Digital signal processors for the measurement of turbulence

Donker, Marcel

Award date:
1997

Eindhoven University of Technology,
Faculty of Applied Physics,
Department of Fluid Dynamics,
Group Turbulence, Chaos and collodial systems.

Digital Signal Processors for the measurement of turbulence

R-1418-A

Marcel Donker

Februari 13, 1997

Thesis of a graduation project carried out in the group Turbulence, Chaos and collodial systems of the Eindhoven University of Technology. This graduation project is done in cooperation with the group Measurement and Computer Science.

Supervisor: dr. ir. W. v/d Water
Professor: Prof. dr. ir. K. Kopinga

Abstract

Turbulence is a common phenomenon that we encounter everyday. There is no closed theory of turbulence and experiments are a prime source of information.

For fundamental turbulence research the prime goal is understanding of its small-scale structure. Two important tools to quantify the small-scale structure are the *energy spectrum* and the *structure function*, the second being equivalent to the correlation function of fluctuating velocities in two different spatial points.

In this graduation project, the measurements of these turbulence characteristics in a windtunnel have succesfully been implemented using a Texas Instruments TMS320C40 Digital Signal Processor. Because of its fast real-time *Digital Signal Processing* (DSP) capability, the DSP processor is an excellent tool to implement these turbulence measurements in real-time. We call the implementation of the real-time measurement of structure functions the *'Structurator'*.

The turbulent wind velocities are measured using *Constant Temperature Anemometry* (CTA) in fully developed turbulent air-flow generated with a grid in the windtunnel. Energy spectra are measured and calculated using a Fast Fourier Transform. The velocity differences needed for the structure functions are calculated from velocities measured in the *longitudinal* direction using the *Structurator* implemented in real-time on the Digital Signal Processor. In this report it is indicated how to extend the real-time techniques towards multiple probe measurements.

Special attention has been paid to *digital signal analysis.* The measurement of turbulence quantities involves the estimation of statistical properties of a time-dependent signal. Because the signal becomes discrete (in time and signal value) after sampling, the question is how the *discretization* will effect the estimate. The used hot-wire velocity probe is sensitive to two perpendicular velocity components. We will discuss the affect of the associated *rectification* principle on measured turbulence characteristics.

Some recommendations are given regarding future research. A proposal is given for a *multi-channel Structurator* using multiple probes, for the real-time measurement of *transverse* structure functions. Also some recommendations regarding the use of a different Analog-to-Digital Converter (ADC) are given.

Dankwoord

Hierbij wil ik iedereen van de vakgroepen FTI en Transportfysica bedanken die mij tijdens mijn afstudeerperiode hebben geholpen. In het bijzonder gaat mijn dank uit naar:

- dr. ir. Willem v/d Water, voor de dagelijkse begeleiding tijdens mijn afstudeerperiode. Tevens veel dank voor de nodige opmerkingen op theoretisch gebied.
- Prof. dr. ir. Klaas Kopinga, voor de algehele supervisie en voor zijn inbreng op het gebied van de TMS320C40 en Real-time data-acquisitie, in het bijzonder op het gebied van de Structurator.
- dr. Doug Binks, voor zijn hulp op het gebied van C en LaTeX en zijn hulp bij het oplossen van praktische problemen.
- Gerard Trines, voor de hulp op het gebied van de hot-wire anemometer en het PhyDAS systeem.

Technology assesment

The relevance of fundamental turbulence research can be found in various technological fields. Turbulence can be found in many industries such as the aircraft industry, the aerospace industry or the process industry. The basic understanding of turbulence can be benificial for all these technological fields of interest.

Digital Signal Processors (DSP) have become one of the most important and promising new digital electronic devices of the past ten years. It is assumed that the growth of the DSP market will continue into the next century and will rise above that of other electronic devices. Digital Signal Procesors are used intensive in a whole range of industries for a lot of purposes.

Contents

Chapter 1

Introduction

This thesis is a description of my graduation project done at the turbulence research group at the physics department of the Eindhoven University of Technology. The project involved is a combination of real-time data-acquisition and turbulence research, however the prime field of interest is the real-time data-acquisition and the digital signal processing and analysis.

Despite its familiar appearance and despite many years of research, there is still no good description of turbulence. An intuitive look at turbulence might come up with the idea that the essential characteristic of turbulence is that the local velocity or pressure fluctuations are *random*. Random means that we cannot precisely predict when and what the next move of the particles will be. Figure 1.1 shows the typical velocity fluctuations of a turbulent flow as is measured in the windtunnel.

It is very difficult to give a precise definition of turbulence. We can make a list of some characteristics of turbulent flows, [26]. A turbulent flow is or exhibits: *Irregularity or randomness*; the turbulent flow is unpredictable in nature, *Diffusivity*; wich causes rapid mixing and increases rates of momentum, heat and mass transfer, *Large Reynolds number*; turbulent flows always occur at high Reynolds numbers, *3-Dimensional vorticity fluctuations*; turbulence is rotational and 3-dimensional, *Dissipation*; turbulent flows are always dissipative, *Continuum*; turbulence is a continuous phenomenon, *Flows*; turbulence is a feature of fluid *flows* and not of fluids.

Usually, turbulent flows are characterized with the dimensionless Reynolds number, which is a characteristic parameter of the flow. At low Reynolds numbers, the flow is called *laminar*, i.e. the variations in the flow are predictable in both space and time. At higher Reynolds numbers the flow becomes unstable, and at large enough Reynolds numbers it becomes fully turbulent. The Reynolds number is characterized as

$$\mathrm{Re} = \frac{\bar{U}L}{\nu}, \tag{1.1}$$

with $\bar{U}$ the mean velocity of the flow, L a characteristic dimension of the flow (e.g. the diameter of the pipe through which the fluid flows) and ν the kinematic viscosity.

For incompressible flow, the fundamental dynamical equations are the Navier-Stokes equations (1.2). This is a system of coupled partial differential equations and must be supplemented with initial and boundary conditions. The problem of the description of turbulence is the simultaneous presence of large- and small-scale motions and the impossibility of a closed statistical theory.

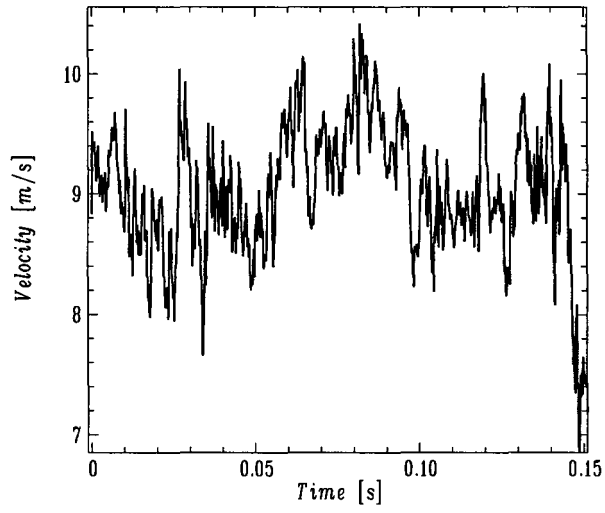


Figure 1.1: *Velocity fluctuations as a function of time, measured in turbulence generated with a grid.*

$$\begin{aligned} &\frac{d\vec{u}}{dt} + \vec{u} \cdot \nabla \vec{u} = -\nabla \frac{p}{\rho} + \nu \nabla^2 \vec{u}, \\ &\nabla \cdot \vec{u} = 0. \end{aligned} \tag{1.2}$$

In this thesis we will consider the case of homogenous and isotropic turbulence, which is the case when the statistical properties of the flow field do not change under spatial translations and rotations. We wil concentrate on the statistical description of turbulence in the limit of infinite Reynolds numbers. In this case turbulence has scale-invariance which allows to short cut the closure problem. The closure problem is the essence of the failure of a closed turbulence theory. It means that one has a set of n - 1 equations with at least n unknown variables in it.

For a statistical description of turbulence we can use different kinds of tools. Valuable information can be taken from the measured turbulence *energy spectrum.* Other tools are the so-called *structure functions* which are correlations of two-point velocity differences. The experimental measurement of these statistical characteristics of turbulent flows is the major field of interest in this thesis. We look into the use of Digital Signal Processors for the real-time measurement of these turbulence statistics.

Digital Signal Processors are processors which allow for the fast data-acquisition needed for real-time turbulence measurements. Digital Signal Processing (DSP) applications such as the Fast Fourier Transform (FFT) can be implemented very fast allowing for an almost real-time measurement of turbulence energy spectra. DSP processor features such as the implementation of circular buffers and parallel instructions make it possible to measure the structure functions in real-time. We successfully implemented these turbulence measurements on the Texas Instruments TMS320C40 DSP processor, and further turbulence research can now be carried out with this real-time data-acquisition system.

1.1 Thesis outline

This thesis starts with a chapter *Turbulence models* covering an important statistical model based in the turbulence cascade assumption. Although there are many models available, we

restrict ourself to the most basic model, the Kolmogorov K41-model. The dynamic theory of the local structure of turbulence is described which results in some important relationships between the second and third order structure functions based on the Navier-Stokes equations. Finally the Batchelor parametrisation is described and the bottleneck phenomenon, which results in an energy pileup in the energy spectrum, is described. The bottleneck phenomenon is hidden in this emperical Batchelor parametrisation and can also be measured with our DSP processor data-acquisition system.

Chapter 3, *Real-time signal processing*, is devoted to the Digital Signal Processor system and the real-time requirements needed for this system for the turbulence experiments. The implementation of the real-time measurement of structure functions is called the *Structurator* and is described in detail.

In the next chapter, *Experimental setup and instrumentation*, the instrumentation is described among which the used hot-wire anemometers for the measurement of the local velocities in the windtunnel. It describes the operating principle, the calibration of the hot-wire probes and the signal conditioning of the anemometer signal. Finally the windtunnel used for our experiments is described.

After the description of the theory, the experiments and the real-time data-acquisition, chapter 5 is devoted to *Signal Analysis*. The measurement of turbulence characteristic quantities involves the estimation of statistical properties of a time-dependent signal. Because the signal becomes discrete (in time and signal value) after sampling, the question is how the discretization will effect the estimate. The principle of the Fast Fourier Transform (FFT), which is used to transform the discrete velocity time-signal to the frequency domain, is described. Yet another effect related to the discretization of the signal, is due to the usage of discrete calibration tables, which together with the Analog-to-Digital converter (ADC) results in a differential non-linearity error. Another error is due to the hot-wire velocity probe which is sensitive to two perpendicular velocity components. We will discuss the effects of this rectification principle on measured turbulence characteristics.

Chapter 6, *Experimental results*, presents some experimental results of measurements done in the windtunnel with the DSP processor data-acquisition system. We present turbulence spectra, velocity probability distribution functions (PDFs), velocity difference PDFs and some structure functions calculated from these velocity difference PDFs.

In chapter 7, *Recommendation for future Research*, some recommendations are made regarding future research. A possible setup is given for a multi-channel Structurator for the measurement of transverse structure functions. Also some recommendations regarding the use of a new Analog-to-Digital converter are given, especially regarding the differential non-linearity of the ADC. The rectification error should also be investigated further.

The thesis ends with some global *Conclusions* in chapter 8. In *appendix A* a detailed description is given of Digital Signal Processors on the whole, the used DSP processor from Texas Instruments (TMS320C40) and the used DSP development PC-board from Transtech Parallel systems. *Appendix B* gives the C-programs and the Structurator assembler routine developed for this project.

Chapter 2

Turbulence models

There are mainly two groups of ideas which have opposing points of view of how to approach turbulence: The first group, the *statistical*, tries to model the flow in averaged quantities. This group follows Kolmogorov and believes in the phenomenology of *cascades* and the possibility of any coherence or order in turbulence. The second group believes in the *coherence among chaos* and considers turbulence from a purely deterministic point of view by studying either the dynamics of isolated "coherent" structures or the stability of flows (that is no turbulence) in various situations. In this thesis the first group will be of interest.

2.1 Cascade models, the legacy of Kolmogorov

In case of cascade modelling, Richardson was the first who put forward some ideas on the theory of fully developed turbulence, [22]. He assumed a hierarchy of turbulent distances on different scales. 'Eddies' of a certain scale would be the result of the instability of larger 'eddies' at a larger scale. Richardson assumed a cascade process of eddies breaking down and in this cascade there is a transmission of energy of the flow motion of smaller and smaller eddies down to the smallest scale, where the breaking-down process is stopped by dissipation. Kolmogorov further developed and formulated the idea of Richardson in 1941. He assumed an *inertial range* in which energy is transported from large eddies to smaller eddies. This range of scales is bounded from above by the size Λ of the eddies at which energy is injected and from below by the size η of the eddies where kinetic energy is dissipated to heat. Kolmogorov assumed a uniform energy distribution over all eddies. Since then many reseachers came op with ideas and models to describe the statistical (cascade) behavior of fully developed turbulence. In these models the energy dissipation per unit of mass per unit of time is an important parameter,

$$\epsilon = \frac{1}{2}\nu \sum_{i,j} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)^2 \tag{2.1}$$

where ν is the kinematic viscosity, u_i and u_j are velocity components and x_i and x_j are components of the position vector. In the inertial range ϵ equals the energy transmission from large eddies, where the energy is injected, to small eddies where the energy is dissipated to heat by viscosity, see Figure 2.1. In the case of homogeneous and isotropic turbulence, on average

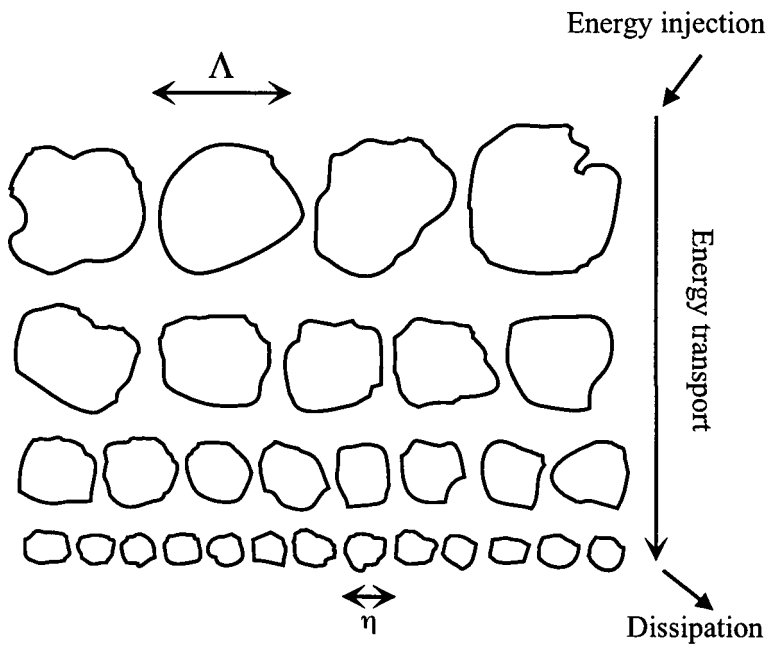


Figure 2.1: *The energy cascade process following Richardson ideas. At eddies of size* Λ *energy is injected, then energy is transmitted to smaller and smaller eddies, until it is dissipated to heat at the smallest eddies of size* η

$$\langle \epsilon \rangle = 15\nu \left\langle \left(\frac{\partial u}{\partial x} \right)^2 \right\rangle. \tag{2.2}$$

The size, Λ, of the largest possible eddy will be bounded from above by the size of the system. The size, η, of the smallest eddy, the cutoff scale, is determined by viscous dissipation. The range of scales $\eta << r << \Lambda$ where no energy is injected and in which no viscous processes occur, is called the *inertial range.*

2.1.1 Kolmogorov 1941

One very important model that was of one of the first developed using the scenario of the energy cascade is from Kolmogorov (1941), usually abreviated with K41. He developed his theory for locally isotropic and homogeneous turbulence, using *structure functions*, [14]. Locally isotropic and homogeneous turbulence means that the probability distribution of the relative velocity differences $\Delta\vec{u}(\vec{r}) = \vec{u}(\vec{x}+\vec{r}) - \vec{u}(\vec{x})$ is independent of $\vec{x}$, independent of time and invariant under translations (which is the same as independence of $\vec{x}$) and rotations, [11]. Structure functions are correlations of two-point velocity differences. A structure function of order p is defined as:

$$G_p(\vec{r}) = \langle [\Delta\vec{u}(\vec{r})]^p \rangle = \langle [\vec{u}(\vec{x}+\vec{r}) - \vec{u}(\vec{x})]^p \rangle, \tag{2.3}$$

Where the angular brackets denote the ensemble average.

