

Semester Project WS 04/05

# Automation of a measurement setup for miniaturised bulge testing

Stefan Fuchs

Supervisor: Bernd Schöberle  
Professor: Prof. Dr. C. Hierold

30th January 2005

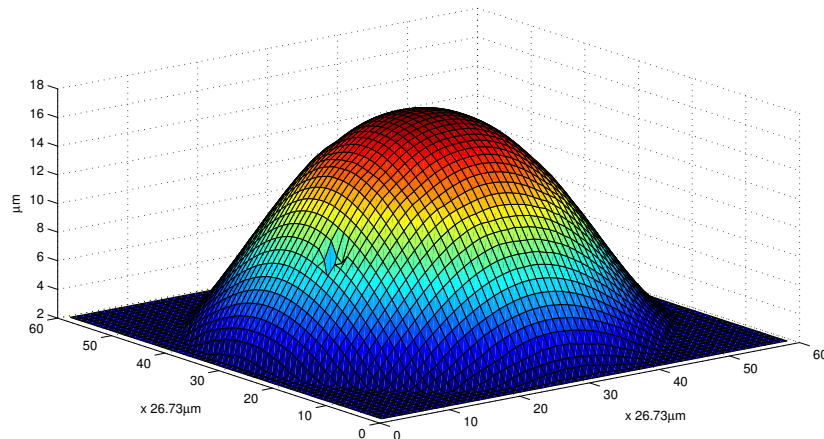
## Abstract

This semester project contributes to a dissertation about the characterisation of polymeric thin films. These films are attached to one side of Si wafers containing holes. Thus each wafer provides 16 to 32 polymeric thin film membranes suitable for measurements. The main factors of influence are the differential pressure applied to the membrane, the temperature and relative air humidity.

Verification of mathematical models is based on statistical methods. These require a large number of measured membranes. The measurements can only be carried out in a serial fashion. Thus the automation of that very time consuming measurement task is highly desirable.

The objective of this semester project is to automate an existing measurement setup. It has to be possible to take measurements without manual intervention after an initial setup phase.

The resulting LabView, MetroPro and C++ applications allow to take over 100 measurements of membranes at various differential pressures per hour. The membranes must be integrated within the same wafer. The wafers get manually exchanged between measurement process runs.



The figure above shows the measured surface map, the heightmap, of a single membrane. Its diameter is  $1.461\text{mm}$ , the peak height is  $16.63\mu\text{m}$ . The walls of the hole in the Si wafer where the membrane is located are not shown.

# Contents

<b>1</b>	<b>Measurement Setup</b>	<b>1</b>
1.1	Purpose of the Setup	1
1.2	Software	2
1.2.1	LabVIEW	2
1.2.2	MetroPro	2
1.3	Hardware	3
1.3.1	NewView 5000	3
1.3.2	Wafer Holder	4
1.3.3	Computers	5
1.3.4	LabJack U12	5
1.3.5	Bronkhorst Flow Controller	5
1.3.6	Burster Pressure Sensor	6
1.3.7	Sensirion SHT71 Temperature and Rel. Humidity Probe	6
<b>2</b>	<b>Assignment</b>	<b>7</b>
2.1	Project Aims	7
2.2	Involved Steps	7
2.3	Changes during the ongoing Project	8
<b>3</b>	<b>Solution</b>	<b>9</b>
3.1	Overview	9
3.2	LabVIEW	11
3.2.1	Overview	11
3.2.2	Automated Measurement Process	11
3.2.3	VI Description	13
3.2.4	Safety Considerations	16
3.3	MetroPro and NewView	18
3.3.1	auto-membranes Script	18
3.3.2	Auto-Membranes.app	18
3.3.3	Autofocus Issues	20
3.4	Post Measurement Processing	22
3.4.1	Requirement and Purpose	22
3.4.2	Sourcecode Availability and Formatting	22
3.4.3	Introduction to the Code	22
3.4.4	Membrane Recognition Algorithms	25
3.4.5	Tilting Issues	33
<b>4</b>	<b>Conclusion</b>	<b>36</b>
<b>A</b>	<b>User Guide</b>	<b>37</b>
A.1	Prerequisites	37
A.2	Starting the Software	37
A.3	The 4 Steps	37
A.4	Collecting Results	40
A.5	Post Processing	40
<b>B</b>	<b>File and Identifier Format Definition</b>	<b>42</b>

B.1	About the Definitions . . . . .	42
B.2	Files and Identifiers . . . . .	42
B.2.1	WaferID and Measurement Process ID . . . . .	42
B.2.2	Files on the LabVIEW PC . . . . .	42
B.2.3	Files on the MetroPro PC . . . . .	43
B.3	Format Definitions in ABNF . . . . .	44
B.4	Example files . . . . .	46
<b>C</b>	<b>Pressure Accuracy</b>	<b>47</b>
C.1	Expected . . . . .	47
C.2	Measurement Results . . . . .	47
<b>D</b>	<b>Datasheet Excerpts</b>	<b>49</b>
<b>E</b>	<b>Script auto-membranes.scr</b>	<b>50</b>

## List of Figures

1.1	Schematic diagram of the measurement setup . . . . .	1
1.2	Two membranes . . . . .	1
1.3	The wafer holder . . . . .	4
3.1	GUI of do-measurement.vi . . . . .	11
3.2	Structogram of do-measurement.vi . . . . .	17
3.3	GUIs of rs232-read-line.vi and block-transient.vi . . . . .	17
3.4	A screenshot of the Auto-Membranes application . . . . .	19
3.5	Focus and scan related offsets and lengths . . . . .	20
3.6	Perfect membrane with a slightly rough surface . . . . .	25
3.7	Si walls showing up in the heightmap . . . . .	26
3.8	Plots of 039a_008-. -0-4000-1-10, Si border visible . . . . .	27
3.9	How invalid region based membrane recognition works . . . . .	28
3.10	Radius detection of 039a_013-. -0-3050-14-14, perfect . . . . .	31
3.11	Radius detection of 039a_013-. -0-3050-16-16, noncircular . . . . .	31
3.12	Radius detection of 039a_008-. -0-4000-1-10, Si border . . . . .	31
3.13	Radius detection of 039a_008-. -1-3000-12-16, bad ring . . . . .	32
3.14	Surfaceplot of 039a_008-. -1-3000-12-16, bad ring . . . . .	32
3.15	Membrane untilting trigonometry . . . . .	34
C.1	Difference of pressures from Burster and Bronkhorst . . . . .	47

## List of Tables

3.1	A typical measurement sequence . . . . .	10
3.2	Purpose of sensor and controller communication VIs . . . . .	14
3.3	Purpose of RS232 communication VIs . . . . .	14
3.4	Purpose of helper VIs . . . . .	15
3.5	Tilt angles of membranes in 160 measured heightmaps . . . . .	35
A.1	Zero pressure measurement configuration . . . . .	39
C.1	Pressure test series . . . . .	48
D.1	Bronkhorst Flow Controller EL-PRESS P-602C-AAA-33-V Data	49
D.2	Burster Pressure Sensor 8263-5100 Data . . . . .	49
D.3	Sensirion SHT71 Data . . . . .	49



## 1.2 Software

### 1.2.1 LabVIEW

LabVIEW<sup>1</sup> is a graphical programming language developed by National Instruments<sup>2</sup>. It allows rapid development of graphical user interfaces, GUIs, for measurement and automation purposes. Its concept is to use Virtual Instruments, in short VIs. A VI consists of a GUI part and its associated block diagram, where the in- and outputs contained in the GUI get connected to each other in sense of dataflow.

A SubVI is just an ordinary VI which gets called from another VI. Whether a VI is a SubVI or not depends on the point of view.

### 1.2.2 MetroPro

MetroPro is the measurement software to control instruments from Zygo [1]. In our setup it is connected to the NewView 5000 (s. 1.3.1), a white light scanning interferometer.

#### *Patterns*

MetroPro allows to specify a pattern of measurement locations. These locations must either be arranged in a rectangular, matrix like fashion or on concentric circles. Only rectangular patterns are used in this project. Once such a pattern is loaded and correctly aligned, it's easy to command the stage to a desired position using MetroScript [2] (s. 1.2.2), the scripting language of MetroPro.

MetroPro even allows to exclude arbitrary pattern positions so they don't get measured. This leads to two different numbers of pattern positions. One enumerates all positions, the other counts only the desired positions.

Mapping positions to numbers enables computers to work with them. There are many different possibilities to associate positions with numbers. In mathematics a *row/column* schema is used to address the elements of a matrix. MetroPro maps the pattern positions to numbers ranging from 1 to the number of positions (all or only the "included" positions).

Enumeration of the positions can happen in different orders configurable by the user. RowSerp, ColSerp, RowRast and ColRast are the possible choices for rectangular patterns. The enumeration order for RowSerp as usually used is: 1/1, ..., 1/n, 2/n, ..., 2/1, 3/1, ... (*row/col*,  $n$  columns, square) This means, the upper left pattern position, in matrix coordinates 1/1, is assigned to 1. The position to the right of it is assigned to 2. The last position on the first row becomes number  $n$ . The position below it has index  $n + 1$ . This continues until index  $n \cdot n$  is reached and all positions are enumerated.

---

<sup>1</sup><http://www.ni.com/labview/>

<sup>2</sup><http://www.ni.com/>

Though the origin of the pattern is configurable, it's assumed to be at the upper left corner, respectively matrix position 1/1, within this document. The Membranes Post Measurement Processing tool (s. 3.4), MPMP, can treat other origin locations also.

The fact that MetroPro uses different, linear indexing modes for patterns leads to confusion. To avoid misinterpretations of results, the actual pattern loaded in MetroPro is saved at the beginning of every measurement process. MPMP can read those pattern files and calculate the *row/col* coordinates with 1/1 at the upper left corner.

### *MetroScript*

MetroPro supports a very simple, Basic like scripting language called MetroScript. Scripts are like small Basic programs or makros in OpenOffice.org and MS Office. The commands for repetitive tasks can be written into a text file, the script. This file can be assigned to a button to cause its execution. It seems the MetroScript manual [2] still contains nasty errors. But scripts are extremely useful to automate small tasks.

MetroScript is designed to control MetroPro, communicate via an RS232 interface and interact with the NewView 5000. It's not suitable at all to process measurement data.

## **1.3 Hardware**

### **1.3.1 NewView 5000**

The NewView 5000 is a white light scanning interferometer developed by Zygo [1]. It allows to generate 3D maps of reflective surfaces.

The NewView uses a multispectral light source. The interfering light reflected by the target surface, e.g. a membrane, and the light reflected by a reference surface, builds a pattern of fringes, the destructive and constructive regions of interference. These fringes will move depending on the vertical movement of the camera. MetroPro acquires the images from the NewViews CCD camera with a framegrabber card. It then analyzes the fringe pattern and its movement in frequency domain when scanning in vertical direction. That way it obtains the relative height of each pixel of the camera. The result is a heightmap containing 640 x 480 pixels. Smaller pictures are possible but not used in this project.

The lateral resolution is limited by the CCD camera being used. Dividing the image width in meters by the number of pixels in x- / horizontal direction gives the lateral resolution. It's the same for both lateral directions. The vertical resolution is 0.1nm.

Typically an objective with 2.5x or 5x magnification and an image size of 640 x 480 pixels is used for the membranes (s. 1.1). This results in a lateral resolution of  $4.46\mu\text{m}/\text{pixel}$  and  $2.23\mu\text{m}/\text{pixel}$  respectively.

Image coordinates on computers are the same as screen coordinates. The 0/0 coordinate is in the top left corner, the  $(width - 1)/(height - 1)$  coordinate at the bottom right corner. The x axis runs from left to right and the y axis from top to bottom. Such coordinates are used by MetroPro, NewView and MPMP (s. 3.4) for images and heightmaps. They're typically stored in row major order, an exception are the .mat files for Matlab, which have their data stored in column major order.

The very limited lateral resolution imposes some limitations regarding steep slopes. When multiple fringes fall on the same pixel of the camera, the NewView 5000 isn't able to derive the height at that position. A solution to that problem is to use a greater magnification. This results in a smaller field of view and it might be necessary to use stitching. That means to join multiple measurements of different regions of the same target to form one big heightmap.

### 1.3.2 Wafer Holder

The wafer holder is a device to apply pressure to the membranes and to measure them at the same time. It's a cylindrical aluminum block with three holes in its side wall (A, B and C, see figure 1.3) and one hole per membrane in its 6mm thick aluminum cover plate. The Si wafer containing the membranes is the top seal of the pressure chamber and lays just underneath the aluminum cover plate.

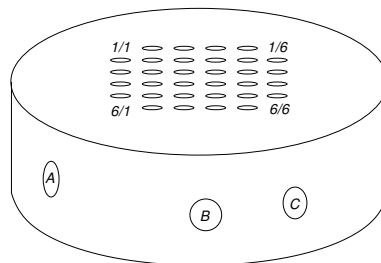


Figure 1.3: The wafer holder

A needle valve is connected to hole A to limit the  $\text{N}_2$  stream flowing out of the pressure chamber. Hole B is for the Burster pressure sensor (s. 1.3.6) which allows to measure the exact pressure in the chamber. Hole C is the inlet for the  $\text{N}_2$  stream supplied by the Bronkhorst flow controller (s. 1.3.5).

The cover plate contains a hole at each position where the silicon wafer contains a membrane. The 1/1, 1/6, 6/1 and 6/6 positions don't have any holes and thus make manual pattern alignment for 6/6 matrices quite a hard task. This is not a problem as usually 6/4 or 4/4 matrices are used. The quality of the holes in the center is better than at the outer region because of the inhomogeneous etching process.

### 1.3.3 Computers

#### *PC running MetroPro*

A Pentium 4 is running MetroPro. It controls the NewView 5000 and executes the actual membrane measurements. All acquired data gets stored locally on the harddrive. Due to security considerations this computer is not connected to the local area network.

#### *PC running LabVIEW*

The other computer controls the whole measurement process. It runs VIs (s. 1.2.1) to communicate with sensors, controllers and the other PC. The relative humidity, temperate and pressure sensors and controllers are all connected to this machine.

### 1.3.4 LabJack U12

The LabJack [3, 4] connects via USB to the PC running LabVIEW. It provides multiple A/D converters and digital in- and outputs. It's thus a simple interface to communicate with sensors and controllers. There's a DLL available for access from within LabVIEW.

### 1.3.5 Bronkhorst Flow Controller

The [Bronkhorst](http://www.bronkhorst.com/)<sup>3</sup> flow controller "EL-PRESS P-602C-AAA-33-V-020R Multi-Bus DEPC(A)" has an integrated pressure controller on its output side. It's connected via RS232 to the LabVIEW PC and communicates using the custom Flowbus protocol developed by Bronkhorst. There's a Windows application, FlowDDE, which provides a DDE service to set and and get parameters of the controller.

The integrated pressure controller uses an own unit, in which 32000 equals the nominal pressure, in this case *4bar*. At some locations a percentage of the nominal pressure is used.

A limitation of this controller is the large death time of the overall transfer function. This makes fast control of high gain systems, that means systems with small volumes as the wafer holder (s. 1.3.2), a difficult task. The factory default PID controller parameters currently used are quite a bit to tight. This can result in small remaining oscillations at operation points in the region of 3000-5000 (internal units). There are at least two possible solutions:

- enlarging the volume: adding a 20l canister to the setup, inserted between the Burster flow controller and the wafer holder

---

<sup>3</sup><http://www.bronkhorst.com/>

- optimising the controller: deriving a mathematical model of the N<sub>2</sub> flow system and finding better PID parameters. This is considered the most promising solution. Due to time restrictions this was not part of this semester project.

