scattering. This can be an effect of index mismatch between the balls and the AFLC, but it can also be an effect of bad alignment of the smectic layers in the cell. Ideally, the smectic layers should everywhere be vertical (normal to the glass plates) and at the same time the tilt-plane of the OAFLC should be parallel to the glass plates. This ideal situation might be very difficult to obtain in the inverted system. These first cells are under evaluation.

# Measurements

## Measurement set-up

Seven PDAFLC components and one commercial PDLC film were investigated. Mainly of interest was the ability of the samples to scatter incoming light in the closed state, while at the same time have a high transmittance in the open state. The speed of switching between the open and closed states was also measured, as the pulse response gives important information about the dynamic properties of the samples. The components were labelled PDAFLC:1-5, W129:1 and W129:3, and we refer to the "Manufacturing of PDOAFLC" chapter for a closer description of them.

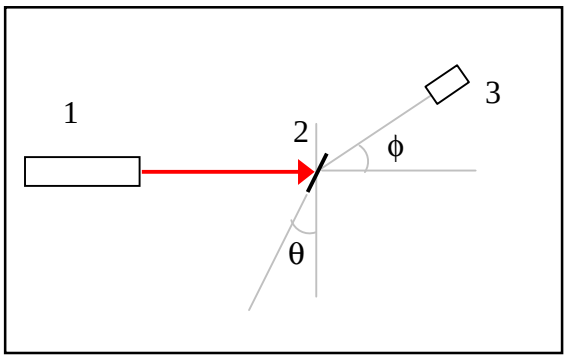



***Figure 2. The principal measurement set-up.***

Each sample was placed in a holder (2) which lets us control the angle θ with a step motor, see figure 2. The sample was then illuminated by a 633 nm HeNe laser (1). The detector (3) was placed on an arm originating from the sample holder, so that the angle ϕ could be adjusted to investigate the angular distribution of the scattered light. The detector lens had a diameter of 5.2 mm, which together with the distance from the sample to the detector (132 mm) gave us an angular resolution better than 1.1°. This configuration was a compromise between angular and amplitude resolution, and it suited our needs quite well.

All samples had an ITO electrode at each side of the PDAFLC. The “voltage” referred to in this report is the potential difference between these two electrodes. The driving voltage was supplied as a square wave at 300 Hz, generated by a function generator and amplified to give a maximum of 320 V. When the temperature dependence was measured the sample was placed inside a temperature regulating cell which made it easy to keep the sample at a specific, constant, temperature for a long time. When the angular properties were measured a hot air gun was substituted for the cell, keeping the sample at 62±5°C.

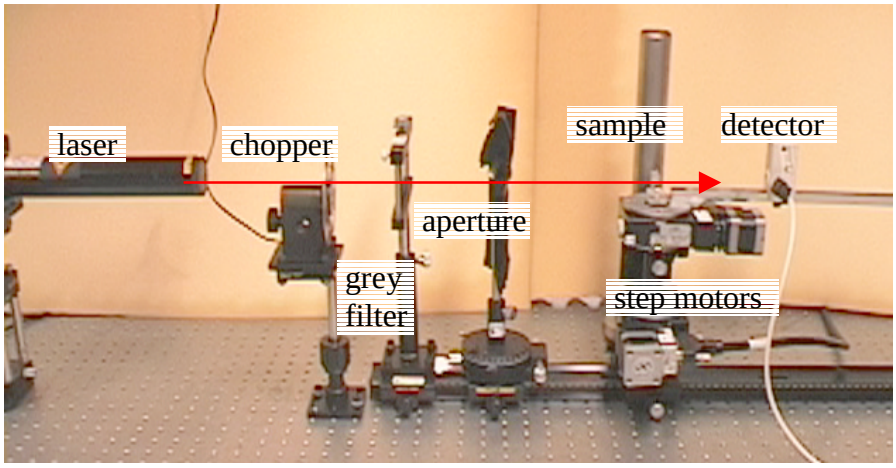



***Figure 3. Photograph of the measurement set-up.***

A chopper was placed in front of the laser source giving a square pulse input to the detector. This made it easy to compare the amplitude of the laser light reaching the detector with the zero-level background. Whenever “intensity” is mentioned, in the graphs or in the text, this refers to the amplitude of this square pulse, as measured by the detector and automatically converted by a digital oscilloscope to the voltage numbers in the report. A grey filter was used to adjust the amplitude of the laser to the sensitivity of the detector. In front of the sample an aperture was placed, mainly to remove reflections from the optical components, figure 3.

## Measurement procedure

Since a lot of measurements were going to be made some custom software was written to perform most of the tasks. The programs are written in C, and use a GPIB board to communicate with the instruments. Tasks that could be performed automatically was the adjusting of the angles $\phi$ and $\theta$ (see figure 1) and controlling of the voltage V. The resulting detector intensity was also measured automatically by an oscilloscope. See appendix 4 for further descriptions and listings of the programs.

## Initial measurement of temperature dependence

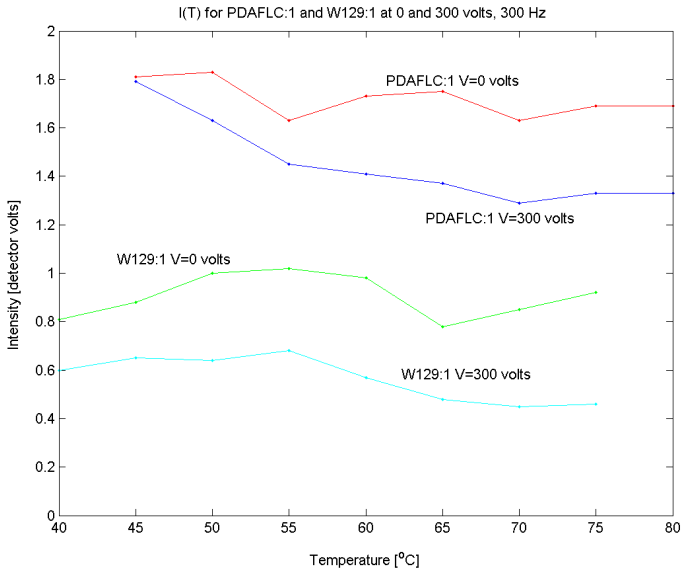



***Figure 4. Determining the temperature required for switching.***

Like for all liquid crystals the dynamic and static properties of the PDAFLCs are highly temperature dependent. We found that a temperature of about 60-65 ºC was suitable to produce a large enough difference between the open and closed states (see figure 4). Only two samples were tested but the result is expected to be valid for all samples as the same materials were used in each group of components.

## Transmittance in the open state

The transmittances at $\theta = \phi = 0$ was measured for all samples, in the open state at room temperature (21ºC), are shown in figure 5It should be noted that some of the samples (PDAFLC:5, and to some extent W129:1 and W129:3) had visible defects (small bubbles) which cause the low transmittance.

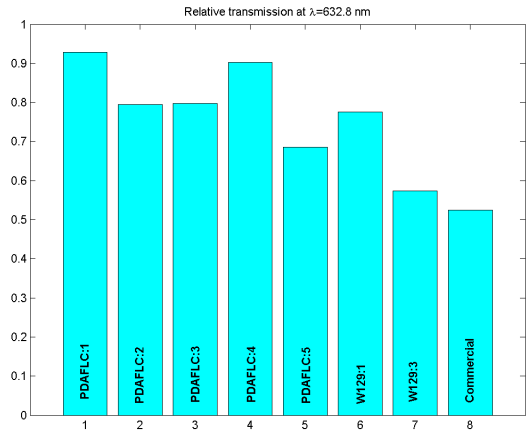



***Figure 5. Relative transmission in open state.***

The transmittance relative to air (no sample in the beam path) varies between the samples. The differences seen among the PDAFLC components can be used, in conjunction with the other findings, to give a complete picture of the samples. All samples tested have a higher (or much higher) transmittance

than the commercial sample, but since the commercial film has a different thickness than the others the results are not directly comparable. The commercial film had a potential difference of 80 volts applied at the electrodes, to put it in the open state.

## Transmittance as a function of voltage

The transmittance as a function of the voltage is a very important property of the PDAFLC, and it is preferable to have a low activation voltage since it reduces the probability of breakdown and makes device integration easier.

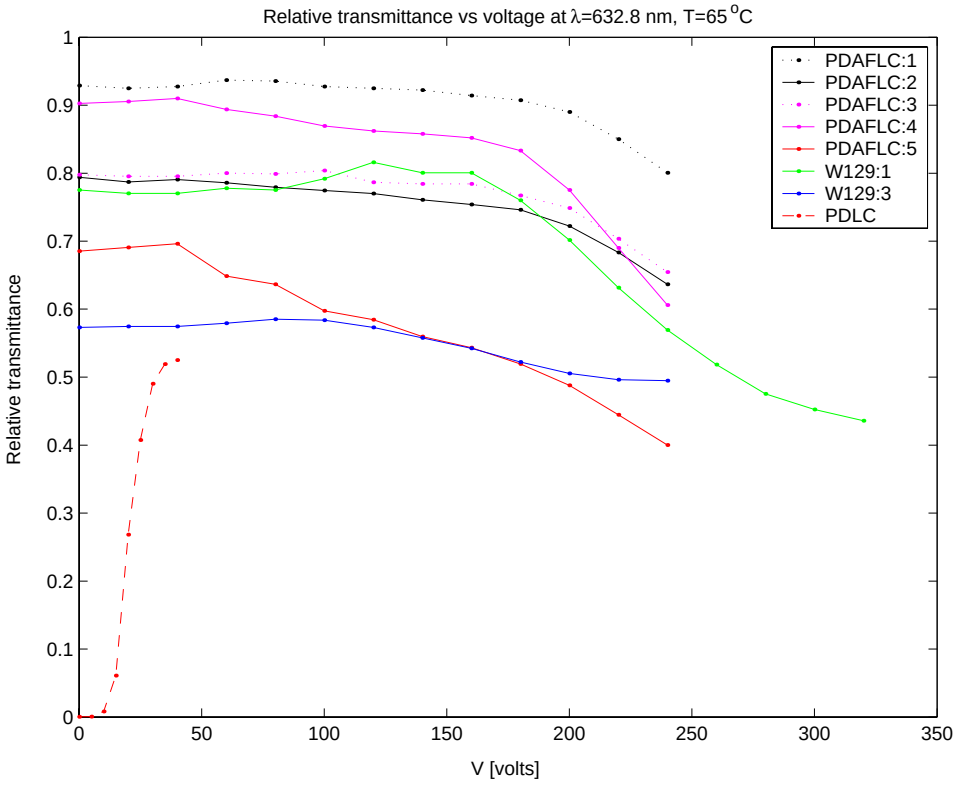



***Figure 6. Transmission as a function of voltage. Shown point are averages of 4 measurements to reduce noise.***

It is clearly favourable to use a voltage > 240 V for most of the samples (see figure 6.), but since the risk for breakdown gets large at these levels this range has not been tested except for PDAFLC:1 and PDAFLC:2[1]. The difference in transmittance between V=0 and V=240 volts are summarised in table 3 to allow for easy comparison.

[1] One sample not otherwise mentioned in this report, W129:2, was accidentally destroyed during the measurements. It was however very promising and required an activation voltage of about 80 volts to achieve maximum attenuation. (About 30 % difference for open vs. closed states.)

| Sample | Relative transmittance difference (closed vs. open state ) |
|---|---|
| **PDAFLC:1** | 0.14 |
| **PDAFLC:2** | 0.20 |
| **PDAFLC:3** | 0.18 |
| **PDAFLC:4** | 0.33 |
| **PDAFLC:5** | 0.41 |
| **W129:1** | 0.27 |
| **W129:3** | 0.14 |

***Table 3. Relative transmission difference.***

As can be seen in table 3 there is a large difference between the different samples, with PDAFLC:5 showing the largest difference between the “open” and “closed” state. Several samples show around 30% lower transmittance in the “closed” state, and this can obviously be increased by using thicker films. At V=320 volts there should be an even greater difference, but due to the problems with breakdown mentioned above this has not been tested.

## Spectral properties

Since the components should be able to protect from a wide range of lasers it is important to investigate the transmittance for the whole laser spectrum. A Mechelle 900 spectrometer, capable of detecting light from 400 nm to 1100 nm, was used to measure the light from a black body source, filtered through the component. The graphs show the relative transmittance difference (contrast) for the samples, combined with the transmittance of the open state (compared to air). The samples were heated to 62±5 °C. The glass substrates does affect the spectral properties, and its transmittance is presented in figure 7 below.

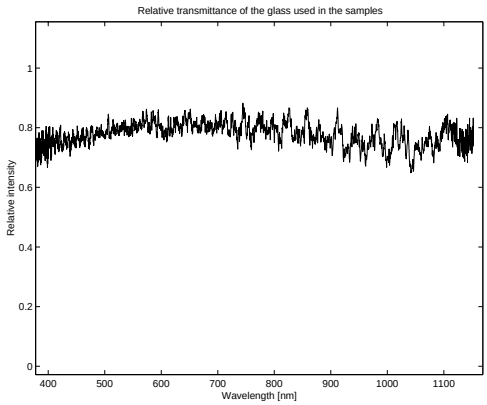



***Figure 7. The spectral properties of the glass.***

The data for the samples and the commercial PDLC is presented below. Please note that the spectra were taken with different exposure times for the different samples and therefore the open state transmittances is not directly comparable between the different samples, i.e. it differs with a scaling factor. Please refer to figure 5 when comparing the transmittance of the components. Due to technical difficulties W129:1 is missing from these experiments.