Implicitly assuming that the energy dissipation is uniformly distributed, Kolmogorov formulated his famous similarity hypotheses:

First similarity hypothesis:
For locally isotropic turbulence the relative velocity distributions are uniquely determined by the kinematic viscosity ν and the average energy dissipation, $\langle\epsilon\rangle$, per unit time and per unit of mass, provided that all distances $r = |\vec{r}|$ are small compared with macroscales, $r << \Lambda$.

Second similarity hypothesis:
When, in addition the distances r are large compared with the dissipation range, so when r is in the inertial range $\eta << r << \Lambda$, then the relative velocity distributions are uniquely determined by the quantity $\langle\epsilon\rangle$ and do not depend on ν.

A functional discription for the moments of velocity differences and thus for the structure functions can be derived using dimensional analysis and the two similarity hypotheses and this will lead to, [11]

$$G_p(r) = C_p[\langle\epsilon\rangle r]^{\frac{p}{3}}, \tag{2.4}$$

in which the C_p are universal constants. For the third-order structure function , $C_3(= -\frac{4}{5})$ follows directly from the Navier-Stokes equations, see section 2.3. This is called Kolmogorov's four-fifths law,

$$G_3(r) = -\frac{4}{5}\langle\epsilon\rangle r. \tag{2.5}$$

It follows from Eq. (2.4) that the structure functions have a scaling behavior given as,

$$G_p \sim r^{\zeta_p} \tag{2.6}$$

where ζ_p is called the *scaling exponent.* For the Kolmogorov theory (K41) $\zeta_p = \frac{p}{3}$.

In tune with the mentioned hypothesis, there is a smallest length scale in turbulence that can be derived from ν and $\langle\epsilon\rangle$ using dimension counting only, taking

$$\eta = \left(\frac{\nu^3}{\langle\epsilon\rangle}\right)^{\frac{1}{4}} \tag{2.7}$$

together with a (Kolmogorov) velocity scale,

$$v_K = (\nu\langle\epsilon\rangle)^{\frac{1}{4}} \tag{2.8}$$

By definition, the Reynolds number based on η and v_K is unity,

$$Re_\eta = \frac{v_K\eta}{\nu} = 1. \tag{2.9}$$

It implies that fluid motion on scale η is dominated by viscosity.

As for the structure functions, a discription for the energy of the turbulent fluctuations per unit of mass of fluid in scales r can be derived from the hypotheses by dimensional analysis. By a dimensional argument, the first similarity hypothesis implies the following universal form for the energy spectrum at large wavenumbers, the famous *minus five-third law,*

$$E(k) \sim \epsilon^{\frac{2}{3}}k^{-\frac{5}{3}}. \tag{2.10}$$

The proportionality constant b in $E(k) = b\epsilon^{\frac{2}{3}}k^{-\frac{5}{3}}$ is usually referred to as the Kolmogorov constant and can experimentally be derived from measured turbulence energy spectra. In a log-log plot of a measured energy spectrum versus the wavenumber (or frequency) a linear behavior can be observed in the inertial range with a slope $-\frac{5}{3}$. In chapter 6 a measured energy spectrum is given having this kind of scaling behavior in the scaling range.

2.2 Other turbulence models

Shortly after Kolmogorov published his two similarity hypotheses, Landau pointed out [17], that the K41 theory could not be right, because he did not take into account *intermittency*. By intermittency he meant that turbulence is not uniformly distributed in space; there are regions with less intense and regions with more intense turbulence. In the *log-normal model* Kolmogorov [15] and Oboukhov [21] tried to reconcile Kolmogorov's original theory with intermittency. They suggested that the logarithm of the local spatially averaged dissipation rate ϵ_r has a normal distribution.

Deviations of the K41 model and the log-normal model from measurements are associated to the description of intermittency in these models, therefore intermittency must be a crucial ingredient of a scaling description of fully developed turbulence. After the ideas of Kolmogorov and others, different models have been developed. It is beyond the scope of this thesis to describe them all in detail, but one model will shortly be mentioned here.

Fractal models are an attempt to capture intermittency in a geometric framework. The key idea is that self-similar cascades do not need to be space filling. Such a process is characterized by a fractal dimension, even by a continuous dimension function. Measurements have shown that the scaling exponents of the structure function depend on the order of the structure function in a nonlinear way. The *multifractal model* seems to give a good discription of the turbulence cascade.

2.3 Exact results for third order structure functions

Whereas dimensional arguments would seem to suffice for predicting the form of structure functions, there is an exact result that explicitly relies on the Navier-Stokes equations. The result is that for the third-order structure function, [20],

$$G_3^L(r) = -\frac{4}{5}\langle\epsilon\rangle r + 6\nu\frac{dG_2^L(r)}{dr}. \tag{2.11}$$

It relates the second- and third-order longitudinal structure functions of locally isotropic turbulence, corresponding to isotropic turbulence with sufficiently large Reynolds number. For anisotropic turbulence we would still expect the general form of Eq. (2.11), but with different prefactors. For example, the factor $-\frac{4}{5}$ would be different in the case of anisotropic turbulence. For $r \gg \eta$ (in the inertial range), when the viscosity is negligible we can write,

$$G_3^L(r) = -\frac{4}{5}\langle\epsilon\rangle r. \tag{2.12}$$

This equation was first found by Kolmogorov and is often referred to as Kolmogorov's *Four-Fifths law*. A popular normalisation of the third-order structure function is the one where we normalize the structure function by the Kolmogorov velocity v_K and devide the distance

scale by the Kolmogorov length η. Then the third-order structure function $G_3^L(r/\eta)/v_K^3$ can be written as (for $r \gg \eta$),

$$\frac{G_3^L(\frac{r}{\eta})}{v_K^3} = -\frac{4}{5}\frac{r}{\eta}, \tag{2.13}$$

where we use that $v_K = (\nu\langle\epsilon\rangle)^{\frac{1}{4}}$ and $\eta = \left(\frac{\nu^3}{\langle\epsilon\rangle}\right)^{\frac{1}{4}}$ so that $v_K^3 = \eta\langle\epsilon\rangle$. If we plot this normalisation of the third-order structure function $G_3^L(r/\eta)/v_K^3$ against r/η on a log-log scale, then we must find according to (2.13) in the inertial range a slope equal to unity and an offset at $\log(4/5)$, for locally isotropic turbulence.

2.4 Batchelor parametrization and the bottleneck phenomenon

The famous $E(k) \sim k^{-\frac{5}{3}}$ form of the energy spectrum is a direct consequence of the form $G_2(r) \sim r^{\frac{2}{3}}$ of the second-order structure function. This can be seen from Fourier-transforming G_2 and using the fact that

$$G_2(r-r') = \langle(v(r)-v(r'))^2\rangle = 2\int_{-\infty}^{\infty}\left(1-e^{if(r-r')}\right)E(f)df, \tag{2.14}$$

where f denotes the frequency.

An interesting problem arises from the finite extent of scaling. In an experiment that is performed at a finite Reynolds number, the inertial range has a finite extent. This affects the quality of scaling of both $G_2(r)$ and $E(k)$. A remarkable observation is that $G_2(r)$ and $E(k)$ are affected in a different way such that, while the scaling of $G_2(r)$ rapidly deteriorates as Re decreases, that of $E(k)$ remains relatively unaffected. Viscosity sets an obvious lower limit on the scaling range.

Near $r \sim \eta$ the second-order structure function can be represented by the Batchelor parametrisation, [18], [19],

$$G_2^{(B)}(r) = \frac{\epsilon r^2/(3\nu)}{\left[1+\left(\frac{1}{3b}\right)^{\frac{3}{2}}\left(\frac{r}{\eta}\right)^2\right]^{1-\frac{\zeta_2}{2}}}, \tag{2.15}$$

where the scaling exponent ζ_2 denotes the asymptotic value of $\zeta_2(r)$ for $r \gg \eta$, ν is the kinematic viscosity, $\eta = (\nu^{3/4})/(\epsilon^{1/4})$ the Kolmogorov length, and b is the Kolmogorov constant. The Kolmogorov constant b can be determined experimentally from the turbulence energy spectrum. Originally Eq. (2.15) was given by Batchelor as a parametrization [1], but recently got theoretical support by Sirovich, Smith and Yakhot [24], who, moreover, find agreement between the Batchelor energy spectrum and numerical simulation spectra for 30 orders of magnitude.

It can be seen from Eq. (2.15) that for large r

$$G_2^B(r) \sim \left(\frac{r}{\eta}\right)^{\zeta_2}, \tag{2.16}$$

whereas for small r

$$G_2^B(r) \sim r^2. \tag{2.17}$$

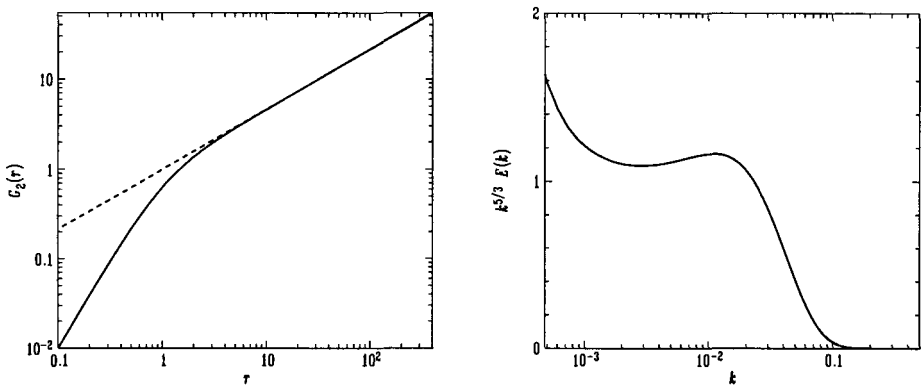


Figure 2.2: *The Batchelor parametrization. (a) Second order structure function according to the Batchelor parametrisation of Eq. (2.15). (b) Batchelor energy spectrum derived from the Fourier transform of the Batchelor parametrization, multiplied with* $k^{5/3}$.

One could say that Eq. (2.15) is an interpolation between these two behaviors. Figure 2.2a shows a plot of the second order structure function according to the Batchelor parametrization of Eq. (2.15). In this plot the simplified function $G_2^{(B)}(r) = r^2/[1+r^2]^{2/3}$ has been plotted to depict only the scaling character of the Batchelor fit.

Next, we want to calculate the Batchelor energy spectrum $E^{(B)}(f)$. The second order velocity structure function for a given energy spectrum can be calculated through the Fourier transformation of Eq. (2.14) and can be expressed as, [20],

$$G_2(r) = 4\int_0^\infty (1-\cos(fr))E(f)df. \tag{2.18}$$

By inverting this equation we can calculate the energy spectrum from a given structure function. First, the second order structure function can be expressed as,

$$\begin{aligned} G_2(r) &= \langle (v(x+r)-v(x))^2\rangle & (2.19)\\ &= 2\langle v^2(x)\rangle - 2\langle v(x)v(x+r)\rangle & (2.20) \end{aligned}$$

We call $\Gamma(r) = \langle v(x)v(x+r)\rangle$ the second order correlation function and according to Eq. (2.20) can be written as,

$$\Gamma(r) = \langle v^2\rangle - \frac{1}{2}G_2(r). \tag{2.21}$$

The second order correlation function is related to the energy spectrum as,

$$E(f) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ifr}\Gamma(r)dr. \tag{2.22}$$

Substituting Eq. (2.21) into Eq. (2.22), gives the following relation,

$$E(f) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ifr}\langle v^2\rangle dr - \frac{1}{4\pi}\int_{-\infty}^{\infty} e^{ifr}G_2(r)dr \tag{2.23}$$

$$= \langle v^2\rangle\delta(f) - \frac{1}{4\pi}\int_{-\infty}^{\infty} e^{ifr}G_2(r)dr \tag{2.24}$$

Here, we use that $\langle v^2(x)\rangle = \frac{1}{2}G_2(\infty) = 0$ according to [19], and we use that $\int_{-\infty}^{\infty} e^{ifr}G_2(r)dr = 2\int_0^{\infty} cos(fr)G_2(r)dr$ and we write

$$E(f) = -\frac{1}{2\pi}\int_0^{\infty} cos(fr)G_2(r)dr \tag{2.25}$$

This relation between the second order structure function and the energy spectrum can be used to derive the Batchelor energy spectrum resulting from the Batchelor second order structure function of Eq. (2.15). Inserting this parametrization into the transformation relation of Eq. (2.25) gives,

$$E^{(B)}(f) = -\frac{1}{2\pi}\int_0^{\infty} cos(fr)\frac{\epsilon r^2/(3\nu)}{\left[1+\left(\frac{1}{3b}\right)^{\frac{3}{2}}\left(\frac{r}{\eta}\right)^2\right]^{1-\frac{\zeta_2}{2}}}dr \tag{2.26}$$

In order to make a graphical representation of this energy spectrum in a log plot, we use again the simplification that $G_2^{(B)}(r) = r^2/[1+r^2]^{2/3}$ and we solve the Batchelor energy spectrum by writing,

$$E^{(B)}(f) = -\frac{1}{2\pi}\int_0^{\infty} \cos(fr)\frac{r^2}{[1+r^2]^{2/3}}dr \tag{2.27}$$

$$= \frac{1}{2\pi}\frac{d^2}{df^2}\int_0^{\infty}\frac{\cos(fr)}{[1+r^2]^{2/3}}dr. \tag{2.28}$$

This will only introduce a singularity at $f = 0$, which we safely discard. Solving this integral numerically we get Figure 2.2b, which gives a log plot of the resulting Batchelor energy spectrum, times $k^{5/3}$. For $k^{-5/3}$ scaling this would give a horizontal line at $k = 1$, but from Figure 2.2b it can be seen that the Batchelor parametrization contains an important physical phenomenon. An energy pileup in the crossover region of $E^B(f)$ becomes noticeable if we compare the Batchelor spectrum with the classical $k^{-5/3}$ Kolmogorov spectrum. We can also see that an energy pileup is visible at the beginning of the spectrum.

Falkovich [6] was the first to introduce the name *bottleneck phenomenon* for this energy pileup. This phenomenon has been discussed recently, in various publications, [4], [6], [9], [10], [18], [19], [23], [24], [25], and its important physical consequences hidden in the transformed Batchelor parametrisation are a major point of discussion.

Lohse and Müller-Goeling, [19], give an analytical solution for the relation of Eq. (2.26) is terms of Bessel functions. For $\zeta_2 = \frac{2}{3}$, the solution these authors found is of the form,

$$E^{(B)}(p) = E_0\epsilon^{2/3}(p')_d^{-5/3}A\left[\frac{2}{3}\tilde{p}^{1/6}K_{11/6}(\tilde{p}) + \tilde{p}^{7/6}K_{5/6}(\tilde{p})\right], \tag{2.29}$$

where A is a dimensionless constant, and $\tilde{p} = pr'_d$ with r'_d the crossover point between the inertial range and the viscous range in the structure function and $K_\nu(x)$ is the modified Bessel

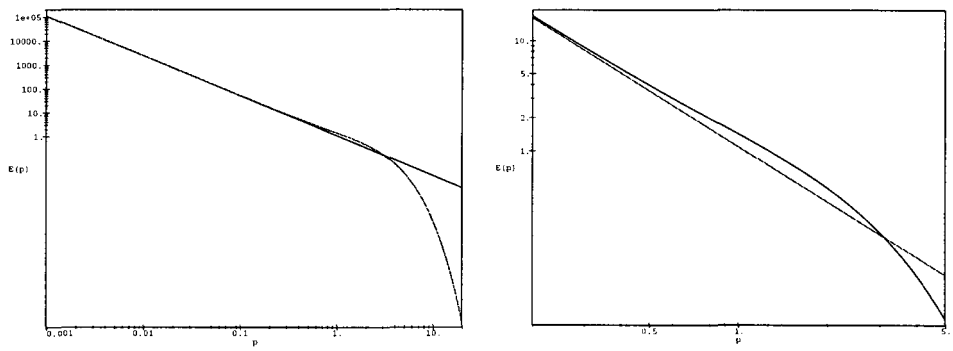


Figure 2.3: *(a) Batchelor energy spectrum according to Eq. (2.29). (b) Enlarged section showing the energy pileup region just before the energy drops off due to viscous dissipation.*

function of the third kind. To view the scaling behavior of this solution for the spectrum we only give a plot without the constants $E_0\epsilon^{2/3}(p')_d^{-5/3}A$. Equation (2.29) is plotted in Figure 2.3.