A rather bad solution is to open the needle valve a bit more. This changes the operation point of the flow controller and increases damping. It helps to reduce the oscillations. But the low impact doesn't justify the additional N<sub>2</sub> consumption. There's of course a point where even a large volume system would get unstable when the flow gets reduced to much.

### 1.3.6 Burster Pressure Sensor

The Burster MTS-8263-5100 pressure sensor is directly attached to the pressure chamber in the wafer holder (s. 1.3.2). That way the pressure on the membranes can be measured more precisely than with the Bronkhorst flow controller (s. 1.3.5). In comparison to the Bronkhorst sensor, the Burster sensor is not affected by pressure losses in the tube between the sensor and the pressure chamber.

The sensor's output current of 4 – 20mA is pulled across a 500Ω resistor parallel to a 12bit A/D converter input of LabJack (s. 1.3.4).

### 1.3.7 Sensirion SHT71 Temperature and Rel. Humidity Probe

The Sensirion SHT71 is a digital, integrated device including a temperature and a relative humidity sensor. It's connected to two digital ports of LabJack and talks a serial, synchronous protocol.

The LabJack DLL contains functions which read the temperature and relative humidity from a SHTxx probe.

Datasheets and further information can be found on the homepage of [Sensirion](http://www.sensirion.com/)<sup>4</sup>.

---

<sup>4</sup><http://www.sensirion.com/>

# Assignment

## 2.1 Project Aims

The aim of this semester project is to be able to automatically measure a large number of membranes (s. 1.1). After an initial, manual configuration phase, the setup must be able to proceed unattended. It should for example be possible to have a measurement process running over night.

## 2.2 Involved Steps

Because manual measurements of membranes were already being made, there wasn't much to do on the hardware side.

An analysis revealed the following necessary steps to reach the aim:

1. getting familiar with LabVIEW (s. 1.2.1) and its capabilities
2. getting familiar with MetroPro (s. 1.2.2) and the NewView 5000 (s. 1.3.1)
3. analysis of and finding solutions for emerging problems, such as overshoots of the pressure controller, possible emergencies and the right reactions to them, switching off the NewViews light source when it's not needed
4. figuring out how to read the different available sensors from within LabVIEW and how to set the reference pressure of the Bronkhorst pressure controller (s. 1.3.5)
5. controlling the NewView 5000 using MetroScript (s. 1.2.2)
6. analysing which measurement parameters and results there are and making sure to include any of them in the measurement results. This includes figuring out requirements and wishes of future users.
7. developing a schema to consistently store the data acquired during a measurement process
8. drawing VIs to control the various sensors and to communicate with MetroPro on the 2nd computer via RS232
9. developing a GUI for measurement process control. The VI containing this GUI calls the VIs indicated in the previous point to access the sensors, controllers and communication interfaces.
10. As MetroPro can't obtain the radius, profile and peak heights without manual interaction, it's necessary to postprocess the raw heightmaps saved during the measurement process. It's probably the easiest way to write an application in C/C++. Due to time restrictions and portability reasons a console application without GUI is the best option. The following points are realised:

- (a) development of two simple but sufficiently accurate image recognition algorithms for circular and rectangular membranes
- (b) the application is to be developed using free<sup>1</sup> tools and compilers
- (c) development under Linux, final usage under Windows, portability matters
- (d) writing an extensible framework of interacting C++ classes which collect measurement results from various logfiles and provide them in a centralised form. This eases writing the results afterwards to a CSV file for later import into MS Excel.
- (e) implementation of the image recognition algorithms in the just mentioned framework
- (f) testing and verification of this tool, especially the image recognition part

11. testing the developed applications, scripts and VIs.

## 2.3 Changes during the ongoing Project

Initially the project started without the intention to write a post measurement processing tool as mentioned in section 2.2, point 10. When the necessity of it became clear, the main part of the project was finished and there was still enough time to write it.

The following tasks should be addressed in the future:

- developing a mathematical model of the pressure system and deriving better PID parameters for the Bronkhorst pressure controller
- building a mechanical tool to fix the wafer holder (s. 1.3.2) to the NewViews motorized X/Y - stage. Until now the wafer holder is just being put on the stage. Care has to be taken not to push against it after pattern alignment (s. 1.2.2). The accelerations due to motorized movement don't pose any problems.

---

<sup>1</sup>GPL (<http://www.gnu.org/>) or similiar license

# Solution

## 3.1 Overview

The automation solution consists of three main parts: LabVIEW related software in the form of VIs (s. 3.2), MetroPro scripts and applications (s. 3.3) and the MPMP tool (s. 3.4) for data postprocessing.

A wafer containing a single type of membranes (s. 1.1) can be measured. The pressure gets specified as a ramp of levels, optionally double sided. Automatic measurements with no applied pressure at the beginning of the ramp are not possible. A single zero pressure measurement without ramp is necessary in this case. Measuring with zero applied pressure at the end of the ramp is possible. The setup is able to measure over 100 membranes per hour. This number varies depending on autofocus and pressure ramp settings chosen by the user. For example to measure a wafer at 10 pressure levels containing 16 membranes takes about 1h 30 in addition to 20min manual configuration.

The LabVIEW PC (s. 1.3.3) controls the measurement process. There's a main VI for user interaction, called `do-measurement.vi`. That VI accesses a number of SubVIs. All sensors and actors, except the NewView 5000 (s. 1.3.1) and its stage, are connected to the LabVIEW PC. The MetroPro PC awaits commands from the LabVIEW PC via the RS232 interface. The commands cause the measurement of a single membrane at the current pressure. After such a measurement the MetroPro PC sends a completion notice to the LabVIEW PC and waits for the next command. This continues until all membranes are measured at the current pressure. Then the LabVIEW PC increases or decreases the pressure, depending on user settings. Once the pressure has stabilised, the whole pattern (s. 1.2.2) of membranes is measured again. Thus a typical sequence looks as shown in table 3.1.

On the MetroPro PC, everything gets handled by the MetroPro script (s. 1.2.2) `auto-membranes.scr` (s. E). This script gets started by the user after he has leveled the membranes, adjusted the light source, loaded and aligned the pattern, set the focus zero position and fine tuned the autofocus parameters [6].

Once the measurement process is finished, there are a number of files on each of the two PCs. The user has to collect them and put them in a single, flat directory. The results of multiple measurement process runs may be put in the same directory for postprocessing in a single step. The MPMP tool (s. 3.4) can then be run in that directory to extract the wanted results and check for erroneous measurements. The results get written into one file per measurement process as comma separated values. There's one line per measured membrane. Its format is described in the file header<sup>1</sup>. [OpenOffice.org](http://www.openoffice.org)<sup>2</sup> and MS Excel can load these files.

---

<sup>1</sup>A detailed specification of filenames and formats can be found in appendix B

<sup>2</sup><http://www.openoffice.org/>

LabVIEW PC	RS232	MetroPro PC	NewView
<i>setup skipped.</i>			
set pressure			
	PAT_FIRST →		
meas. temp, pressure, rh		autofocus	
			measure
		save data	
	← LOC_DONE		
	LOC_NEXT →		
meas. temp, pressure, rh		autofocus	
			measure
		save data	
	← LOC_DONE		
	LOC_NEXT →		
meas. temp, pressure, rh		autofocus	
			measure
		save data	
	← LOC_DONE		
	LOC_NEXT →		
meas. temp, pressure, rh		autofocus	
			measure
		save data	
	← LOC_DONE		
	LOC_NEXT →		
meas. temp, pressure, rh		autofocus	
			measure
		save data	
	← PAT_DONE		
change pressure			

*until all pressure levels done: goto third row (PAT\_FIRST)*

Table 3.1: A typical measurement sequence

## 3.2 LabVIEW

### 3.2.1 Overview

There are two VIs for the user. The `do-measurement.vi` shown in figure 3.1 controls an automated measurement process, and `manual-control.vi` helps controlling the sensors and actors manually, e.g. for autofocus fine tuning.

Both VIs access the same set of SubVIs for assistance. There are SubVIs designed to show their GUI when called from an other VI. These are

```
pressure-monitor.vi
flowctrl-p-set.vi
block-transient.vi
rs232-read-line.vi
```

Their function is explained in following sections. Only the first of those VIs requires manual interaction to return to the caller. All of them need a certain amount of time to finish, so it's desirable to show the user what's going on. They contain an abort button to stop the current measurement process and switch off the N<sub>2</sub> flow.

### 3.2.2 Automated Measurement Process

One full execution of `do-measurement.vi` is equivalent to that what the author calls a **measurement process**. A measurement process gets its unique identification number, the measurement ID, as specified in appendix B. The process gene-

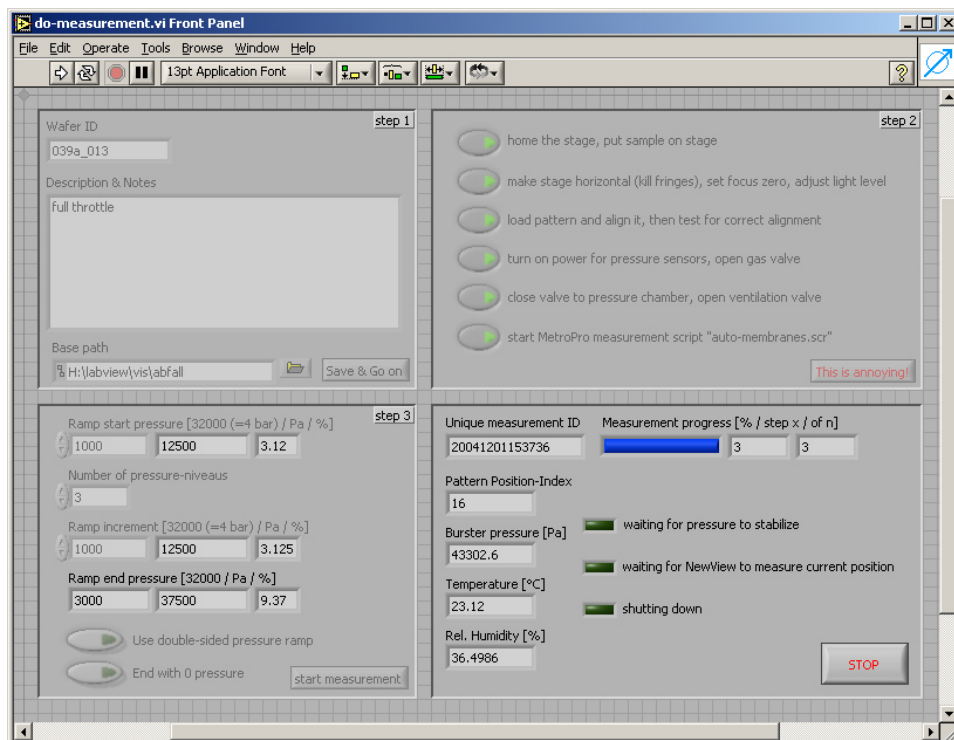


Figure 3.1: GUI of `do-measurement.vi`

rates a logfile containing information about each measured membrane on a line. Errors get logged to that file also. The file's exact format and name is specified in appendix B. MPMP (s. 3.4) should be used to parse this logfile and extract meaningful information.

A measurement process requires manual interaction during four steps, indicated by four boxes in the `do-measurement.vi` GUI (fig 3.1)<sup>3</sup>:

1. filling in the wafer ID, additional freeform information about the wafer, the membrane geometry and the directory, where generated files should be placed.
2.
  - (a) homing the stage, putting the wafer holder containing the wafer on it.
  - (b) leveling the stage, setting focus zero, adjusting the light level.
  - (c) loading the pattern and aligning it, testing for correct alignment.
  - (d) turning on the power for the sensors, opening the N<sub>2</sub> valve.
  - (e) closing the valve to the pressure chamber, opening the ventilation valve.
  - (f) starting the MetroPro script `auto-membranes.scr` (appendix E).
3. configuring the desired pressure ramp.
4. starting the automated measurement process.

After completing these steps, the measurement process runs.

The process gets aborted on severe errors only. If the autofocus fails at a membrane, that measurement gets an "ERROR: ..." line appended in the logfile defined by `bnf.logfilename` (s. B.2.2) and the process continues with the next membrane or pressure level. MPMP (s. 3.4) recognises that "ERROR: ..." line and tags the according membrane<sup>4</sup> as invalid.

Termination of the measurement process causes the N<sub>2</sub> flow and the NewViews light source to be switched off. The rest of the measurement setup continues to run as automatic power off is not possible<sup>5</sup> at the moment and would only make sense for the last measurement process of a series.

Both PCs must be configured to not enter suspend or standby mode during the measurement process. It's recommended to use a black screen as screen saver instead of CPU intensive graphics. Switching off the screens is recommended too, for example by using their power saving facilities.

Extensive flow of N<sub>2</sub> is intensionally not considered an error and the measurement process continues. It might happen that a membrane explodes. It's assumed that the user still wants to measure the remaining ones. A proper way to terminate the measurement process if the N<sub>2</sub> flow exceeds some limit would be as follows. A SubVI

---

<sup>3</sup>Screenshots might look very bad when viewed on a monitor. The screenshots are included as high quality jpeg images. Zooming in until the image gets displayed with the same resolution as the monitor shows the full quality.

<sup>4</sup>Referring to a membrane in a measurement process context means a membrane (s. 1.1) at a certain pressure level. This is equivalent to a single line of values in the logfile.

<sup>5</sup>requires additional equipment

needs to be created which has at least error in- and outputs and probably in- and outputs for a DDE session handle. Errors on error input, which have their `status` field equal `true`, have to be forwarded unaltered to error output. If the `status` field of error input is `false`, then the N<sub>2</sub> flow has to be read from the Bronkhorst flow controller (s. 1.3.5). If it exceeds the desired limit, which should depend on the pressure, an error with `code 101xx` and `status := true` has to be generated. A human readable error string like "Defect membranes suspected." is suggested. This new VI has to be inserted into the error flow of the innermost `while` loop in `do-measurement.vi`. Switching off the Bronkhorst flow controller is not necessary as it's done upon termination of the measurement process. The supplied error string gets written as an "ERROR: ..." line to the logfile. This VI should be integrated the same way into the pressure ramping loop of `flowctrl-p-set.vi` too, as that's the point where bursting is usually expected.

### 3.2.3 VI Description

#### *Main GUIs*

`do-measurement.vi` (fig 3.1) is the largest VI of this project. It reads all sensors, controls the pressure and commands the MetroPro PC (s. 1.3.3). The user has to specify the path for files generated during a measurement process in step 1. This path must not end with a backslash. The pressure ramp must be parametrised in step 3. The parameters are `ramp start pressure`, `number of pressure levels` and the `difference between 2 levels`. There are options for a double sided pressure ramp (e.g. 1000, 2000, 1000 instead of 1000, 2000)<sup>6</sup> and for a zero pressure measurement after the ramp (e.g. 1000, 2000, 0 instead of 1000, 2000). These options can be combined as needed.

Using `manual-control.vi` it's possible to read sensor values and set the pressure manually. This is useful for fine tuning the MetroPro autofocus parameters and tracking down measurement problems with applied pressure.

#### *Sensor and Controller Communication*

Table 3.2 gives an overview of the VIs used to communicate with the sensors and controllers. Some of these VIs feature specialities requiring further illustration.

`flowctrl-p-set.vi` is able to show a GUI and to ramp the pressure in increments of 50 unit steps up to the new reference value. This ramping can optionally get skipped. If the new reference value is lower than the old one, no ramping is performed. That's a simple solution to minimize pressure overshoots caused by the tight PID parameters of the controller.