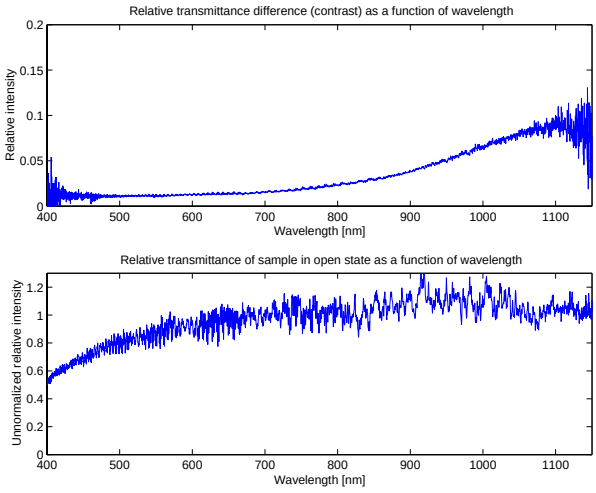



***Figure 8. Spectral properties for the commercial PDALC***

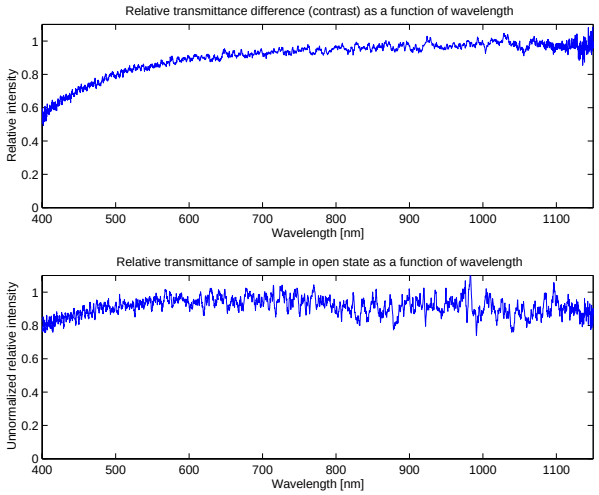



***Figure 9. Spectral properties for PDAFLC:1***

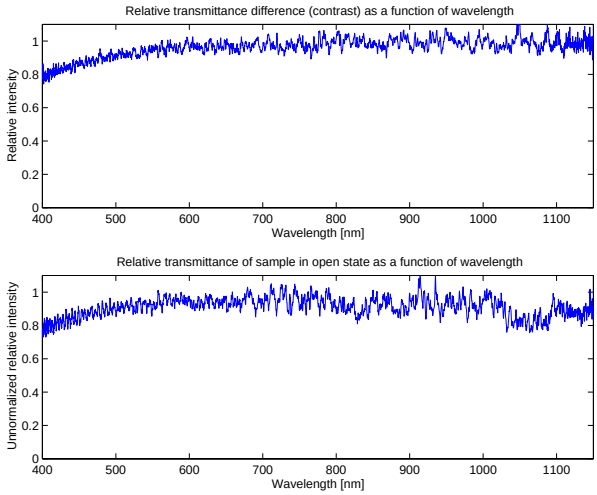



***Figure 10. Spectral properties for PDAFLC:2***

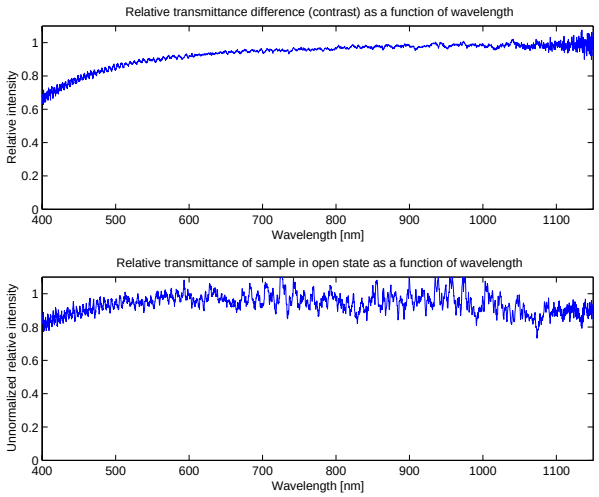



***Figure 11. Spectral properties for PDAFLC:3***

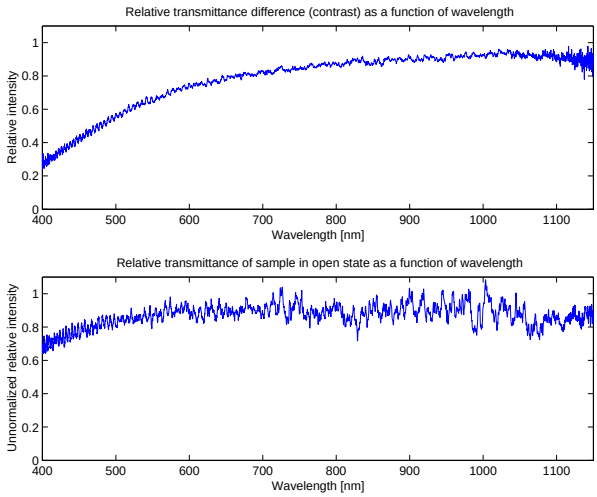



***Figure 12. Spectral properties for PDAFLC:4***

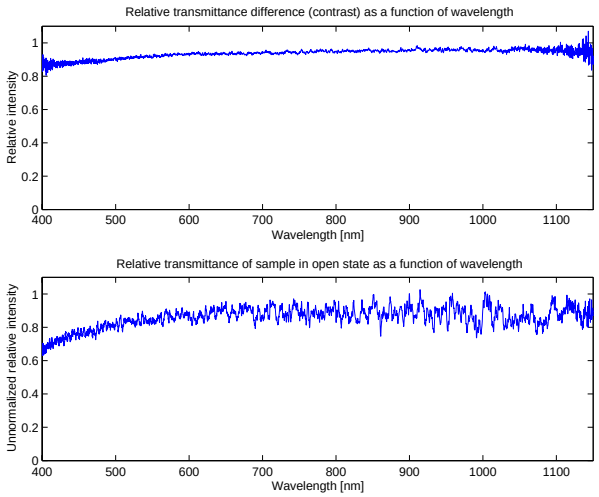



***Figure 13. Spectral properties for PDAFLC:5***

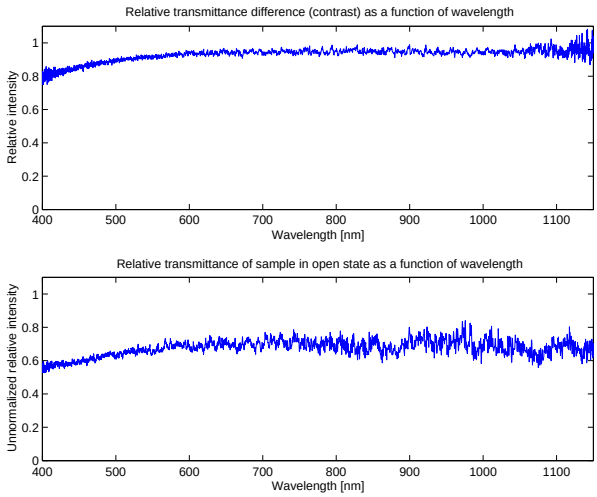



***Figure 14. Spectral properties for W129:3***

## Scattering properties

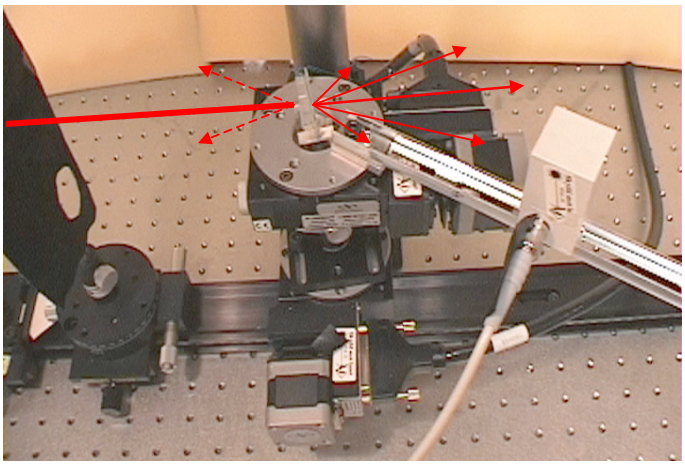


***Figure 15. Detail of set-up used to measure angular properties.***

The scattering properties were measured for different angles and the results are presented in a graph for each sample (See appendix 1). The most important finding of these measurements was that the scattering is almost independent of the incident angle θ. This is a desired property since it lets the component protect against harmful light coming from any direction. The scattering for small and large angles ϕ are shown separately since they required different grey filters in the set-up to get proper amplitude resolution. Figure 15 shows the measurement set-up.

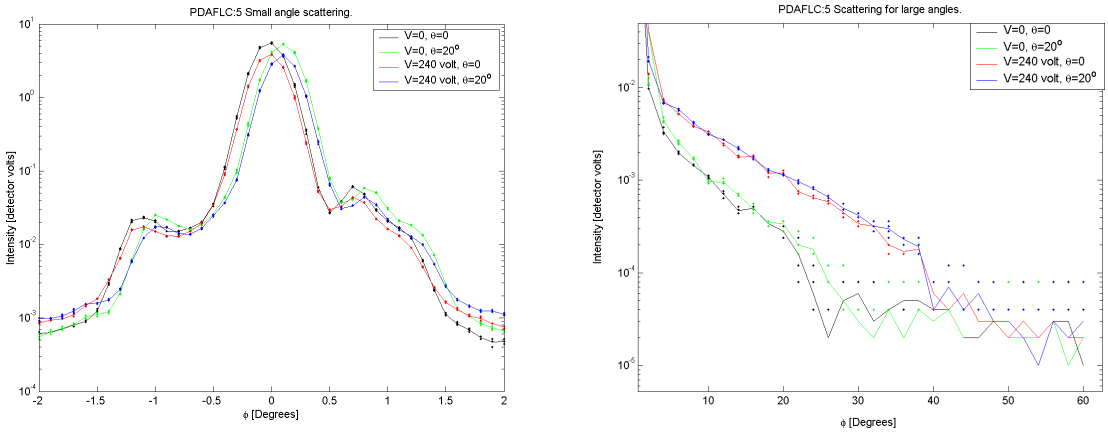



***Figure 16. Scattering for PDAFLC:5. Measurements were made for both small and large angles. Note that the ϕ angle was not calibrated to the same zero in the two measurements.***

An example of the distribution of the scattered light is shown in figure 16. The dots shown are individual measurements, and the solid lines indicate the average of four measurements. Notice that below an intensity of about $10^{-4}$ the values become somewhat unreliable due to detector noise and oscilloscope quantization error. All measurements where performed at about T=62±5 ºC (Sample heated by the hot air gun). See appendix 1 and 2 for the complete results.
As can be seen in the graphs PDAFLC:5 have a fair amount of scattering for angles in the range 0-40º, even in the ”open” state (V=0). This is probably due to the defects in the sample previously mentioned. Regarding the scattering at small angles (see appendix 2.) we notice primarily that there is little difference in the transmittance for the different angles θ. An interference effect for small angles is also visible in all samples.

## Dynamic properties

Response time was measured as a function of temperature and voltage. A single square pulse was generated by a signal generator and amplified to produce 160 and 320 volts temporary potential difference between the electrodes. The sample was kept at a specific temperature by the heat cell, which lets us test both the temperature and the voltage dependence at the same time. As the pulse was applied the detector signal was measured on an oscilloscope and four characteristic times, defined in figure 17, were calculated.

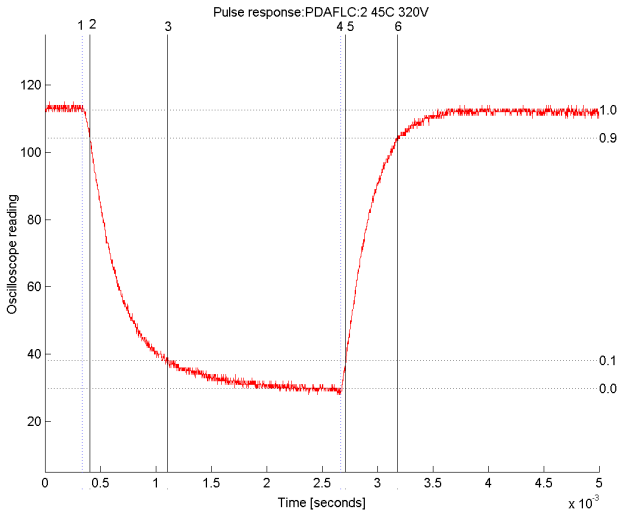



***Figure 17. Pulse response for PDAFLC:2.***

The pulse is applied at the first dashed vertical line (1). The **activation delay** is defined as the time between the start of the pulse and the moment where the curve reaches the 0.9 level at (2), relative the lowest level. The **activation time** is defined as the time between (2) and the 0.1-level at (3). When the pulse ends at (4) two corresponding times are measured, referred to as **deactivation delay** (4 - 5) and **deactivation time** (5 - 6) respectively.
Sometimes these definitions are not appropriate, or there are other phenomena present that needs comment; see below (figures 18-21).

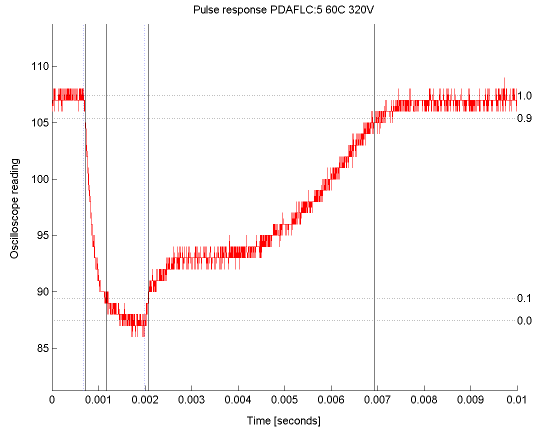



***Figure 18. Pulse response example.***

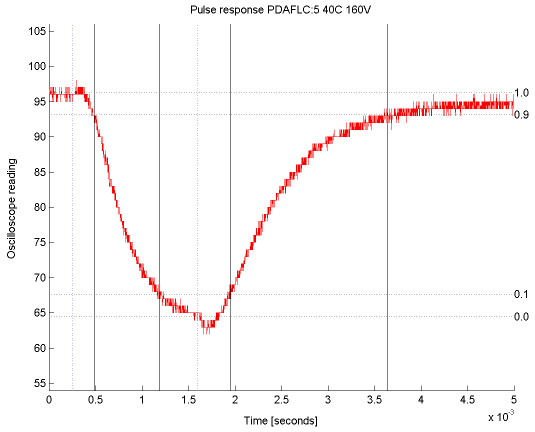



***Figure 19. Pulse response example.***

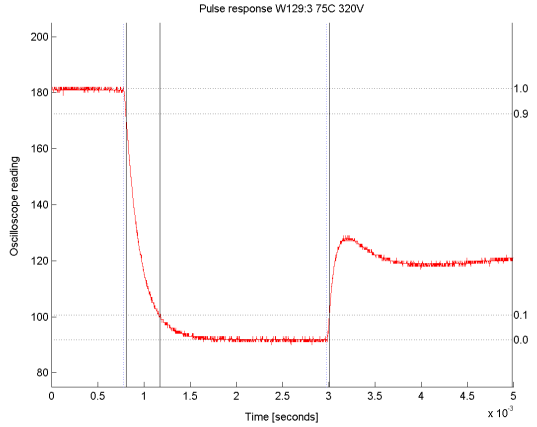



***Figure 20. Pulse response example.***

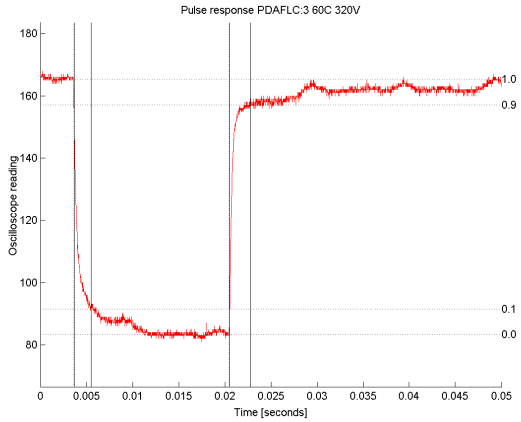



***Figure 21. Pulse response example.***

In figure 18 we see that there are two processes involved as the voltage is removed. First there is a steep increase, after which a slower, probably thermodynamic, process takes over. Because of the definition of the deactivation time there might be unfair cases where the signal reaches, for example, 0.8 very quickly and therefrom ascend slowly towards 1.0. This means that important information can be lost if only the response times, as previously defined, are considered.

Another interesting phenomena can be observed in figure 19, where we actually get lower transmission after the pulse has ended (Notice the small dip at about 1.7 ms). This is probably due to imperfect optical density matching between the droplets and the matrix material. Figure 20 shows yet another example of the two distinct events after the voltage was removed, and in figure 21 we see a similar process as the voltage is turned on. Besides the responses shown in figure 18-21 all samples behaved similar to figure 16 with respect to the shape of the response function.

| **Property** | **Typical value [µs]** | **Commercial PDLC at 21 °C [µs]** |
|---|---|---|
| **Activation delay** | 50 | 3 000 |
| **Activation time** | 500 | 13 000 |
| **Deactivation delay** | 50 | 500 |
| **Deactivation time** | 500 | 14 000 |

***Table 4. Typical response times compared with the commercial PDLC.***

In table 4 we see typical values for the various characteristic times of the different samples, as extracted from appendix 3. Corresponding values for the commercial PDLC at room temperature are listed for

reference. The “activation time” for the PDLC is the time from the closed to the open state, i.e. the time to “activate” after the voltage is turned on. It is notable that the difference in voltage (160 V vs. 320 V) did not affect the response times much. The reason might be that even though the process was actually faster with the higher voltage, the change in transmission was greater (i.e. more work had to be done), and these two effects might have cancelled each other.

No sample was noticeably faster than the typical values, however some were definitely slower at some temperatures (see below and appendix 3). For most samples it was seen that a higher temperature produced shorter response times, and this was consistent with how other LC components are known to work. Due to the issues associated with figure 18 (Two processes with different time scales) there are some noise present in the graphs, but the general trend should be visible for each sample. Some data points have been deleted due to technical difficulties in the measurement process, but these provided no additional information.

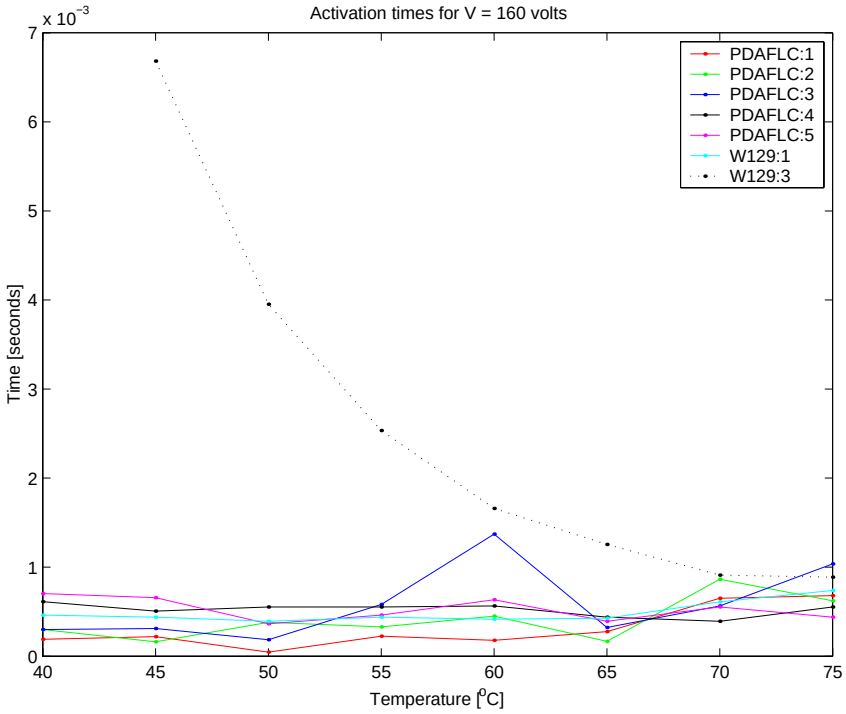



***Figure 22. Activation times as a function of temperature for V=160 volts.***

As can be seen in figure 22, the activation time is in the 0.5 ms range for most of the samples. The notable exception is W129:3, which shows a large increase in activation time as the temperature is reduced. This behaviour is repeated for V = 320 volts, as seen in figure 23.