The enlarged section in Figure 2.3b shows clearly the energy pileup in the crossover region. The question remains to what kind of physical effect the bottleneck phenomenon originates from.

2.5 Connecting turbulence theory and experiments

The statistics of velocity fluctuations on the flow are of main interest in turbulence research. We are interested in the scaling properties of both the energy spectrum and the structure functions. The experimental quantification of the small scale turbulent fluctuations is important for comparing numerical simulations or to value (new) turbulence models, [12]. Therefore, it is necessary to be able to measure energy spectra and structure functions with high statistical accuracy.

Experimentally measured energy spectra are inherent to have a low signal-to-noise ratio, and the improvement of the statistical accuracy of these measurements is very important. The suspicion arises that there may be phenomena visible in the spectrum, with energy levels that sofar have remained undetected.

Structure functions $G_p(r)$ of order p are also a key tool in the statistics of the small scale motion of turbulence. However, for higher orders these are an average over increasingly rare instances and increasingly large velocity differences, so the statistical accuracy is also here a problem and extremely long-time averages are needed. The deviation from the K41 model and other models are visible in this higher order region of the structure functions.

This is the major motivation for developing a system based on a digital signal processor to measure the energy spectra and the structure functions with high statistical accuracy.

Chapter 3

Real-time signal processing

This chapter deals with the initial goal of this work; the implementation of real-time measurement of turbulence characteristics using a digital signal processor (DSP). First the requirements for the measurement of structure functions are described. We look at the structure functions on the whole and at longitudinal structure functions in particular. Also transverse structure functions, although not applicable to this thesis, are mentioned. Then the motivation for using a digital signal processor for this kind of measurements is discussed. And finally the actual 'Structurator' which was developed, is explained in detail.

3.1 Structurator requirements

3.1.1 Measuring Structure functions

The structure function of order p can be calculated as the integral over the probability distribution function (PDF) of velocity differences $\Delta u(r) = u(x+r) - u(x)$ that are measured a distance r apart.

$$G_p(r) = \langle (\Delta u(r))^p \rangle = \int P(\Delta u)(\Delta u)^p d(\Delta u) \tag{3.1}$$

In real-time measurements it suffices to accumulate the PDF's of the velocity differences for a range of distances and to compute the structure functions $G_p(r)$ later on. The structure functions have scaling behavior in the form of $G_p(r) \sim r^{\zeta_p}$ and we will be interested in the scaling exponent ζ_p which can be calculated from

$$\zeta_p = \frac{d \log(G_p(r))}{d \log r}. \tag{3.2}$$

This exponent can be found by plotting $G_p(r)$ in a log-log plot, and measuring the slope of the resulting straight line. Therefore it is advantageous to select the points r exponentially.

The various configurations for measuring longitudinal and transverse structure functions are shown in Figure 3.1.

The correlation function can be expressed in the second-order structure functions, see Eq. (2.21). Whereas for real-time measurement of the correlation function it may still be feasible to actually perform multiplications such as $u(x)u(x+r)$, this is no longer possible for structure functions of arbitrary order p, and the route via storage of the velocity difference distribution

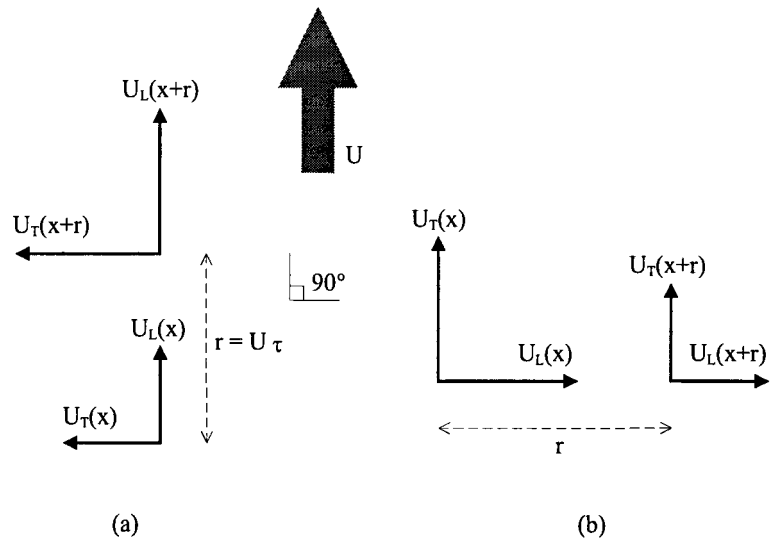


Figure 3.1: *The longitudinal velocity components $u_L(x)$, $u_L(x+r)$ are defined as the velocities on the direction of the distance vector $\vec{r}$, whereas the transverse velocity components $u_T(x)$, $u_T(x+r)$ are defined as the velocities perpendicular to the distance vector $\vec{r}$. (a) The longitudinal velocities are in the direction of the mean flow. (b) The transverse velocities are in the direction of the mean flow.*

functions $P(\Delta u)$ is unavoidable. Also the PDFs are an important source of information in turbulence research, so all is not lost.

3.1.2 Longitudinal Structure functions

The longitudinal structure function of order p is defined as the pth moment of the *longitudinal* velocity difference over a distance r. In Figure 3.1 we can see that for the longitudinal structure functions the component of the velocity which is in the direction of the difference vector $\vec{r}$ is measured.

In turbulent flows, the turbulent velocity fluctuations u' are a small fraction of the mean flow velocity $\bar{U}$. A velocity measuring probe in a turbulent flow, therefore, registers a velocity signal whose time dependence is mainly caused by the spatial fluctuations of the turbulent velocity field that is swept across the probe by the mean flow. This situation is the subject of Taylor's *frozen turbulence* hypothesis:

$$\frac{\partial}{\partial t} = -\bar{U}\frac{\partial}{\partial x} \tag{3.3}$$

This hypothesis implies that temporal differences can be interpreted as spatial differences through $r = \tau U$ (Figure (3.1a)). A straightforward recipe for which flows the Taylor hypothesis is applicable can not be given. A frequently used guideline is the ratio between the standard deviation σ_u of the velocity and the mean velocity $\bar{U}$. For $\sigma_u/\bar{U} < 0.3$ the use of Taylor's hypothesis is permitted, [11].

If Taylor's hypothesis is applicable, longitudinal structure functions can be computed from a *time series* of velocity readings $u(t)$ using the velocity differences,

$$\Delta u(r, t+\tau) \;=\; u(x+r, t+\tau) - u(x, t+\tau) \tag{3.4}$$

$$= u(x,t) - u(x,t+\tau) \quad (3.5)$$
$$= -\Delta u(\tau) \quad (3.6)$$

in which τ ($=r/\bar{U}$) is in practice equal to multiples of the sample time. The measured velocity signal $u(t)$ is digitized and read by the DSP processor running the *Structurator* routine. This is a DSP processor assembler routine that is capable of *real-time* measurement of longitudinal structure functions. The workings of the Structurator is described in detail in section 3.3, but a brief description is given here. Because the used probe voltage has a non-linear relation to the actually measured wind velocity, an integer calibration table is used to convert the voltage-word to a velocity-word. For 32 different exponentially distributed time delays the velocity differences are then calculated and the velocity difference probability distribution functions (PDFs) for each time delay are created and updated. These activities take place in real-time. From these distribution functions the structure functions of arbitrary order can be calculated off-line, using Eq. (3.1).

3.1.3 Transverse Structure functions

Transverse or lateral structure functions are calculated from *transverse* velocity differences. The transverse velocity component is perpendicular to the direction of the distance vector $\vec{r}$ over which the velocity difference is calculated (Figure 3.1).

Measuring transverse structure functions is more complicated than measuring longitudinal structure functions. For longitudinal structure functions Taylor's frozen turbulence hypothesis can be used and only a one-wire probe is needed. For transverse structure functions, however an X-wire probe in combination with Taylor's hypothesis should be used to measure the lateral velocity increments or an array of detectors perpendicular to the mean flow. In such a configuration there is no need to invoke Taylor's hypothesis. For isotropic and homogenous turbulence the second-order transverse structure function, $G_2^T(r)$, is related to the second-order longitudinal structure function, $G_2^L(r)$, as

$$G_2^T(r) = G_2^L(r) + \frac{r}{2}\frac{dG_2^L(r)}{dr}. \quad (3.7)$$

The degree to which Eq. (3.7) is satisfied for measured $G_2^T(r)$ and $G_2^L(r)$, therefore, indicates the validity of Taylor's hypothesis and the isotropy and homogeneity of the turbulent fluctuations. Eq. (3.7) can be derived from the relation between the second-order longitudinal and transverse correlation functions, [11].

In this thesis we will not go into these type of transversal measurements and we will restrict ourselves to longitudinal measurements, but in chapter 7 a possible setup is given to do these transverse measurements using a multi-wire setup.

3.2 Motivation for using a digital signal processor

In this section the motivation for chosing a digital signal processor for the kind of measurements described in this thesis are pointed out. A detailed description of the DSP-system which has been used, is given in appendix A. Here we will only give a summary of the main characteristics of the digital signal processor and the implications for using it as a device for measuring turbulence.

3.2.1 Realtime requirements

Digital Signal Processors are processors which are specially designed for doing fast real-time digital signal processing. For the measurements described in this thesis we ask ourselves what the real-time requirements of the hardware should be, if we want to use a special device for doing real-time turbulence measurements. There are three main types of turbulence measurements in which we are interested:

1. *Measuring turbulence energy spectra,*
2. *Measuring characteristics of the turbulent flow,*
3. *Measuring structure functions.*

For the first type of measurements there doesn't seem to be a real-time requirement. It seems straightforward to just sample a block of velocity samples in the time-domain and to calculate the Fourier transformation to the frequency domain off-line using a Fast Fourier Tranformation (FFT) algorithm. However, because turbulence energy spectra inherents a high noise level, we want to improve the signal-to-noise ratio of these spectra. A way to do this is to measure a high number of blocks of velocity data in the time domain, calculate the Fourier transform of each time-block and updating an average energy spectrum with the resulting frequency data-block. In this way the signal to noise ratio can be improved significantly by averaging out the noise. It is then possible to measure a huge time-history record of velocities and calculate the average energy spectrum of these blocks afterwards off-line. But improving the signal-to noise ratio even better would require storing vast amounts of data which is limited by available diskspace and is not very efficient.

With a DSP processor however, it is possible to do the Fourier transformation on-line; sample a block of velocity data, calculate the Fourier transform of this block using a FFT routine, update an average energy spectrum and sample the next block of data and so on. In this procedure information is missed between succesive time-blocks but for measuring the energy spectrum this is not a problem. If the turbulence flow is measured far enough behind the grid, the flow is statistically homogeneous and isotropic and the turbulence is independent of the way it was generated, see section 4.5. Thus the statistical properties of the flow do not change in the time between measuring successive time-blocks of data and calculating the FFT algorithm. The program *spectrum.c*, see appendix B.1, uses this setup to measure the energy spectra of the turbulent flow.

The second type of measurements, the calculation of the turbulence characteristics, can be incorporated in the former, the measurement of energy spectra. When a velocity time-block is measured, before the FFT is calculated, some statistics can be done on the time-block and the resulting statistical quantities can be used to update running averages, just as is done for the average energy spectrum. The extra time it takes to calculate these statistics is neglectable compared with the time it takes to do a FFT algorithm.

For the last type, the measurement of structure functions, real-time requirements are high. One way of measuring structure functions, or better, the measurent of velocity difference probability distribution functions (Δu PDFs), is again to take a long time-history record of the velocities and to calculate the Δu PDFs and the structure functions afterwards off-line. However, we want to measure large velocity differences which are necessary for structure functions of higher order, and for that we need to sample a long time-history record. These

large velocity differences are rare events and the probability of these occuring is small, so in order to measure them with sufficient statistical accuracy, we need to sample for a long integration time. This would again require vast amounts of data storage which is not very efficient.

Here, real-time data-acquisition will provide for data-reduction and the ability to measure for a long duration of time. Therefore the Structurator has to form distribution functions of (longitudinal) velocity differences $\Delta u(\tau)$ measured a time τ (conform the Taylor's hypothesis in section 3.1.2) apart. Because the interest is in the *scaling* properties of the structure functions, the time delays τ need to be spaced exponentially. The expected dynamical range of the scaling behavior is such that the ratio of the largest to the smallest time delay is at most a factor of 10^3. The required delay times $\tau = n\tau_s$, in units of the sample time τ_s, have to be stored in a lookup table. For generating the velocity differences a long circular buffer can be used. The result of a difference calculation of two 12-bit velocity values at a certain delay τ corresponds to an address in (fast) memory. Each time a certain difference at that delay occurs, the contents of the memory at that address has to be incremented.

3.2.2 TMS320C40 DSP processor characteristics

The requirements mentioned above have lead to the choise of using a Digital Signal Processor for the kind of measurements described. Some of the characteristics of a DSP processor link up with the real-time requirements needed. A detailed description of DSP processors on the whole, in particular of the TMS320C40 DSP processor and the Transtech DSP development board is given in Appendix A. Here, only some of the most important characteristics of the TMS320C40 that are beneficial to our turbulence measurements will be mentioned:

- *Pipelined processing and Harvard architecture*; for fast processing capability,
- *32-Bit wide instruction word*; for parallel instructions or more-operand instructions,
- *Bit reversed addressing*; for FFT algorithm implementation,
- *Circular addressing*; for implementing circular buffers,

Pipelining is a technique which allows for two or more operations to overlap during execution. In pipelining, a task is broken down into a number of distinct subtasks which are overlapped during execution. It is used extensively in digital signal processors to increase speed. The TMS320C40 DSP processor has a four stage pipeline in which during a given machine-cycle, four different instructions may be active at the same time, although each will be at a different stage of completion. Pipelining is used together with a *Harvard architecture* of the DSP processor. In a Harvard architecture the access to data and instruction memory is done with seperate channels so that the information streams do not interfere. Data and instructions are placed in seperate memory spaces. For a detailed description of the Harvard architecture and pipelining see Appendix A.

The TMS320C40 has a *32-bit wide instruction word* which makes it possible for two parallel instructions or two- or three-operand instructions to be executed in one machine-cycle. This parallel instruction capability also increases the speed of the DSP processor.

The capability of *bit reversed addressing* makes it possible for the TMS320C40 to implement Fast Fourier Transforms (FFTs). If the data to be transformed is in the correct order,

the final result of the FFT algorithm is in bit-reversed order, see section 5.4.1. To recover the frequency-domain data in the correct order, certain memory locations must be swapped. The bit-reversed addressing mode makes swapping unnecessary. The next time data must be accessed, it is accessed in a bit-reversed manner rather than sequentially. In the TMS320C40, this bit-reversed addressing can be implemented through both the CPU and DMA.

In our Structurator assembler routine the velocity differences are generated using a circular buffer storing old velocity samples. The circular buffer acts as a sliding window that contains the last 1024 sampled velocities which are compared with the current new velocity sample. As a new velocity sample is brought in, the new data overwrites the oldest velocity data. The key to using a circular buffer is the implementation of a circular addressing mode of the TMS320C40. The block-size register (BK) (see App.A) specifies the size of the circular buffer. In assembler code, using the circular addressing mode is implemented by using the '%' character in the code when addressing the circular buffer, see the assembler code in the next section.