`flowdde-check-run.vi` checks if it can obtain a DDE session handle to the service provided by `FlowDDE.exe`. If failing to do so, it starts `FlowDDE.exe` and waits for 30 seconds. At the end it returns two valid DDE session handles or an error. The handles must be closed using `flowdde-close-session.vi` after use. Employing the same handle for two parallel tasks won't work because of collisions.

---

<sup>6</sup>These are pressure values in Bronkhorst flow controller internal units (s. 1.3.5)

<code>flowctrl-p-read.vi</code>	Reads pressure from the Bronkhorst flow controller.
<code>flowctrl-p-set.vi</code>	Sets the pressure reference value of the Bronkhorst flow controller.
<code>flowctrl-switch-off.vi</code>	Turn off the N <sub>2</sub> flow.
<code>flowdde-check-run.vi</code>	Runs FlowDDE.exe if necessary, returns 2 valid DDE session handles.
<code>flowdde-close-session.vi</code>	Closes a DDE session handle.
<code>psens-p-read-04-AISample.vi</code>	Reads the value of the Burster pressure sensor.
<code>read-temp-rh.vi</code>	Reads the temperature and relative humidity.

Table 3.2: Purpose of sensor and controller communication VIs

`psens-p-read-04-AISample.vi` reads the pressure from the Burster (s. 1.3.6) sensor. It averages 10 sample points to calculate the result. It uses the function `AISample` of `ljackuw.dll`, the DLL provided with LabJack (s. 1.3.4), to do this. A faster solution is to use `AIBurst`, which makes multiple A/D conversions at a high frequency and returns all results at once. Unfortunately this doesn't work. The remaining solution is slow. `psens-p-read-04-AISample.vi` can't be called with more than 5 Hz. The VI generates an error if the sensor is switched off. This is possible because the sensor outputs an offset current of  $4mA$  at zero relative pressure.

### Communication via RS232

Table 3.3 shows the VIs used for asynchronous, serial communication with Metro-Pro.

<code>rs232-metropro-setup.vi</code>	Opens the RS232 interface to Metro-Pro, returns a handle to it.
<code>rs232-read-line.vi</code>	Blocks until a complete line has been received.
<code>rs232-write-line.vi</code>	Writes a line to the interface.

Table 3.3: Purpose of RS232 communication VIs

When calling `rs232-read-line.vi` it should be configured to show its GUI. This is a blocking call, the user might think something is not working. So the received characters and a blinking LED are shown as in figure 3.3. Received lines need to have unix style LF line termination. CR characters get passed on to the VI's output.

`rs232-write-line.vi` sends one line over the RS232 interface to the MetroPro PC (s. 1.3.3). The input string must not contain a line termination character. The necessary CR LF line termination gets appended by the VI.

## Helpers

Table 3.4 shows VIs not fitting into the other categories. Some of them require further explanation.

<code>block-transient.vi</code>	Waits until pressure has stabilised.
<code>clip-value.vi</code>	Saturates an input value at a certain maximum and minimum.
<code>debug-logline.vi</code>	Writes a debugging note to a certain file.
<code>error-condenser.vi</code>	Collects errors and handles them.
<code>generate-abort-error.vi</code>	Generates a pseudo error for measurement process abortion.
<code>lf2sp.vi</code>	Crunches a multi line string onto a single line.
<code>pressure-monitor.vi</code>	Shows a pressure graph to the user, logs pressure to a file.
<code>vars-global.vi</code>	Global variables for the measurement process.

Table 3.4: Purpose of helper VIs

`vars-global.vi` contains some global variables used for configuration and parameter passing. The debugging mode can be switched on and off from here. This controls the operation of `debug-logline.vi`. That VI writes an arbitrary input string to some file specified in `vars-global.vi`. But this is only done if the debugging mode is switched on. This mode was being used to investigate the communication between the two PCs.

`block-transient.vi` blocks execution until the pressure at the Burster pressure sensor (s. 1.3.6) has settled to steady state. The VI blocks execution for at least 10 seconds. If the pressure has not stabilised after 3 minutes, the measurement process aborts with an error message. The criteria for stability are the three differences *maximum – average*, *average – minimum* and *maximum – minimum*. The running *average* of the pressure at the Burster sensor is calculated over the previous 50 samples taken at  $\approx 5Hz$ . The limit is dynamic within small bounds. It gets changed depending on the current pressure. The first two differences have their limit clamped to the interval  $180 \dots 250Pa$ . The *maximum – minimum* difference needs to be within  $360$  and  $500Pa$ . Which criteria are currently met is indicated by three LEDs in the VI (fig 3.3).

`pressure-monitor.vi` shows a chart of the current reference and effective pressures of the Bronkhorst flow controller (s. 1.3.5) and the Burster pressure sensor (s. 1.3.6). It logs the pressure measured with the Burster pressure sensor about 4 times per second to a file, named as defined by `bnfPlogfname` (s. B.2.2). The frequency of 4 Hz is not exact because Windows is not a real time operating system.

### 3.2.4 Safety Considerations

There are two possible dangers:

- The motorized stage of NewView could crash into something, e.g. touching an objective with a part of the wafer holder.
- The N<sub>2</sub> flow could become too large.

The first issue is easily solved by assuring, that no valve or sensor attached to the wafer holder is higher than the cover plate of it. NewView has a Z-Stop which needs to be set properly. Once this is set, it's not possible anymore to crash into an objective. It's not possible to touch anything by laterally moving the stage if the wafer holder is properly placed and leveled.

The second problem is more difficult. The pressure from the 200bar N<sub>2</sub> bottle gets reduced to 4bar by a valve directly attached to the bottle. Further there's a ventilation in the laboratory and the bottle is of finite size. Bursted membranes won't destroy the NewViews objectives as it's just N<sub>2</sub> at a low pressure blowing against them. The N<sub>2</sub> flow gets switched off at the end of every measurement process. This won't be the case if one of the PCs crashes during the measurement process, communication with the Bronkhorst flow controller gets interrupted or the communication between the PCs stops working. Nothing like that happened during the test phase. In case anything happens and the user is present, it's possible to stop the measurement process by clicking on a large, red STOP button in the active window. This will turn off the N<sub>2</sub> flow within about 1 second. Finally it's always possible to manually turn off the main valve of the N<sub>2</sub> bottle.

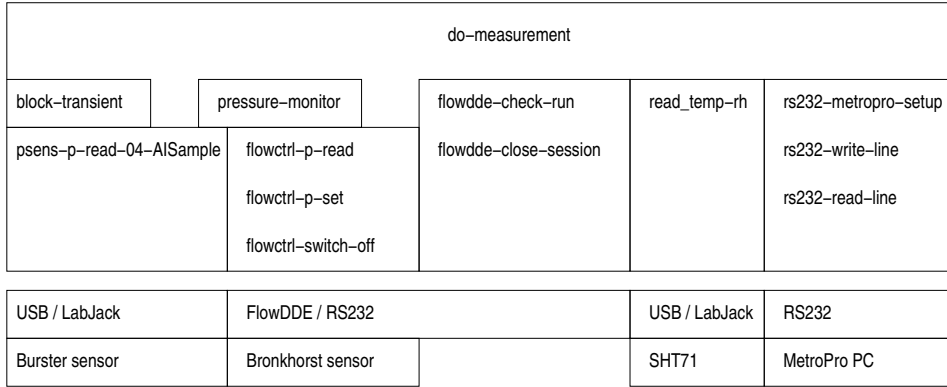


Figure 3.2: Structogram of do-measurement.vi

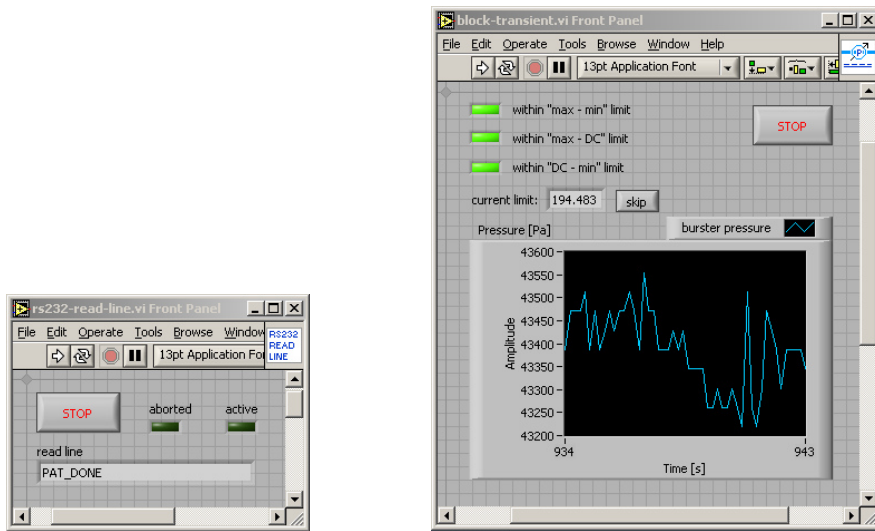


Figure 3.3: GUIs of rs232-read-line.vi and block-transient.vi

## 3.3 MetroPro and NewView

### 3.3.1 auto-membranes Script

The main part of the control software on the MetroPro PC (s. 1.3.3) consists of the `auto-membranes.src` script. This script has to be executed by the user after he set up everything. Its source code is listed in appendix E.

First the script sets necessary parameters of MetroPro. Then it listens to the LabVIEW PC (s. 1.3.3). It receives commands and partial filenames for the `.pat` and `.dat` files to be created. As the LabVIEW PC doesn't know about the loaded pattern, it's up to the script which membrane gets measured. Receiving a `PAT_FIRST` command causes the script to start with the first position of a pattern. Every measured position gets confirmed by a `LOC_DONE` command, except the last position of the pattern, which gets answered with `PAT_DONE`. A `LOC_NEXT` command tells the script to step to the next position within the pattern. Every `PAT_FIRST` and `LOC_NEXT` cause the script to measure a membrane. If something fails it responds with `LOC_SKIP` and an error message before it sends the `LOC_DONE` or `PAT_DONE` confirmation. Three further commands the script understands are `PAT_SAVE` to save the actually loaded pattern and `MEAS_DONE` or `MEAS_ABORT` to terminate the script and switch off the NewViews light source. The latter two are nearly identical, they just print different status messages on the MetroPro GUI.

One pattern file gets created per measurement process (s. 3.2.2). It allows to calculate the matrix coordinates (s. 1.2.2) of membranes by knowing their position index. Using the `RowSerp` enumeration order is recommended, as it's the most tested mode MPMP (s. 3.4) can treat. Both possible position indices, "all" and "included positions only" (s. 1.2.2), are part of the filename of each `.dat` file.

Two `.dat` files get saved for each membrane. One contains the raw data and the other contains the data after MetroPro analysed it. The raw data is the same as if "Save Data" gets clicked in the MetroPro application window. The analysed data is the same as if "Save Data" gets clicked in the "Surface Wavefront Map (T+R)" window. This enables the user to specify what happens when the data gets analysed.

All generated files are saved in the subdirectory `output` of the directory specified with the MetroPro control "Controls/Custom/Text 1". The path entered in that control must not have a trailing backslash, `"\"`. Network paths don't work. An exact specification of the filenames is available in appendix B (s. B.2.3).

### 3.3.2 Auto-Membranes.app

MetroPro allows the user to compose so called "applications". That means to instantiate windows, buttons, controls and attributes and to suitably arrange them. All those items exist within MetroPro but the user decides whether they are displayed or not. This makes it possible to configure for every task a custom GUI. The `Auto-Membranes.app` application (fig 3.4) is a GUI assembled for this project.

The `auto-membranes.scr` script requires some special controls and attributes to be present in the caller application, in this case the `Auto-Membranes.app`:

- "Controls/Custom/Text 1" must contain the base path, in whose sub-directory "output" the data gets saved. The base path must not end with a "\".
- "Controls/Custom/Text 2" is the place where the script prints its status messages.
- "Controls/Miscellaneous/Comment" contains the .dat filename without suffix after each membrane measurement. This control can be included in report windows to allow proper linkage of report entries to data files.

After the measurement of a membrane, the function "logreports" gets called. This causes all report windows to be logged. The user is free to create and modify such windows to suit his needs.

MPMP (s. 3.4) requires for proper functionality that MetroPro truncates measured heightmaps to a minimal, rectangular area containing all valid pixels. The application `Auto-Membranes.app` (fig 3.4) has this by default, like many other MetroPro applications. This ensures that the center of the stored heightmap lays within the membrane, not outside of it.



Figure 3.4: A screenshot of the Auto-Membranes application

### 3.3.3 Autofocus Issues

#### *Top and Bottom Surface*

The autofocus feature of MetroPro has problems with semi transparent surfaces whose top and bottom side are both within the autofocus scan range. It's possible that the autofocus sticks to the bottom surface instead of choosing the top surface. As the vertical position of a membrane varies with applied pressure, it's not possible to solve the problem by limiting the autofocus scan range.

With knowledge of the autofocus parameters [6] it's possible to fine tune the autofocus algorithm to find the top surface in most cases. Very important parameters are Focus Data, Focus Min Mod (%), Focus Max Adjust, and Focus Retry Max Adjust. The parameters of the provided application are already tuned. Adjusting them for new membranes not yet available is necessary.

#### *Offsets and Ranges*

Figure 3.5 illustrates the offsets and lengths used for autofocus and scan. MetroPro starts at Focus Zero, searches for optimal modulation within  $\pm$  Focus Max Adjust and retries with  $\pm$  Focus Retry Max Adjust (not shown). Then it adds Focus Offset to the position found to obtain the central point for a bipolar scan.

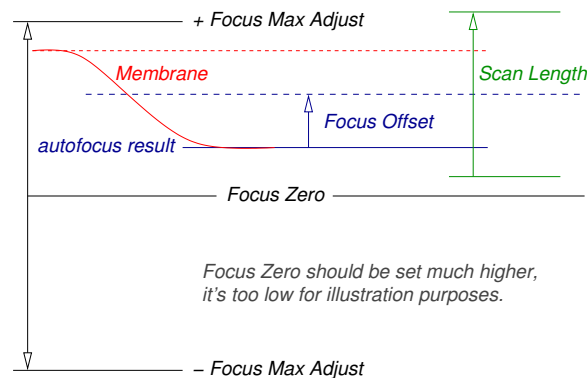


Figure 3.5: Focus and scan related offsets and lengths

If MetroPro fails to find a point of focus after trying twice, it bails out with an error. This causes the current membrane to get skipped. An invalid measurement gets logged on the LabVIEW PC. Adding about  $20\mu m$  to the Focus Zero position prevents the membrane from falling out of the Focus Max Adjust range when pressure gets applied.

To have the whole membrane scanned after auto focusing, it's crucial to have correct Focus Offset and Scan Length values. On a membrane which is bent upwards to have a relative peak of  $16\mu m$ , the autofocus can either stick to the valley at the periphery or lock at the top. If Focus Offset = 0 and Scan Length =  $20\mu m$  bipolar, MetroPro scans from  $-10\mu m$  to  $+10\mu m$  relative to the focus

position it has found. If the autofocus is stuck to the valley, then the top  $6\mu m$  are cut off. It's not possible to tell where the autofocus sticks to and it even varies from membrane to membrane. Taking the Scan Length more than twice as large as necessary solves the problem. Though this requires a lot of measurement time. A better solution is to fine tune the autofocus so it locks to the periphery, which is most often the largest area.