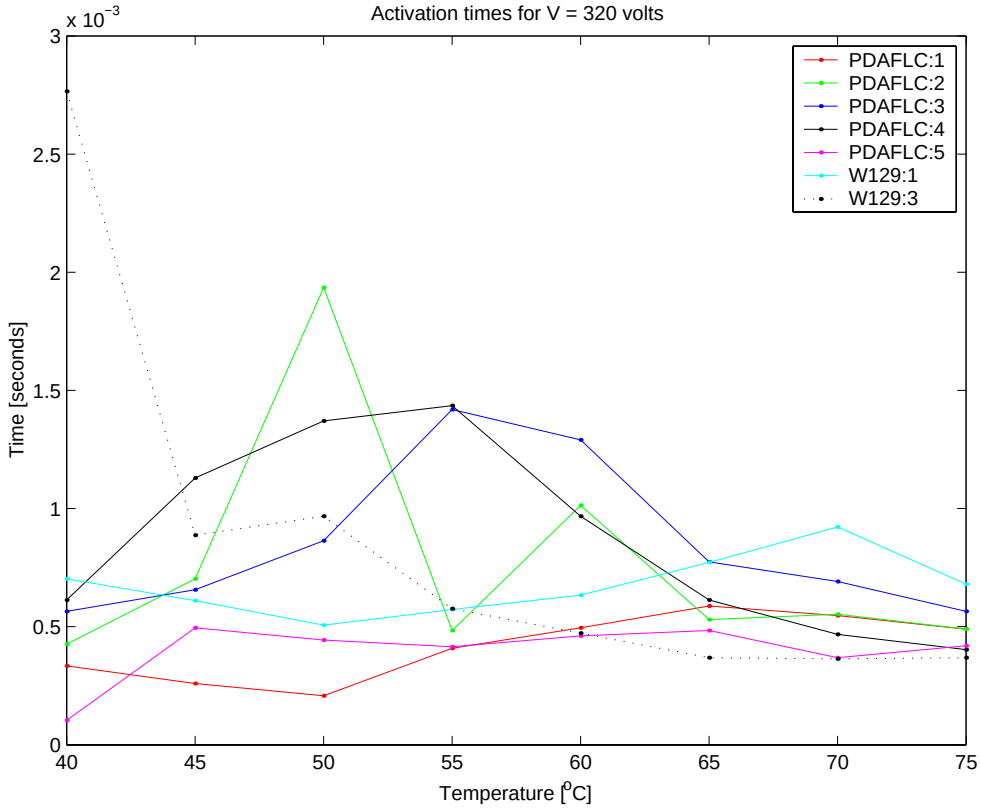



***Figure 23. Activation times for V=320 volts.***

Again we see some components firmly at or below the 0.5 ms level. PDAFLC:4 shows decreasing activation time with higher temperature. Notice that even though the activation times seem to be on the same order for 160 and 320 volts there is a subtle difference. Since the maximum attenuation is larger for higher voltage (see figure 6) the time to achieve the same absolute level of attenuation is considerably shorter for V=320 volts. The same is true for temperature. At 40 °C the difference in attenuation between the open and the closed state is smaller, and this might hide the temperature dependence in figure 22 and 23. Apparently these effects cancel and we get almost constant response times as temperature and voltage vary, except for W129:3.

PDAFLC:2, PDAFLC:3 and PDAFLC:4 shows an increase in activation time at around 55 °C. This could possibly indicate a phase transition at these temperatures, but nothing definite can be said about this without further investigation. For activation delay, deactivation delay and deactivation times the reader is referred to appendix 3.

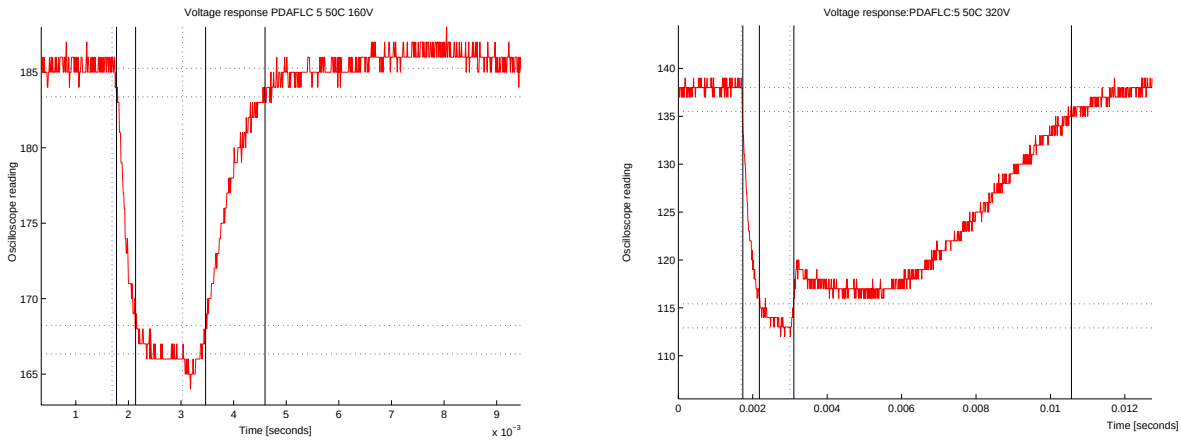



***Figure 24. Pulse response for PDAFLC:5 at 50 °C, V=160, 320 volts.***

An interesting phenomena was discovered when comparing the pulse response at different voltages for PDAFLC:5 at 50 °C. (See figure 24 above.) Apparently the higher voltage at V = 320 volts affects the crystal in such a way that the recovery time increases dramatically when compared to the V = 160 volts level. This increase in deactivation time for this sample is also clearly visible in figure 25 and 26 below.

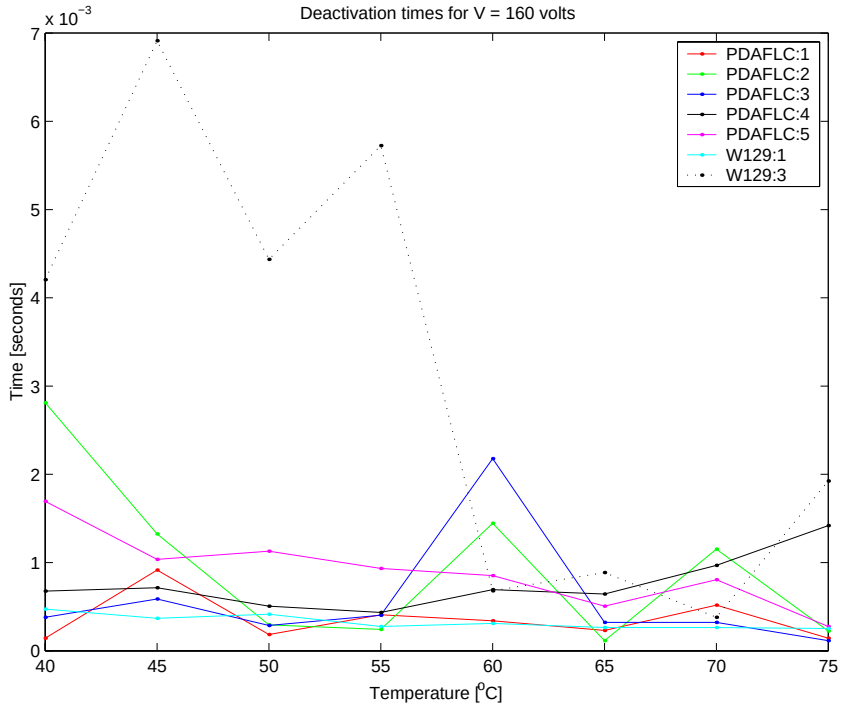



***Figure 25. Deactivation times for V=160 volts.***

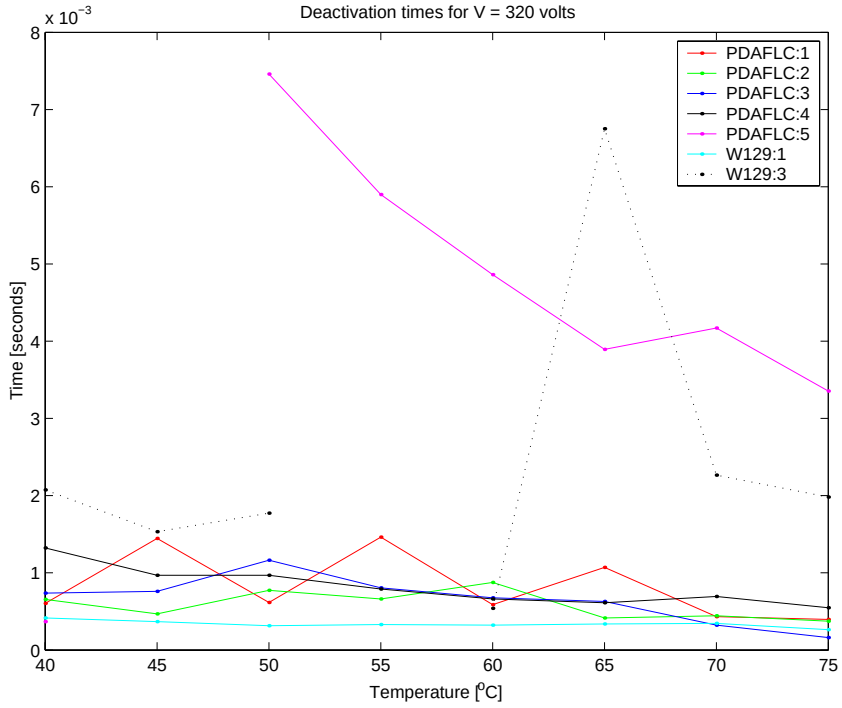



***Figure 26. Deactivation times for V=320 volts.***

## Measurement uncertainties

The purpose of this study is to compare the different components and get a depiction of their properties relative to each other. A rigorous measurement uncertainty analysis is therefore not necessary.

All spectra and most transmission values were measured in a relative manner, eliminating the influence of instrument sensitivity. The spectrometer was calibrated in the wavelength regime each day of measurement using a mercury lamp.

Temperature values, measured with a liquid thermometer, have a standard uncertainty of 5 K. Incidence angles and scattering angels were measured on the rotation stages, within a standard uncertainty of 1 degree.

The dynamic properties were measured using a traceably calibrated oscilloscope and a traceably calibrated function generator generated voltages.

# Theoretical analysis of PDOAFLC[2]

In order to evaluate the potential for orthoconic antiferroelectric liquid crystals in PDLC systems we have calculated the scattering properties of PDOAFLCs and made a comparison of the scattering properties of a conventional PDLC.

For a PDLC system the apparent extinction of light beam in the forward direction is caused by the scattering of light into directions other than the forward direction. The forward direction intensity is given by[iv]

$$I = I_0 e^{-(\sigma_t \beta_N x)} \quad (1)$$

where $I_0$ is the incident intensity, $\sigma_t$ is the total scattering cross section of the droplets and $\beta_N$ the density of the droplets in the PDLC, and $x$ is the distance travelled through the medium. Thus, the scattering of single droplets is directly connected to the total scattering and therefore the cross section of a single droplet gives good information about the scattering of the whole system.

In the case that the scattering objects have a characteristic size $kR >> 1$, where $k = 2\pi n_p/\lambda$, and $R$ is the radius of the (spherical) droplet we can use the anomalous diffraction approach[v], which has already successfully been applied to nematic PDLCs[vi] and is here adapted to the PDOAFLCs[vii]. The polymers used have $n_p$ of about 1.5 which gives $R >> \lambda/2\pi n_p \approx 50$nm, well below $\lambda \approx 500$nm.

By means of the approach described above the relative scattering cross sections for the anticlinic and the synclinic states have been calculated. The results from our calculations are given in the figures below. The used values of the relevant refractive indices in the molecular frame, as measured in the synclinic state, are $n_1=1.52$ and $n_3=1.7$. The results, figures 27-31, show that provided that the droplet size in the future can be increased to make $kR \approx 35$, we would, in fact, be able to achieve stronger scattering in PDOAFLC than in normal nematic PDLCs. However, experimental investigations indicates that we today have $kR \approx 5$ which could explain the very weak scattering measured so far on our PDOAFLC systems. The scattering efficiency can also be increased by increasing the birefringence of the OAFLC material and increasing the cell thicknesses. However, both these factors require further materials development. Thicker cells require a significantly lower threshold voltage for switching than what the present materials have in order to keep the driving voltage low enough to be used in an eye protection application.

[2] This is a very short summary of the extensive analysis on scattering of PDOAFLCs performed by Koen D'havé in his PhD thesis, which will be presented in November 2001.

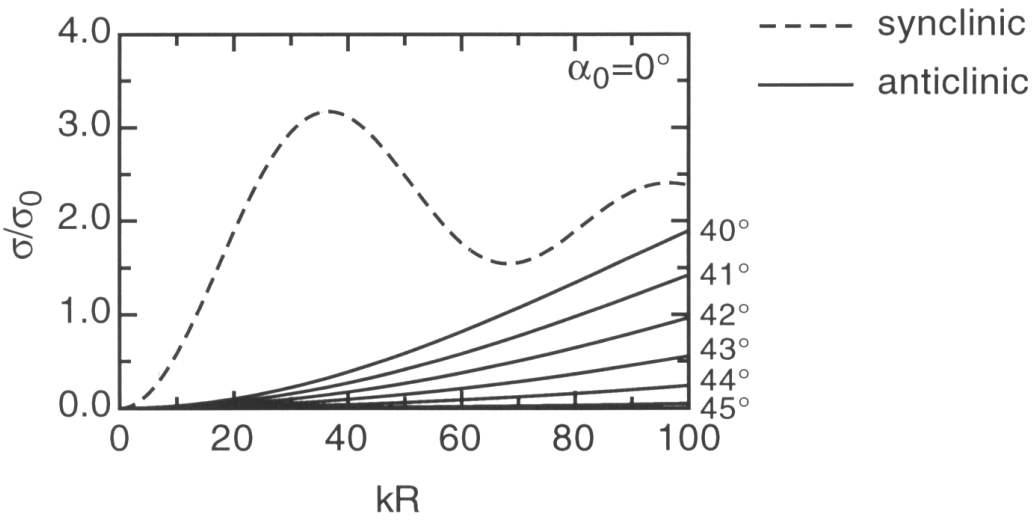



***Figure 27 Relative scattering cross section for PDOAFLC as function of characteristic size* kR*, for different values of molecular tilt. For θ=45° virtually no scattering occurs. Polymer refractive index used $n_p$ = 1.61. For the components investigated in this report kR is estimated to be between 5 and 10.***

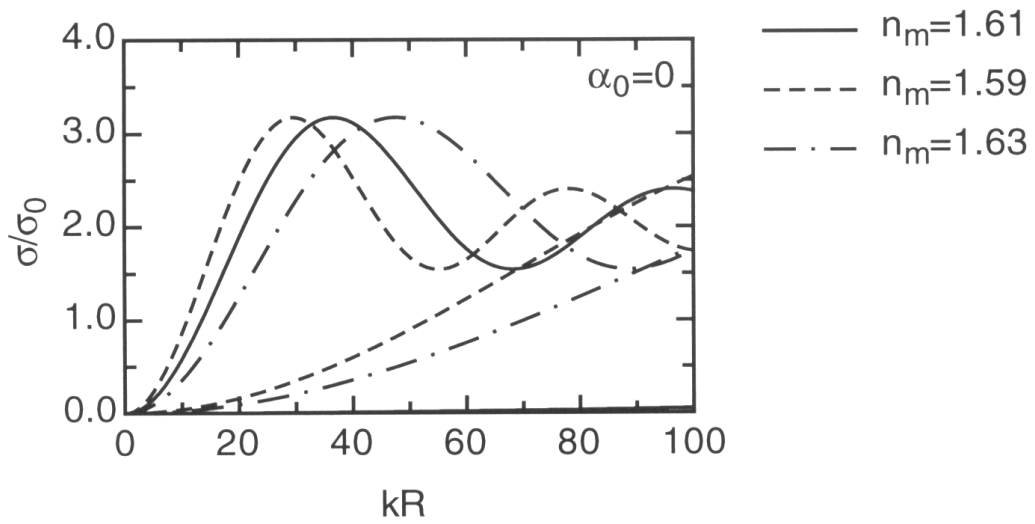



***Figure 28 Influence of index matching on the relative scattering cross section for PDOAFLC. Here the polymer refractive index $n_p$ is denoted $n_m$. Lower curves anticlinic state, upper curve synclinic state.***

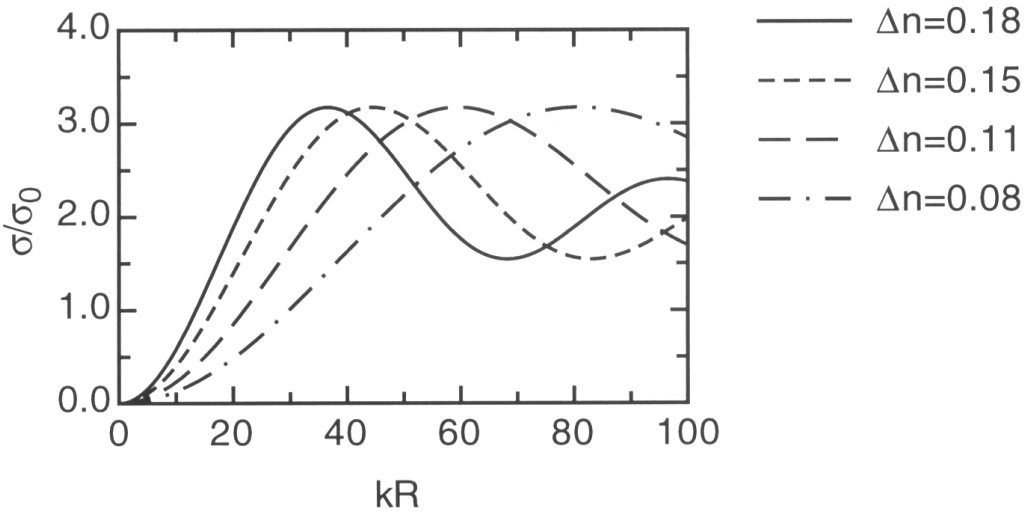



***Figure 29 Influence of the birefringence (in the molecular frame, synclinic state) on the relative scattering cross section of the field-on state for PDOAFLCs***

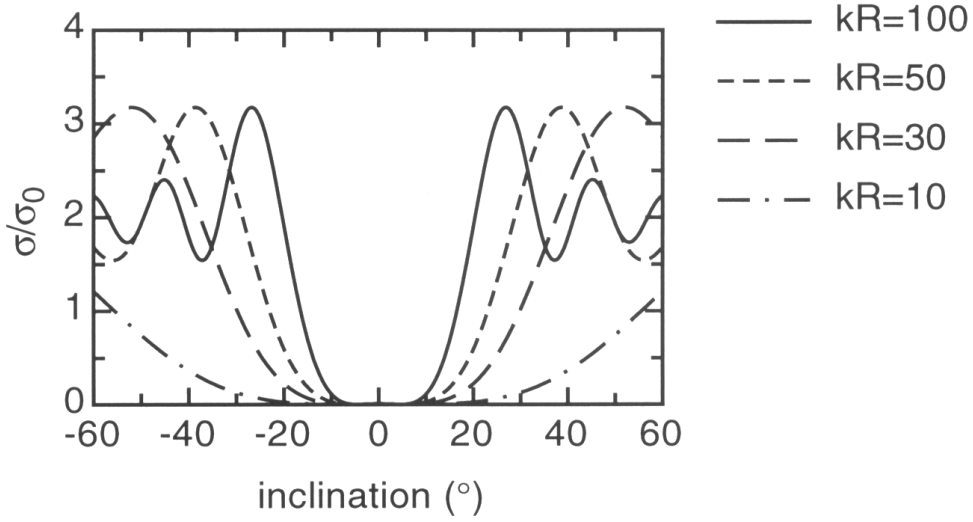



***Figure 30 Viewing angle dependence of the relative scattering cross section in nematic PDLC in the transparent state for different characteristic droplet sizes. Here*** $n_p = 1.521$.