3.3 Implementation of the Structurator

We now turn to the actual program which has been developed to implement the real-time *Structurator* on the C40 DSP processor. In what follows the working of the real-time structurator routine is explained, [16].

The *Structurator* enables the computation of (longitudinal) structure functions of arbitrary order from a line measurement in real-time. The structurator allows for evaluation of structure functions of *one* 12-bits time signal through the accumulation of probability distribution functions (PDFs) of discretized velocity differences. As was already mentioned in section 3.1.2, it suffices to measure distribution functions of velocity differences and afterwards generate the structure functions. The Structurator, therefore, forms distribution functions of (longitudinal) velocity differences $\Delta u(\tau)$ measured a time τ apart (conform the Taylor's hypothesis in section 3.1.2). Because the interest is in *scaling* properties of the structure function, the time delays τ need to be spaced exponentially. The required delay times $\tau = n\tau_s$, in units of the sample time τ_s, are stored in a look-up table. The velocity differences are generated using a circular buffer of 1024 positions long.

In appendix B.2 the assembler code for the Structurator routine *c40struc.asm* is given, together with the C-code *structur.c*, which is used to call the structurator routine and to control the flow of data to and from the PC's harddisk and the DSP processor. The assembler routine has three main blocks which are the *initialisation*, the *main-loop* and the *end-phase*. The call sequence for the structurator routine is:

```
int c40struc(MCATBL,RELOFS,CALIBR,BUFFER,SIZE)
```

with the following arguments:

int *MCATBL:	pointer to location of MCA[1]memory block
int *RELOFS:	pointer to table with relative offsets
int *CALIBR:	pointer to location of calibration table
int *BUFFER:	pointer to 1024-size circular buffer
int SIZE:	total number of samples to take before program end
On return:	int R0 = status of ADC FIFO buffer

3.3.1 Initialisation

At startup of the structurator routine the arguments mentioned above are passed into registers AR2, R2, R3, RC and RS respectivelly, for the case of *register based* argument passing. If *stack based* argument passing is chosen, then the arguments are loaded from the stack and put into these registers. After the passing of the arguments, the pointers in these arguments are copied into named variables for convenience (variables @MCATBL, @RELOFS, @CALIBR, @BUFFER, @SIZE). Now the *initialisation* part of the structurator can commence. The initialisation code is given below, see for the complete code appendix B.2,

```
INIT01:   LDA     1024,BK
          LDI     @MCATBL,R5
          LDA     @RELOFS,AR5
          LDA     @CALIBR,AR1
          LDA     @BUFFER,AR4
          LDI     @SIZE,R8
          LDI     @maxval,R1
          LDI     0,R9
          LDI     1,R7
          ADDI3   R1,R5,R6
          LDI     @adc,ADC
          LDI     @cmd1,CMD
          STI     CMD,ADC_OUT
          LDI     ADC_IN,R10
          LDI     *AR4--%,R0
          LDI     @cmd2,CMD
          LDI     1023,RC
          RPTB    FILL
          STI     CMD,ADC_OUT
          LDI     ADC_IN,R10
          AND     0FFFFh,R10
          LDA     R10,IR1
          LDI     *+AR1(IR1),R4
FILL:     STI     R4,*AR4--%
          ADDI    1024,R9
```

The variables containing the pointers, are copied into processor registers. The variable @maxval is equal to 4095 and is the maximum value possible in the calibration table (12 bits ADC). Register R6 is loaded with the base address of the MCA-memory block added with the maximum value of the calibration table. After that, the ADC registers are initialized to enable external triggering which also will empty the ADC FIFO buffer of old data. Register AR4 is set to point to the oldest value in the circular buffer and after the command-word for the ADC is set for single word read of samples from the ADC FIFO buffer, the circular buffer can be filled with the first 1024 samples. The filling of the buffer is done by requesting for the next sample from the ADC, after that the sample value is used as an index for the calibration lookup table. After converting the voltage integer value to a velocity integer value via the calibration table, this velocity value is stored into the circular buffer and the buffer pointer is

[1]The term MCA, which stands for Multi Channel Analyser, is an artifact of the old Structurator which was implemented on the PhyDAS system. A Multi Channel Analyser was used to store and to increment the PDF memories, hence the term MCA for the block of DSP processor memory which stores the velocity difference PDFs.

updated for the next sample. The '%' character in the instruction LDI *AR4−−%,R0 is used to store the contents of register R0 into the buffer using the circular address mode. After the buffer is filled with 1024 velocity integer values the sample counter is increased with 1024.

3.3.2 Main structurator loop

The *main Structurator loop* calculates the subsequent MCA memory block address to be incremented during the acquisition of the next samples. The main loop code is given as:

```
        STI     CMD,ADC_OUT
        LDI     ADC_IN,R10
        AND     0FFFFh,R10
        LDA     R10,IR1
        ADDI    R7,R9
NXTSMP: LDA     *AR5,IR0
        LDA     AR4,AR6
        LDI     7,RC
        LDI     *+AR1(IR1),R4
||      LDI     *AR6++(IR0)%,R0
        RPTBD   STRUCT
        STI     R4,*AR4−−%
        LDA     *++AR5,IR0
        ADDI    R6,R4
        LDA     *++AR5,IR1
        SUBI3   *AR6++(IR0)%,R4,AR0
        LDA     *++AR5,IR0
        ADDI    8191,R4
        ADDI3   R7,*AR0,R0
        SUBI3   *AR6++(IR1)%,R4,AR1
        LDA     *++AR5,IR1
        ADDI    8191,R4
        ADDI3   R7,*AR1,R1
||      STI     R0,*AR0
        SUBI3   *AR6++(IR0)%,R4,AR2
        ADDI    8191,R4
        SUBI3   *AR6++(IR1)%,R4,AR3
        ADDI3   R7,*AR2,R2
||      STI     R1,*AR1
        ADDI    8191,R4
        LDA     *++AR5,IR0
        ADDI3   R7,*AR3,R3
        STI     R2,*AR2
STRUCT: STI     R3,*AR3
        LDA     @CALIBR,AR1
        LDA     @RELOFS,AR5
        CMPI    R8,R9
        BZ      END
        STI     CMD,ADC_OUT
        LDI     ADC_IN,R10
        BD      NXTSMP
        AND     0FFFFh,R10
        LDA     R10,IR1
        ADDI    R7,R9
```

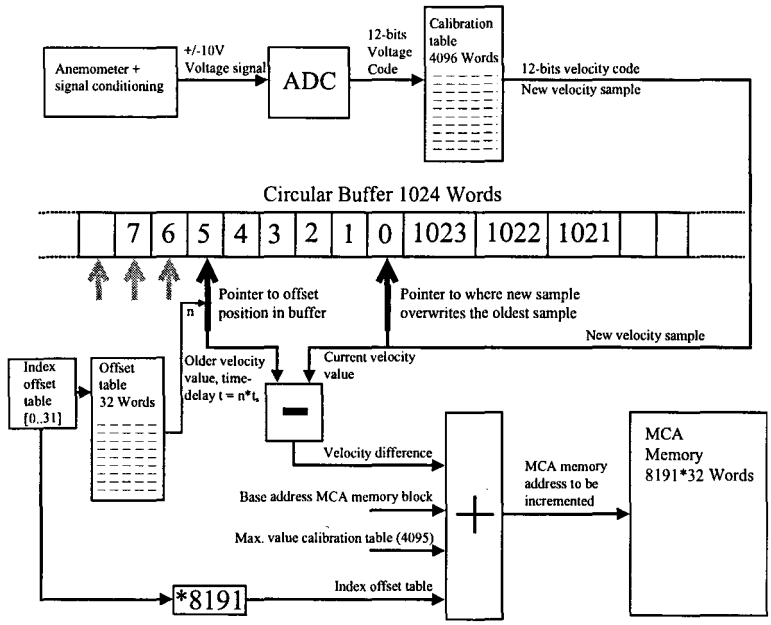


Figure 3.2: *Schematic overview of structurator routine*

The *main structurator loop* is illustrated in Figure 3.2 and is described as follows. The main-loop is situated between the global labels NXTSMP and STRUCT, see code above. Before the loop is executed, after the *initialisation* block, the next sample is read from the ADC FIFO into index register IR1 and the sample counter is updated. At the beginning of the loop at label NXTSMP, the first relative offset from the offset table is read into index-register IR0. Register AR6 is set to point to the running index in the circular buffer, which is the pointer to the position in the circular buffer, which is the value of the offset earlier relative to the new sample. Inside the main-loop there is a second loop which increments the 32 MCA-memory addresses, and is situated between instructions RPTB STRUCT and STRUCT. This loop processes the 32 offset values from the offset table in 8 runs. The offset values are thus processed in 8 groups of 4 offsets more or less interleaved, to avoid register conflicts. Before the offset-loop can commence pointers are set and the pipeline is flushed (the first three instructions following RPTB STRUCT are not part of the loop). The next new sample is used as index (IR1) for the calibration table to convert the voltage integer to a velocity integer, which is stored in register R4. Register AR6 is updated to point to the position in the circular buffer of the sample at the first relative offset. The velocity integer is stored in the circular buffer overwriting the oldest sample in the buffer, and the next relative offset is read into index register IR0. In register R4 the MCA base address added with the velocity integer and added with the maximum value of the calibration table is stored. After that, the loop that increments the 32 MCA memory addresses can begin.

In this loop the subsequent offset values are loaded into IR0 or IR1, alternately. And register AR6 which points to the next sample in the circular buffer conform the offset value, is updated accordingly. The MCA memory addresses are calculated by substracting register R4 (= max. value calibration table + MCA base address + velocity integer) with the sample in the circular buffer conform the offset value and adding (n*8191) (with n = number of the

offset being processed). This memory is then stored into registers AR0, AR1, AR2, or AR3. These MCA memory locations are read, incemented and the result (= value of MCA memory location) is stored into registers R0, R1, R2 or R3, which are stored into the acccording MCA memory locations. This loop is run for 8 times, processing the 8x4=32 offset values, stored in the offset table.

When all 32 offsets are processed, register AR2 is reloaded with the start address of the calibration table and register AR5 is reloaded with the baseaddress of the offset-tabel. Then the program checkes whether or not the number of required samples is already taken, if this is not the case, then next new sample is read from the ADC, the sample counter is updated and a branche occurs to the loop beginning at label NXTSMP (but first doing the three instructions after instruction BD NXTSMP, pipeline flush).

3.3.3 Program end phase

The *program end* part of the structurator routine is given as,

```
END:    LDI     @stop,CMD
        STI     CMD,ADC_OUT
        LDI     ADC_IN,R0
```

The STOP command word is send to the ADC and the FIFO status is read from the ADC and stored into register R0. The structurator routine returns to the C-environment making the ADC FIFO status available to the user. In the C-program (*'structur.c'*) the FIFO status is checked for overflow of the FIFO buffer, if so the MCA memory tabel is useless because samples were missed and the MCA table is not correct. A lower sample frequency should then be chosen.

The MCA memory block which is available after runnning the structorator routine is saved to disk to be processed further. The MCA memory block which has a size of 32x8192x4 bytes = 1Mb, consists of 32 arrays each representing a velocity difference probability distribution function. Equation (3.1) can then be used to calculate the structure functions using these 32 velocity difference PDFs.

In chapter 6 some structure functions are given, measured and calculated with the Structurator.

Chapter 4

Experimental setup and instrumentation

In this chapter the experimental setup and the instrumentation for the turbulence measurements are described. First, an overview is given of the total experimental setup and the instrumentation. Secondly, the different parts of this setup are described in more detail. The instrumentation among which the hot-wire detectors and their calibration is discussed. The signal conditioning and the data conversion is described and finally the windtunnel in which the experiments were done is described.

4.1 Overview of instrumentation

Figure 4.1 shows an overview of the total experimental setup and the instrumentation used for turbulence experiments.

For the longitudinal measurements described in this thesis, a single velocity hot-wire probe is used. This hot-wire is connected to a DISA 55M01 constant temperature anemometer (CTA), which gives a voltage signal representing the windvelocity in the windtunnel. The principle of hot-wire anemometry is described in section 4.2.

The signal conditioning unit adapts the resulting anemometer signal to the input of the A/D-converter by substracting the mean voltage and amplifying the resulting fluctuations, see section 4.4 for a detailed description. This signal is sent via an anti-aliasing filter to the ADC.

The used Digital Signal Processor (DSP) is from Texas Instruments, type TMS320C40. This is a programmable floating-point DSP capable of fast digital signal processing. A detailed description of the DSP is given in Appendix A. We used a DSP development board from Transtech Parallel Systems, which is a DSP-board for AT-class Personal Computers (PC). This so-called motherboard can be placed in one of the 16-bit expansion slots of the PC. The DSP is situated on this motherboard and can be controlled and programmed by downloading programs made on the PC to the DSP processor. The processor executes the downloaded program and the results can be transmitted back to the PC to be processed further. The Analog-to-Digital converter (ADC) can be placed on the same DSP motherboard and is connected to one of the six communication ports of the TMS320C40 Digital Signal Processor.

For calibration of the hot-wire probes the situation is somewhat different. The calibration procedure involves the use of the PhyDAS system for which this procedure originaly was de-

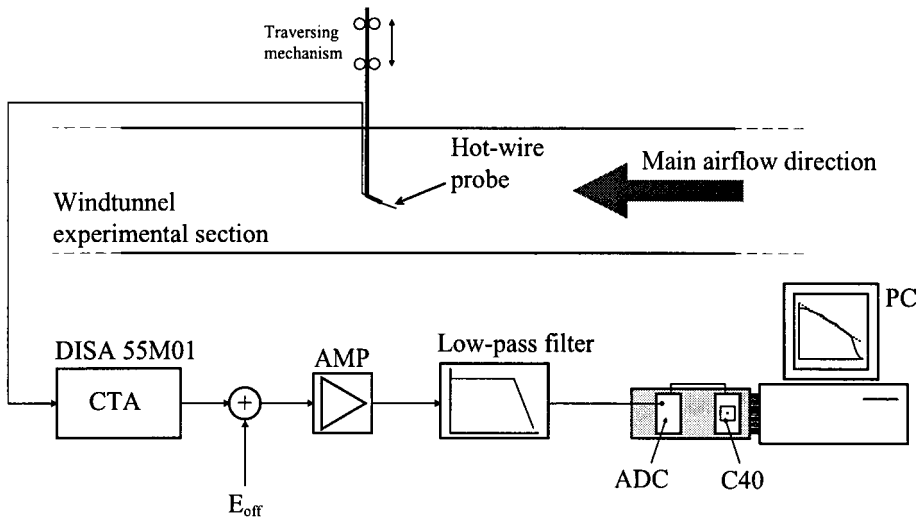


Figure 4.1: *Schematic overview of total experimental setup. The local velocities are measured with a hot-wire probe and a constant temperature anemometer (CTA) in the experimental section of the windtunnel. Via the signal conditioning unit consisting of a voltage offset (E_{off}), an amplifier (AMP) and an anti-aliasing low-pass filter, the voltage signal is sampled with the Analog-to-Digital converter (ADC) and processed by the Digital Signal Processor (C40). The final data is transfered to the Personal Computer (PC) to be processed further.*

signed and is described in detail in section 4.3. Here only a brief description of the PhyDAS system is given. PhyDAS (Physics Data-Acquisition System) is developed at the Physics department at the Eindhoven University of Technology. The system is controlled by a Motorola M68030 microprocessor and a dedicated bus structure (PhyBUS) that can accommodate a variety of instrumentation modules, such as ADC's, stepper motor interfaces, etc. For the measurements described in this thesis the PhyDAS system has not been used, other than for the control of the hot-wire calibration setup.

4.2 Hot-wire anemometry

Our basic interest is in the measurement of velocity fluctuations in the air-flow. The hot-wire anemometer has been succesfully used for many years to measure turbulent velocity fluctuations, because it can handle these fast velocity fluctuations with ease. In its simplest form, a hot-wire consists of a small metallic wire, usually of platinum or tungsten, which is heated by an electric current that flows through it. If the wire is cooled by air flowing around it, its electrical resistance is reduced and this reduction in resistance can be related to the local air velocity.