## 3.4 Post Measurement Processing

### 3.4.1 Requirement and Purpose

It's not possible to obtain the final measurement results, e.g. diameter and radius, directly from MetroPro. Obtaining them from MetroPro would require manual control. The large number of heightmaps requires automatic processing of them. This is the purpose of MPMP. The tool collects information from measurement process outputs. It analyses heightmaps to calculate the peak height, the diameter and some further, relevant values. Finally it writes these results to a CSV file for later import into OpenOffice.org, StarOffice or MS Excel.

### 3.4.2 Sourcecode Availability and Formatting

The source code of MPMP, the Membranes Post Measurement Processing tool, is not printed within this documentation. It's available on <http://srf.ch/bitbytes.php>.

A tab width of 4 characters and an editor<sup>7</sup> which can handle unix style linebreaks are recommended to view the source code files.

Thorough understanding of the `mpmp` source code requires a certain knowledge of C++. "The C++ Programming Language" [7] is a good reference book. The only libraries used are the standard C library and the `iostreams` part of the C++ Standard Template Library. It's proved to work with the [GNU C Library 20040808 rel, version 2.3.4](#)<sup>8</sup> and the C++ library from [GCC](#)<sup>9</sup> or the libraries and headers from [MinGW](#)<sup>10</sup>. The latter employs Windows native C libraries and a statically linked C++ library.

### 3.4.3 Introduction to the Code

#### *Two Tools*

Besides MPMP, which has its main function in the file `mpmp.cc`, there's another small tool called `mpmp_dat2mat`. It can be built using the `dat2mat` target of the supplied `Makefile`<sup>11</sup>. This tool converts the loaded `.dat` files to `.mat` files which can be loaded with [Matlab](#)<sup>12</sup>. To succeed, it needs two special subdirectories in its working directory, `mat-filtered` and `mat-unfiltered`. The generated `.mat` files contain the heightmap as a matlab matrix. Each matrix element, a pixel of the heightmap, contains a "connected phase value" (s. [B.2.3](#)), an integer type. These values are linearly proportional to the height of the pixel relative to the start

---

<sup>7</sup>KDevelop, nedit, emacs, Borland C++ Builder, MS Visual C++, ...

<sup>8</sup><http://www.gnu.org/software/libc/libc.html>

<sup>9</sup><http://gcc.gnu.org/>

<sup>10</sup><http://www.mingw.org/>

<sup>11</sup>The target gets usually specified as the first option. On a unix system using GNU make [8] this means `make dat2mat`

<sup>12</sup><http://www.mathworks.com/>

position of the measurement scan. The value 2147483640 indicates an invalid pixel, which contains no height information.

Both tools, the main tool `mpmp` and the conversion tool `mpmp_dat2mat`, use the same set of classes. They only differ in their `main` function. The code base is divided into units, each consisting of a header file `.h` and an implementation file `.cc`. There's one class per unit. Exceptions are classes working tightly together. The filenames are the same as the unit's main class name changed to lowercase.

### *Input Format and Selection*

When starting the program `mpmp` it expects a full set of measurement results in the current working directory. This includes the LabVIEW logfiles (s. B.2.2) named according to `bnf_logfilename`, the raw and analysed data files (s. B.2.3) `bnf_rawfilename` and `bnf_datfilename` as well as the pattern files (s. B.2.3) `bnf_patfilename`. All these files need to have the format as specified in appendix B. There's a line similar to

```
const char *Measurement::fnpostfix = ".r.dat";
```

in `measurement.cc`. It decides whether `mpmp` uses the `dat` files containing raw or analysed data. The string `".r.dat"` needs to be changed to `".a.dat"` to have `mpmp` working with analysed instead of raw data.

### *Class Interaction, the Big Picture*

The class `ResultParser` is responsible for creating a list of `Measurement` objects basing on the content of the current working directory. Each measurement process in the directory gets represented by a `Measurement` object. When such an object gets created, it reads the according log file (s. B.2.2) and instantiates a number of `Membrane` objects. Every membrane measured during a measurement process gets thus represented by a `Membrane`<sup>13</sup> object. The content of the current working directory is then stored as a convenient set of objects in memory. The `.dat` files aren't loaded yet.

The `mpmp_dat2mat` tool calls of each `Measurement` object the function `generateMatFiles()`. That executes of every `Membrane` object in memory the `generateMatFiles()` function. This function loads the according heightmap from a `.dat` file (s. B.2.3), makes a copy of it which gets low pass filtered, and then it saves both heightmaps to `.mat` files. The subdirectories `mat-filtered` and `mat-unfiltered` have to exist for this to work. The function to write the `.mat` files is in the file `heightmap.cc`:

```
template<class MapType>
bool HeightMap<MapType>::writeMATFile(const char *fn);
```

---

<sup>13</sup>`Membrane` is an abstract parent class of `MembraneCirc` and `MembraneRect`. Which class finally gets instantiated depends on the geometry type stored in the process logfile.

The class `FIRFilter` does the filtering part. It uses a raised cosine windowed sinc<sup>14</sup> to perform a convolution in time domain. Every column and row of the heightmap gets filtered once as a onedimensional function. The filter compensates the delay to keep the heightmap at its place. Border effects get eliminated by preloading the filter with a number of border pixel values.

The `mpmp` tool calls the `doNumberCrunching(.)` function of each `Measurement` object. That function passes the call on to the `Membrane` objects. Their `doNumberCrunching(.)` function loads the according heightmap, reads the header structure from the `MetroPro .dat` file, performs the `CenterHeight(.)` test as described later and calculates the minimum height of the heightmap. Finally execution gets passed on to the `doGeometryCalculations(.)` function of the childclasses `MembraneCirc` or `MembraneRect` to perform geometry dependent calculations. The heightmap must be deleted after that function returns to save memory resources. Relevant values are stored in the `values_t` and `results_t` structures to be written to a CSV file at a later point.

Most algorithms working on the membranes have limits in their intermediate results. Membranes mark themselves as invalid (`values->valid = false;`) if intermediate results exceed their limits<sup>15</sup>. Only membranes passing all limit tests remain valid. An invalidated membrane doesn't get further processed, control returns to the caller. The corresponding `Measurement` object continues with the next membrane or returns control to its caller, a `ResultParser` object. Work is now finished or continues with the next `Measurement` object if there's data from multiple measurement processes saved in the current working directory.

### Generic Tests

The `Membrane::CenterHeight(.)` function performs a test. It takes a rectangular area in the center of the heightmap. The lateral dimensions of that area are each `CENTER_AVG_PERCENT` of the lateral dimensions of the heightmap. That value is currently<sup>16</sup> set to 30% and can be modified in `membrane.h`. The central area must contain at least `CENTER_MIN_VALID_PTS` (98) % of valid pixels for the test to succeed. This test invalidates membranes which have their cap not measured due to wrong autofocus and scan length parameters (s. 3.3.3). It also recognises a part of the holes where the polymer is not visible because the Si isn't fully etched away. These are no membranes and have to be invalidated. The part wrongly not recognised as invalid gets narrowed down in later, geometry dependent tests.

All other tests and calculations depend on the membrane geometry. They're part of the `MembraneCirc` and `MembraneRect` classes.

---

<sup>14</sup> $\text{sinc}(\omega t) := \frac{\sin(\omega t)}{\omega t}$ , time domain transferfunction of an IIR low pass; in this case multiplied with a Hanning window, also called raised cosine window, to obtain the coefficients for a FIR filter.

<sup>15</sup>Many limits depend on the algorithm being used. Generic limits are explained in section "Generic Tests" on page 24. The limits of the invalid region based algorithm implemented in MPMP are closer explained on pages 28f.

<sup>16</sup>MPMP revision 26, 9th Jan 2005

### 3.4.4 Membrane Recognition Algorithms

#### *Membrane Geometries*

The characterisation of polymeric thin film membranes requires more than one shape of membranes. Thus MPMP needs to be able to not only treat circular but also rectangular membranes. It obeys this fact by providing an abstract base class `Membrane` which gets inherited for every type of geometry, as mentioned above. The decision, which child class gets finally instantiated for every measured membrane, is up to the user. He specifies the geometry type in the measurement process GUI (s. 3.2.3). This information is included in the logfile, where it gets parsed by MPMP.

`MembraneCirc` handles circular membranes and is well tested. `MembraneRect` is implemented but it's not tested yet, because there are no functional, rectangular membranes so far. Thus the following treatment of geometrical issues and image recognition algorithms covers mainly the circular membranes. The problems are the same for rectangular membranes. The algorithms have to be adapted but the ideas remain almost the same.

The aim of every algorithm proposed later is to obtain the radius or width of the polymer membrane excluding any Si border. In addition the peak height relative to the lowest point has to be calculated.

#### *A Perfect Heightmap*

Figure 3.6 shows the membrane 039a\_013-20041213131858-0-3050-16-16. Its height is  $16\mu m$ , the diameter  $1.461mm$ . It was measured at a relative pressure of roughly  $43900Pa$ . The surface plot to the left contains only reduced information as squares of six by six pixels are averaged to form one new pixel. All invalid pixels are set to the same height as the lowest valid pixel of the heightmap. They form the flat bottom plane. The obtained heightmap is plotted. Thus the lateral  $x$  and  $y$  coordinates are also scaled by a factor 6.

The plot on the right side shows a part of the same heightmap but it contains all pixels within that area. The  $z$  axis is converted from "connected phase values" (s. B.2.3), as being used by MetroPro and MPMP, to  $\mu m$ .

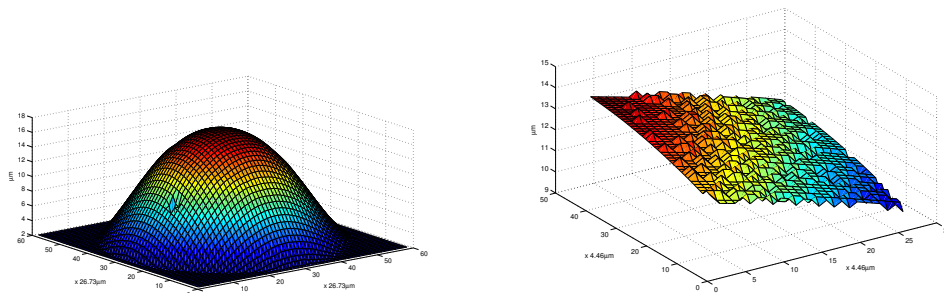


Figure 3.6: Perfect membrane with a slightly rough surface

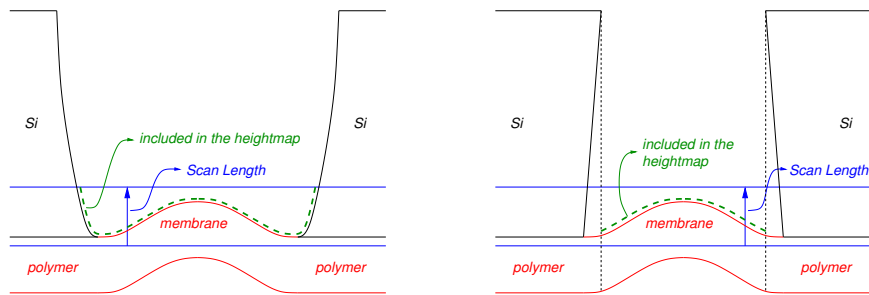


Figure 3.7: Si walls showing up in the heightmap

Perfectly measured heightmaps have a circular area of valid pixels within a rectangular heightmap. The inner, circular area represents a circular membrane. The pixels around are invalid because the scan has not reached a surface since the Si wafer is thicker than the scan length. If the area of valid pixels would be perfectly round and exactly the border of the membrane, it wouldn't be any problem to figure out its center and radius. But this is not always the case.

### *Problematic Heightmap Characteristics*

Regions with steep slope can result in regions of invalid height information. On circular membranes this is a ring of invalid pixels concentric with the membrane. The inner ring might be partially, radially connected to the outer region of invalid pixels.

Membranes are not always deflected upwards. Measurements without applied pressure can show membranes slightly bent downwards. It's important to sort out such membranes.

The hole in the Si wafer might not be fully etched through down to the polymer. This results in membranes halfway covered with Si. The etched Si is rougher than the polymer surface.

Whether a part of the Si hole wall gets measured or not depends on the etching process parameters used to fabricate the membranes. MPMP has to work with both results, with wafers where NewView can see the wall and such where it just gets invalid pixels at the membrane border.

Figure 3.7 shows on the left hand side a membrane where the Si wall is included in the heightmap. Desired is only the region where the red, thick line indicating the membrane and the green, dashed line overlap. Additionally included in the heightmap is the region where the green, dashed and the black, thick lines overlap. On the right hand side the etching process generated walls NewView can't see. The membrane area not visible for NewView is in practice so small that it can be neglected.

Figure 3.8 shows a surfaceplot and a profileplot of membrane 039a\_008-20041201133103-0-4000-1-10. This surfaceplot contains only  $\frac{1}{36}$  of all

pixels of the original. Invalid pixels are set to the same value as the lowest, valid pixel. The profileplot contains full height information. The high, flat regions at the left and right border of the profile indicate invalid height information.

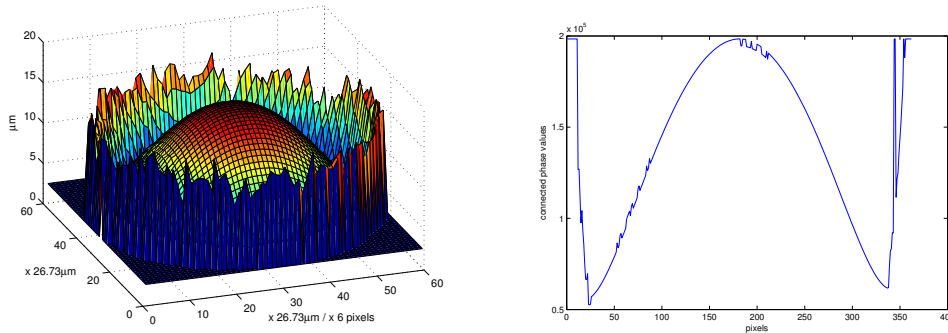


Figure 3.8: Plots of 039a\_008-. -0-4000-1-10, Si border visible

Yet another problem are particles and defects on the membrane surface. Sometimes they give random height information and sometimes they just introduce invalid regions at unexpected locations. The right plot in figure 3.8 has such an invalid spike at the Si wall on the right hand side.

The aim of the following algorithms is to find the center of the membrane to determine the radius and profile afterwards. The following sections illustrate three different approaches. The first doesn't work because it assumes a mathematically perfect, continuous membrane. The second explained is implemented in MPMP. It uses the outer regions of invalid pixels to find the center of the membrane and subsequently acquire more information about the membrane. The third is an illustration of some ideas for a possible future algorithm.

### *Slope based*

The simplest, mathematical way of finding local minima is to differentiate a function and set it equal 0. Subsequent solving delivers the desired position of the minima. This works fine as long as the function is continuous. Because the heightmaps used here are discrete, this doesn't work any more. So the adoption is to take some pixel and always walk towards a lower pixel.

But such algorithms don't take the dimensions of the membrane into account. They treat noise<sup>17</sup> minima the same way as the peripheral minima of the membrane. Only the latter are wanted, all others disturb. Even after filtering the heightmap with a low pass filter, results don't get much better. There might be invalid regions disturbing the filter. Finally the problems remain. Algorithms detect minima and maxima that aren't of interest.

<sup>17</sup>in this context noise is the same as surface roughness

### Invalid Region based

The second approach is much more promising. It's the solution currently used by MPMP.