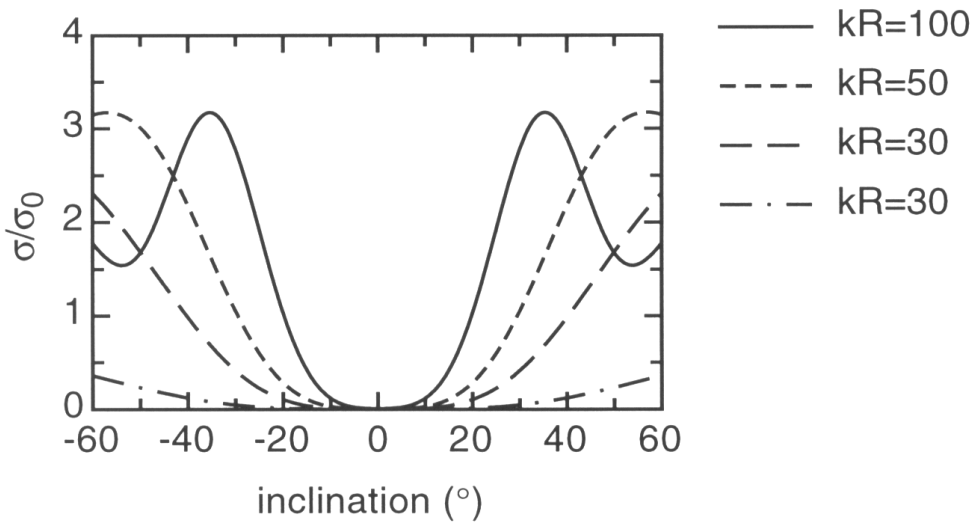



***Figure 31 Viewing angle dependence of relative scattering cross section in PDOAFLC in the transparent state for different characteristic droplet sizes. Here $n_p = 1.611$.***

# Conclusions

PDOAFLCs have been manufactured by PIPS and SIPS.

The electrooptic mechanism works.

The activation time is notably shorter than for commercial PDLCs.

The transmission in the open state is considerably higher than for both commercial PDLCs and for the devices based on field controlled polarisation investigated in the first phase of the photonics project.

The scattering efficiency must be significantly increased.

Theoretical analysis show that the relative scattering cross section can be significantly increased if we

1) increase and control the droplet size

2) use (find) materials with larger birefringence

3) bring down the threshold voltages

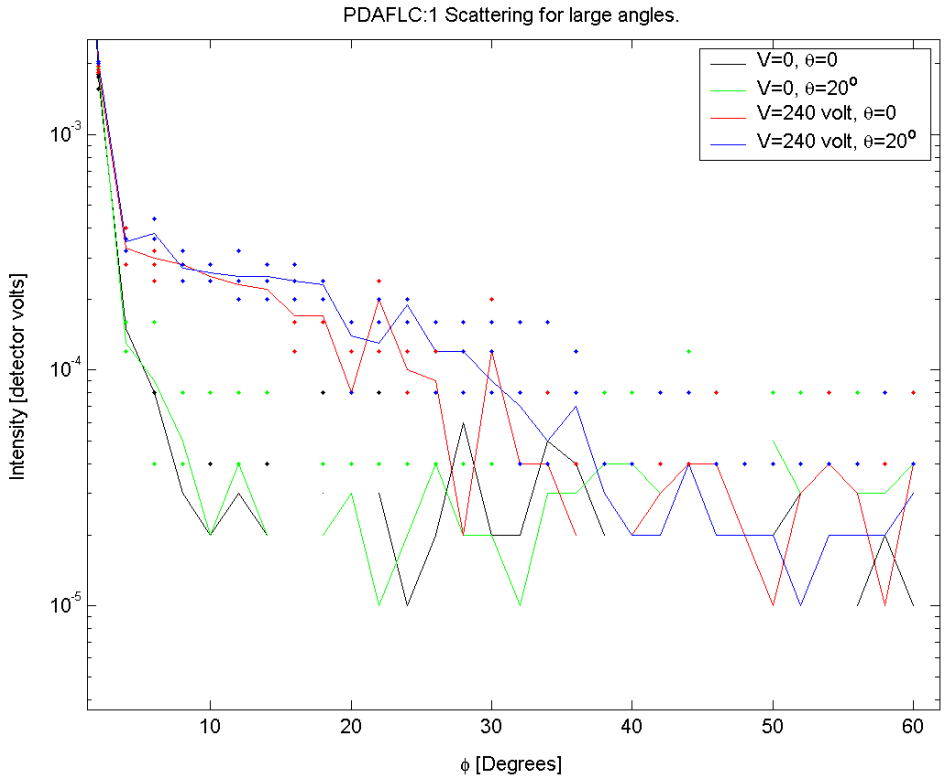
4) increase the density of droplets

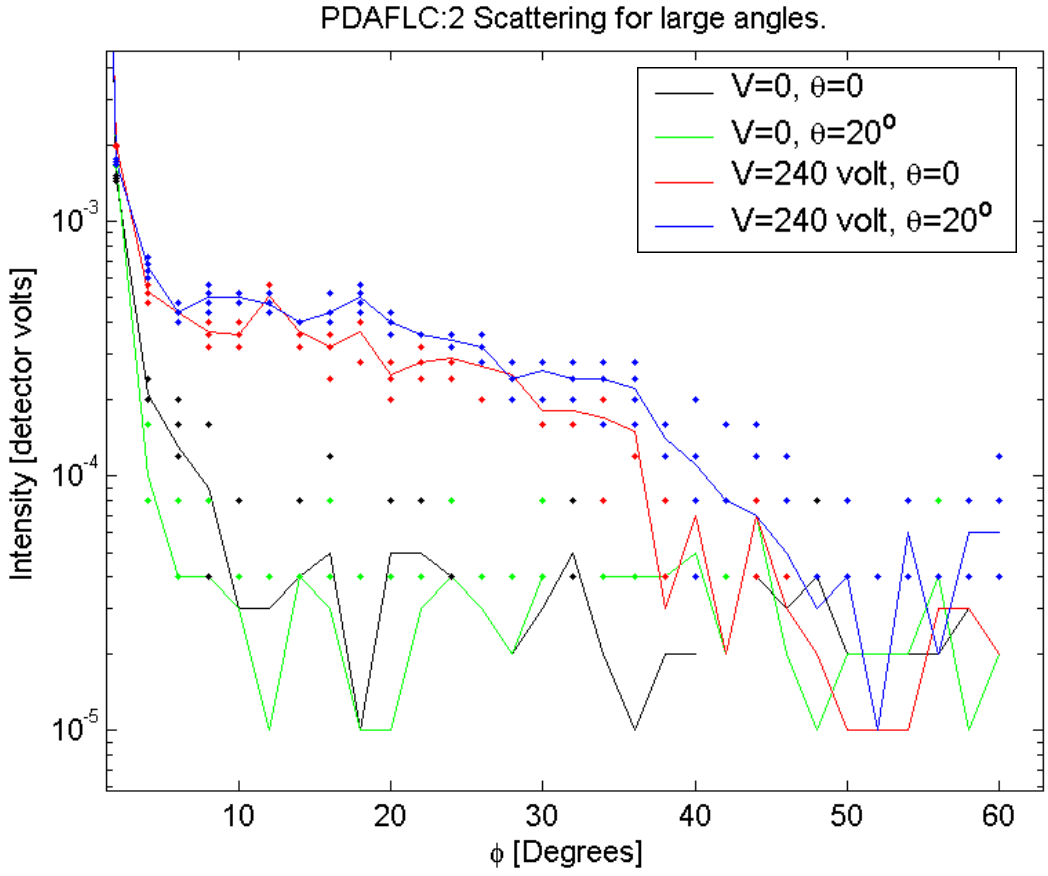
One interesting option to control the size of the scattering object is to use the inverted system with spheres dispersed in an OAFLC material.

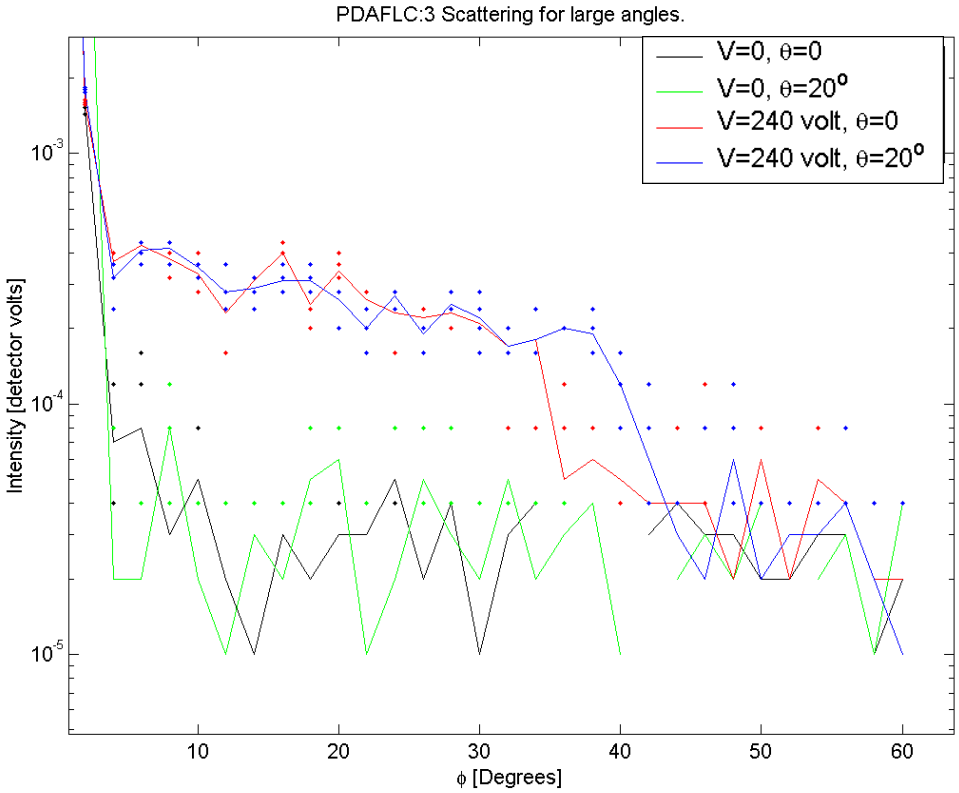
# References

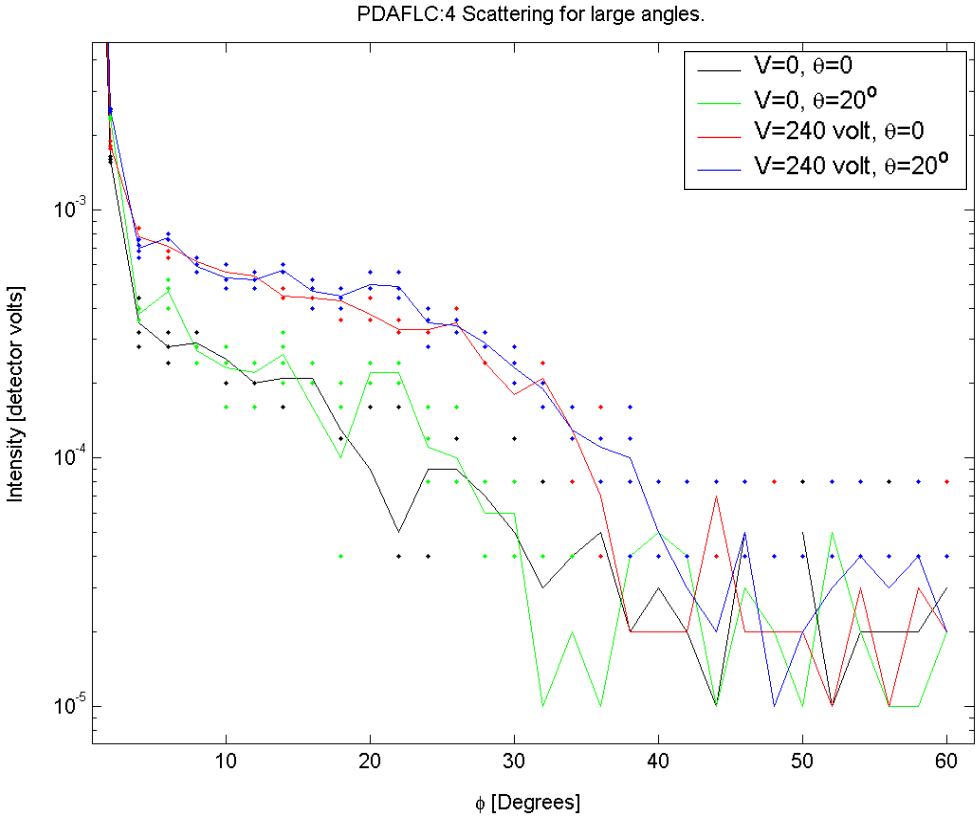
[i] Moberg, Wiss (eds.), *FOI orienterar om elektromagnetiska vapen och skydd*, FOI, Stockholm 2001
[ii] Eriksson, Lopes, Lindgren, Svensson, McKay, Davy, *Nonlinear Optics* **25**, 297-302 (2000)
[iii] Eliasson, Evekull, Kariis, *Laserskydd med vätskekristallkomponenter*, FOA-R--00-01537-612--SE, FOI 2000
[iv] S.Elston, R.Sambles (eds.) *The optics of thermotropic liquid crysrals*, Taylor & Francis 1998
[v] H.C.van de Hulst, *Light scattering from small particles*, wiely, 1957.
[vi] S.Zumer, Light scattering from Nematic Droplets: Anomalous-Diffraction Approach, Phys. Rev. A, 34, pp3373, 1986.
[vii] K.D'havé, Thesis, Chapter 6. *Light scattering polymer dispersions of OAFLC* 2001.