4.2.1 Operating principle

The operating principle can be understood as follows, [5]: Assume that the wire has initially the same temperature T_f as the air and that it has an electrical resistance R_f. Suppose that the wire is heated to a temperature T_w, the resulting resistance will then be given by

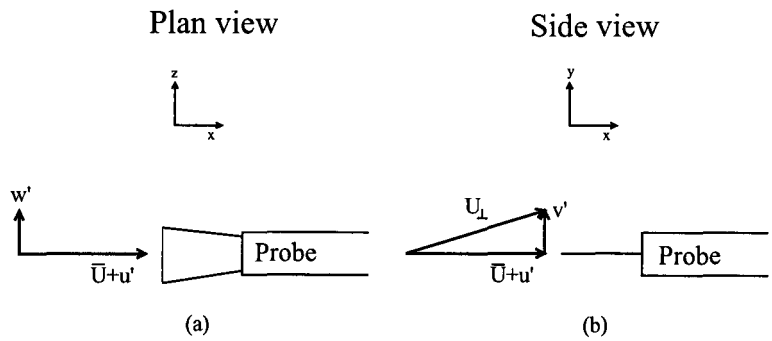


Figure 4.2: *Hot-wire configuration to measure the streamwise turbulent fluctuations in an unidirectional mean flow U. (a) planview. (b) sideview.*

$$R_w = R_f \left[1 + c_T \left(T_w - T_f\right)\right], \tag{4.1}$$

where c_T is the temperature coefficient of the wire. If a current I flows through the wire, heat is generated at a rate $I^2 R_w$. The heat is transferred from the wire to the fluid at a rate $c_h S \left(T_w - T_f\right)$, where c_h is the heat transfer coefficient and S is the surface area of the wire. For thermal equilibrium the two rates can be equated and, using Eq. (4.1) for the temperature difference, we get

$$I^2 = c_h S \left(T_w - T_f\right) \tag{4.2}$$

$$= \frac{c_h S \left(R_w - R_f\right)}{c_T R_f}. \tag{4.3}$$

Heat is transferred from the wire to the perpendicular (laminar) flow $U_\perp$ by forced convection and to the support of the wire by conduction. Therefore,

$$\frac{I^2 R_w}{R_w - R_F} = A + B U_\perp^{\frac{1}{2}} \tag{4.4}$$

where A and B are measured to be independent of the air velocity. Eq. (4.4) is often referred to as *King's Law*. But in practice we assume a more general polynomial relation with coefficients that are determined in a static calibration procedure that uses a laminar flow with known $U_\perp$, see section 4.3.

When a hot-wire is used for measuring turbulence fluctuations, it is to first order only sensitive to the components u' in the direction of the mean flow. The situation is illustrated in Figure 4.2, where it is assumed that the wire is aligned along the z-axis. Now, suppose that only the components perpendicular to the hot-wire affect its cooling. This means that the z-component w' does not contribute to the cooling of the wire. Thus, when the x-component u' is the turbulent velocity component in the direction of the main flow $\bar{U}$ and v' is the lateral turbulent velocity component perpendicular to the wire in the y-direction, we can write

$$U_\perp = \left[\left(\bar{U} + u'\right)^2 + v'^2\right]^{\frac{1}{2}}. \tag{4.5}$$

A series expansion gives

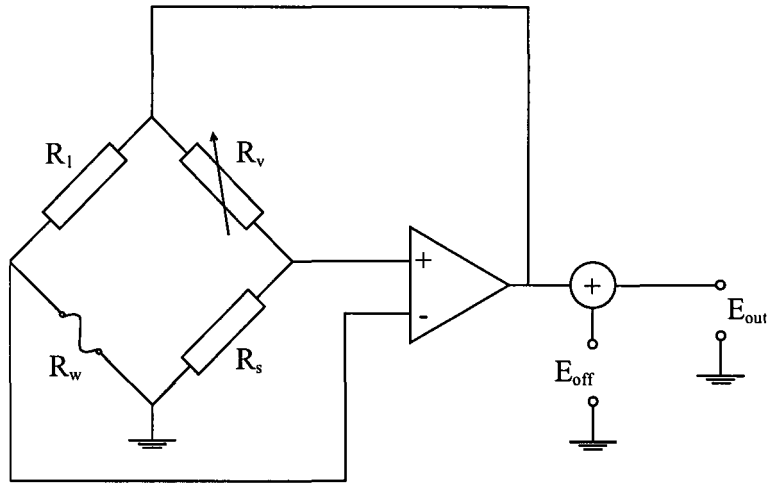


Figure 4.3: *Schematic overview of an anemometer. It shows the Wheatstone bridge with a wire, R_w, two fixed resistors, R_1 and R_s, and a variable resistor R_v. The constant resistance of the wire is controlled by the feedback amplifier. The voltage E_{out} is a measure of the flow velocity.*

$$U_\perp = \bar{U}\left(1 + \frac{u'}{\bar{U}} + \frac{v'^2}{2\bar{U}^2} - \frac{u'v'}{2\bar{U}^3} + \cdots\right). \tag{4.6}$$

Thus to first order in small quantities

$$U_\perp = \bar{U} + u'. \tag{4.7}$$

In section 5.6 we will worry about the neglect of the perpendicular fluctuating velocity.

For measuring turbulent velocities two different methods can be applied: *constant-current* anemometry and *constant temperature* anemometry. The first method is the oldest. Here the electric current I is kept constant and the temperature of the wire (and hence its electrical resistance) changes due to the fluctuating cooling caused by the fluctuating velocity. Constant-current anemometers are less convenient to operate because the response of the wire falls at high frequencies. In the second case, the electrical resistance of the wire and so its temperature is kept constant. Now, it is the electric current that fluctuates due to the air-velocity fluctuations. This last method is more convenient to operate and higher frequencies can be measured. The common method of measuring the resistance change of the hot-wire is to build the probe into an arm of a Wheatstone-bridge.

4.2.2 Constant temperature anemometry and hot-wire probe

For the measurents in this thesis Constant-Temperature Anemometry (CTA) was used to measure velocity fluctuations. The anemometer used was a type DISA 55M10 anomometer. In Figure 4.3 the electrical circuit of the anemometer is schematically shown. The Wheatstone-bridge consists of two fixed resistors R_1 and R_s, the probe (hot-wire) resistance R_w and a variable resistor R_v.

The resistance of the wire has a positive temperature coefficient and therefore it will increase when the wire is heated by an electric current. When the probe is cooled through an airflow the feedback amplifier controls the current through the probe and therefore its

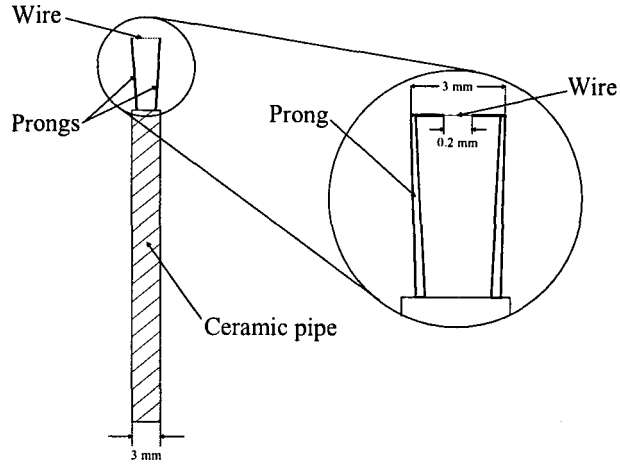


Figure 4.4: *Schematic view of the hot-wire probe with its dimensions.*

(constant) temperature and resistance. The values of the resistors R_1, R_s and R_v determine the resistance of the probe for which the bridge is in equilibrium. The working temperature (resistance) of the hot-wire is determined by

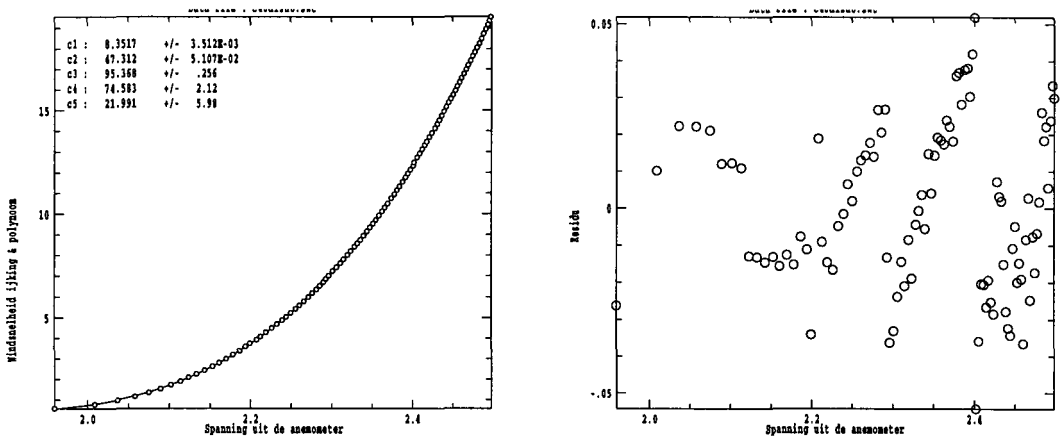
$$\frac{R_w}{R_s} = \frac{R_1}{R_v}. \tag{4.8}$$

A higher working temperature gives a better sensitivity of the anomometer. The set temperature ($R_w = 1.6R_f$) is a compromise between sensitivity and mechanical stability, where R_f is the resistance of the hot-wire for which the bridge is set in equilibrium with variable resistor R_v. The voltage E_{out} is a measure of the velocity U of the flow.

The probe used for the experiments is shown in Figure 4.4. The probe is positioned in a holder at the front end of a long pipe that points into the flow. The diameter of the tube of the probe in which the prongs are fixed is 3 mm and the prongs are 10 mm long.

The effective lenght l_p of the probes is an important parameter. If this length is to small in relation to the diameter d of the wire, the heat loss to the prongs might affect the dynamic behavior of the wire. A frequently used rule of thumb is that the ratio l_d/d should be larger than 200, [5]. In this case it is assumed that the ends of the wire are at a constant temperature. However, the temperature of the ends of the wire fluctuates. The ratio of the heat lost from the wire by convection to that lost by conduction to the prongs changes under dynamic conditions and the temperature distribution along the wire changes, [11]. The latter is subject to the restriction that the mean wire temperature is constant, which is imposed by the CTA circuitry. The frequency dependence of the heat lost to the prongs implies that the static calibration might not be completely representative for the dynamic behavior of the probe. The dynamic behavior of hot-wire probes has been studied many times, and it is known that the influence of the prongs is insignificant for probes with $l_d/d \geq 200$, [11].

In our measurements we used a wire with $l_d = 0.2mm$ and $d = 2.5\mu m$, which results in a ratio of $l_d/d = 80$. The size $l_d = 0.2mm$ is chosen to be of the order of the Kolmogorov length scale η, to minimize averaging effects. This implies that the above mentioned dynamic effects may become important. However, a smaller diameter $d < 2.5\mu m$ for the wire is physically difficult to make.



(a) (b)

Figure 4.5: *A typical hot-wire probe calibration curve. (a) The fitted curve shows the relation between the wind velocity and the measured voltage from the anemometer. (b) The residues of the 4th order polynomial to the calibration points*

4.3 Calibration of the hot-wire probe

The output voltage (V_{out}) of the anemometer is a function of the air velocity $\bar{U} + u'$. As the relation between the velocity and the voltage can not be calculated on a theoretical basis, the wires are usually calibrated in a laminar flow.

In the experiments the wire has been calibrated before each measurement, using portable automated calibration equipment. The static calibration of the detector including the filters, amplifiers, ADC, etc was done in-situ. In this way, the probe was not moved after the calibration and possible deviations of the instrumentation were included in the calibration. A calibration nozzle is used to generate a known laminar flow perpendicular to the wire. The output voltage of the anemometer is measured over a range of air-velocities. The measured calibration functions $U_\perp(V)$ are parameterized using a fourth-order polynomial,

$$U_\perp(V) = c_0 + c_1 V + c_2 V^2 + c_3 V^3 + c_4 V^4, \tag{4.9}$$

with $c_0 \ldots c_4$ the fit parameters. This polynomial is used to generate the calibration tables that are needed for the real-time data processing. Figure 4.5 shows a measured and fitted calibration curve. The residues for this polynomial fit seem to have some kind of correlation between them, which is of course undesirable. This is likely to be caused by the pressure transducer used during the calibration procedure. During calibration the pressure transducer has to be switched through different scale-ranges and this probably causes the effect depicted in Figure 4.5b.

In ref. [8] the calibration procedure is described for the PhyDAS system. Because the ADC used as input device for the digital signal processor, was mounted on the Transtech DSP PC-board, it was neccesary to measure the voltage of the anemometer with this ADC and not with the ADC used for the PhyDAS system. Without having to do the whole procedure of rewriting the software for the calibration procedure for the Transtech DSP system, a method

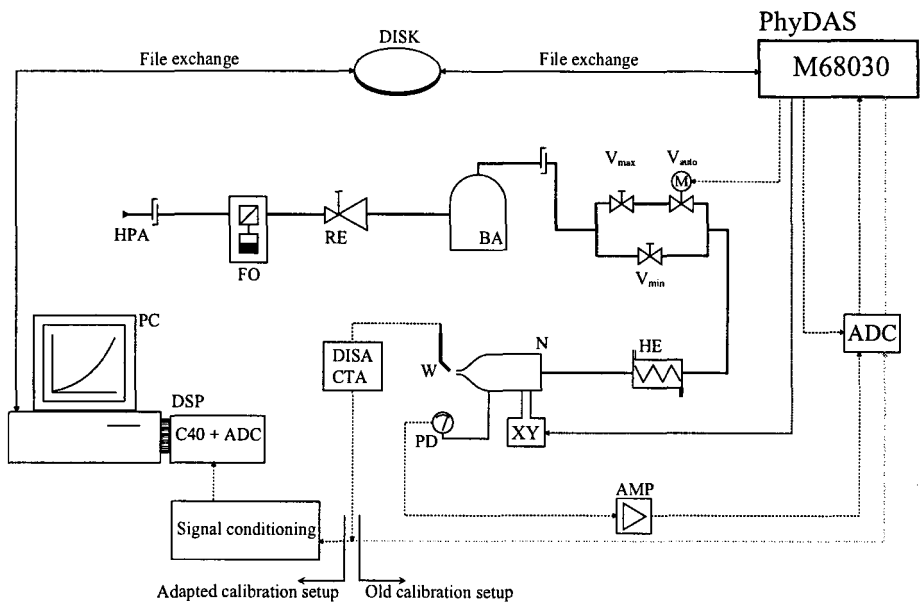


Figure 4.6: *Blockdiagram of the calibration setup. The old calibration setup with the PhyDAS system has been adapted to incorporate the DSP/PC system. The wire voltage is now read with the ADC from the DSP system and a mounted disk is used to transfer the voltage information to the PhyDAS system by means of a file exchange.*

was developed to combine the ADC and the DSP-system with the already existing calibration procedure on the PhyDAS system.

In order to make it ourselves not too difficult it was decided that a means of communication had to be made to let the PhyDAS-calibration program take the voltage readings from the ADC/DSP-system and use it in the calibration of the hot-wire. Figure 4.6 is a schemetical representation of the adapted calibration setup in comparison with the old setup.