As MetroPro truncates the heightmap to a minimal rectangle containing the valid pixels, it's assured that the center of the heightmap lays somewhere within the membrane. A *step angle* of about  $1^\circ$  or  $2^\circ$  (degrees,  $30^\circ$  in figure 3.9) is then defined. For every increment of *step angle*, starting at angle  $0^\circ$ , the algorithm walks from the border radially towards the center of the heightmap. It stops on the first occurrence of a valid pixel (red circles in figure 3.9). For every *step angle* the coordinates of the first valid pixel get saved. At the end there's an array of about 200 coordinates of valid pixels. All lay on the border of the circular hole where the membrane resides in the Si wafer. The hole and the membrane are concentric, whether there are Si walls on the heightmap or not doesn't matter yet. Having that it's possible to calculate the center of the membrane. It's the point of gravity of the border coordinates. The next step is to calculate the average radius of the hole, based on the same border coordinates. After that the radius gets optimised. In other words, it gets moved from the border of the hole to the border of the membrane. The center remains the same. Optimisation is done by scanning over a range of radii. At each radius the average, peripheral height gets calculated. The radius pointing to the lowest average, peripheral height is expected to be the radius of the membrane.

This algorithm is implemented in MPMP. It's enhanced by a number of plausibility tests. Membranes get marked as invalid if one of the tests fails. Entry point of the algorithm is the function `doGeometryCalculations(.)` of the class `MembraneCirc`. The function `getXYPhi(.)` searches the points at the border of the hole. For each angle it calls the `seekRadius(.)` function to find the ac-

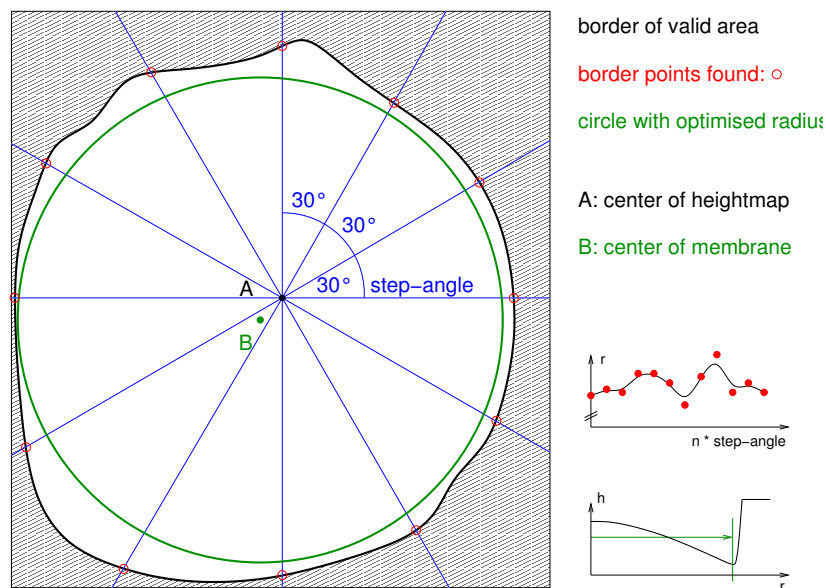


Figure 3.9: How invalid region based membrane recognition works

coding border point. The point gets stored in the `pos` array. The number of border points searched for is `RADIUS_TEST_PHI_NUM`, defined as 200 in `membrane.h`.

It follows the function `getCenterOfGravity(.)` to determine the center of the membrane. The result gets stored in `results->center`. The knowledge about this point gives the possibility for a first plausibility test. The function `getRadiusPhi(.)` creates an array of radii between each hole border point and the center just found. The result of that step is shown in figures 3.10, 3.11 and 3.12 in the left plot<sup>18</sup>. Wafer 039a\_013 has very steep Si hole walls. They don't show up on the heightmap as it would be the case with wafer 039a\_008. This is the reason why the outer ring of circles is very close to the inner, regular ring of circles in figures 3.10 and 3.11. The inner ring shows the radius found after optimisation. The outer circles represent the points found in the first step of the algorithm, the hole border points. The inner ring of circles is so regular because it's plotted as a function of the radius found. Figure 3.12 shows a big distance between the inner and outer ring because it's a measurement at wafer 039a\_008.

The rounder a hole is, the less differ maximum and minimum of the obtained radius signal, respectively function. After filtering that signal with a low pass filter, the average  $r_{avg}$ , maximum  $r_{max}$  and minimum  $r_{min}$  get calculated. The *angularity* obtained with

$$angularity = \frac{r_{max} - r_{min}}{r_{avg}}$$

shows whether the hole found looks like a circle or not. Membranes with an  $angularity > 0.1242640687 = (\sqrt{2} - 1) \cdot 0.3$  get marked as invalid. An *angularity* of  $\sqrt{2} - 1$  is the minimum for a rectangle, which would be a square. The initial intention of this value was to automatically distinct between circular and rectangular membranes. But this is put under user control and the *angularity* remains as a plausibility test. The test is mainly useful to sort out heightmaps containing rings or partial rings of invalid regions. This invalidity happens when the autofocus locks to the membranes top, and the scan length is too short to measure the peripheral valley. This is shown in figures 3.13 and 3.14.

Once the radius is obtained the function `handleCircular(.)` does the rest. First the radius gets optimized to point to the peripheral valley of the membrane. The results are the green, inner rings of circles in the plots on the right of figures 3.10 - 3.13.

It follows the next plausibility check. The original radius of the hole and the optimized radius get compared. The ratio must fit the following rule to pass the test:

$$\frac{r(valley\ optimized)}{r(original\ hole)} \geq RADIUS\_CHECK\_MIN\_VALID := 0.7$$

The remaining part of the `handleCircular(.)` function handles the membranes tilt, calculates the peak heights and extracts a profile through the center of the membrane. The extracted profile is being kept until MPMP gets terminated, whereas the heightmap gets deleted from memory.

<sup>18</sup>These plots were created by just executing the `r*_r.m` Matlab script belonging to the desired membrane. The files get written by MPMP after working on a circular membrane. The x and y axes of the right plots are in pixels.

Results of the algorithm get written to the `results` and `values` storage structures. Relevant results are written to a file at a later point in the program. Responsible are the functions `writeHeaders(.)` and `writeData(.)`, both part of the class `MembraneCirc`. Every child class of `Membrane` has to implement these two functions.

The `MembraneRect` class works nearly the same way. Instead of searching and optimising radii, it lays tangents at the Si trench and optimises them. Unfortunately there aren't any useful, rectangular membranes available until now. So this part of MPMP is not tested. It compiles and doesn't crash.

Manual inspection of results from circular membranes showed that MPMP is tuned to throw away nearly all strange looking heightmaps / membranes. But there seems to be no mathematical way to prove its correctness. The number of different heightmaps is large and they're unpredictable. It's possible to get incorrect measurement results. A critical look at them and comparison with already verified and expected results is thus necessary.

This algorithm might give wrong results when the membrane is tilted. As treated in a later section, membranes are usually tilted less than  $1^\circ$ . The algorithm invalidates those tilted more than  $1^\circ$  out of vertical direction.

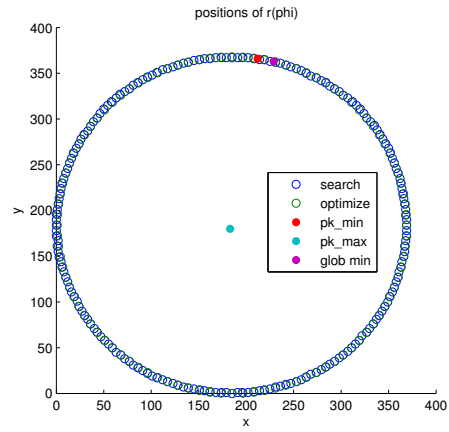
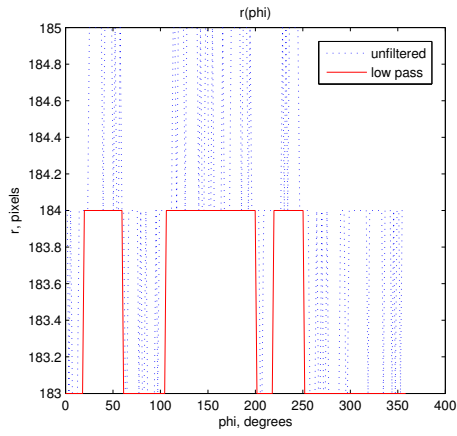


Figure 3.10: Radius detection of 039a\_013-. -0-3050-14-14, perfect

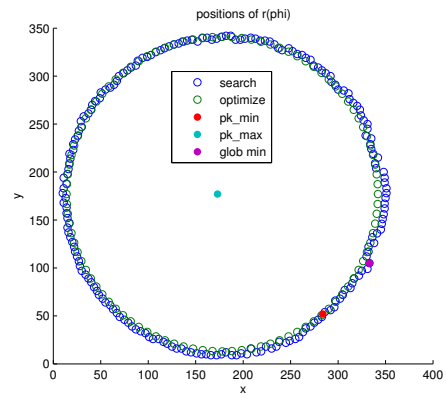
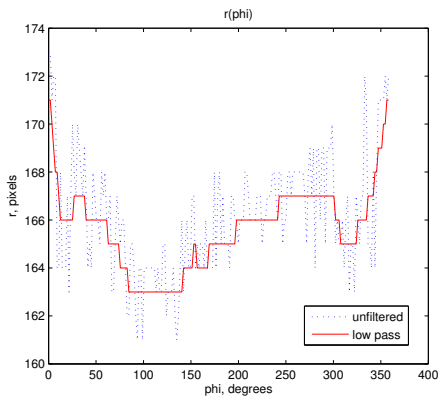


Figure 3.11: Radius detection of 039a\_013-. -0-3050-16-16, noncircular

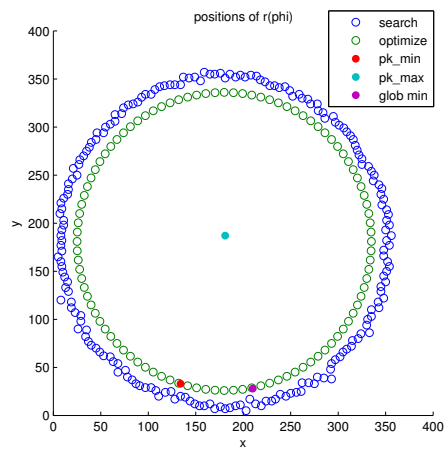
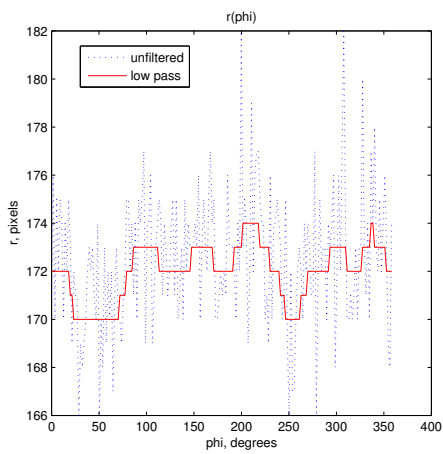


Figure 3.12: Radius detection of 039a\_008-. -0-4000-1-10, Si border

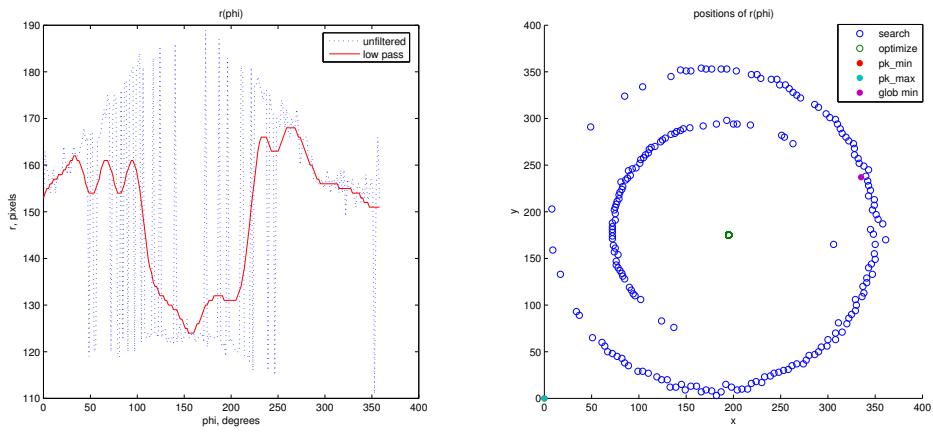


Figure 3.13: Radius detection of 039a\_008-. -1-3000-12-16, bad ring

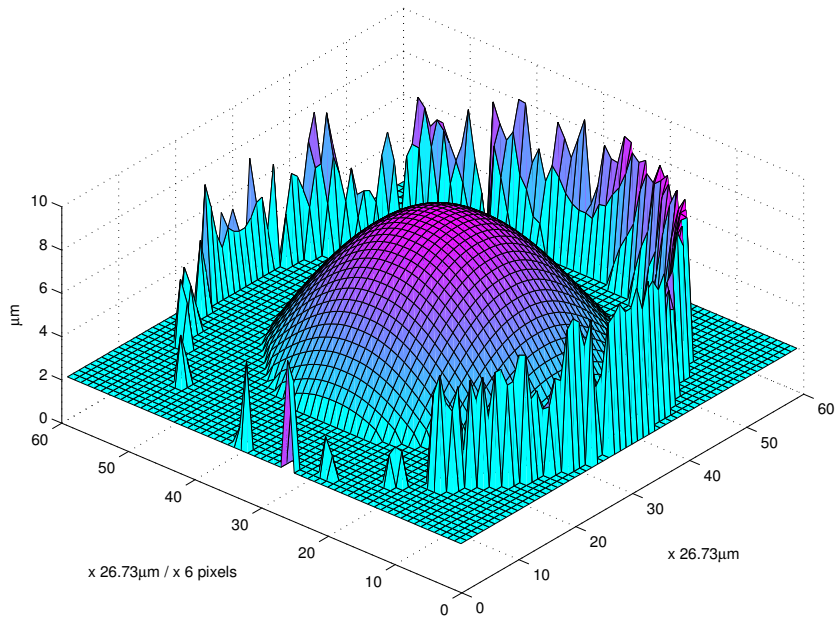


Figure 3.14: Surfaceplot of 039a\_008-. -1-3000-12-16, bad ring

### *Roughness based*

There's an idea for another possibility to differentiate polymer from Si wall surface. Due to a lack of time this algorithm has neither been implemented nor verified in another way.

By partitioning a heightmap into squares of  $n$  by  $n$  pixels and calculating the frequencies on each square, a value for the roughness of the surface at the squares location is obtained. Calculation of the frequencies on a square area can either happen via a 2D FFT<sup>19</sup> or by calculating RMS values of the difference between each pixel of the square and the plane where the square lays in, defined by a least square fitting or something similar. It's probably necessary to shift the whole square pattern by one pixel in each lateral direction and recalculate the characteristic roughness value again for each square. Repeating this step for every possible pixel offset between 0 and  $n - 1$  in each lateral direction should lead to a finer grained localisation of the membrane border. Every pixel of the heightmap is the center of a square once during shifting. That's a possible way to recombine the frequency values into a new frequency map. The problem of locating a certain region in a heightmap remains the same, but this time with other data. Using a suitable threshold value it could be possible to locate the membrane border. Pixels residing on the membrane get lower frequency respectively roughness values than those on the Si wall.

### **3.4.5 Tilting Issues**

#### *The Problem*

Before starting an automated measurement process, the membranes get leveled. This is very exact, as it's done by observing the fringes (s. 1.3.1) on a membrane. But it's only being done for one membrane on the wafer. When pressure gets applied, the whole wafer will buckle a bit upwards and thus pull the membranes out of the lateral plane.