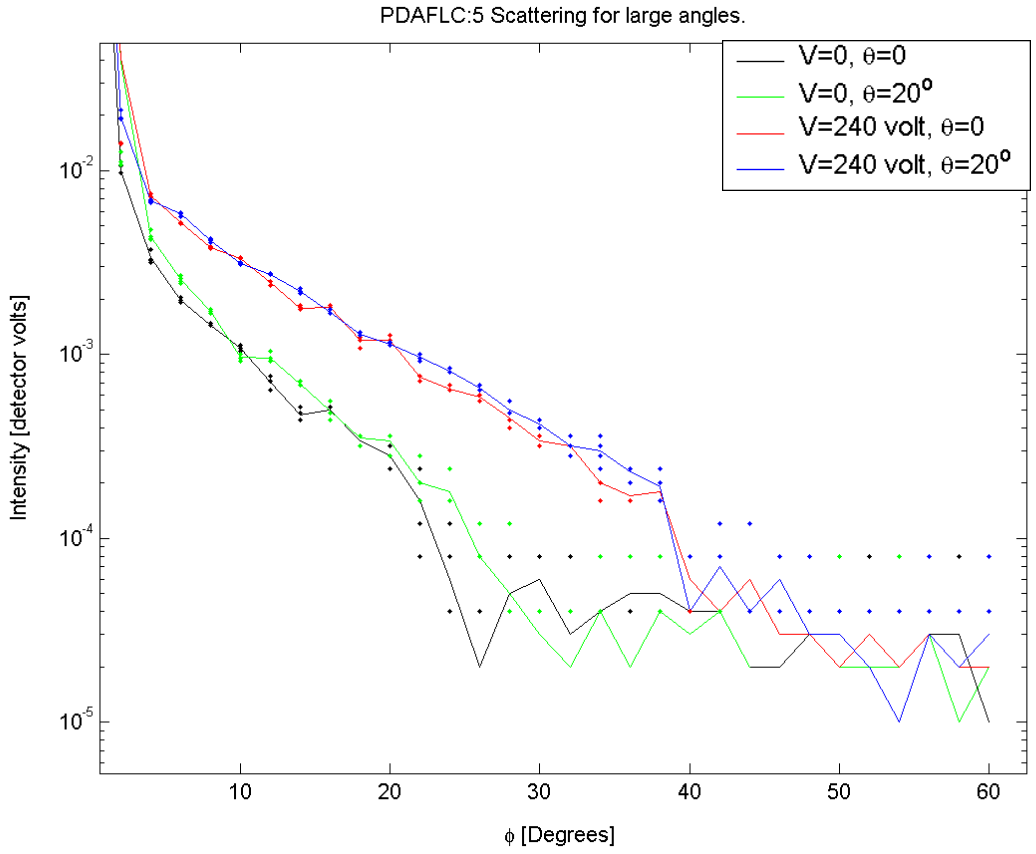
# Appendix 1. Scattering for large angles

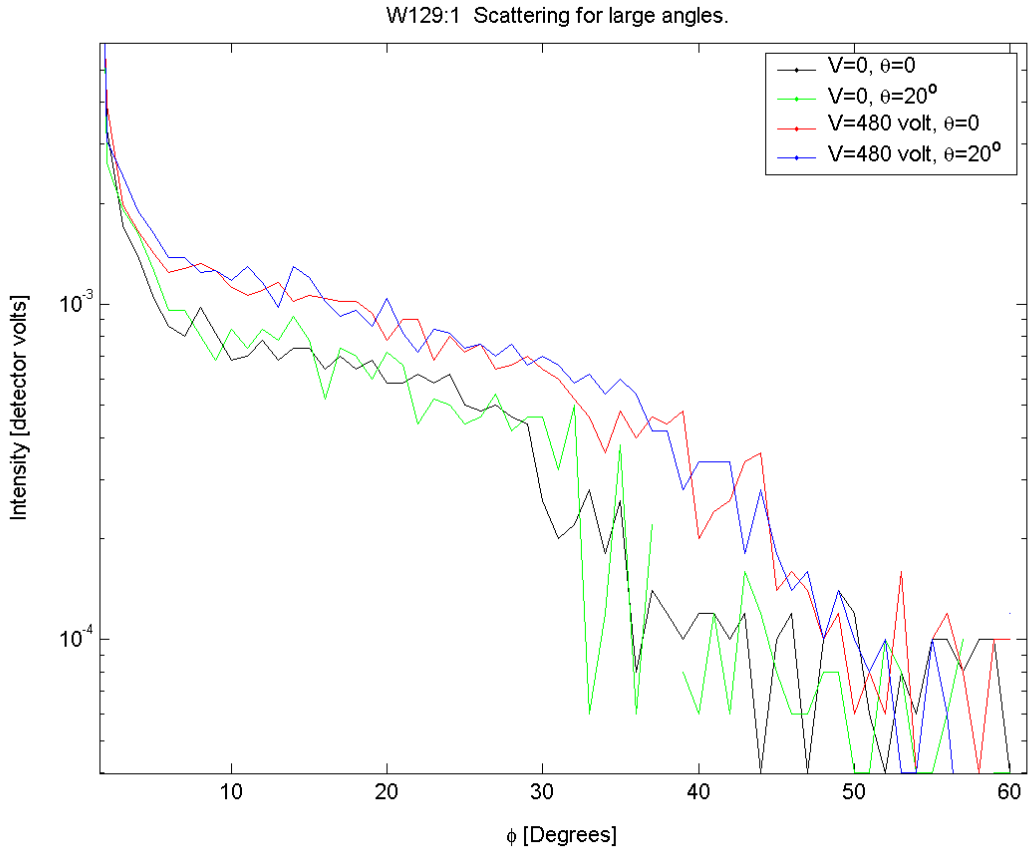


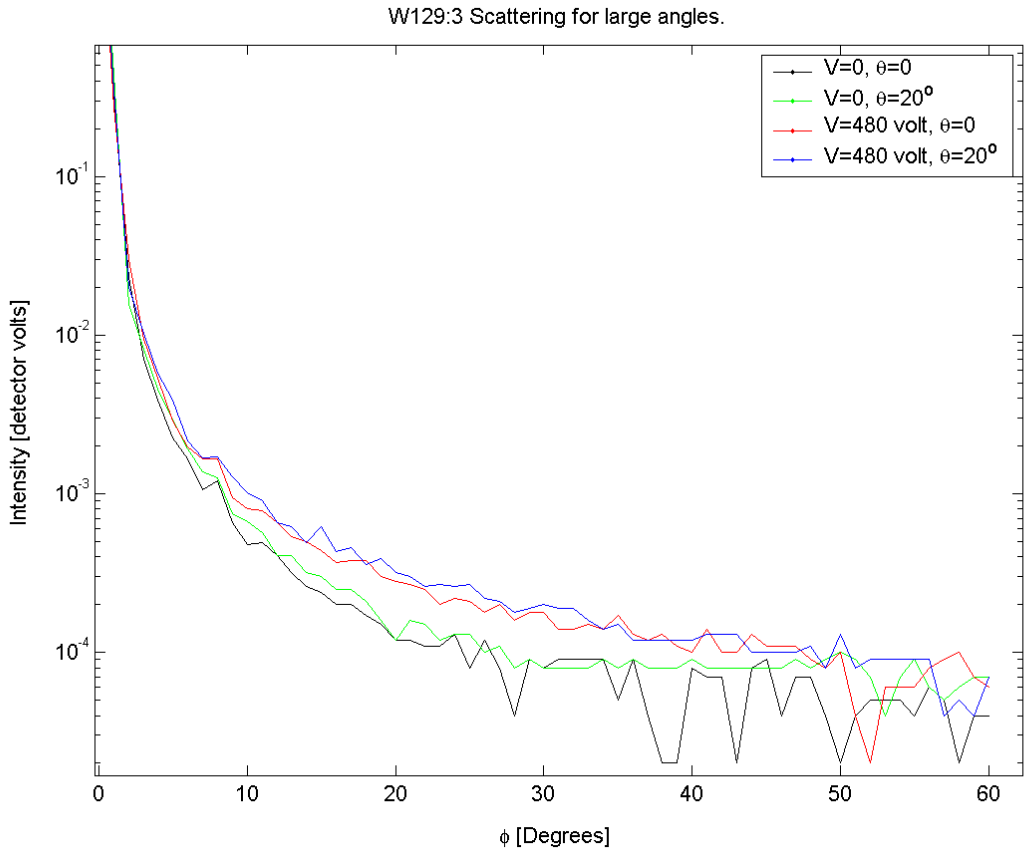




PDAFLC:3 Scattering for large angles.
V=0, θ=0
V=0, θ=20°
V=240 volt, θ=0
V=240 volt, θ=20°
Intensity [detector volts]
10^-3
10^-4
10^-5
10
20
30
40
50
60
ϕ [Degrees]

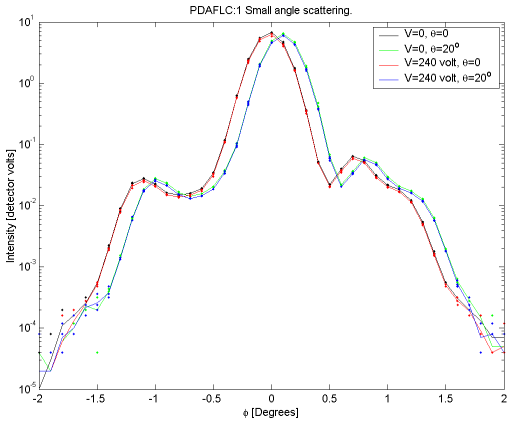

PDAFLC:4 Scattering for large angles.
V=0, θ=0
V=0, θ=20°
V=240 volt, θ=0
V=240 volt, θ=20°
Intensity [detector volts]
10^-3
10^-4
10^-5
10
20
30
40
50
60
ϕ [Degrees]

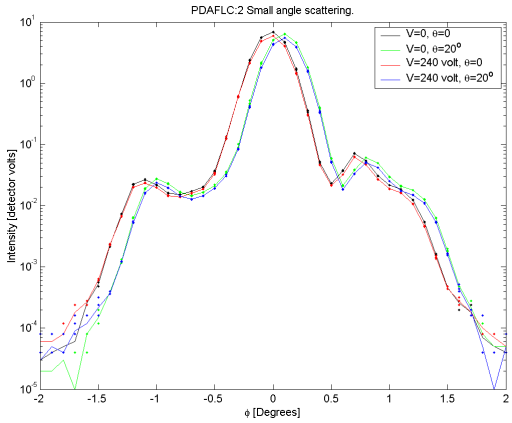

PDAFLC:5 Scattering for large angles.
V=0, θ=0
V=0, θ=20°
V=240 volt, θ=0
V=240 volt, θ=20°
Intensity [detector volts]
10^-2
10^-3
10^-4
10^-5
10
20
30
40
50
60
ϕ [Degrees]

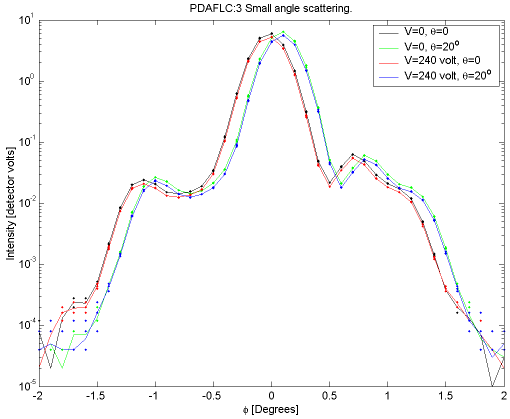

W129:1 Scattering for large angles.
V=0, θ=0
V=0, θ=20°
V=480 volt, θ=0
V=480 volt, θ=20°
Intensity [detector volts]
10^-3
10^-4
10
20
30
40
50
60
ϕ [Degrees]

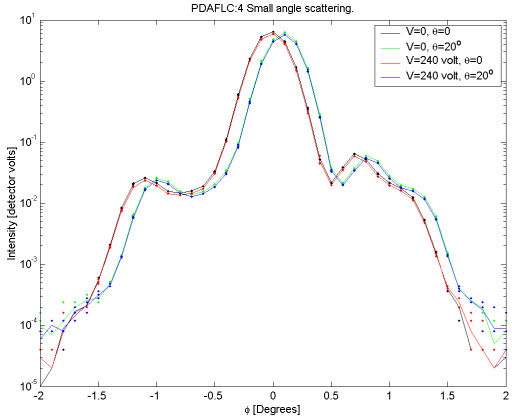

W129:3 Scattering for large angles.
V=0, θ=0
V=0, θ=20°
V=480 volt, θ=0
V=480 volt, θ=20°
Intensity [detector volts]
10-1
10-2
10-3
10-4
0
10
20
30
40
50
60
ϕ [Degrees]

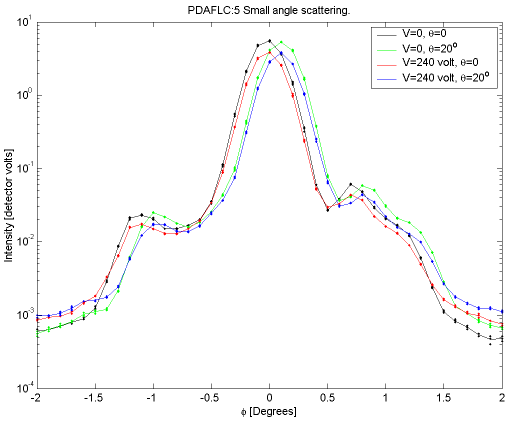
# Appendix 2. Scattering for small angles