The original calibration procedure has the following setup [8]: the airflow for the nozzle is generated with high pressure air (6bar) (HPA). The air is cleaned with a filter and an oil trap (FO). The pressure is reduced to 4bar with a reducer (RE) to dim the long time pressure variations and then led into a barrel with a capacity of 100dm^3 (BA) to dim the high frequency pressure veriations. Next, the air is led to valves to regulate the airflow. One of the valves (V_{auto}) is controlled by the PhyDAS computer (M68030). The other two (V_{min} and V_{max}) determine the minimum and the maximum flowspeed. By introducing these two valves, the whole range of the automated valve is used independent of the range of the flow. After these valves, the airflow is led through a heat exchanger (HE), to allow the air to gain the temperature of the surroundings. Finally, the air reaches the nozzle (N). The nozzle is mounted on a computer controlled x-y-table (XY) for calibrating multiple wires. The velocity of the air-flow from the nozzle can be calculated from the pressure in the nozzle relative to the air pressure outside the nozzle. The pressure is measured with a pressure transducer (PD). The output voltage of the pressure transducer is within the range of 0-1Volt. The ADC is designed for a range of $\pm$10Volt, therefore an amplifier with an offset (AMP) adapts the output voltage of the pressure transducer to the range of the A/D-converter (ADC). This converter is triggered by the PhyDAS computer (M68030). After a scan the computer writes the calibration data to disk to be processed further.

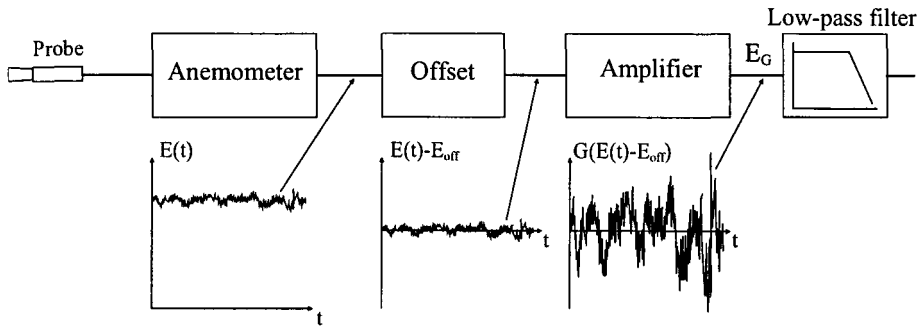


Figure 4.7: *The principle of the signal conditioning unit.*

The calibration procedure was left the same except for the part where the voltage is measured from the hot-wire. For our experiments it was neccesary to use a different constant temperature anemometer (CTA, DISA 55M01) and to use the Transtech ADC together with the C40 digital signal processor. The transfer of the voltage reading from the CTA, to the PhyDAS computer is done by means of a so called *'file-handshake'* between the PhyDAS and the DSP-PC system. When the PhyDAS calibration program is ready to take a voltage reading from the hot-wire probe, a *'ready file'* is written to a directory on a mounted disk which is accessible for both systems, and the DSP-PC waiting for this *ready-file*, then takes 1024 samples from the probe with the Transtech ADC, calculates the average, and writes the mean voltage value to a *data file* and writes an *'acknowledge file'* to the same directory. In the mean time the PhyDAS system waits for the *acknowledge file* and when it receives it, the voltage data is available. The mean voltage value of the probe read from the *data file* is used as a point in the calibration curve.

This adapted calibration procedure works well, only the time it takes to do a complete probe calibration is increased in comparison with the original calibration setup due to the extra time needed for the *'file-handshake'*. It depends on future developments if this calibration procedure is maintained or if it will be completely transfered to the PC-DSP system. The only extra equipment needed will be a steppermotor controler to control the different motors, so this will not be a huge problem. Then only rewriting of the software for the C40 DSP and PC has to be done.

4.4 Signal conditioning and data conversion

Before data-acquisition can commence, the parameters of the digital measurement system must be set. The purpose of the signal-conditioning unit is considered first. An n-bit A/D converter operates in discrete voltage steps, ΔE, with 2^n different output values. A 12-bit converter can therefore only select 4096 different output values, and if the voltage range of the A/D converter is ± 10 V then $\Delta E = 4.9$ mV. Consequently, if a nonlinear voltage signal with small fluctuations like a turbulence signal from an anemometer, is fed directly into the A/D converter, then significant resolution errors may occur in the fluctuating part of the acquired signal.

A practical way of minimizing this problem is to match the fluctuating part of the hot-wire signal to the A/D converter's voltage range, by using a *signal-conditioning unit.* The procedure is illustrated in Figure 4.7. The signal-conditioning steps are as follows: first, an

offset voltage, E_{off}, which is approximately equal to the time-mean value, $\bar{E}$, is substracted from the direct hot-wire signal, $E(t)$. Provided that the value of E_{off} is known, then any reasonable value of E_{off} can be used. The signal, $E(t) - E_{off}$, is then amplified, with a gain G, so that the fluctuating output signal, $E_G(t)$, from the conditioning unit,

$$E_G(t) = G(E(t) - E_{off}), \tag{4.10}$$

nearly covers the complete A/D converter voltage range (±10Volt). Having set the offset, E_{off}, and the gain, G, of the signal-conditioning unit and having set the low-pass filter (anti-aliasing filter) with the appropriate cutoff frequency (= half of the sampling frequency), the only remaining step is to specify the sampling rate and the sampling time (or the number of samples). Data-acquisition can then be initiated, and on completion a digital time-series record, $E_G(i)$, will be available.

The nonlinear voltage signal, $E(t)$, has to be converted into the corresponding velocity time-history, $U(t)$, using the calibration curve. The required statistical velocity information can then be obtained by an analysis of $U(t)$. The data acquired by the DSP processor is a time-series record of the voltage, $E_G(i)$. The corresponding anemometer voltage values, $E(i)$, can be obtained by inverting Eq. (4.10) giving

$$E(i) = \frac{1}{G}E_G(i) + E_{off}. \tag{4.11}$$

To evaluate the required velocity information it is necessary to apply the calibration equation specified as the 4th-order polynomial curve fit of Eq. (4.9). It would be very time consuming to calculate each time the 4th-order polynomial to convert the voltages to velocities. A time-saving solution to this problem is to combine digital data acquisition with a look-up table method.

The most direct look-up table method utilizes the n-bit nature of an A/D converter. Consider the following two A/D principles. Firstly, an n-bit A/D converter can only identify 2^n different output values over the complete input voltage range, in our case ± 10V, and for a typical 12-bit converter there are only 4096 different output values. Secondly, the direct output from the A/D converter is an integer value, I. For a 12-bit, ± 10V, ADC the relationship between I and the ADC output voltage, E_{out}, is

$$E_{out} = -10 + I\frac{20}{4095}, \tag{4.12}$$

with $0 \leq I \leq 4095$. For each input voltage, E_G to the A/D converter, the nearest value of I which will minimize $E_G - (-10 + 20I/4095)$ is selected[1], and at the end of the data acquisition an integer time history record , $I(i)$, will be available. Setting $E_{out}(i)$ equal to the A/D input voltage record, $E_G(i)$, we have

$$E_G(i) = -10 + I(i)\frac{20}{4095}. \tag{4.13}$$

To obtain a velocity history record, it is neccessary to convert the voltage time-record to a velocity time-record. This can be done with the 4th-order polynomial fit, like Eq. (4.9), which is acquired during the calibration procedure, see section 4.3. As mentioned before it

[1]This type of analog-to-digital conversion is called a successive approximation ADC. In section 5.5 we will consider the differential non-linearity, inherent for this type of ADC

would be time-consuming to calculate the 4th-order polynomial for each data point, but this can be done with a table look-up method. If we write the calibration equation (4.9) as

$$U_\perp(E(i)) = c_0 + c_1E(i) + c_2E^2(i) + c_3E^3(i) + c_4E^4(i) = f(E(i)), \tag{4.14}$$

and inserting Eq. (4.11) into Eq. (4.14) the following relationship between the perpendicular velocity $U_\perp(i)$ to the hot-wire and the acquired voltage, $E_G(i)$, can be obtained:

$$U_\perp(E_G(i)) = f\left(\frac{1}{G}E_G(i) + E_{off}\right) \tag{4.15}$$

Finally, by introducing the digital relationship between $E_G(i)$ and $I(i)$ of Eq. (4.13), the relation for $U_\perp(I(i))$ becomes

$$U_\perp(I(i)) = f\left(\frac{1}{G}\left(-10 + I(i)\frac{20}{4095}\right) + E_{off}\right) \tag{4.16}$$

In Eq. (4.14) the coefficients c_0, c_1, c_2, c_3 and c_4 are known calibration constants determined from the calibration of the hot-wire probe, and in Eq. (4.16) G and E_{off} are known preset values for a set of measurement points. The only variable in Eq. (4.16) is the integer variable I, and for a 12-bit ADC I can only have 4096 different values. Therefore, if a large number of data conversions are required, it is worthwhile using the look-up table method due to savings in the data-conversion time. The table look-up method can be implemented in the following way [5]:

1. The probe is first calibrated and the signal-conditioning unit set. A look-up table, $V(j), 0 \leq j \leq 4095$, is then created using Eq. (4.16).

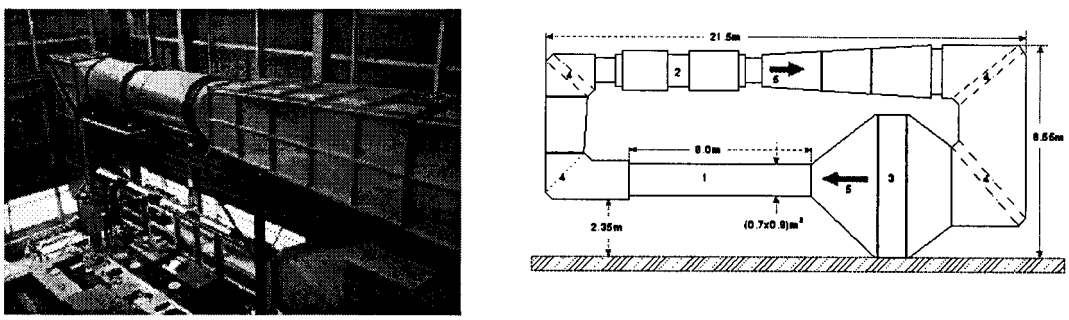
2. For each data acquisition, the input voltage record, $E_G(i)$, is stored in its equivalent ADC *integer* format, $I(i)$.

3. Data conversion is achieved by poking each value of $I(m)$ into the look-up table and retrieving and storing the related velocity integer representation.

4.5 The windtunnel

The measurements in this thesis were done in the large windtunnel of the Eindhoven University of Technology situated in the building 'Warmte en Stroming' and is shown in Figure 4.8. This tunnel is a closed windtunnel and can generate a flow with a maximum speed of $25^m/_s$. It has an experiment section of 8m lenght with a 0.7 x 0.9m^2 cross-section. In this tunnel we can generate grid turbulence by a grid in the flow. Since there is no continuing source of turbulent energy, the turbulence decays slowly with the distance downstream. Far enough behind the grid the flow is statistically homogeneous and isotropic and the turbulence is independent of the way it was generated, [2].

Measurements for the longitudinal structure functions and for the turbulence spectra where done in this windtunnel. The grid consists of a planar mesh with square rods. Grids are usually characterized by the solidity of the grid, which is the ratio of open to blocked area, and the mesh size. Several investigators have found that the flow should be fully developed and thus independent of the initial conditions for $x/M > 30$, where M is the mesh size of



(a) (b)

Figure 4.8: *Schematic view of the Windtunnel 'Goliath' at the Eindhoven University of Technology. (a) Photograph. (b) Schematic view: 1) experiment section, 2) blower, 3) contraction, 4) lamellae, 5) flowdirection.*

the grid, [11]. The used grid in this thesis had a mesh size M = 0.1m and 36% solidity. The probe was situated at about $x/M = 30$ behind the grid.

In grid-turbulence, the situation of isotropic turbulence is most closely approached at the centerline of the grid. However, in an attempt to find a compromise between isotropy and the value of the Reynolds number Re_λ the probe were placed away from the centerline closer to the wall. So higher Reynolds numbers could be measured while losing isotropy of the turbulence flow.

The vertical position of the hot-wire probe in the windtunnel can be varied by means of a traversing mechanism, see also Figure 4.1. This mechanism makes it possible to move the probe from the bottom to the top of the experiment section by means of a stepper motor.

Chapter 5

Signal analysis

In this chapter a description of the digital signal analysis is given. The measurement of turbulence quantities involves the estimation of statistical properties of a time-dependent signal. Because the signal becomes discrete (in time and signal value) after sampling, the question is how the discretization will effect the estimate. Another, related, effect will be due to the usage of discrete calibration tables. The used hot-wire velocity probe is sensitive to two perpendicular velocity components. It is most sensitive to the component u' in the direction of the mean flow, the contribution of the v' component being of the order of $(v')^2/\bar{U}$, where $\bar{U}$ is the mean flow velocity. We will discuss the effect of this rectification principle on measured turbulence characteristics.

5.1 Turbulence flow characteristic quantities

Turbulence flow is characterized by the following quantities. We use $\langle\rangle$ to denote ensemble averages. It is assumed that those can be approximated by a time average. The *mean velocity* $\bar{U}$ is thus defined as

$$\bar{U} = \langle u(t) \rangle \,. \tag{5.1}$$

Due to stationarity of the flow the result will not depend on t. The *root-mean-square velocity* σ_u is defined as

$$\sigma_u = \left\langle (u(t) - \bar{U})^2 \right\rangle^{\frac{1}{2}} \tag{5.2}$$

and the *mean energy dissipation* $\langle\epsilon\rangle$,see Eq. (2.2),

$$\langle\epsilon\rangle = 15\nu\bar{U}^{-2} \left\langle \left(\frac{du}{dt}\right)^2 \right\rangle . \tag{5.3}$$

From these quantities we can calculate; the *Kolmogorov lenght* η,

$$\eta = \left(\frac{\nu^3}{\langle\epsilon\rangle}\right)^{\frac{1}{4}} , \tag{5.4}$$

the *kolomogorov velocity* v_k,

$$v_K = (\nu\langle\epsilon\rangle)^{\frac{1}{4}}, \tag{5.5}$$

the *kolmogorov time-scale* t_K,

$$t_K = \left(\frac{\nu}{\langle\epsilon\rangle}\right)^{\frac{1}{2}}, \tag{5.6}$$

the *Kolmogorov frequency* f_K,

$$f_K = \frac{\bar{U}}{2\pi\eta}, \tag{5.7}$$

the *internal length scale* λ (= the Taylor micro scale),

$$\lambda = \bar{U}\sigma_u \left\langle \left(\frac{du}{dt}\right)^2 \right\rangle^{-\frac{1}{2}}, \tag{5.8}$$

and the *internal Reynolds number* Re_λ based on the Taylor micro scale,

$$Re_\lambda = \frac{\lambda\sigma_u}{\nu}. \tag{5.9}$$

These quantities are neccesary to characterize the flow in the windtunnel. An important quantity is thus the *RMS derivative* of the turbulence signal.

5.2 Discrete approximation of signal characteristics

If *u(t)* is the measured velocity, then we need the mean velocity ($\bar{U}$), the RMS velocity (u_{rms}) and the RMS derivative velocity ($\dot{u}_{rms}$) to calculate the turbulence characteristics of the flow. The *mean velocity* is

$$\bar{U} = \langle u(t)\rangle = \frac{1}{T}\int_0^T u(t)dt \tag{5.10}$$

where we have invoked the ergodic theorem that says that the average $\langle\rangle$ can be approximated by a time average. The *RMS velocity* and the *RMS derivative velocity* are

$$u_{rms} = \langle (u(t) - \bar{U})^2\rangle^{\frac{1}{2}} = \left[\frac{1}{T}\int_0^T u^2(t)dt - \bar{U}^2\right]^{\frac{1}{2}}, \tag{5.11}$$

$$\dot{u}_{rms} = \left\langle \left(\frac{du(t)}{dt}\right)^2 \right\rangle^{\frac{1}{2}} = \left[\frac{1}{T}\int_0^T \left(\frac{du(t)}{dt}\right)^2 dt\right]^{\frac{1}{2}} \tag{5.12}$$

These quantities can be extracted from both the time-series data *u(t)* and from the turbulence power spectrum $|u(\omega)|^2$. The companion spectral estimates are, for the *RMS velocity*,

$$u_{rms} = \langle u^2\rangle^{\frac{1}{2}} = \frac{1}{2\pi}\left[\int_{-\infty}^{\infty} |u(\omega)|^2 d\omega\right]^{\frac{1}{2}}, \tag{5.13}$$

and for the *RMS derivative velocity*,

$$\dot{u}_{rms} = \langle \dot{u}^2 \rangle^{\frac{1}{2}} = \frac{1}{2\pi} \left[\int_{-\infty}^{\infty} \omega^2 |u(\omega)|^2 d\omega \right]^{\frac{1}{2}}, \tag{5.14}$$

where $|u(\omega)|^2$ represents the measured turbulence power spectrum and $\omega = 2\pi f$ denotes angular frequency.