The peak height of a tilted membrane is not the difference between the lowest and highest points of the heightmap. The heightmap would have to be rotated in 3D space to put the peripheral valley of the membrane back onto the lateral plane before the peak difference gets calculated. But there's only a very limited lateral resolution and exact rotation is not possible using discrete values.

#### *Current Solution*

MPMP doesn't rotate the heightmap. As they're only slightly tilted, calculations are made while ignoring the tilt angle. At a certain point, when enough about the location and radius of the membrane are known, an untilting function comes in effect. The original radius is optimised again and results depending on the tilt angle get recalculated.

---

<sup>19</sup>small effort using available libraries such as FFTW: <http://www.fftw.org/>

It turned out that the effect of the untilting function is near zero. Tilt angles are so small that they could be neglected.

The untilting function uses 8 byte floating point values without additional discretisation. That way the original height resolution gets maintained. The resulting corrected height values get rounded to fit into the heightmap of type integer. The error introduced is less than the vertical resolution of NewView.

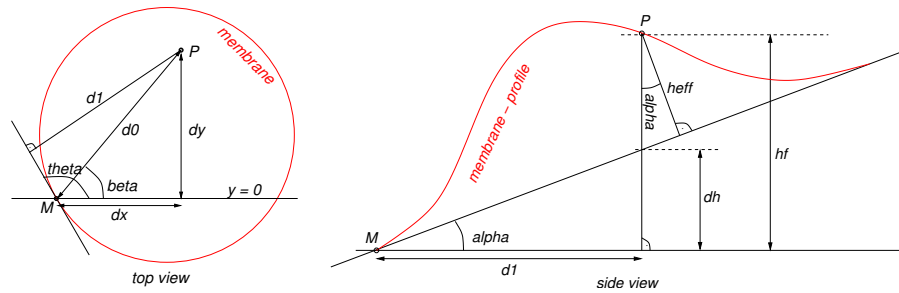


Figure 3.15: Membrane untilting trigonometry

The lengths and angles further used are illustrated in figure 3.15. First, the lowest point of the original heightmap gets determined. This is the touch-down point  $M$ . Its height is not affected by the tilt angle  $\alpha$ .  $P$  is the point whose effective height, relative to point  $M$ , has to be calculated. To obtain  $\theta$ , pairs of opposite pixels at the radius are taken. Their height difference leads to a local tilt angle  $\alpha_i$ , their lateral orientation to  $\theta_i$ . The line through the pixel pair with the lowest  $\alpha_i$  is parallel to the tangent at the membrane in point  $M$ . Thus  $\theta$  is equal to the  $\theta_i$  corresponding to the lowest  $\alpha_i$ .  $\theta$  and  $\alpha$  are a parametrisation of the plane in which the membrane lays.  $\alpha$  is of course the same as the tilt angle `MembraneCirc::results->tilt`. MPMP internally uses *radians*. The effective height  $heff$  of  $P$  can be calculated in four simple steps:

$$\beta = \arctan\left(\frac{dy}{dx}\right) \quad d1 = d0 \cdot \sin(\theta - \beta)$$

$$dh = d1 \cdot \tan \alpha \quad heff = (hf - dh) \cdot \cos \alpha$$

MPMP uses the same equations in a modified form reducing the number of required trigonometrical functions.

Table 3.5 is the result of measurement 039a\_013-20041213131858. It includes 16 membranes measured at pressures between 800 and 3050 in steps of 250 Bronkhorst internal units (s. 1.3.5). This gives 160 measurements, one of them was correctly determined as invalid. The second and third columns show how many membranes are tilted more than the angle listed in the first column. The largest deflection of these membranes is  $17.6\mu m$  at a pressure of  $43471Pa^{20}$ . The diameter is  $1.64mm$ . The radius detection result is shown in figure 3.10. The membrane's tilt angle is  $0.103^\circ$ .

<sup>20</sup>Low accuracy, see appendix C for details

tilt angle	# memb.	%
> 0.0125°	158	99
> 0.025°	142	89
> 0.05°	115	72
> 0.075°	81	51
> 0.1°	53	33
> 0.15°	25	16
> 0.2°	7	4
> 0.3°	0	0

Table 3.5: Tilt angles of membranes in 160 measured heightmaps

Calculating the lateral shift of the topmost point of a membrane with  $17.6\mu m$  deflection and a tilt angle of  $0.5^\circ$  by a rotation to level the membrane reveals  $\approx 3.4\%$  of the lateral resolution ( $4.455\mu m/pixel$ ). The center of the rotation is assumed to be the central point of the circle defined by the lowest points forming the peripheral valley.

$$\text{shift} = \text{height} \cdot \sin \alpha$$

At a tilt angle of  $1^\circ$ , the maximum allowed, a peak height of  $\approx 127\mu m$  is required to dislocate the center of the membrane, the topmost point, by  $\frac{1}{2}$  pixel. Measuring membranes deflected to that height with a membrane thickness of  $\approx 50\mu m$  would rise problems with NewView, because multiple, transparent surfaces, top and bottom of the membrane, are within the scan length. Such measurements get marked as invalid by MPMP because of too many invalid pixels.

It's thus save to neglect the lateral shift of positions due to tilting.

## Conclusion

The goal is to automatically measure a large number of membranes with an existing measurement setup. The developed software provides an extensible platform allowing the unattended measurement of over 100 membranes per hour.

The developed MetroPro script and the assembled GUI leave as much freedom as possible to the user. They require advanced knowledge about MetroPro features, especially the autofocus parameters. This makes the setup more complicated but at the same time more powerful and flexible. It can be adapted to new patterns of membranes and different types of polymers.

The need for a post processing tool added a new aim to the initial assignment. The developed and implemented image recognition algorithm gives promising, experimental results. The MPMP tool is designed as a flexible, portable framework of C++ classes.

## Appendix A

# User Guide

### A.1 Prerequisites

Before starting to take measurements it's necessary to get familiar with MetroPro [6]. Thorough understanding of the autofocus parameters is highly recommended. Slightly wrong parameters can lead to a lot of invalid measurements.

This is no introduction on how to operate NewView. Users should get authorised to work with it before taking measurements.

For each of the two PCs a separate user account is required.

### A.2 Starting the Software

Open the `do-measurement.vi` and close all additionally appearing VIs. They will pop up automatically when needed.

Power up and connect LabJack, the Burster pressure sensor and the Bronkhorst flow controller. Open the N<sub>2</sub> bottles main valve and the valve at the exit of the pressure reduction valve.

Start the FlowDDE application and let it connect to the Bronkhorst flow controller (s. 1.3.5). `do-measurement.vi` will start it if it's not started already. So this step might be omitted. Skipping this step causes the VI to wait about 30 seconds until the FlowDDE application is running.

Start MetroPro and run the home script. This initialises the absolute coordinates of the stage axes. It doesn't need to be done again after restarting MetroPro. It's only necessary after powering up the NewView and its motion controller.

Fix the wafer in the holder. Put the wafer holder on the stage, centered, and immediately set the **Z-Stop** position at its correct height.

The `do-measurement.vi` is already open, start it.

### A.3 The 4 Steps

#### Step 1

Each step has a corresponding box in the GUI of `do-measurement.vi`. There's no way to omit a step or change the order of their execution.

Fill in the wafer ID. Keep it short and simple as it's a part of the filename for files being created. Names like 039a\_008 are suggested. An exact definition of characters allowed is in appendix B.

Set the correct geometry type. This gets written to the logfile for later processing by MPMP (s. 3.4).

In the `comment` field, a textual description of the wafer can be entered. For example a history of the wafer and how it was produced should be declared there. Multiple lines are allowed but they get collapsed to one line in the log file.

The `base path` field must contain the path to a directory where you're allowed to create files. It must not end with a "\" (backslash). All logfiles (s. B.2.2) will be stored there. They will contain the pressure, temperature, relative humidity and information about each measured membrane.

## Step 2

There's a number of buttons in step 2. Each button corresponds to a configuration step in MetroPro or the N<sub>2</sub> flow system. Work through all points, click the buttons as you've done what is requested. The step is finished when all buttons are clicked. Don't use the express button at the bottom right corner unless you're really sure that everything is properly done.

1. Homing the stage has probably already been done before setting the Z-Stop. There's no need to do this twice, mark it as done.
2. Leveling the stage is the same as "killing fringes". Take a membrane somewhere in the center of the wafer holder. Make sure you got the top surface. It might be impossible that the fringes disappear. But it's very important that the fringes build circles concentric with the membrane if they don't disappear. When focus zero is set, drive the z axis 20 to 25 $\mu$ m upwards and set focus zero there. This prevents the membranes from falling out of the `Focus Max Adjust` range once pressure gets applied.
3. Turn to MetroPro. The control labeled "`Base Dir:`" tells where the measurement process data files will get saved. They go to the directory `output` within the directory entered in that control. The directory entered in the control must not end with a backslash "\".

Use the "Edit Pattern" button to display the pattern editor. Load a pattern and modify it to have the desired number of columns and rows. It must be a rectangular pattern. Set "Origin Loc: Upper Left" and "Order: Rows (Serp)" to use the best tested coordinate transformation of MPMP. Then move the stage to get the upper left membrane of the pattern into view. Center it on the NewView Monitor. Then click "Set Origin". Close the pattern editor.

Now click the "Align Pattern" button. On the first dialog click "ok". Then choose "Goto Origin". Select twice "Yes" to calculate row and column spacing. Use the joystick to move to the last column, then click "ok". Take again the joystick to move to the last row of the last column and click again "ok". Make sure the membrane is in the center of the picture as exactly as possible

when clicking "ok". Say "Both" when asked for the alignment type. Click "Done" to complete the alignment.

Next you might want to check for proper alignment. Click "View Pattern" and have the stage moved by clicking on the desired location symbols. At this point it's possible to exclude arbitrary pattern positions from the measurement process. Use the "Edit Pos" button to display the pattern position editor. Change the "I" at the unwanted locations to an "X" to have them excluded. Just click on the location's symbol to change it.

4. It's now the last moment to switch on the sensors and open the valves for the N<sub>2</sub> if not done yet.
5. There are two valves at the inlet to the wafer holder (s. 1.3.2). One blocks the way to the wafer holder. Close this valve. The other lets the N<sub>2</sub> escape to enable a flow through the Bronkhorst controller but not through the wafer holder. Open this valve only a little bit.
6. Start the MetroPro script `auto-membranes.scr` by clicking on the "Give Control to LavVIEW" button. Don't touch MetroPro anymore before the measurement process terminated. Aborting the script is possible by pressing ESC twice and waiting a while. But this is absolutely the wrong way to abort a measurement process. Use the "STOP" button of active VIs (s. 1.2.1) to do so, if necessary.

### Step 3

Before measuring, the pressure ramp has to be configured. It's possible to use a ramp, occasionally containing only a single pressure level, or to measure a pattern at zero pressure. Starting the ramp at zero pressure and then ramping the pressure up is not possible because manual intervention would be necessary. Every position included in the pattern gets measured once per pressure level.

A zero pressure measurement is done as in table A.1. It's of course not zero pressure but zero relative pressure...

Ramp start pressure	0
Number of pressure levels	1
Ramp increment	0
double sided ramp	off
end with 0 pressure	off

Table A.1: Zero pressure measurement configuration

A pressure ramp gets configured by giving the start pressure level, the number of pressure levels and the pressure difference between two subsequent levels. Specifying ramps going downwards is not possible. All units get specified in Bronkhorst flow controller internal units (s. 1.3.5). The minimal, working start pressure not treated as a zero ramp is 650 units. Once entered, the pressures get displayed in % and Pa too.

In addition to the normal ramp there are two options. "Use double sided pressure ramp" will measure all levels twice except the highest one. It will ramp up from the start pressure to the highest pressure using the number of pressure levels specified. Then it will ramp down to the start pressure and measure everything again. The other option can be used to append a measurement at zero relative pressure. These two options are independent. In this context, a measurement means to measure every included pattern position once.

Clicking the "start measurement" button passes control to the VI until the whole measurement process is finished.

## Step 4

At the very beginning of the measurement process, the user's presence is required. Initially the N<sub>2</sub> stream doesn't flow through the wafer holder. Once the Bronkhorst flow controller opens its valve, the green ones at the wafer holder need to be switched. You're prompted to do this. During the intervention, a pressure chart is shown and the pressure gets logged to a file about 4 times per second. Click "ok" when the valves got switched. This switching of the N<sub>2</sub> flow is necessary, because the Bronkhorst controller generates a large overshoot when it turns on the flow. Now you might want to walk away... just do it.

Status information is shown. The percentage of the measurement process already finished is not correct at the beginning. It becomes correct at the second pressure level. Initially, `do-measurement.vi` doesn't know anything about the pattern loaded. The actual number of included pattern positions has to be counted during the first pressure level.

## A.4 Collecting Results

Collect all files generated during the measurement process. Each measurement process creates two files on the LabVIEW PC in the base path and a lot of files in the `output` directory on the MetroPro PC. Put all the files into one directory. Optionally, it's good to make one directory per measurement process.

The `.a.dat` files (s. [B.2.3](#)) contain only analysed data. These files can be recreated by loading the according `.r.dat` file into MetroPro and analysing the data. Besides a few hours of work you don't loose anything when deleting the `.a.dat` files.

## A.5 Post Processing

Open a console and change to the directory containing all output data of at least one measurement process. Run MPMP in that directory. The output will be one

file per measurement process. It's name matches the expression `*.csv.txt`. Use OpenOffice.org or MS Excel to open it.

How MPMP has to be executed depends on the location of the `mpmp.exe` file. Sources are available at <http://srf.ch/bitbytes.php>. It can be compiled on GNU Linux using `g++` or on Windows using [MinGW](#)<sup>1</sup>. There's a separate Makefile for each system. Use `Makefile.w32` to compile it on Windows.

---

<sup>1</sup><http://www.mingw.org/>

## Appendix B

# File and Identifier Format Definition

## B.1 About the Definitions

The measurement process (s. 3.2.2) generates a lot of data. A certain consistency is required within that data.

There's a special syntax, the [Augmented Backus Naur Form](#) [9], in short ABNF, to define or describe syntactical relations and formats. The ABNF is a nice tool to describe IDs, filenames and file formats.

## B.2 Files and Identifiers

All ABNF elements used in the following explanations are defined in section B.3. They're prefixed with `bnf` if occurring within the main text.

### B.2.1 WaferID and Measurement Process ID

Each measurement process (s. 3.2.2) started gets its unique measurement process identification number, in short measurement ID. The format of the measurement ID is defined as `bnfmeasid`. It's a timestamp of the time when the measurement process was started. The measurement ID is unique for sure as there's only one instrumentation which can't have more than one measurement process running at the same time.

The wafer ID is the first part of most log- and datafilenames as this makes sorting of measurement results easier. This is relevant when entering a wafer ID as defined by `bnfwaferid` into the GUI. The OS and MPMP (s. 3.4) should be able to handle all IDs valid according to `bnfwaferid`. But it makes sense to use only letters, numbers and underscores.

### B.2.2 Files on the LabVIEW PC

#### *Measurement Process Logfile*

All data acquired during a measurement process (s. 3.2.2) on the PC running LabVIEW (s. 1.3.3) gets logged to the file named as `bnflogfile`. The file has a header `bnfllhead` of three lines which contain the wafer ID, the measurement ID, the comment entered by the user and the ramp settings. The rest of the file consists of one line of data per membrane measurement. The format of such a line is `bnflldata`. If it's followed by a line defined by `bnfllerror`, then the previous

`bnf11data` line indicates an invalid measurement. All measurement results in relation with that line must be ignored. The `bnf11data` and `bnf11error` lines might be interspersed by any number of `bnf11empty` and `bnf11comment` lines.