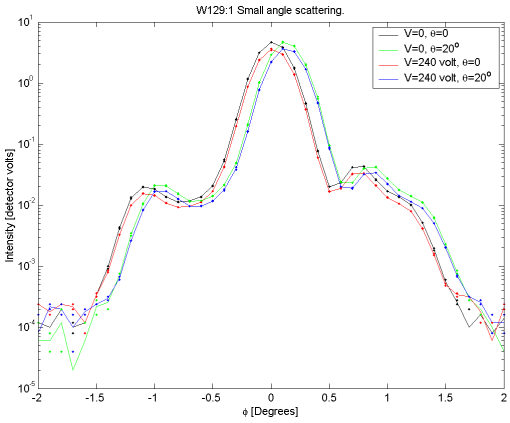


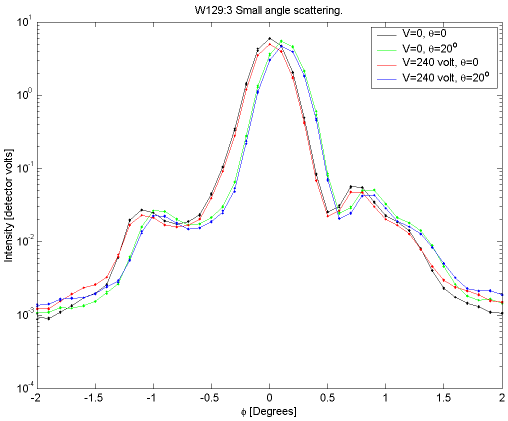



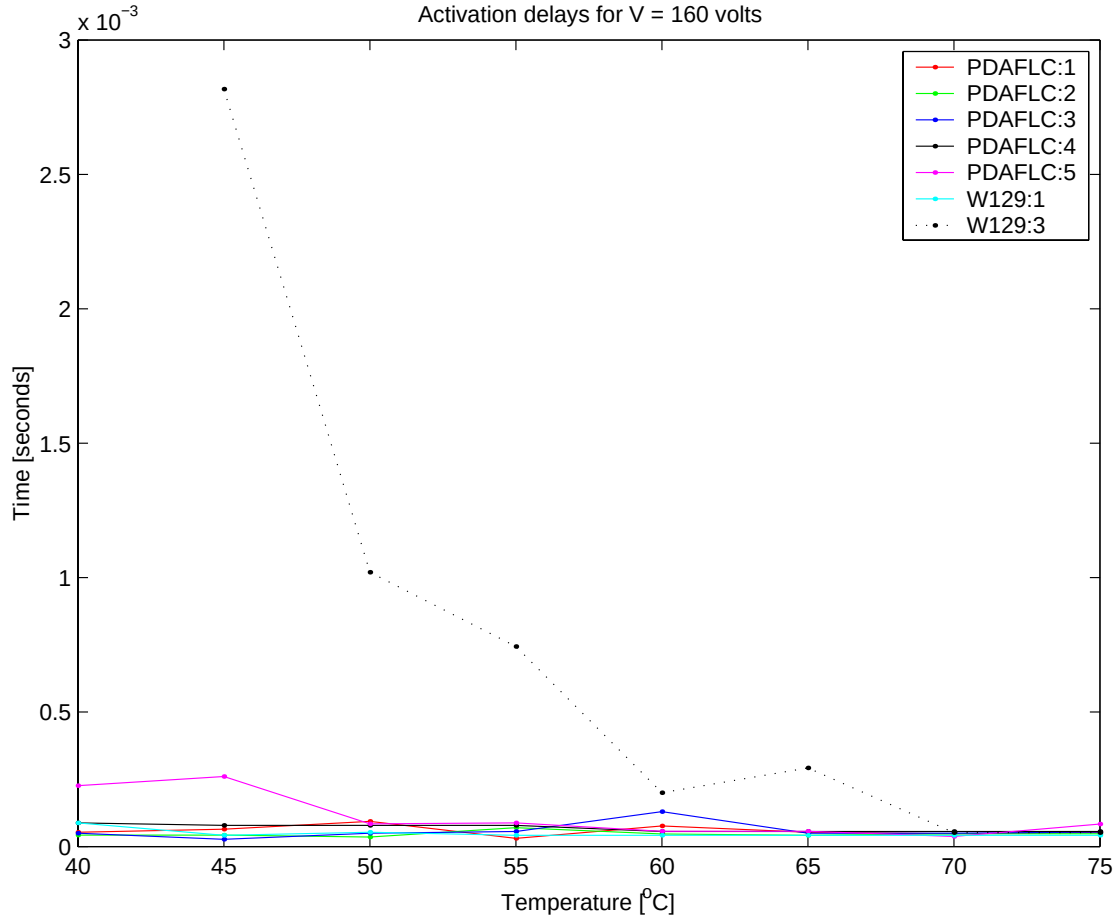



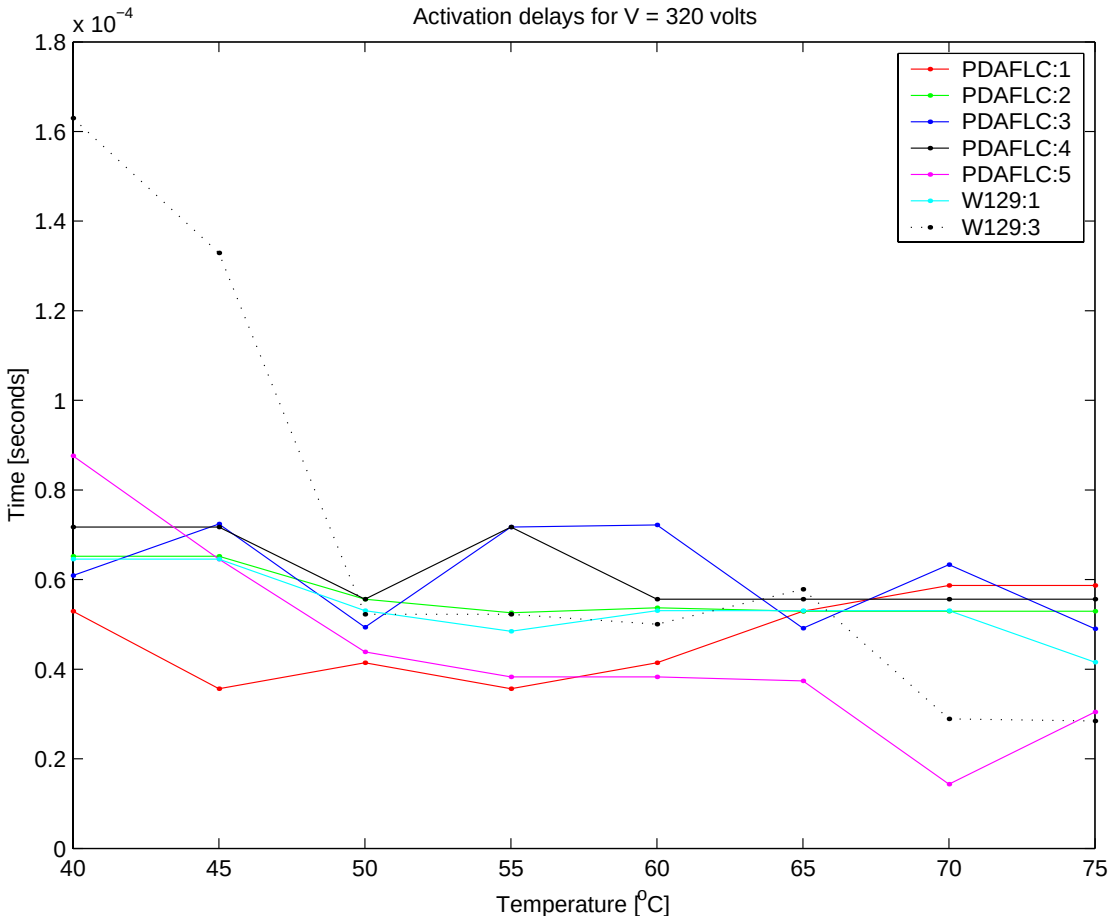

PDAFLC:5 Small angle scattering.
V=0, θ=0
V=0, θ=20°
V=240 volt, θ=0
V=240 volt, θ=20°
Intensity [detector volts]
ϕ [Degrees]

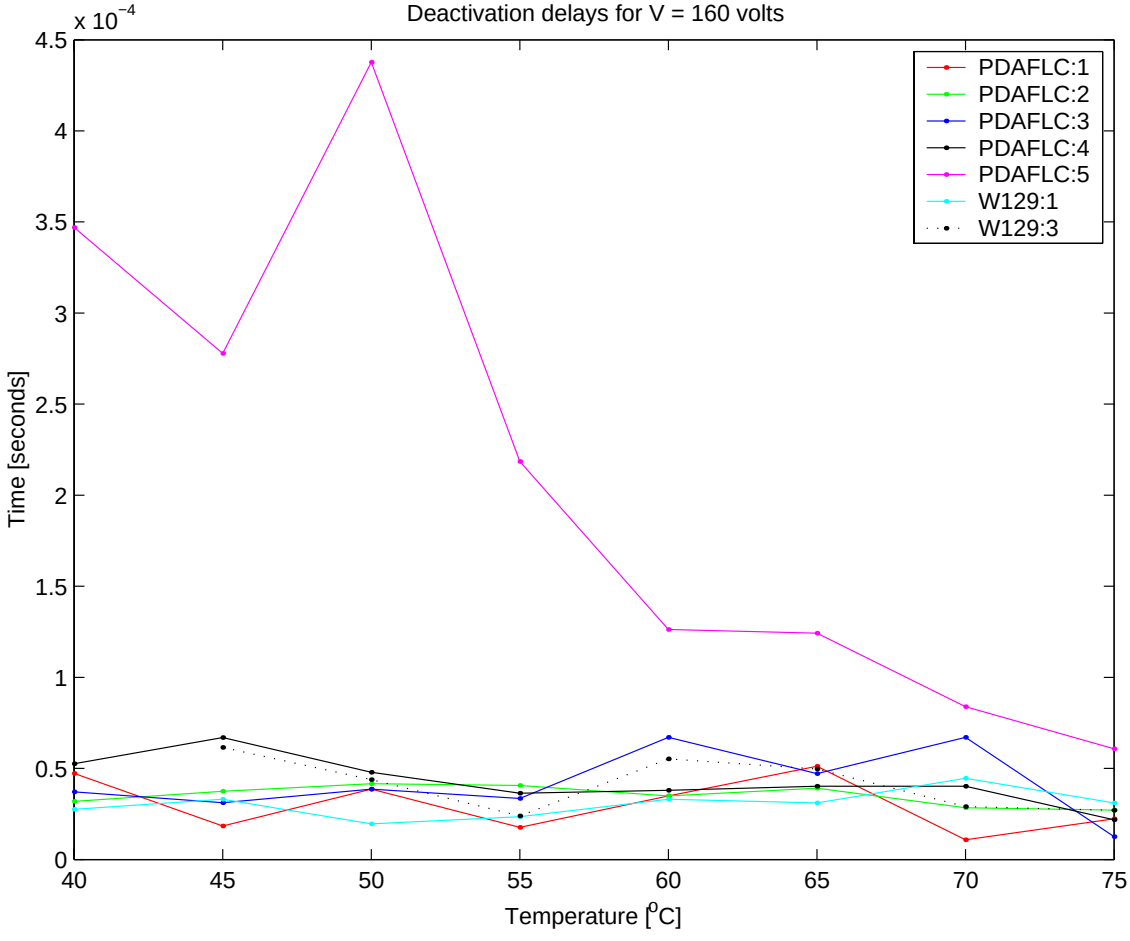

W129:1 Small angle scattering.
V=0, θ=0
V=0, θ=20°
V=240 volt, θ=0
V=240 volt, θ=20°
Intensity [detector volts]
ϕ [Degrees]

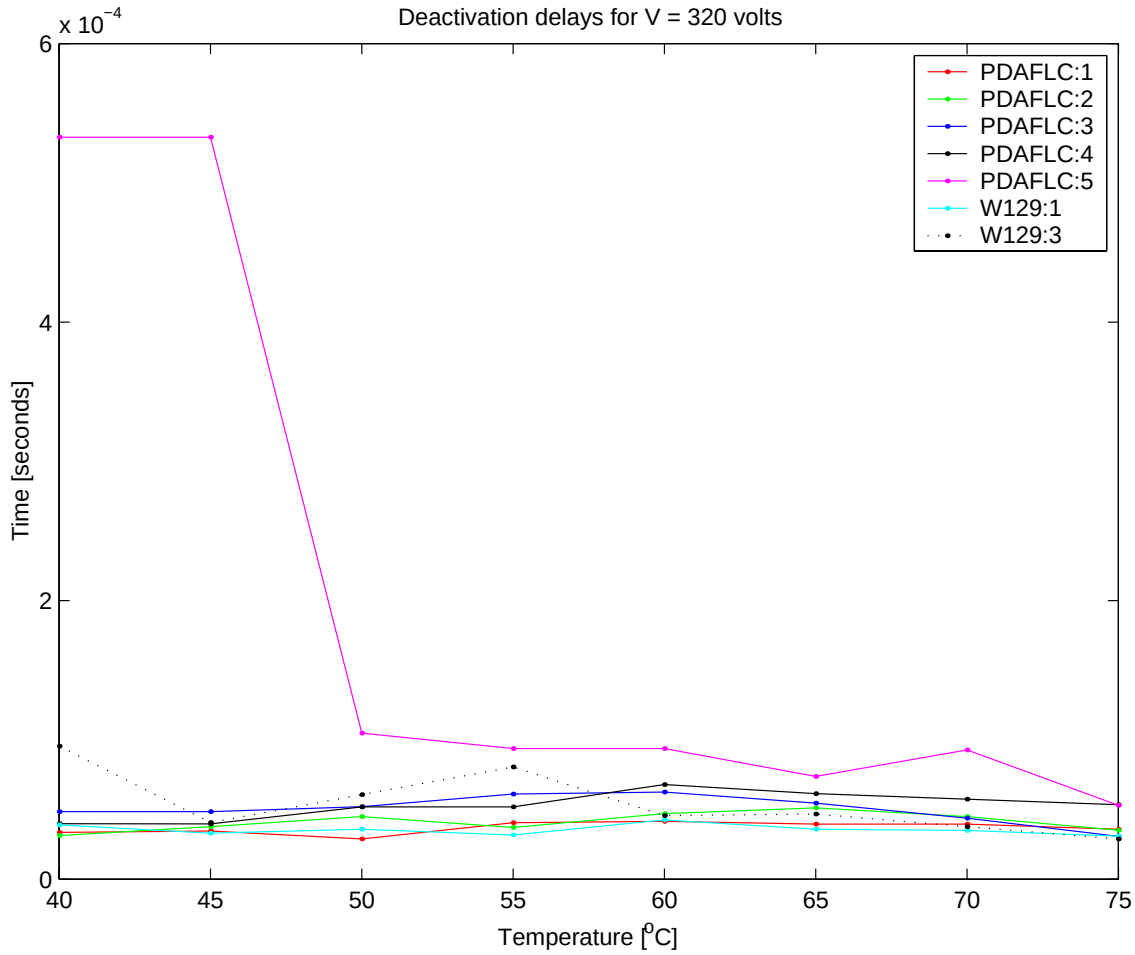

W129:3 Small angle scattering.
V=0, θ=0
V=0, θ=20°
V=240 volt, θ=0
V=240 volt, θ=20°
Intensity [detector volts]
ϕ [Degrees]

# Appendix 3. Characteristic response delays




Activation delays for V = 320 volts
x 10^-4
Time [seconds]
Temperature [°C]
PDAFLC:1
PDAFLC:2
PDAFLC:3
PDAFLC:4
PDAFLC:5
W129:1
W129:3


Deactivation delays for V = 160 volts
x 10^-4
Time [seconds]
Temperature [°C]
PDAFLC:1
PDAFLC:2
PDAFLC:3
PDAFLC:4
PDAFLC:5
W129:1
W129:3


Deactivation delays for V = 320 volts
x 10^-4
6
4
2
0
Time [seconds]
40
45
50
55
60
65
70
75
Temperature [°C]
PDAFLC:1
PDAFLC:2
PDAFLC:3
PDAFLC:4
PDAFLC:5
W129:1
W129:3

# Appendix 4. Source code for measurement automation

The programs have been implemented as a three level architecture with the two lower levels implemented in C and the user level as Matlab functions. The first layer defines some general functions for communicating with the instruments. Below is a table of the supported functions:

| Instrument | Implemented functions |
|---|---|
| Tektronix TDS200-series oscilloscopes | Automatic pk2pk measurement on any channel. (Includes automatic volts/div scaling.)<br>Waveform acquisition |
| Tektronix TDS400-series oscilloscopes | Automatic pk2pk and 'amplitude' ('smart' pk2pk) measurement on any channel. (Includes automatic volts/div scaling.) |
| Newport motion controller model MM3000 | Set/get angle on any of four step motors.<br>Waiting for motors to finish |
| Stanford Research Systems DS335 signal generator | Amplitude acquisition/control<br>Frequency acquisition/control |

The code is organised in such a way that it is easy to add new functions and/or instruments. The only immediate limitation is that it is currently awkward to control several instruments of the same type, and that the utility programs require a recompilation if the GPIB addresses of the instruments change. An object oriented approach might be better if one wishes to address these issues.

If you want to reuse this code for another set-up follow the steps below:

1) Decide which instrument to use and use the GPIB-spy program from National Instruments to get their GPIB addresses.
2) Check if the functions you need are included in the code below, otherwise you have to extend the routines.
3) Edit src/gpibconf.h to reflect the instrument addresses from 1).
4) Write a small C program to perform the measurements. It might be easiest to modify one of the existing programs like "src/angle.c".
5) Compile the program using either the included Makefile or using the Visual C++ environment.
6) Test the program by itself, and then write a small matlab function to call it. Take a look at "matlab/gpib_angle.m" for an example.

```
---------------------files:----------------------------------------------------------

gpib/
     README                    Compilation and usage instructions.
     src/                      Contains the C source code.
         GPIB.h                Basic initialization functions.
         GPIB.c
         gpibconf.h            Contains adresses to the instruments.
         TDS200.h              Interface to TDS200-series oscilloscopes.
         TDS200.c
         TDS400.h              Interface to TDS400-series oscilloscopes.
         TDS400.c
         MM3000.h              Interface to Newport motion controller MM3000.
         MM3000.c
         DS335.h               Interface to Stanford research systems signal generator.
         DS335.c
         curve.c               Reads a waveform from the TDS200 oscilloscope.
         angle.c               Sets angles on axis 1 and 2 on the MM3000 and reads the amplitude on
                               channel 2 of the TDS400.
         voltage.c             Sets the amplitude on the DS335 and reads the amplitude on channel 2 of the
                               TDS400.
         angle_voltage.c       Sets angles on axis 1 and 2 on the MM3000 and the amplitude on the DS335
                               and reads the amplitude on channel 2 of the TDS400.
         Makefile              Compiles and builds the executable and places them in bin/.
```

```
matlab/                   This directory contains functions for use in matlab. See
                          the help text inside the functions for further information.
      extern.m            Used by the other functions to call external programs.
      comb.m              Prepares a matlab matrix for output to the external programs.
      gpib_curve.m        Gets a waveform from the TDS200.
      gpib_voltage.m      Calls voltage.exe
      gpib_scatter.m      Calls angle.exe
      gpib_scatter_voltage.m Calls a_v.exe
      gpib_path.m         Returns the path to the gpib directory.

bin/                      These files are built automatically by the Makefile.

    curve.exe             Built from curve.c

    voltage.exe           Built from voltage.c
    angle.exe             Built from angle.c
    a_v.exe               Built from angle_voltage.c
```

----------src/Makefile----------

```
# Ulf Ekström 2001.06.05-2001.07...
# Makefile för att bygga spridningsmätningsprogrammen.
# 'angle' vrider båda axlarna, 'voltage' sätter spänningen.
# alla programmen läser av amplituden från oscilloskopet.
#
# Kom ihåg att köra vcvars32.bat först, annars hittar den inte programmen. Vcvars32 ligger i
# Visual c++-katalogen. Gör sedan
# cd src
# nmake
# för att bygga programmen och automatiskt flytta dem till rätt katalog, eller
# nmake curve.exe
# för att bygga curve.exe (etc..)


CC=cl # kompilator som ska användas.
GPIB_INCLUDE="..\Microsoft C" #sökväg till includefilen från National Instruments
GPIB_OBJ="..\Microsoft C\Gpib-32.obj" #sökväg till objektfilen -"-

install: angle.exe voltage.exe a_v.exe curve.exe
       copy voltage.exe ..\bin
       copy angle.exe ..\bin
       copy a_v.exe ..\bin
       copy curve.exe ..\bin
       nmake clean

angle.exe: GPIB.c TDS400.c MM3000.c angle.c GPIB.h TDS400.h MM3000.h GPIB.obj TDS400.obj MM3000.obj
       $(CC) /I $(GPIB_INCLUDE) /o "angle.exe" angle.c GPIB.obj TDS400.obj MM3000.obj $(GPIB_OBJ)

voltage.exe: GPIB.c TDS400.c DS335.c voltage.c GPIB.h TDS400.h DS335.h GPIB.obj TDS400.obj DS335.obj
       $(CC) /I $(GPIB_INCLUDE) /o "voltage.exe" voltage.c GPIB.obj TDS400.obj DS335.obj $(GPIB_OBJ)

a_v.exe: MM3000.c GPIB.c TDS400.c DS335.c angle_voltage.c GPIB.h TDS400.h DS335.h MM3000.h GPIB.obj
TDS400.obj DS335.obj MM3000.obj
       $(CC) /I $(GPIB_INCLUDE) /o "a_v.exe" angle_voltage.c GPIB.obj TDS400.obj DS335.obj MM3000.obj
$(GPIB_OBJ)

curve.exe: GPIB.c GPIB.h TDS200.c TDS200.h curve.c GPIB.obj TDS200.obj
       $(CC) /I $(GPIB_INCLUDE) /o "curve.exe" curve.c GPIB.obj TDS200.obj $(GPIB_OBJ)


GPIB.obj: GPIB.c GPIB.h
       $(CC) /c /I $(GPIB_INCLUDE) GPIB.c

TDS400.obj: TDS400.c TDS400.h
       $(CC) /c /I $(GPIB_INCLUDE) TDS400.c

TDS200.obj: TDS200.c TDS200.h
       $(CC) /c /I $(GPIB_INCLUDE) TDS200.c

DS335.obj: DS335.c DS335.h
       $(CC) /c /I $(GPIB_INCLUDE) DS335.c

MM3000.obj: MM3000.c MM3000.h
       $(CC) /c /I $(GPIB_INCLUDE) MM3000.c


clean:
       del *.obj
       del *.exe
```