A typical turbulence spectrum shows a rapid drop off (as $f^{-\frac{5}{3}}$) to the noise level, equivalent with the K41 theory. It is crucial to understand the way in which various time-discrete approximations are sensitive to noise and allow an adequate discrimination of the true turbulence and noise components of the measured signal.

The mean velocity and the rms velocity can be directly calculated from the time-series data $u(t)$ but are most efficiently calculated from stored histograms of discrete signal values. For the mean and rms values of velocities, a lookup in a discrete table F is needed to convert the discrete voltage signal to discrete velocities. Let us call i the signal voltage value that is discretized with ${}^2log(N+1)$ bits, then

$$\bar{U} = \frac{\sum_{i=0}^{N-1} F(i)P(i)}{\sum_{i=0}^{N-1} P(i)}, \tag{5.15}$$

and the rms velocity $u_{rms} = \sigma_u$ is given by,

$$u_{rms} = \sigma_u = \left(\frac{\sum_{i=0}^{N-1} F(i)^2 P(i)}{\sum_{i=0}^{N-1} P(i)} - \bar{U}^2 \right)^{\frac{1}{2}} \tag{5.16}$$

In section 5.5 differential non-linearity is described which influences the velocity PDFs $P(i)$ from which these quantities are calculated.

5.3 Approximation of derivatives

For calculating the rms derivative velocity ($\dot{u}_{rms}$) we need the first derivative of the velocity signal $u(t)$. We use a numerical differentiating formula of some order n. A second order ($n = 2$) discrete approximation of the time-derivative is given as

$$\tilde{\dot{u}} = \frac{u(t+\Delta t) - u(t-\Delta t)}{2\Delta t}, \tag{5.17}$$

where the tilde denotes approximation. Calculating the rms derivative velocity of the time-series signal then results in the following:

$$\tilde{\dot{u}}_{rms} = \left\langle \left(\frac{u(t+\Delta t) - u(t-\Delta t)}{2\Delta t} \right)^2 \right\rangle^{\frac{1}{2}} \tag{5.18}$$

For the time averaged value of the derivative we use a discrete averaging. For a test let us use a simpel cosinus input signal for the velocity: $u(t) = \cos(\omega t)$. This testsignal has a true rms derivate value of: $\dot{u}_{rms} = \left(\frac{1}{2}\right)^{\frac{1}{2}} \omega$. We can see that this value is depending on the frequency of the input signal. If we look at the $\dot{u}_{rms}$ value of the testsignal in a discrete way, we find:

$$\tilde{\dot{u}}_{rms} = \left[\frac{1}{N} \sum_{i=1}^{N} \left[\frac{\cos((i+1)\omega\Delta t) - \cos((i-1)\omega\Delta t)}{2\Delta t} \right]^2 \right]^{\frac{1}{2}}. \tag{5.19}$$

Working out the square and letting $S = \omega\Delta t$, gives:

$$\tilde{\dot{u}}_{rms} = \left[\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1+\frac{1}{2}\cos((i+1)2S) - 2\cos((i+1)S)\cos((i-1)S) + \frac{1}{2}\cos((i-1)2S)}{4(\Delta t)^2}\right]\right]^{\frac{1}{2}}. \tag{5.20}$$

Using that $-2\cos((i+1)S)\cos((i-1)S) = -(\cos 2iS + \cos 2S)$, some rearranging of the cosinus gives:

$$\tilde{\dot{u}}_{rms} = \left[\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1+\frac{1}{2}\cos((i+1)2S) + \cos((i-1)2S) - \cos(2iS) - \cos(2S)}{4(\Delta t)^2}\right]\right]^{\frac{1}{2}}. \tag{5.21}$$

Then using that $\sum_{i=1}^{N}\cos(i+1)2S = \sum_{k=1}^{N}\cos 2Sk - cos2S$ and $\sum_{i=1}^{N}\cos(i-1)2S = \sum_{k=1}^{N}\cos 2SK + 1$, gives the result:

$$\tilde{\dot{u}}_{rms} = \frac{1}{2\Delta t}\left[1 + \frac{1}{2N}\sum_{k=1}^{N}\cos 2Sk - \frac{1}{2N}\cos 2S + \frac{1}{2N}\sum_{k=1}^{N}\cos 2Sk + \frac{1}{2N} - \frac{1}{N}\sum_{k=1}^{N}\cos 2Sk - \cos 2S\right]^{\frac{1}{2}} \tag{5.22}$$

Striking away the sums in the last equation gives the result:

$$\tilde{\dot{u}}_{rms} = \frac{1}{2\Delta t}\left(\frac{2N+1}{2N}\right)^{\frac{1}{2}}(1-\cos(2S))^{\frac{1}{2}}\ ;\ \text{with } S = \omega\Delta t \tag{5.23}$$

If we approximate the cosinus to the first order with: $\cos(2S) \simeq 1 - 2S^2$ for small S, then it follows with $S = \omega\Delta t$:

$$\tilde{\dot{u}}_{rms} = \omega\left(\frac{1}{2}\right)^{\frac{1}{2}}\left(\frac{2N+1}{2N}\right)^{\frac{1}{2}} \tag{5.24}$$

For $N \to \infty$, this will give: $\dot{u}_{rms} = \omega\left(\frac{1}{2}\right)^{\frac{1}{2}}$, which is the value mentioned earlier for the true rms value of the cosinus testsignal. This tells us that the error in the rms derivative verlocity is depending on N, which is very small. A much more grave error results if S is no longer small. The latter error is catastrophic if $2S = \pi$, or $\Delta t = \frac{\pi}{\omega} = \frac{1}{2f}$, where f is the frequency of the signal. Of course, this Δt would violate the Shannon sampling theorem.

In order to understand these errors better, we will compute the workings of our simple 2nd order differencing formula in the frequency domain. Therefore it is useful to recall the formula for the inverse Fourier Transform

$$u(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega t}u(\omega)d\omega \tag{5.25}$$

and Parseval's identity

$$\langle u^2(t)\rangle = \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{i(\omega_1+\omega_2)t}\langle u(\omega_1)u(\omega_2)\rangle d\omega_2 d\omega_1 \tag{5.26}$$

$$= \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty} E(\omega)d\omega, \tag{5.27}$$

with the energy spectrum $E(\omega) = \langle u(\omega)u(-\omega)\rangle = \langle |u(\omega)|^2\rangle$. From Eq. (5.25) we also see that

$$\langle (\dot{u}(t))^2\rangle = \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\omega^2 E(\omega)d\omega, \tag{5.28}$$

equivalent to Eq. (5.14) mentioned earlier. The working of the differencing operator can be written as a convolution

$$\begin{aligned}\tilde{\dot{u}}(\tau) &= \int H(\tau - t)u(t)dt &(5.29)\\ &= \int \frac{1}{2}\left[\delta(\tau + \Delta t - t) - \delta(\tau - \Delta t - t)\right]u(t)dt &(5.30)\end{aligned}$$

for the case of the simple 2nd order differencing formula. So it follows that:

$$H(t) = \frac{1}{2}\left[\delta(t + \Delta t) - \delta(t - \Delta t)\right]. \tag{5.31}$$

The spectrum of the approximation to the derivative $\tilde{u}$ then is the product of the original spectrum and $H(\omega)$, with,

$$H(\omega) = \frac{1}{2}\left(e^{i\omega\Delta t} - e^{-i\omega\Delta t}\right), \tag{5.32}$$

and

$$\begin{aligned}\left\langle \left(\tilde{\dot{u}}(t)\right)^2\right\rangle &= \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}|H(\omega)|^2 E(\omega)d\omega &(5.33)\\ &= \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\frac{\sin^2(\omega\Delta t)}{(\Delta t)^2}E(\omega)d\omega &(5.34)\\ &\simeq \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\omega^2 E(\omega)d\omega \text{ ; for the case that } \omega\Delta t \ll 1. &(5.35)\end{aligned}$$

This means that the ideal differentiation has $H(\omega) = \omega^2$, and the differencing formula is only correct for $\omega\Delta t \ll 1$. We can extent these arguments for higher order differencing formula's. Figure 5.1(a) shows formula's of order 2 (Eq. (5.17)), 4 and 6 in the spectral domain. Figure 5.1(b) shows what happens if we multiply these spectral representations of the differencing formula's with the turbulence power spectrum $E(\omega)$. The turbulence signal is measured with a sampling frequency of 40 KHz and with an ADC resolution of 12 bits. As can be seen from Figure 5.1(b) the grid generated turbulence signal stops at about 10KHz and the higher frequency region can only be accounted for by noise from the analog to digital converter and other electronic equipment in the experimental setup. On first sight it seems that a simple 2nd order differencing formula is less accurate then the higher order versions if compared with the ideal case of ω^2. But because of the fact that the measured turbulence signal has (unwanted) noise in the higher frequency range when sampled with a sampling frequency of 40KHz, a 2nd order differencing formula can be used as a filter, to eliminate the unwanted noise. It seems paradoxal that a lower order differencing formula gives a more accurate result than a higher order formula, but this is only valid when measuring with a high sample frequency such as 40KHz. If a lower sample frequency, for example 20 KHz, is used then the opposite is valid; a higher order differencing formula gives a more accurate result.

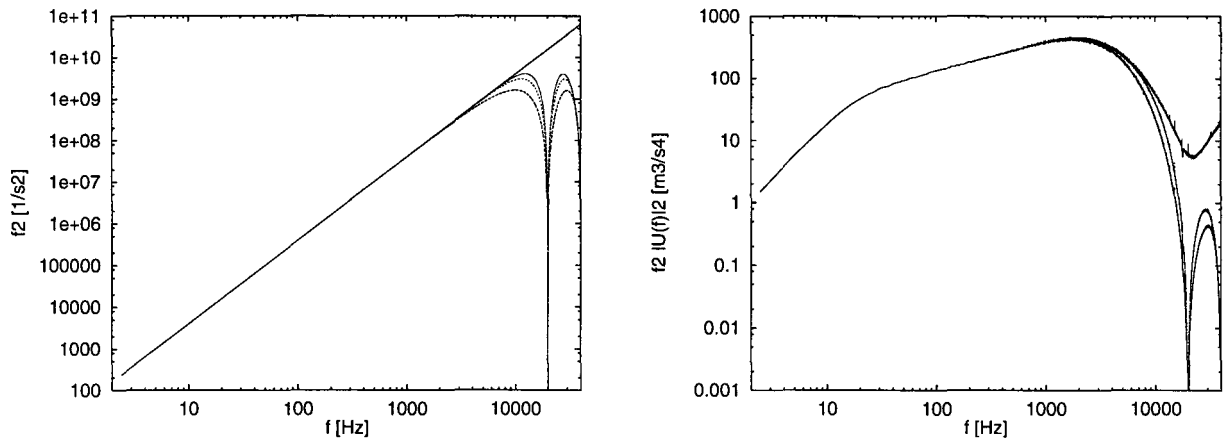


Figure 5.1: *Representation of the discrete differencing formula in the spectral domain. (a) The 2nd, 4th and 6th order differencing formulas are compared with the ideal case of ω^2, lower, 2nd, 3th and upper curve respectively. (b) The same spectral representations of the 2nd and 4th order differencing formula's multiplied with the measured turbulence spectrum, compared with the exact solution.*

5.4 Spectrum, Fast Fourier Transform and windowing

In appendix B.1 the C program *spectrum.c* is given which is used to calculate the turbulence characteristic quantities and the energy spectrum of the turbulent flow. The calculation of the energy spectrum requires the use of a Discrete Fourier Transform (DFT) to transform the sampled velocity time-blocks to the frequency domain.

5.4.1 Fast Fourier Transform

When a signal in the frequency domain has to be processed by digital components such as the Digital Signal Processor, the signal has to be sampled similar to the sampling of signals in the time-domain. Sampling in the time-domain results in a periodic signal in the frequency domain, likewise a sampled signal in the frequency domain results in a periodic signal in the time domain. Therefore the sampling of a signal in the frequency domain is only usefull for signals that are limited in time. To derive the Discrete Fourier Transform we start of by recalling again the standard Fourier Transform:

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt. \tag{5.36}$$

The standard Fourier Transform of an analog signal $x(t)$ determines the associated Fourier Spectrum $X(\omega)$. As mentioned above, both signals $X(\omega)$ and $x(t)$ have to be sampled to enable digital processing. If the input signal $x(t)$ is sampled the Fourier Transform of the sampled signal $x(n)$ takes the form:

$$X(e^{i\omega}) = \sum_{-\infty}^{\infty} x(n) e^{-i\omega n}. \tag{5.37}$$

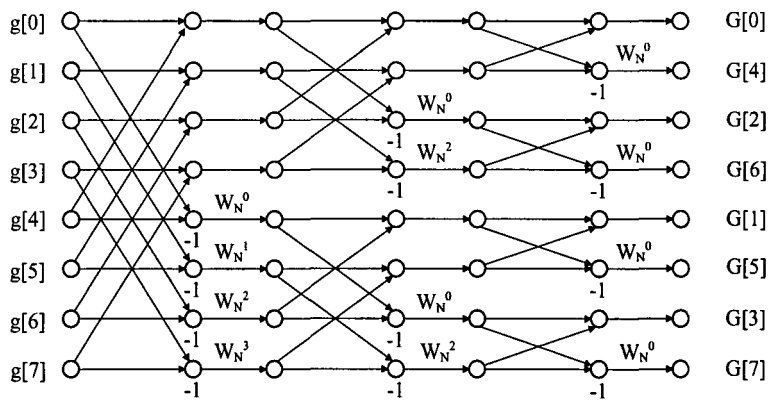


Figure 5.2: *Schematic overview of a 8-point Fast Fourier Transform, the socalled butterfly. The output array G[i] is bit-reversed.*

If this Fourier Transform is calculated at discrete points in the ω-domain it will result in the Discrete Fourier Transform (DFT):

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \tag{5.38}$$

where $k, n = 0, 1, \cdots, N-1$ and W_N^{nk} is determined by:

$$W_N^{nk} = e^{-\frac{i2\pi nk}{N}}. \tag{5.39}$$

If this Discrete Fourier Transform is calculated directly the complexity of the algorithm is of order

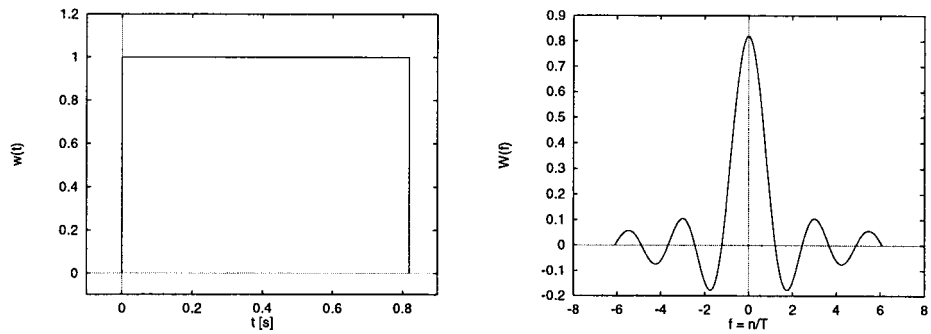
$$\mathcal{O}(C) = N^2. \tag{5.40}$$

When the number of points used to calculate the DFT increases, the number of operations that are to be performed increases quadratically, hence the Discrete Fourier Transform will become extremely inefficient for large N. To overcome the computational problems of the DFT a much more efficient algorithm, called the Fast Fourier Transform, has been developed.