Each valid `bnf11data` line has a number of corresponding data files on the MetroPro PC. The most important one is `bnfrawfilename` (s. B.2.3) which contains the raw, measured heightmap.

### *Switch-On Pressure Transient Logfile*

When starting a measurement process which uses a pressure ramp, it's necessary to manually change the N<sub>2</sub> flow by turning valves so it goes through the wafer holder (s. A). This is required due to the unavoidable switch-on overshoot of the Bronkhorst flow controller (s. 1.3.5). While doing this, an unreproducible pressure overshoot is generated. That's the reason why the pressure measured with the Burster sensor (s. 1.3.6) needs to be logged. That pressure gets logged in *Pa* approximately 4 times per second to the file named as `bnfplogfname` with format `bnfplogformat`.

## **B.2.3 Files on the MetroPro PC**

Every measurement of a membrane at a certain pressure generates two files. One file contains the raw measurement data and the corresponding measurement parameters. The other file of the same MetroPro `.dat` file format [6] contains analysed data. What is being done when MetroPro analyses the data can be freely configured by the user. The default of the `Auto-Membranes.app` application is to remove a plane to level the measured membrane.

Additionally, a pattern file is saved at the beginning of every measurement process.

All files generated during a measurement process end up in the `output` subdirectory of the configured base directory. The base directory can be specified in the MetroPro application `Auto-Membranes.app`. The control is labeled `Base Dir:`. Its internal object path is `"Controls/Custom/Text 1"`.

### *MetroPro DAT Files*

These files have a custom, binary format described in the MetroPro Reference Guide [6]. An extensible C++ class implementation to load them is available in the MPMP (s. 3.4) source code<sup>1</sup>. Errors occurring in the manual are corrected in that class. The MetroPro DAT files are in big endian format! They contain a full set of MetroPro parameters and NewView settings besides the measured data.

The heightmap (s. 1.3.1) in these data files is stored in a NewView internal unit called "connected phase value". This unit is in linear proportional relation to the

---

<sup>1</sup>this class assumes to run on a little endian machine and thus performs byte swapping. Little endian means that the byte containing the least significant bit of a multi byte value gets stored at the lowest memory address occupied by that value (x86, ...). The opposite is the big endian convention where the byte containing the most significant bit of the value comes first (Sparc, 68000 family, ...).

height relative to the start of the scan length<sup>2</sup>. MPMP (s. 3.4) calculates the multiplication factor for the conversion from "connected phase values" to meters by using values in the .dat files. Each valid pixel of the stored heightmap has to be multiplied by that factor to get its height in meters.

Saved in the dat files is usually only a rectangular area containing all valid pixels instead of the whole heightmap. The rest of the heightmap, which contains only invalid pixels without height information, is truncated. Actual behaviour depends on the MetroPro configuration.

Valid filenames for these .dat files in a measurement process context are `bnfrawfilename` for those containing raw data and `bnfdatfilename` for files containing analysed data. What happens during analysis is up to the user and freely configurable within the limitations of MetroPro.

### *PAT Files*

Pattern files contain the parameters of the loaded pattern. This allows reconstruction of the indexing scheme which was used during the measurement process. They're named according to `bnfpatfilename`. The content is saved in an ASCII formatted, human readable `<key> <value>` manner.

### *MPMP Output*

The MPMP tool (s. 3.4) creates files containing comma separated values. It doesn't make sense to describe the content here as it can easily be specified in the sourcecode of the tool. The files created by MPMP are named according to `bnfmpmpcsvfile`. There's a header line describing its columns.

## **B.3 Format Definitions in ABNF**

```
FLOAT1      = 1 *DIGIT "." DIGIT
FLOAT3      = 1 *DIGIT "." 3DIGIT
YEAR        = 4DIGIT
MONTH       = 2DIGIT           ; 1-12
DAY         = 2DIGIT           ; day of month
HOUR        = 2DIGIT           ; 0-23
MINUTE      = 2DIGIT           ; 0-59
SECOND      = 2DIGIT           ; 0-59

waferid     = 1*( ALPHA / DIGIT / "_" / "-" )
measid      = YEAR MONTH DAY HOUR MINUTE SECOND
              ; e.g. 20050102111541
```

---

<sup>2</sup>for building a heightmap, NewView scans a certain vertical range from bottom to top.

```

geometry      = "1" / "2"      ; 1:circular / 2:rectangular
rampside     = "0" / "1"      ; 0:down / 1:up
bronksoll    = 1*5DIGIT      ; internal unit
timeofs      = 1*DIGIT       ; seconds since start of process
pos          = 1*DIGIT
posfull      = pos           ; all pos enumerated
posincl      = pos           ; included pos only
bronkist     = FLOAT1        ; [Pa]
bursterist   = FLOAT3        ; [Pa]
temp         = FLOAT3        ; [°C]
relhum       = FLOAT3        ; [%]

datprefix    = waferid "-" measid "-" rampside "-"
              bronksoll "-" posincl "-" posfull
patfilename  = waferid "-" measid ".pat"
rawfilename  = datprefix ".r.dat"
datfilename  = datprefix ".a.dat"
llerror      = "ERROR:" *VCHAR CRLF
llcomment    = "#" *VCHAR CRLF
lldata       = timeofs ", " geometry ", " posincl ", "
              rampside ", " bronksoll ", " bronkist ", "
              bursterist ", " temp ", " relhum CRLF
llhead       = "#" " waferid " - " measid CRLF
              llcomment
              "# start=" bronksoll ", increment=" bronksoll
              ", ende=" bronksoll
              ", [Bronkhorst sensor-units]; steps=" 1*DIGIT
              ", double-sided-ramp=" ( "FALSE" / "TRUE" ) CRLF
llempty      = CRLF
logline      = lldata / llerror / llcomment / llempty
logfile      = llhead 1*logline
logfilename  = waferid "-" measid ".txt"

plogfname    = waferid "-" measid ".p-log.txt"
plogline     = FLOAT1 CRLF
plogformat   = *plogline

mpmpcsvfile  = waferid "-" measid ".csv.txt"

```

## B.4 Example files

### logfile

This is an example of a `bnf` logfile generated by `do-measurement.vi`. The `bnf` `timeofs` values are faked. Most lines are excluded and replaced by "...".

```
# 039a_013 - 20041213131858
# testrun with circular membranes
# start=800, increment=250, ende=3300, [Bronkhorst sensor-units]; steps=10, double-sided-ramp=FALSE
1, 1, 1, 1, 800, 9975.0, 15780.847, 22.790, 36.124
30, 1, 2, 1, 800, 9987.5, 15865.012, 22.780, 36.123
60, 1, 3, 1, 800, 9950.0, 15822.929, 22.770, 36.122
90, 1, 4, 1, 800, 10012.5, 15949.176, 22.770, 36.191
120, 1, 5, 1, 800, 9975.0, 15738.765, 22.740, 36.256
150, 1, 6, 1, 800, 9987.5, 15738.765, 22.700, 36.252
180, 1, 7, 1, 800, 9975.0, 15780.847, 22.710, 36.116
210, 1, 8, 1, 800, 9987.5, 15822.929, 22.720, 36.186
240, 1, 9, 1, 800, 9987.5, 15780.847, 22.730, 36.187
270, 1, 10, 1, 800, 9987.5, 15780.847, 22.740, 36.256
300, 1, 11, 1, 800, 9962.5, 15780.847, 22.740, 36.188
330, 1, 12, 1, 800, 9987.5, 15780.847, 22.750, 36.189
360, 1, 13, 1, 800, 10050.0, 15822.929, 22.740, 36.188
390, 1, 14, 1, 800, 9987.5, 15865.012, 22.720, 36.117
420, 1, 15, 1, 800, 9975.0, 15738.765, 22.720, 36.083
450, 1, 16, 1, 800, 9975.0, 15907.094, 22.720, 35.911
480, 1, 1, 1, 1050, 13112.5, 19147.428, 22.710, 36.116
510, 1, 2, 1, 1050, 13100.0, 19105.346, 22.730, 36.187
540, 1, 3, 1, 1050, 13125.0, 19021.181, 22.720, 36.049
570, 1, 4, 1, 1050, 13100.0, 19105.346, 22.720, 36.014
...
4710, 1, 14, 0, 3050, 37962.5, 43470.973, 22.740, 36.050
4740, 1, 15, 0, 3050, 37987.5, 43555.138, 22.740, 36.016
4770, 1, 16, 0, 3050, 37937.5, 43891.796, 22.750, 36.017
```

### MPMP output

These are the results obtained by parsing the above file with MPMP. The third line, containing the column descriptions, is broken into two lines to fit onto this page. Most lines are not included. They're replaced by "...".

```
"20041213131858", "039a_013"
"cam_res [m/pixel]:", 4.455308e-06, "phase2height [m]:", 7.910157e-11

"id", "valid", "row", "col", "ramp_up", "brnk_soll[32000]", "brnk_ist[Pa]", "burster_ist[Pa]",
"temp[degC]", "rel_hum[%]", "diameter[mm]", "peak[um]", "tilt[deg]", "prof_len", "profile[um]"
"1-800-5", "yes", 2, 4, "up", 800, 9975.0, 15738.8, 22.74, 36.256, 1.586, 6.6784, 1.342385e-01
"1-800-15", "yes", 4, 2, "up", 800, 9975.0, 15738.8, 22.72, 36.083, 1.622, 6.4523, 8.213242e-02
"1-800-6", "yes", 2, 3, "up", 800, 9987.5, 15738.8, 22.70, 36.252, 1.622, 6.7964, 7.009654e-02
"1-800-1", "yes", 1, 1, "up", 800, 9975.0, 15780.8, 22.79, 36.124, 1.577, 8.0389, 1.593411e-01
"1-800-9", "yes", 3, 1, "up", 800, 9987.5, 15780.8, 22.73, 36.187, 1.613, 6.4785, 5.849786e-02
"1-800-7", "yes", 2, 2, "up", 800, 9975.0, 15780.8, 22.71, 36.116, 1.631, 6.4641, 6.254305e-02
"1-800-12", "yes", 3, 4, "up", 800, 9987.5, 15780.8, 22.75, 36.189, 1.640, 6.4763, 7.736478e-02
"1-800-10", "yes", 3, 2, "up", 800, 9987.5, 15780.8, 22.74, 36.256, 1.640, 6.0692, 2.745992e-02
"1-800-11", "yes", 3, 3, "up", 800, 9962.5, 15780.8, 22.74, 36.188, 1.648, 5.8201, 2.789715e-02
"1-800-3", "yes", 1, 3, "up", 800, 9950.0, 15822.9, 22.77, 36.122, 1.577, 7.0504, 1.269874e-01
"1-800-8", "yes", 2, 1, "up", 800, 9987.5, 15822.9, 22.72, 36.186, 1.613, 7.5453, 7.541959e-02
"1-800-13", "yes", 4, 4, "up", 800, 10050.0, 15822.9, 22.74, 36.188, 1.631, 6.9061, 1.098143e-01
"1-800-2", "yes", 1, 2, "up", 800, 9987.5, 15865.0, 22.78, 36.123, 1.604, 6.2513, 3.698211e-02
"1-800-14", "yes", 4, 3, "up", 800, 9987.5, 15865.0, 22.72, 36.117, 1.631, 6.7499, 1.091926e-01
"1-800-16", "yes", 4, 1, "up", 800, 9975.0, 15907.1, 22.72, 35.911, 1.568, 5.7562, 5.590258e-02
"1-800-4", "yes", 1, 4, "up", 800, 10012.5, 15949.2, 22.77, 36.191, 1.488, 7.4151, 1.739729e-01
"1-1050-3", "yes", 1, 3, "up", 1050, 13125.0, 19021.2, 22.72, 36.049, 1.586, 7.8476, 1.342194e-01
"1-1050-11", "yes", 3, 3, "up", 1050, 13112.5, 19063.3, 22.72, 35.980, 1.648, 7.3157, 3.019875e-02
"1-1050-4", "yes", 1, 4, "up", 1050, 13100.0, 19105.3, 22.72, 36.014, 1.497, 8.3853, 1.863622e-01
...
"0-3050-15", "yes", 4, 2, "down", 3050, 37987.5, 43555.1, 22.74, 36.016, 1.408, 14.6590, 1.761292e-02
"0-3050-2", "yes", 1, 2, "down", 3050, 38100.0, 43555.1, 22.74, 35.948, 1.461, 15.0590, 3.176331e-02
"0-3050-5", "yes", 2, 4, "down", 3050, 38025.0, 43555.1, 22.74, 35.982, 1.568, 15.8642, 1.525285e-01
"0-3050-11", "yes", 3, 3, "down", 3050, 37987.5, 43597.2, 22.74, 36.050, 1.470, 15.8581, 1.858974e-02
"0-3050-6", "no", 2, 3, "down", 3050, 37987.5, 43639.3, 22.72, 36.049, 0.000, 0.0000, 0.000000e+00
"0-3050-10", "yes", 3, 2, "down", 3050, 38062.5, 43639.3, 22.73, 36.050, 1.488, 16.4905, 1.236134e-02
"0-3050-3", "yes", 1, 3, "down", 3050, 38262.5, 43765.6, 22.74, 35.982, 1.479, 15.0659, 9.314301e-02
"0-3050-16", "yes", 4, 1, "down", 3050, 37937.5, 43891.8, 22.75, 36.017, 1.461, 14.8766, 1.761930e-02
"0-3050-13", "yes", 4, 4, "down", 3050, 38212.5, 43891.8, 22.73, 36.050, 1.506, 15.5870, 3.686081e-02
"0-3050-1", "yes", 1, 1, "down", 3050, 37975.0, 43891.8, 22.74, 36.016, 1.506, 17.3133, 1.391252e-01
"0-3050-8", "yes", 2, 1, "down", 3050, 38187.5, 43933.9, 22.75, 36.017, 1.488, 16.6296, 2.206337e-02
```

## Appendix C

# Pressure Accuracy

### C.1 Expected

There are two possibilities to measure the pressure in the chamber of the wafer holder. The Burster pressure sensor (s. 1.3.6) is directly attached to the chamber. This sensor is not disturbed by pressure differences due to flow and turbulences. Only pressure changes in the chamber dynamically affect the sensor. The other way to measure the pressure is to use the pressure sensor integrated in the Bronkhorst flow controller (s. 1.3.5).

It's expected that the Burster sensor shows a lower pressure than the Bronkhorst controller, as there's a pressure drop over the tube (radius  $3mm$ , length  $1m$ ) connecting the controller and the pressure chamber.

### C.2 Measurement Results

The Burster sensor's values are always about  $45mbar$  above the pressure at the Bronkhorst sensor. This is certainly wrong. A set of measured values is shown in figure (fig C.1) and table C.1. The measurement are taken at intervals of 200 for the low range series and 500 Bronkhorst internal units for the large range series. All values are converted to  $mbar$ .