----------src/GPIB.h----------

```
#ifndef GPIB_H
#define GPIB_H

void GPIBCleanup(int ud, char* ErrorMsg);
```

```
int GPIB_init(int board_nr);
int GPIB_err();
void GPIB_exit();

//Letar efter och skriver ut alla anslutna instrument på stderr
void GPIB_find_instruments();

#endif

---------------------src/GPIB.c-----------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <windows.h>
#include "Decl-32.h"
#include "GPIB.h"


Addr4882_t ud;
int Num_Instruments,            // Number of instruments on GPIB
    PAD,                        // Primary address
    SAD,                        // Secondary address
    loop,                       // Loop counter
    GPIBID;
Addr4882_t Instruments[32],            // Array of primary addresses
           Result[31];                 // Array of listen addresses
char ErrorMnemonic[21][5] = {"EDVR", "ECIC", "ENOL", "EADR", "EARG",
                             "ESAC", "EABO", "ENEB", "EDMA", "",
                             "EOIP", "ECAP", "EFSO", "", "EBUS",
                             "ESTB", "ESRQ", "", "", "", "ETAB"};



void GPIBCleanup(int ud, char* ErrorMsg)
{
    fprintf(stderr,"Error : %s\nibsta = 0x%x iberr = %d (%s)\n",
            ErrorMsg, ibsta, iberr, ErrorMnemonic[iberr]);
    fprintf(stderr,"Cleanup: Taking board offline\n");
    ibonl (ud, 0);
}

int GPIB_init(int board_nr)
{
  GPIBID = board_nr;
  SendIFC(GPIBID);
    if (ibsta & ERR)
    {
       GPIBCleanup(GPIBID, "Unable to open board");
       return -1;
    } else return 0;
}

int GPIB_err()
{
  if (ibsta & ERR)
    return -1;
  else
    return 0;
}

void GPIB_exit()
{
  ibonl(GPIBID, 0);
}

//Letar efter och skriver ut alla hittade instrument på stderr
void GPIB_find_instruments()
{
  for (loop = 0; loop < 30; loop++) {
    Instruments[loop] = (Addr4882_t)(loop + 1);
  }
  Instruments[30] = NOADDR;

  fprintf(stderr,"Finding all instruments on the bus...\n\n");

  FindLstn(GPIBID, Instruments, Result, 31);
  if (ibsta & ERR)
    {
      GPIBCleanup(GPIBID, "Unable to issue FindLstn call");
      return;
    }
  Num_Instruments = ibcntl;
  fprintf(stderr,"Number of instruments found = %d\n\n", Num_Instruments);
  Result[Num_Instruments] = NOADDR;
  for (loop = 0; loop < Num_Instruments; loop++)
    {
```

```
      PAD = GetPAD(Result[loop]);
      SAD = GetSAD(Result[loop]);

      if (SAD == NO_SAD)
        {
          fprintf(stderr,"The instrument at Result[%d]: PAD = %d SAD = NONE\n",
                loop, PAD);
        }
      else
        {
          fprintf(stderr,"The instrument at Result[%d]: PAD = %d SAD = %d\n",
                loop, PAD, SAD);
        }
    }
}
```

----------src/gpibconf.h----------

```
#ifndef GPIBCONF_H
#define GPIBCONF_H

/* GPIB adresses of the various instruments used by the different programs.
   You need to recompile for changes in this file to take effect! */


#define TDS200_PAD 1
#define TDS200_SAD 0

#define TDS400_PAD 7
#define TDS400_SAD 0

#define MM3000_PAD 2
#define MM3000_SAD 0

#define DS335_PAD 22
#define DS335_SAD 0

/* Id nr of the gpib interface card, usually 0 for the first card. */
#define GPIB_ID 0

#endif
```

----------src/TDS200.h----------

```
/*
  Kod för att kommunicera med 'Tektronix TDS-200'-oscilloskop.
*/

#ifndef TDS200_H
#define TDS200_H

int TDS200_init(int board,int PAD,int SAD);

// ger/sätter volt/div för en viss kanal
float TDS200_get_scale(int channel);
void TDS200_set_scale(int channel,float scale);

// ger/sätter 0-nivån i rutor för en viss kanal.
float TDS200_get_pos(int channel);
void TDS200_set_pos(int channel,float pos);


//förbereder mätning av pk2pk på en viss kanal
void TDS200_init_pk2pk(int channel);

// Ger nuvarande immediate measurement (pk2pk om den initierats) på kanalen
// som valdes med init_pk2pk.
// Ska ge svaret i volt, dock inte automatiskt ännu. (beror på osc. inställn.)
// zoomar ut/in automatiskt. Dock måste signalens bas ligga vid 0-nivån för att det
// ska funka bra.
float TDS200_measure();

float TDS200_waveform_xincr();

unsigned char *TDS200_waveform_data();

int TDS200_aquire_waveform(int channel);

#endif
```

----------src/TDS200.c----------

```
#include "TDS200.h"
#include <windows.h>
#include "decl-32.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include <assert.h>

//hur länge man ska vänta på att mätvärdet stabiliserat sig,
//används i pk2pk mätningen. 1500 ms funkar bra empiriskt, men om zoomen inte
//funkar bra så ökas lämpligen värdet till 2000 ms eller nåt.
#define TDS200_MEASUREMENT_DELAY 2000

extern void GPIBCleanup(int ud, char* ErrorMsg);

Addr4882_t TDS200_ud;
int TDS200_board;

float ranges[13] = {1e-3f,2e-3f,5e-3f,10e-3f,20e-3f,50e-3f,1e-1f,2e-1f,5e-1f,1,2,5,1e19f};

unsigned char waveform[2501];


//tar ut det värde ut ranges som ligger närmast value.
float _get_closest(float value)
{
       int i = 1;
       while (ranges[i] < value) i++;
       if (fabs(ranges[i] - value) < fabs(ranges[i-1] - value))
              return ranges[i];
       else return ranges[i-1];
}

float _get_upper(float value)
{
       int i = 1;
       while (ranges[i] < value) i++;
       return ranges[i];
}

int TDS200_init(int board,int PAD,int SAD)
{
  TDS200_board = board;
  TDS200_ud = ibdev(board,PAD,SAD,T10s,1,0);
  ibclr(TDS200_ud);
  if (ibsta & ERR)
    {
      GPIBCleanup(board, "Error in TDS200_init()\n");
      return -1;
    } else return 0;
}

void TDS200_autoset()
{
  ibwrt(TDS200_ud,"AUTO",4);
  _sleep(TDS200_MEASUREMENT_DELAY);
}


// ger volt/div för en viss kanal
float TDS200_get_scale(int channel)
{
  char buffer[100];
  float result;
  sprintf(buffer,"CH%i:SCALE?",channel);
  ibwrt(TDS200_ud,buffer,strlen(buffer));
  ibrd(TDS200_ud,buffer,100);
  if (sscanf(buffer,"%f",&result)<1)
    {
      fprintf(stderr,"Error in TDS200_get_scale(pk2pk_channel)\n");
    }
  return result;
}

// väljer automatiskt tillåtna värden på scale
void TDS200_set_scale(int channel,float scale)
{
  char buffer[100];
  sprintf(buffer,"CH%i:SCALE %f",channel,_get_closest(scale));
  ibwrt(TDS200_ud,buffer,strlen(buffer));
  _sleep(TDS200_MEASUREMENT_DELAY);
}


// ger 0-nivån i rutor för en viss kanal.
float TDS200_get_pos(int channel)
{
  char buffer[100];
  float result;
  sprintf(buffer,"CH%i:POSITION?",channel);
  ibwrt(TDS200_ud,buffer,strlen(buffer));
  ibrd(TDS200_ud,buffer,100);
  if (sscanf(buffer,"%f",&result)<1)
    {
      fprintf(stderr,"Error in TDS200_get_pos()\n");
```

```
    }
  return result;
}
void TDS200_set_pos(int channel,float scale)
{
  char buffer[100];
  sprintf(buffer,"CH%i:POS %f",channel,scale);
  ibwrt(TDS200_ud,buffer,strlen(buffer));
  _sleep(TDS200_MEASUREMENT_DELAY);
}



int pk2pk_channel = -1;
int reclevel = 0;


void TDS200_init_pk2pk(int channel)
{
  char buffer[100];
  sprintf(buffer,"MEASUREMENT:IMMED:SOURCE CH%i",channel);
  ibwrt(TDS200_ud,buffer,strlen(buffer));
  ibwrt(TDS200_ud,"MEASUREMENT:IMMED:TYPE PK2PK",29);
  TDS200_set_pos(channel,-3.5);
  pk2pk_channel = channel;
  reclevel = 0;
}

// hjälpfunktion, läser pk2pk-värdet rakt av.
float _pk2pk()
{
  char buffer[100];
  float result;
  ibwrt(TDS200_ud,"MEASUREMENT:IMMED:VALUE?",24);
  ibrd(TDS200_ud,buffer,100);
  if (sscanf(buffer,"%f",&result)<1)
    {
      fprintf(stderr,"Error in TDS200_pk2pk()\n");
    }
  return result;
}



// TODO: Gör mer generell, nu funkar den bara med
// positiva signaler. Förutsätter att nollan ligger vid -3.5 rutor,
// men det fixas i init.

float TDS200_measure()
{
  float pk2pk;
  float scale;
  reclevel++;
  _sleep(TDS200_MEASUREMENT_DELAY);
  pk2pk = _pk2pk();
  scale = TDS200_get_scale(pk2pk_channel);
  if (pk2pk > scale*6.5) //om vi fyller ut mer än 6 rutor ska vi zooma ut
    {
      if (scale >= 5)
        {
          fprintf(stderr,"Signal out of range in TDS200_pk2pk(). Sorry.\n");
          return 0;
        }
      TDS200_set_scale(pk2pk_channel,scale*5.0f);
      return TDS200_measure();
    } else
      if ((pk2pk < scale*3) && (scale > ranges[1]) && reclevel < 15) //om vi fyller ut mindre än 3 rutor
ska vi zooma in
        {
          if ((_get_upper(pk2pk/6.5f) != scale)&&(_get_upper(pk2pk/6.5f)*6 > pk2pk)) //zoomar bara in om
vi inte behöver zooma ut direkt sen.
            {
              TDS200_set_scale(pk2pk_channel,_get_upper(pk2pk/6.5f));
              return TDS200_measure();
            } else
              {
                reclevel = 0;
                return pk2pk;
              }
        }
  reclevel = 0;
  return pk2pk;
}


unsigned char *TDS200_waveform_data()
{
  return waveform;
```

```
}

//returnerar tid/sample i sekunder.
float TDS200_waveform_xincr()
{
  char buffer[20];
  float f;
  buffer[19] = 0;
  ibwrt(TDS200_ud,"WFMP:XIN?",9);
  ibrd(TDS200_ud,buffer,19);
  assert(sscanf(buffer,"%f",&f));
  return f;
}

//returns nr samples read
int TDS200_aquire_waveform(int channel)
{
  char buffer[20];
  int i;
  ibwrt(TDS200_ud,"DAT:WID 1",9); //en byte per punkt
  sprintf(buffer,"DAT:SOU CH%i",channel);
  ibwrt(TDS200_ud,buffer,strlen(buffer));
  ibwrt(TDS200_ud,"DAT:ENC RPB",11);//kör binär, unsigned 8 bit/sample
  ibwrt(TDS200_ud,"DAT:STAR 1",10);
  ibwrt(TDS200_ud,"DAT:STOP 2500",13);
  ibwrt(TDS200_ud,"CURV?",5);
  ibrd(TDS200_ud,buffer,2); //läser antalet siffror i nästa tal.
  assert(sscanf(buffer,"#%i",&i));
  ibrd(TDS200_ud,buffer,i); //läser antalet samples
  assert(sscanf(buffer,"%i",&i));
  ibrd(TDS200_ud,(char *)waveform,i);
  return i;
}

----------------------src/TDS400.h----------------------------------------------------------------

/*
  Kod för att kommunicera med 'Tektronix TDS-200'-oscilloskop.
 */

#ifndef TDS400_H
#define TDS400_H

int TDS400_init(int board,int PAD,int SAD);

// ger/sätter volt/div för en viss kanal
float TDS400_get_scale(int channel);
void TDS400_set_scale(int channel,float scale);

// ger/sätter 0-nivån i rutor för en viss kanal.
float TDS400_get_pos(int channel);
void TDS400_set_pos(int channel,float pos);


//förbereder mätning av pk2pk på en viss kanal
void TDS400_init_pk2pk(int channel);

//förbereder mätning av amplituden på en viss kanal (finns inte hos TDS 200).
void TDS400_init_amplitude(int channel);

// Ger nuvarande immediate measurement (pk2pk om den initierats) på kanalen som valdes med init_pk2pk.
// Ska ge svaret i volt, dock inte automatiskt ännu. (beror på osc. inställn.)
// zoomar ut/in automatiskt. Dock måste signalens bas ligga vid 0-nivån för att det
// ska funka bra.
float TDS400_measure();

#endif

----------------------src/TDS400.c----------------------------------------------------------------

include <windows.h>
#include "decl-32.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>

//hur länge man ska vänta på att mätvärdet stabiliserat sig,
//används i pk2pk mätningen. 1500 ms funkar bra empiriskt, men om zoomen inte
//funkar bra så ökas lämpligen värdet till 2000 ms eller nåt.
#define TDS400_MEASUREMENT_DELAY 2000

extern void GPIBCleanup(int ud, char* ErrorMsg);

Addr4882_t TDS400_ud;
int TDS400_board;

float ranges[14] = {-1e19f,1e-3f,2e-3f,5e-3f,10e-3f,20e-3f,50e-3f,1e-1f,2e-1f,5e-1f,1,2,5,1e19f};
```

```
//tar ut det värde ut ranges som ligger närmast value.
float _get_closest(float value)
{
       int i = 1;
       while (ranges[i] < value) i++;
       if (fabs(ranges[i] - value) < fabs(ranges[i-1] - value))
              return ranges[i];
       else return ranges[i-1];
}

float _get_upper(float value)
{
       int i = 1;
       while (ranges[i] < value) i++;
       return ranges[i];
}

int TDS400_init(int board,int PAD,int SAD)
{
  TDS400_board = board;
  TDS400_ud = ibdev(board,PAD,SAD,T10s,1,0);
  ibclr(TDS400_ud);
  if (ibsta & ERR)
    {
      GPIBCleanup(board, "Error in TDS400_init()\n");
      return -1;
    } else return 0;
}

void TDS400_autoset()
{
  ibwrt(TDS400_ud,"AUTO",4);
  _sleep(TDS400_MEASUREMENT_DELAY);
}


// ger volt/div för en viss kanal
float TDS400_get_scale(int channel)
{
  char buffer[100];
  float result;
  sprintf(buffer,"CH%i:SCALE?",channel);
  ibwrt(TDS400_ud,buffer,strlen(buffer));
  ibrd(TDS400_ud,buffer,100);
  if (sscanf(buffer,"%f",&result)<1)
    {
      fprintf(stderr,"Error in TDS400_get_scale(pk2pk_channel)\n");
    }
  return result;
}

// väljer automatiskt tillåtna värden på scale
void TDS400_set_scale(int channel,float scale)
{
  char buffer[100];
  sprintf(buffer,"CH%i:SCALE %f",channel,_get_closest(scale));
  ibwrt(TDS400_ud,buffer,strlen(buffer));
  _sleep(TDS400_MEASUREMENT_DELAY);
}


// ger 0-nivån i rutor för en viss kanal.
float TDS400_get_pos(int channel)
{
  char buffer[100];
  float result;
  sprintf(buffer,"CH%i:POSITION?",channel);
  ibwrt(TDS400_ud,buffer,strlen(buffer));
  ibrd(TDS400_ud,buffer,100);
  if (sscanf(buffer,"%f",&result)<1)
    {
      fprintf(stderr,"Error in TDS400_get_pos()\n");
    }
  return result;
}
void TDS400_set_pos(int channel,float scale)
{
  char buffer[100];
  sprintf(buffer,"CH%i:POS %f",channel,scale);
  ibwrt(TDS400_ud,buffer,strlen(buffer));
  _sleep(TDS400_MEASUREMENT_DELAY);
}



int pk2pk_channel = -1;
int reclevel = 0;
```

```
void TDS400_init_pk2pk(int channel)
{
  char buffer[100];
  sprintf(buffer,"MEASUREMENT:IMMED:SOURCE CH%i",channel);
  ibwrt(TDS400_ud,buffer,strlen(buffer));
  ibwrt(TDS400_ud,"MEASUREMENT:IMMED:TYPE PK2PK",29);
  TDS400_set_pos(channel,-3.5);
  pk2pk_channel = channel;
  reclevel = 0;
}

void TDS400_init_amplitude(int channel)
{
  char buffer[100];
  sprintf(buffer,"MEASUREMENT:IMMED:SOURCE CH%i",channel);
  ibwrt(TDS400_ud,buffer,strlen(buffer));
  ibwrt(TDS400_ud,"MEASUREMENT:IMMED:TYPE AMPLITUDE",29);
  TDS400_set_pos(channel,-3.5);
  pk2pk_channel = channel;
  reclevel = 0;
}


// hjälpfunktion, läser measurement-värdet rakt av.
float _pk2pk()
{
  char buffer[100];
  float result;
  ibwrt(TDS400_ud,"MEASUREMENT:IMMED:VALUE?",24);
  ibrd(TDS400_ud,buffer,100);
  if (sscanf(buffer,"%f",&result)<1)
    {
      fprintf(stderr,"Error in TDS400_pk2pk()\n");
    }
  return result;
}



// TODO: Gör mer generell, nu funkar den bara med
// positiva signaler. Förutsätter att nollan ligger vid -3.5 rutor,
// men det fixas i init.

float TDS400_measure()
{
  float pk2pk;
  float scale;
  reclevel++;
  _sleep(TDS400_MEASUREMENT_DELAY);
  pk2pk = _pk2pk();
  scale = TDS400_get_scale(pk2pk_channel);
  //fprintf(stderr,"scale: %f\n",scale);
  //fprintf(stderr,"pk2pk: %f\n",pk2pk);
  if (pk2pk > scale*6.5) //om vi fyller ut mer än 6 rutor ska vi zooma ut
    {
      if (scale >= 5)
        {
          fprintf(stderr,"Signal out of range in TDS400_pk2pk(). Sorry.\n");
          return 0;
        }
      //fprintf(stderr,"zoomar ut\n");
      TDS400_set_scale(pk2pk_channel,scale*5.0f);
      return TDS400_measure();
    } else
      if ((pk2pk < scale*3) && (scale > ranges[1]) && reclevel < 15) //om vi fyller ut mindre än 3 rutor
ska vi zooma in
        {
          //fprintf(stderr,"zoomar in kanske\n");
          if ((_get_upper(pk2pk/6.5f) != scale)&&(_get_upper(pk2pk/6.5f)*6 > pk2pk)) //zoomar bara in om
vi inte behöver zooma ut direkt sen.
            {
              //             fprintf(stderr,"zoomar in på riktigt\n");
              TDS400_set_scale(pk2pk_channel,_get_upper(pk2pk/6.5f));
              return TDS400_measure();
            } else
              {
               reclevel = 0;
               return pk2pk;
              }
        }
  reclevel = 0;
  return pk2pk;
}

----------------------src/DS335.h----------------------------------------------------------------
```

```
/*
Kod för att styra Stanford Research Systems DS335, signalgenerator.
*/


#ifndef DS335_H
#define DS335_H

/* Initierar kommunikationen med DS335
Returnerar 0 om det gick bra. */
int DS335_init(int board,int PAD,int SAD);

/* gets peak to peak voltage */
float DS335_get_vp();
void DS335_set_vp(float v);


/* gets frequency in Hz */
float DS335_get_frequency();
void DS335_set_frequency(float f);

#endif

--------------------src/DS335.c----------------------------------------------------------------

#include "DS335.h"
#include <windows.h>
#include "decl-32.h"
#include <stdio.h>
#include <stdlib.h>

extern void GPIBCleanup(int ud, char* ErrorMsg);

Addr4882_t DS335_ud;
int DS335_board;


/* Initierar kommunikationen mot DS335, stegmotorkontroller */
int DS335_init(int board,int PAD,int SAD)
{
  DS335_board = board;
  DS335_ud = ibdev(board,PAD,SAD,T10s,1,0);
  ibclr(DS335_ud);
  if (ibsta & ERR)
    {
      GPIBCleanup(board, "Error in DS335_init()\n");
      return -1;
    } else return 0;
}



void DS335_set_vp(float v)
{
  static char buff[20]; //20 tecken borde räcka för flyttal..
  sprintf(buff,"AMPL %.2fVP",v);
  ibwrt(DS335_ud,buff,strlen(buff));
}

void DS335_set_frequency(float f)
{
  static char buff[20]; //20 tecken borde räcka för flyttal..
  sprintf(buff,"FREQ %.2f",f);
  ibwrt(DS335_ud,buff,strlen(buff));
}

float DS335_get_vp()
{
  float ampl;
  char buff[20];
  ibwrt(DS335_ud,"AMPL?",5);
  ibrd(DS335_ud,buff,40);
  if (sscanf(buff,"%fVP", &ampl)<1)
    {
      fprintf(stderr,"Error: DS335_get_angle() failed\n");
      return -1;
    }
  return ampl;
}

float DS335_get_frequency()
{
  float ampl;
  char buff[20];
  ibwrt(DS335_ud,"FREQ?",5);
  ibrd(DS335_ud,buff,40);
  if (sscanf(buff,"%f", &ampl)<1)
    {
      fprintf(stderr,"Error: DS335_get_angle() failed\n");
```

```
      return -1;
    }
  return ampl;
}
```