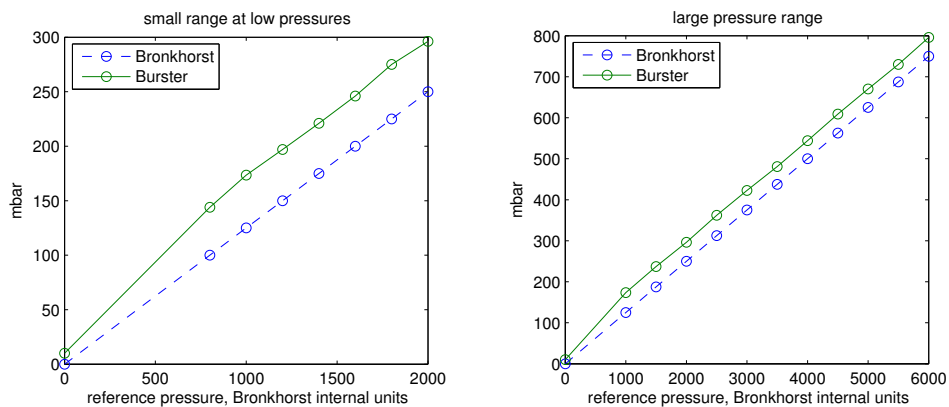


Figure C.1: Difference of pressures from Burster and Bronkhorst

At zero relative pressure it's obvious that the Burster pressure sensor is wrong. It indicates  $\approx 10mbar$  too much. An offset of the A/D converter of LabJack would show the same error for all pressures.

The other measurements can't be verified as there's currently no suitable, calibrated pressure sensor available.

A future investigation of the problem requires to measure each intermediate result. A table has to be created containing one row per pressure measured. Each row contains in its columns the current from the sensor, the voltage across the  $500\Omega$  resistor, the digitised voltage obtained from the LabJack DLL and the pressure in  $Pa$  indicated by `manual-control.vi`. These intermediate results can be converted to  $Pa$ . A comparison of these pressures could reveal the conversion step introducing the error.

Bronkh. [mbar]	Burster [mbar]
0	10
100	144
125	174
150	197
175	221
200	246
225	275
250	296

Bronkh. [mbar]	Burster [mbar]
0	10
125	174
188	237
250	296
313	362
375	423
438	481
500	544
563	609
625	670
688	730
750	796

Table C.1: Pressure test series

## Appendix D

### Datasheet Excerpts

nominal pressure	4 32000	bar internal	
maximal reference	32767	internal	
minimal reference	640 2	internal %	configurable lower := 0

Table D.1: Bronkhorst Flow Controller EL-PRESS P-602C-AAA-33-V Data

nominal pressure	100 6.895	<i>PSI</i> <i>bar</i>	maximal measurable pressure
offset current	4	<i>mA</i>	output current at 0 <i>bar</i>
max current	20	<i>mA</i>	output current at 100 <i>psi</i>
accuracy	0.05	%	$\approx 345 Pa$
conversion resistor	500	$\Omega$	
accuracy	0.01	%	
A/D range	$\pm 10$	<i>V</i>	
A/D bits	12 + 1		12 plus 1 sign bit
discretisation error	1.22 105	<i>mV</i> <i>Pa</i>	$\frac{1}{2}$ LSB

Table D.2: Burster Pressure Sensor 8263-5100 Data

humidity accuracy	$\pm 3.0$	%RH	
temperature accuracy	$\pm 0.4$	$^{\circ}C$	at 25 $^{\circ}C$

Table D.3: Sensirion SHT71 Data

## Appendix E

# Script auto-membranes.scr

This is the source code of `auto-membranes.scr`, the automation script written for MetroPro.

Tabulator characters (ASCII value 8, tab) got replaced by ". " for better readability. "<br>" sequences indicate linebreaks not existing in the original source code. Restoring the original lines is possible by removing all ". " strings at the beginning of a line following a "<br>" and then replacing the "<br>" by that line.

The tab characters are of great importance in MetroScript! Code lines require at least 2 tabs ahead of them, label lines at least 1 tab. The "#" and "!" signs are both comment sequences. "#" is used at the very beginning of the line, "!" is to comment out code lines with 2 tab characters at the beginning of them.

```
# 2004 by S. Fuchs, http://srf.ch/
# This script belongs to a collection of LabView
# files which allow to automate the measurement of
# membranes arranged in a rectangular pattern.
# The measurement setup (pattern alignment, focus, ...)
# needs to be done manually before starting the
# measurement process.
#
# The communication to the computer running LabView
# is done via RS232 (9600 baud, 8N1). The following
# commands control the operation of this script
# (one line per command / parameter):
#
# "PAT_SAVE": saves the current pattern, next line must
#   be the filename (without path and suffix)
# "PAT_FIRST": goto first location and measure it,
#   answer with "LOC_DONE", "PAT_DONE" or "LOC_SKIP"
#   whereas LOC_SKIP gets followed by 2 further lines:
#     - a reason (english sentence on a separate line,
#       no numerical code)
#     - LOC_DONE or PAT_DONE
# "LOC_NEXT": move stage to next location in pattern
#   and measure it, same answers possible as with "PAT_FIRST".
# Every PAT_FIRST and PAT_NEXT command gets followed by a
#   filename without suffix on a single line.
# "MEAS_DONE": terminate the script and return control
#   to the user

. . dim basedir$ [100]
. . dim misc$ [100]
. . dim to_send$ [100]
. . dim pot_err_msg$ [100]
. . dim cmd$ [100]

#=== config, change as needed ===

# just dump all data to one directory. There's a C/C++ program
# which extracts the necessary information and ignores invalid
# measurements (those with an "ERROR:" line after them in the
# logfile generated by LabVIEW)
```

```

. . data_dir$ = "output"
. . pattern_dir$ = data_dir$
. . report_dir$ = data_dir$

#=== config end

. . basedir_id = getid("Controls/Custom/Text 1")
. . basedir$ = getstr$(basedir_id) & "\\\"

. . setdir("Pattern", basedir$ & pattern_dir$)
. . setdir("Data", basedir$ & data_dir$)
. . setdir("Report", basedir$ & report_dir$)

. . metric

. . ! switch off automatic autofocus *lol* before measurement,
. . ! we command it separately to take care of errors
. . focus_id = getid("Controls/Focus/Focus")
. . setnum(focus_id, 0, "")

. . ! a control to display the current measurement id
. . ! this control can be logged in the report file to
. . ! get a proper association.
. . meas_id_id = getid("Controls/Miscellaneous/Comment")

. . status_id = getid("Controls/Custom/Text 2")

. . tr_auto_app_id = getid("Surface Wavefront Map (T+R) / <br>
. . . Controls / Auto Aperture Ref / Auto Aperture")
. . tr_remove_mode_id = getid("Surface Wavefront Map (T+R) <br>
. . . / Controls / Remove Mode")
. . tr_remove_id = getid("Surface Wavefront Map (T+R) <br>
. . . / Controls / Remove")
. . tr_filter_id = getid("Surface Wavefront Map (T+R) <br>
. . . / Controls / Filter / Filter")
. . tr_smwin_id = getid("Surface Wavefront Map (T+R)")

. . setstr(tr_auto_app_id, "Off")
. . setstr(tr_remove_mode_id, "On")
. . setstr(tr_remove_id, "Plane")
. . setstr(tr_filter_id, "Off")

. . ! use 9600/8N1 on RS232 port 3
. . assign @serdev to "com3" "9600,8,0,1"

. GET_COMMAND:

. . setstr(status_id, "Waiting for a command from LabVIEW\n\n")

. . on error goto on_RxError
. . cmd$ = ""
. . enter @serdev; cmd$
. . off error

. . setstr(status_id, "Handling cmd: " & cmd$)

. . if cmd$ = ("PAT_FIRST") then gosub HANDLE_PAT_FIRST
. . if cmd$ = ("LOC_NEXT") then gosub HANDLE_LOC_NEXT
. . if cmd$ = ("MEAS_ABORT") then gosub HANDLE_MEAS_ABORT

```

```

. . if cmd$ = ("MEAS_DONE") then gosub HANDLE_MEAS_DONE
. . if cmd$ = ("PAT_SAVE") then gosub HANDLE_PAT_SAVE

. . goto GET_COMMAND

. on_RxError:
. . if pos(cmd$, chr$(27)) <> 0 then goto on_RemoteAbort
. . goto GET_COMMAND

. on_RemoteAbort:
. . dummy = dialog("\n Measurement process remotely aborted.\n\n",3)
. . goto risk_life

#=== end of main loop =====

. GET_MISC:
. . on error goto GET_MISC_WAIT
. . misc$ = ""
. . enter @serdev; misc$
. . off error
. . !message("\n GET_MISC got '" & misc$ & "'\n\n", 1, 1)
. . return
. GET_MISC_WAIT:
. . ! message("\n Waiting for commands from LabVIEW\n\n", 1, 1)
. . goto GET_MISC

. HANDLE_PAT_FIRST:
. . pat_loc = 1
. . getpatval("NumPosIncl", pat_loc_meas, "")
. . dim pat_array_id (pat_loc_meas)
. . getpatval("PosArrayId", pat_array_id, "")
. . gosub DO_MEASUREMENT
. . return

. HANDLE_LOC_NEXT:
. . pat_loc = pat_loc + 1
. . gosub DO_MEASUREMENT
. . return

. DO_MEASUREMENT:
. . ! disable joystick and move stage to desired pattern position
. . joyoff
. . setstr(status_id, "Moving stage to new location...")
. . gotopatpos(pat_array_id(pat_loc))

. . ! receive part of filename
. . ! (yes, assume we're using a secure connection)
. . setstr(status_id, "Receiving filename...")
. . ofmtr("%f")
. . ofmti("%d")
. . gosub GET_MISC ! receive filename without postfix
. . dim filename_nosuff$ [300]
. . filename_nosuff$ = misc$ & "-" & val$(int(pat_loc)) <br>
. . . & "-" & val$(int(pat_array_id(pat_loc)))
. . setstr(meas_id_id, filename_nosuff$)

```

```

. . ! let the autofocus do it's work
. . on error goto on_position_failure
. . setstr(status_id, "Running autofocus procedure...")
. . pot_err_msg$ = "autofocus failed, position skipped."
. . autofocus
. . off error

. . ! do the measurement
. . setstr(status_id, "Measuring membrane...")
. . on error goto on_position_failure
. . pot_err_msg$ = "measurement failed (MetroScript <br>
. . . command 'measure'), position skipped."
. . measure
. . off error

. . ! save the obtained data
. . savedata(0, basedir$ & data_dir$ & "\\\" & <br>
. . . filename_nosuff$ & ".r.dat")

. . ! analyze data and save data from
. . ! "Surface/Wavefront Map (Test+Ref)"
. . ! where a plane got removed --> horizontal membrane
. . analyze
. . savedata(tr_smwin_id, basedir$ & data_dir$ <br>
. . . & "\\\" & filename_nosuff$ & ".a.dat")

. . ! now either get logreports working in a nice fashion
. . ! (filename needs to fit into our schema).
. . ! or get each value separately to write them into the right
. . ! file.
. . ! savedata(winid, fname.txt) does not work for report windows
. . on error goto oops_dont_log
. . !! logfile is basedir$ & report_dir$ & "\\\" <br>
. . . & filename_nosuff$ & ".log"
. . ! 20041201: "logfile is" doesn't seem to work with
. . ! metropro 7.15.1 as expected.
. . logreports
. . off error
. . oops_dont_log:

. do_measurement_finish:
. . ! enable joystick control and return to caller
. . joyon

. . if pat_loc < pat_loc_meas then
. . . to_send$ = "LOC_DONE"
. . . gosub SEND_FUNC
. . else
. . . to_send$ = "PAT_DONE"
. . . gosub SEND_FUNC
. . endif

. . disable abort @serdev
. . return

. on_position_failure:
. . off error
. . to_send$ = "LOC_SKIP"
. . gosub SEND_FUNC
. . to_send$ = pot_err_msg$

```

```

. . gosub SEND_FUNC
. . goto do_measurement_finish

. HANDLE_PAT_SAVE:
. . setstr(status_id, "Saving current pattern...")
. . gosub GET_MISC
. . savepattern(basedir$ & pattern_dir$ & "\\\" & misc$)
. . return

. SEND_FUNC:
. . setstr(status_id, "sending " & to_send$)
. . output @serdev; to_send$
. . return

. HANDLE_METROPRO_ERROR:
. . to_send$ = "MEAS_ABORT"
. . gosub SEND_FUNC
. . goto risk_life
. HANDLE_MEAS_ABORT:
. . setstr(status_id, "Measurement remotely aborted.")
. . goto risk_life
. HANDLE_MEAS_DONE:
. . setstr(status_id, "Measurement successfully finished.")
. risk_life:
. . off error
. . setlight(0)
. . assign @serdev to ""
. . end

```

## Glossary

<i>psi</i>	Pounds per Square Inch, $1 \text{ psi} \approx 68.9475729 \text{ mbar}$
A/D	Analog to Digital conversion / converter
ABNF	Augmented Backus Naur Form
ASCII	American Standard Code for Information Interchange
BNF	Backus Naur Form
class	object oriented programming language structure [7]
CSV	Comma Separated Values, a file format, encoded in ASCII
DDE	Dynamic Data Exchange, an interprocess communication system
DLL	Dynamic Link Library, MS Windows specific
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
GUI	Graphical User Interface
IIR	Infinite Impulse Response
object	an instantiation of a class; at runtime in memory
OS	Operating System
PID	Proportional, Integral, Differential → industrial controller design
RMS	Root Mean Square
RS232	commonly used serial communication protocol lacking a standardisation

## Index

*angularity*, 29

ABNF, 42

auto-membranes.scr, 50

block-transient.vi, 15

Bronkhorst Flow Controller, 5

Burster Pressure Sensor, 6

clip-value.vi, 15

coordinates heightmap, 4

coordinates image, 4

coordinates matrix, 2

coordinates pattern, 2

debug-logline.vi, 15

do-measurement.vi, 13

error-condenser.vi, 15

file formats, 42

flowctrl-p-read.vi, 14

flowctrl-p-set.vi, 14

flowctrl-switch-off.vi, 14

FlowDDE, 5

flowdde-check-run.vi, 14

flowdde-close-session.vi, 14

formats, 42

fringes, 3

generate-abort-error.vi, 15

heightmap, 3

ID formats, 42

LabJack, 5

LabVIEW, 2

lf2sp.vi, 15

manual-control.vi, 13

measurement ID, 42

MetroPro, 2

MetroScript, 3

MPMP, 22

needle valve, 4

NewView 5000, 3

pattern, 2

pressure chamber, 4

pressure-monitor.vi, 15

psens-p-read-04-AISample.vi, 14

read-temp-rh.vi, 14

Resolution of NewView 5000, 3

rs232-metropro-setup.vi, 14

rs232-read-line.vi, 14

rs232-write-line.vi, 14

Sensirion SHT71, 6

vars-global.vi, 15

VI, 2

Virtual Instruments, 2

## Bibliography

- [1] Zygo Homepage  
<http://www.zygo.com/>
- [2] Zygo, *MetroScript Programming Language, Version 8.00, OMP-0399M*  
available at <http://www.zygo.com/>, ask Zygo for a login.
- [3] LabJack U12, product homepage  
<http://www.labjack.com/labjack%5Fu12.html>
- [4] LabJack Corporation, *LabJack U12 User's Guide*  
<http://www.labjack.com/files/LabJack%5FU12%5FUsers%5FGuide.PDF>
- [5] G. F. Franklin, *Feedback Control of Dynamic Systems*  
Prentice Hall, fourth ed., 2002  
ISBN 0-13-098041-2
- [6] Zygo, *MetroPro Reference Guide, OMP-0347H*  
available at <http://www.zygo.com/>, ask Zygo for a login.
- [7] B. Stroustrup, *The C++-Programming Language*  
Addison-Wesley, third ed., 1998  
ISBN 0-201-88954-4
- [8] FSF, *GNU make Manual*  
<http://www.gnu.org/software/make/manual/html%5Fchapter/make%5Ftoc.html>
- [9] RFC-2234: Augmented BNF for Syntax Specifications: ABNF  
<ftp://ftp.rfc-editor.org/in-notes/rfc2234.txt>