----------src/MM3000.h----------

```
/*
Kod för att styra 'Newport motion controller model MM3000' via GPIB.
*/


#ifndef MM3000_H
#define MM3000_h

/* Initierar kommunikationen med MM3000, stegmotorkontroller.
Returnerar 0 om det gick bra. */
int MM3000_init(int board,int PAD,int SAD);

/* Returnerar 0 om allt funkar som det skall, annars något annat */
int MM3000_err();

/* Returnerar nollskiljt om någon axel rör sig, annars 0 */
int MM3000_busy();

/* vinklar i grader */
void MM3000_move_absolute(int axis_nr,float angle);
void MM3000_move_relative(int axis_nr,float angle);
float MM3000_get_angle(int axis_nr);

/* Returnerar statusbyten, bit 0-3 beskriver axelstatusen */
unsigned char MM3000_get_status();

/* Funktionen väntar tills alla motorer stannat innan den returnerar*/
void MM3000_wait_all_axis();
#endif
```

----------src/MM3000.c----------

```
#include "MM3000.h"
#include <windows.h>
#include "decl-32.h"
#include <stdio.h>
#include <stdlib.h>

extern void GPIBCleanup(int ud, char* ErrorMsg);

Addr4882_t MM3000_ud;
int MM3000_board;


/* Initierar kommunikationen mot MM3000, stegmotorkontroller */
int MM3000_init(int board,int PAD,int SAD)
{
  MM3000_board = board;
  MM3000_ud = ibdev(board,PAD,SAD,T10s,1,0);
  ibclr(MM3000_ud);
  if (ibsta & ERR)
    {
      GPIBCleanup(board, "Error in MM3000_init()\n");
      return -1;
    } else return 0;
}


/* Returnerar 0 om allt funkar som det skall, annars något annat */
int MM3000_err()
{
  return (MM3000_get_status() & 16); //kollar om bit 5 (error) är satt
}

void MM3000_move_absolute(int axis_nr,float angle)
{
  static char buff[20]; //20 tecken borde räcka för flyttal..
  sprintf(buff,"%iUP%f",axis_nr,angle);
  ibwrt(MM3000_ud,buff,strlen(buff));
}

void MM3000_move_relative(int axis_nr,float angle)
{
  static char buff[20]; //20 tecken borde räcka för flyttal..
  sprintf(buff,"%iUR%f",axis_nr,angle);
  ibwrt(MM3000_ud,buff,strlen(buff));
}

/* Pollar tills alla axlar står stilla*/
void MM3000_wait_all_axis()
{
```

```
   while(MM3000_get_status() & 15) _sleep(200);
}

unsigned char MM3000_get_status()
{
  unsigned char buffer[5];
  ibwrt(MM3000_ud,"TS",2);
  ibrd(MM3000_ud,buffer,5);
  return buffer[0];
}

float MM3000_get_angle(int axis_nr)
{
  static char buff[40]; //20 tecken borde räcka för flyttal..
  float angle;
  sprintf(buff,"%iTP",axis_nr);
  ibwrt(MM3000_ud,buff,strlen(buff));
  ibrd(MM3000_ud,buff,40);
  if (sscanf(buff,"%f COUNTS", &angle)<1)
    {
      fprintf(stderr,"Error: MM3000_get_angle() failed\n");
      return -1;
    }
  return (float)(angle/1000.0);
}
```

----------src/curve.c----------

```
#include "stdio.h"
#include "stdlib.h"
#include "assert.h"
#include "GPIB.h"
#include "TDS200.h"
#include "gpibconf.h"

/* Reads an integer from stdin ('1' or '2') and gets the
 curve from the corresponding channel on the oscilloscope.
 The curve should be visible and properly aligned on the oscilloscope.
 Data is output in two columns, the first is time in seconds
 from the first sample, and the second is the integer
 sample value (0..255). /Ulf Ekström 2001-07-16 */

int main(int argc,char *argv[])
{
  unsigned char *dat;
  int i,j,k,channel;
  float t,dt;
  assert(!GPIB_init(GPIB_ID));
  assert(!TDS200_init(GPIB_ID,TDS200_PAD,TDS200_SAD));
  _sleep(500);
  assert(scanf("%i",&channel));
  i = TDS200_aquire_waveform(channel);
  dat = TDS200_waveform_data();
  dt = TDS200_waveform_xincr();
  t = 0;
  for (j=0;j<i;j++)
    {
      k = dat[j];
      printf("%f\t%i\n",t,k);
      t+=dt;
    }
}
```

----------src/angle.c----------

```
#include "stdio.h"
#include "stdlib.h"
#include "assert.h"
#include "GPIB.h"
#include "TDS400.h"
#include "MM3000.h"
#include "gpibconf.h"

/*Expects two floats on every line from stdin, first float is axis 1
  and second float is axis 2. Outputs the actual angles and oscilloscope
  amplitude on stdout */
int main(int argc,char *argv[])
{
       float u,v;

       assert(!GPIB_init(GPIB_ID));
       assert(!TDS400_init(GPIB_ID,TDS400_PAD,TDS400_SAD));

       //vi ska mäta amplitud på kanal 2:
       TDS400_init_amplitude(2);

       assert(!MM3000_init(GPIB_ID,MM3000_PAD,MM3000_SAD));

       //läser från stdin:
```

```
        while (scanf("%f%f",&u,&v)>1)
          {
            MM3000_move_absolute(1,u);
            MM3000_move_absolute(2,v);
            MM3000_wait_all_axis();
            printf("%f\t%f\t%f\n",MM3000_get_angle(1),MM3000_get_angle(2),TDS400_measure());
          }


        //går tillbaka till ursprungsläget
        MM3000_move_absolute(1,0);
        MM3000_move_absolute(2,0);
        GPIB_exit();
}
```

----------src/voltage.c----------

```
#include "stdio.h"
#include "stdlib.h"
#include "assert.h"
#include "GPIB.h"
#include "TDS400.h"
#include "DS335.h"
#include "gpibconf.h"

/*Expects one floats on every line from stdin, representing p-p voltage.
  Outputs the actual p-p voltage and oscilloscope amplitude on stdout */

int main(int argc,char *argv[])
{
        float u,v;

        assert(!GPIB_init(GPIB_ID));
        assert(!TDS400_init(GPIB_ID,TDS400_PAD,TDS400_SAD));

        //vi ska mäta amplitud på kanal 2:
        TDS400_init_amplitude(2);

        assert(!DS335_init(GPIB_ID,DS335_PAD,DS335_SAD));

        //läser från stdin:
        while (scanf("%f",&u)>0)
          {
            if (u>16) u = 16;
            if (u<-16) u = -16;
            DS335_set_vp(u);
            _sleep(250);
            printf("%f\t%f\n",DS335_get_vp(),TDS400_measure());
          }
        _sleep(500);
        DS335_set_vp(0);
        GPIB_exit();
}
```

----------src/angle_voltage.c----------

```
#include "stdio.h"
#include "stdlib.h"
#include "assert.h"
#include "GPIB.h"
#include "TDS400.h"
#include "MM3000.h"
#include "DS335.h"
#include "gpibconf.h"

/*Expects three floats on every line from stdin, first float is axis 1
  and second float is axis 2, third is p2p voltage. Outputs the actual angles,
  voltage and oscilloscope amplitude on stdout */
int main(int argc,char *argv[])
{
        float u,v,w;

        assert(!GPIB_init(GPIB_ID));
        assert(!TDS400_init(GPIB_ID,TDS400_PAD,TDS400_SAD));

        //vi ska mäta amplitud på kanal 2:
        TDS400_init_amplitude(2);

        assert(!MM3000_init(GPIB_ID,MM3000_PAD,MM3000_SAD));
        assert(!DS335_init(GPIB_ID,DS335_PAD,DS335_SAD));

        //läser från stdin:
        while (scanf("%f%f%f",&u,&v,&w)>2)
          {
            MM3000_move_absolute(1,u);
            MM3000_move_absolute(2,v);
```

```
            if (w>16) w = 16;
            if (w<-16) w = -16;
            DS335_set_vp(w);
            _sleep(250);

            MM3000_wait_all_axis();

printf("%f\t%f\t%f\t%f\n",MM3000_get_angle(1),MM3000_get_angle(2),DS335_get_vp(),TDS400_measure());
          }


        //går tillbaka till ursprungsläget
        MM3000_move_absolute(1,0);
        MM3000_move_absolute(2,0);
        DS335_set_vp(0);
        GPIB_exit();
}
```

----------matlab/comb.m----------

```
function result = comb(A,B)
% function result = comb(A,B)
%
% Creates a matrix containing all combinations
% of all rows of matrices A and B.
% The resulting matrix will have dimensions (size(A,1)*size(B,1))x(size(A,2)+size(B,2))

temp = zeros(size(A,1)*size(B,1),size(A,2)+size(B,2));
for i = 0:size(B,1) - 1
     for j = 1:size(A,1)
     temp(i*size(A,1)+j,1:size(A,2)) = A(j,:);
     temp(i*size(A,1)+j,(size(A,2)+1):end) = B(i+1,:);
     end
end
result = temp;
```

----------matlab/extern.m----------

```
function res=extern(command_name,argument)
% Calls an external program with the argument as a text file.
% Reads the output from the program in ASCII format
% and returns it as a matrix.
% This function requires writing permission and some temporary space
% in the current working directory! Please check the source if
% you encounter errors!

save mlab_in.tmp argument -ASCII;
eval(strcat('!',command_name,' < mlab_in.tmp > mlab_out.tmp'));
load mlab_out.tmp
res = mlab_out;
delete mlab_in.tmp;
delete mlab_out.tmp;
```

----------matlab/gpib_curve.m----------

```
function res=gpib_curve(N)
% function result = gpib_curve(N)
%
% Returns the waveform from channel N on the TDS200
% oscilloscope. First column of the returned matrix
% is time in seconds, second column is waveform
% data in the range 0..255. Note that the channel
% must be visible on the oscilloscope for this command
% to work.

res = extern(strcat(gpib_path,'\bin\curve.exe'),N);
```

----------matlab/gpib_scatter.m----------

```
function res=gpib_scatter(axis1,axis2)
% function result = gpib_scatter(axis1,axis2)
%
% Measures the oscilloscope amplitude on channel 2 for
% different angles on axis 1 and 2. Angles must be row matrices or real numbers.
% The first and second column of the returned matrix are the actual angles, and
% the third column is the amplitude measurements.

axis1 = axis1';
axis2 = axis2';

if size(axis1,2)~=1 or size(axis2,2)~=1
   error 'scatter: angles must be 1xN row matrices!';
   return;
end

angles = comb(axis1,axis2);
res = extern(strcat(gpib_path,'\bin\sprid.exe'),angles);

if size(angles,1)~=size(res,1)
```

```
     error 'gpib_scatter: An error occurred in the measurement. Check instruments!';
     return;
end

--------------------matlab/gpib_scatter.m-----------------------------------------------------

function res=gpib_scatter_volts(axis1,axis2,V)
% function result = gpib_scatter_volts(axis1,axis2,V)
%
% Measures the oscilloscope amplitude on channel 2 for
% different angles on axis 1 and 2 and voltages V.
% Angles and V must be row matrices or real numbers.
% The first and second column of the returned matrix are
% the actual angles, in degrees, and the fourth column
% is the amplitude measurements.

axis1 = axis1';
axis2 = axis2';
V=V';

nr_of_samples = size(axis1,1)*size(axis2,1)*size(V,1)
estimated_nr_minutes = nr_of_samples/30

if size(axis1,2)~=1 or size(axis2,2)~=1 or size(V,2)~=1
   error 'gpib_scatter_volts: Arguments must be 1xN row matrices!';
   return;
end

angles = comb(comb(axis1,axis2),V);
res = extern(strcat(gpib_path,'\bin\a_v.exe'),angles);

if size(angles,1)~=size(res,1)
     error 'gpib_scatter_volts: An error occurred in the measurement. Check instruments!';
     return;
end

---------------------matlab/gpib_scatter.m-----------------------------------------------------
function res=gpib_voltage(V)
% function result = gpib_voltage(V)
%
% Measures the oscilloscope amplitude on channel 2 for
% different peak to peak voltages on the DS335. V must be a
% row matrix or a real number.
% The first column of the returned matrix are voltages, and
% the second column is the amplitude measurements.

V = V';

if size(V,2)~=1
   error 'scatter: voltage must be a 1xN row matrix!';
   return;
end

res = extern(strcat(gpib_path,'\bin\voltage.exe'),V);

if size(V,1)~=size(res,1)
     error 'gpib_voltage: An error occurred in the measurement. Check instruments!';
     return;
end
---------------------matlab/gpib_scatter.m-----------------------------------------------------
```