The definition of the Fast Fourier Transform (FFT) is identical to the definition of the Discrete Fourier Tranform. The only difference can be found in the algorithm used for calculating the spectrum $X(k)$. When the DFT series, given by Eq. (5.38), is expanded it can be seen that identical *twiddle factors* (W_N^{nk}) occur. Through smart bookkeeping, Eq. (5.38) can be calculated as indicated in Figure 5.2. The complexity of the FFT algorithm is given by

$$\mathcal{O}(C) = N log_2(N), \tag{5.41}$$

therefore the FFT algorithm is much more efficient then a direct calculation of the DFT for large N, [3]. The FFT algorithm depicted in figure 5.2 results in a *bit reversed* output signal $X(k)$. *Bit reversed* output means that the indexes of the array representing the samples in the frequency domain are bit reversed, that is, the binairy representation of the indexes have to be reversed to produce the right sequence of the array in the frequency domain. The TMS320C40 Digital Signal Processor has the ability to address arrays in a bit reversed manner, so no time is lost in converting the bit reversed output into the correct output.



(a) (b)

Figure 5.3: *A rectangular analysis window. (a) A time window. (b) A spectral window.*

5.4.2 Windowing

Analysis of a time-history record of a finite time, T, will result in *spectral leakage*, and windowing techniques can be applied to minimize this phenomenon, [5]. The transformation of a time-history signal of a finite duration T to the frequency domain, results in *spectral leakage* in the frequency domain. A time-history record, $x_k(t)$, specified for a finite time T, can be viewed mathematically as an unlimited time-history record, $x(t)$, viewed through a rectangular window, $w(t)$, with

$$w(t) = \begin{cases} 1 & ; \text{for } 0 \leq t \leq T \\ 0 & ; \text{otherwise} \end{cases} \tag{5.42}$$

Consequently, for the complete time interval $(-\infty, \infty)$

$$x(t)w(t) = \begin{cases} x_k(t) & ; \text{for } 0 \leq t \leq T \\ 0 & ; \text{otherwise.} \end{cases} \tag{5.43}$$

The finite Fourier transform of the function $x(t)w(t)$ is again given by:

$$X(f,T) = \int_{-\infty}^{\infty} x(t)w(t)e^{-i2\pi ft}dt. \tag{5.44}$$

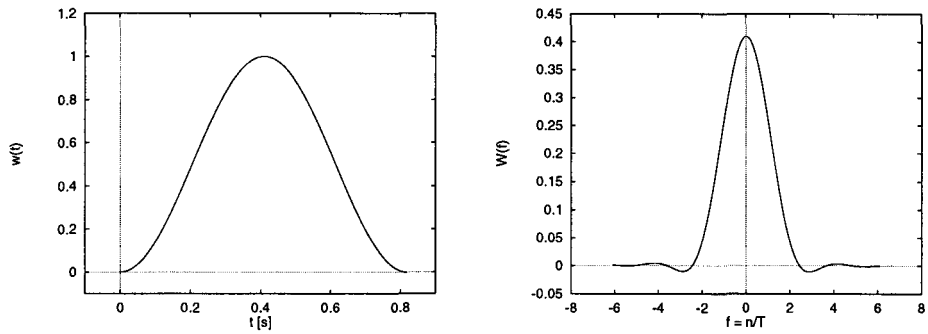
The convolution theorem states that the Fourier Transform of a product of two functions is equivalent to the transform of one of the functions convolved with the transform of the other; that is

$$X(f,T) = \int_{-\infty}^{\infty} X(\xi)W(f-\xi)d\xi. \tag{5.45}$$

For the rectangular function, $w(t)$ defined by Eq. (5.42), the Fourier transform, $W(f)$, is given by

$$W(f) = T\frac{\sin \pi fT}{\pi fT} \tag{5.46}$$

A plot of $W(f)$ is depicted in Figure 5.3. The large side lobes of $W(f)$ allow leakage of power at frequencies which are well separated from the main lobe of the spectral window, and this



(a) (b)

Figure 5.4: *A Hanning analysis window. (a) A time window. (b) A spectral window.*

may introduce a significant distortion of the estimate spectrum. If it is considered necessary to suppress the leakage problem then it is common practice to introduce a data window that tapers the time-history data to eliminate the discontinuities at the beginning and the end of the record being analysed. The window used in the program *spectrum.c* is called a *Hanning window* or also called the *cosine squared window.* It is defined as

$$w_H(t) = \begin{cases} \frac{1}{2}\left[1 - \cos\left(\frac{2\pi t}{T}\right)\right] = 1 - \cos^2\left(\frac{\pi t}{T}\right) & ; \text{for } 0 \leq t \leq T \\ 0 & ; \text{otherwise.} \end{cases} \tag{5.47}$$

Figure 5.4 shows the hanning window in the time and in the frequency domain. The Fourier Transform of Eq. (5.47) is

$$W_H(f) = \frac{1}{2}W(f) + \frac{1}{4}W(f - \frac{1}{T}) + \frac{1}{4}W(f + \frac{1}{T}), \tag{5.48}$$

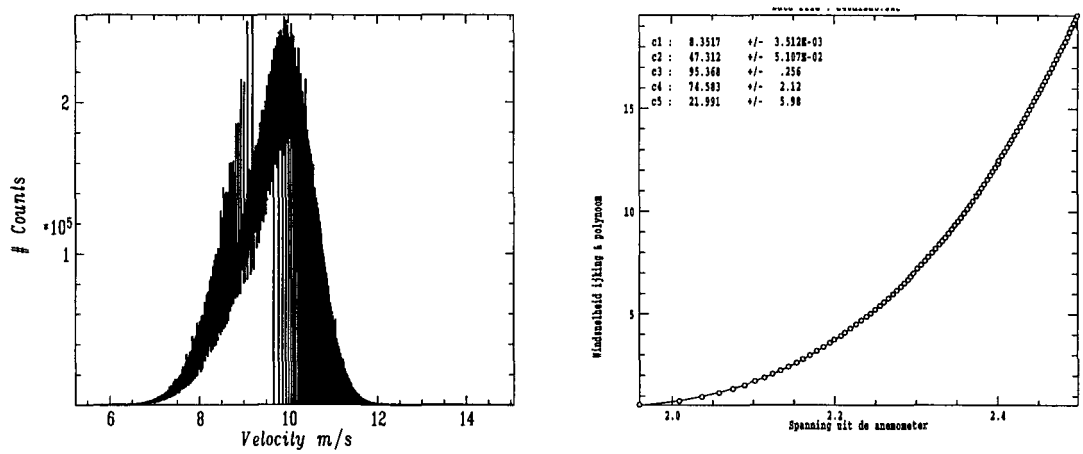
where $W(f)$ is defined as in Eq. (5.46).

Comparing Figure 5.3 and 5.4 it can be observed that the Hanning window has smaller side lobes and a larger bandwidth for the main lobe than the rectangular window. In general, the use of any tapering operation to suppress side-lobe leakage will increase the bandwidth of the main lobe of the spectral window in an autospectral density analysis.

Finally, if a tapered window, $w(t)$, is used in Eq. (5.44) to evaluate a spectral estimate then a loss factor must be introduced. As discussed in [5], when a Hanning window is used then a scale factor of $(8/3)^{1/2}$ should be introduced to obtain the correct magnitude of the spectral density estimate.

5.5 Differential non-linearity

A point of concern is the differential non-linearity in the process of converting analog voltages to digital velocities. Figure 5.5a shows the dramatic effect of differential non-linearity on measured velocity probability functions. The PDF is the result of a very long experimental integration time, and it does not change when averaging longer. Therefore, the noise in the PDF is not statistical and is entirely due to differential non-linearity.



(a) (b)

Figure 5.5: *(a) Velocity probability distribution function as measured in the windtunnel. The distribution function shows large and small spikes and dips which are a result of differential non-linearity. (b) Example of a hot-wire probe calibration curve. The curve represents the relation between the measured voltage and the windvelocity in the windtunnel. The curve has clearly a non-linear behavior.*

There are two sources of non-linearity present. The first is the differential non-linearity of the analog-to-digital converter (ADC). The second source is the non-linear integer transformation of digital voltages to digital velocities. The first source is by far outweighted by the second source of differential non-linearity.

5.5.1 Influence of non-linear calibration

The transformation of digital voltages to digital velocities is determined by the calibration procedure that precedes each experiment. The calibration determines the relation between voltages v and velocities u. This relation is a fourth order polynomial $u = f(v)$ (Eq. (4.9) and it is used to generate a look-up table that maps the 12-bits voltages from the ADC to 12-bits velocities. In Figure 5.5b we can see that the calibration curve is clearly non-linear.

This non-linearity causes the effect that different voltages representing different velocities are mapped on the same integer velocity value whereas other integer velocity values do not correspond to any binary voltage. As a consequence, a measured PDF (histogram) of velocities will show large spikes at the velocity values that correspond to two integer voltages and dips at the skipped velocity values.

However, this effect can be minimised because we know exactly which binary codes in the calibration table are double and which codes are missing. With this information we can make a smoothing procedure to smooth the histogram of recorded velocities without changing the statistical properties of the signal we want to evaluate. The recept for smoothing the histogram of recorded velicities is as follows: Let Δj be the difference between two successive velocity codes in the calibration table. If $u = f(v)$ is the relation between analog voltage and velocity, then Δj is:

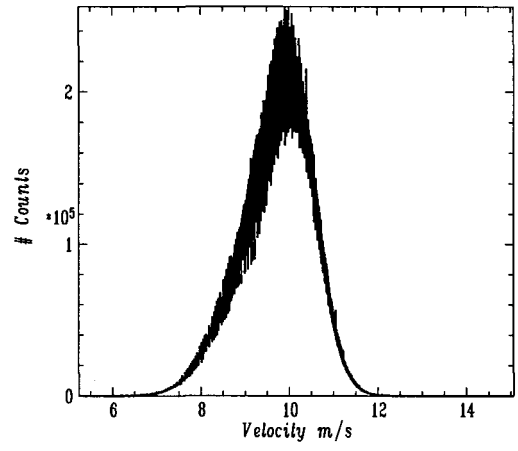


Figure 5.6: *Smoothed velocity probability distribution function. The effects of the non-linear calibration table are removed, but the effects of the ADC differential non-lineariry are still present.*

$$\Delta j(i) = \left[4096 f\left(\frac{i+1}{4096}\right)\right] - \left[4096 f\left(\frac{i}{4096}\right)\right], \tag{5.49}$$

where $i \in [0 \ldots 4095]$ is the ADC voltage code. If we calculate Δj for each $i \in [0 \ldots 4095]$ then the actions to smooth the histogram are:

$$\begin{cases} \Delta j(i) = 1 & ; \text{do nothing} \\ \Delta j(i) = 0 & ; \text{divide by two and add extra bin} \\ \Delta j(i) = 2 & ; \text{skip, no information is lost} \end{cases}$$

This recept will only remove the effects of the non-linear calibration table. The effects of the differential non-linearity caused by the ADC are still present and this will be topic of the next subsection. Figure 5.6 shows the smoothed histogram which is the same measurement as in Figure 5.5a

5.5.2 ADC differential non-linearity

In an ideal ADC, code transitions are 1 LSB (Least Significant Bit) apart. Differential non-linearity (DNL) is the maximum deviation expressed in LSB's, from this ideal value. It is often specified in terms of the maximum number of bits for which no missing codes (NMC) are guaranteed.

As can be seen from Figure 5.6 the effect of differential non-linearity which remains after the removal of the effects of the non-linear calibration curve, is still reasonably high. In the used ADC-card from Transtech (TDM431) there are anolog to digital converters from Burr-Brown (type ADS7810). Burr-Brown gives for the differential non-linearity error a maximum of ± 1 LSB.

Could this error cause such an effect on the velocity PDFs? To answer this question we try to find out what the effect of the differential non-linearity is on the mean of the velocity PDF curve. A second order polynomial fit to a normalised PDF is shown in figure 5.7a, and the residues of this polynomial fit are shown in figure 5.7b. The PDF is normalised such that $\sum_{i=0}^{N-1} P(i) = 1$. Burr-Brown, the manufacturer of the ADCs, gives the following

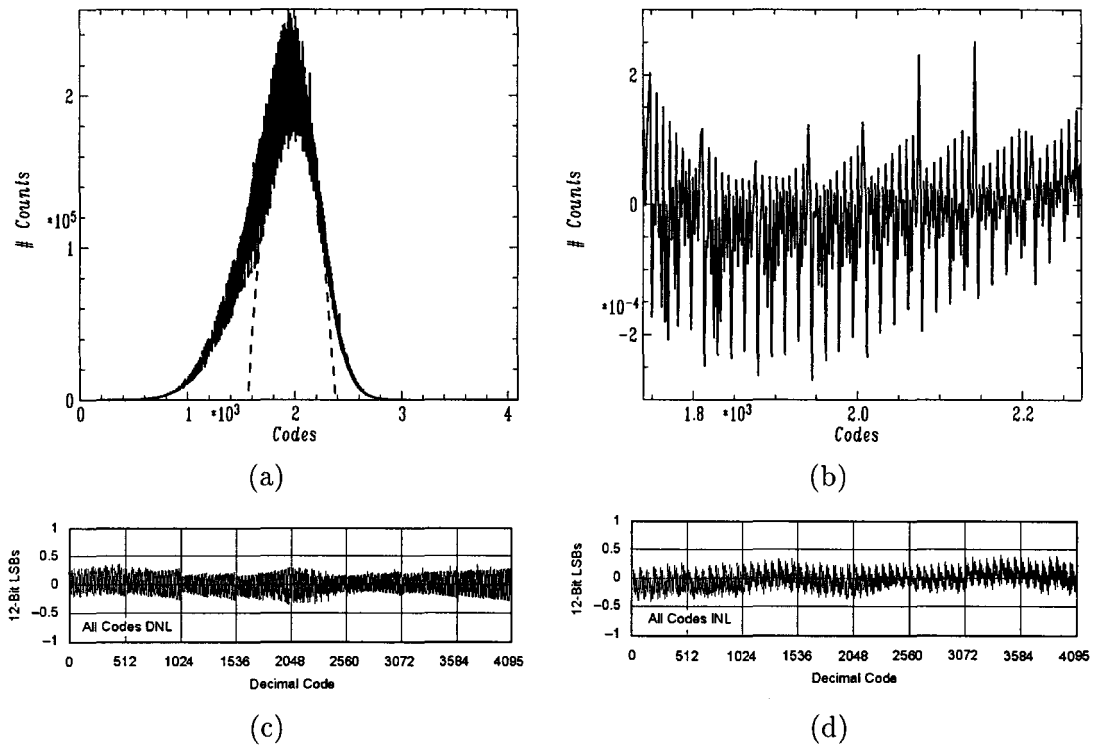


Figure 5.7: *(a) Velocity PDF nomalised such that* $\sum_{i=0}^{N-1} P(i) = 1$ *and a second order polynomial fit. (b) Residues of the second order polynomial fit of the normalised PDF. (c) Data from Burr-Brown ADC7810 data-sheet, ADC differential non-linearity (DNL). (d) ADS7810 integral non-linearity (INL)*

specifications for the differential non-linearity (DNL), see Figure 5.7c. This DNL is measured by Burr-Brown by applying a full-scale sine wave to the ADC and taking a large number of samples (several million for a 12-bit converter). The number of occurrences of each code is recorded on a histogram plot. The data is normalized by comparison with the U-shaped ideal probability density distribution of a sine wave. From it, the DNL in Figure 5.7c can be plotted, and integral nonlinearity can be determined by compiling a cumulative histogram. The INL is shown in Figure 5.7d.

As can be seen from comparison from both figures 5.7b and c the differential non-linearity is comparable in nature. The ADS7810 is an Analog-to-Digital converter of the succesive approximation type, which is a bad performer where DNL is of interest. For more accurate determination of velocity PDFs, it should be considered to use another type of ADC, see also section 7.2.2 for some recommendations.

5.5.3 Influence on velocity difference PDFs

When measuring the structure functions we are not interested in the velocity PDFs, but in the *velocity difference* PDFs. The question arises how the differential non-linearity affects these PDFs of Δu. In ref. [11] it was shown that apart from a series of peaks at discrete Δu, almost all the error caused by the differential non-linearity is piled up at $\Delta u = 0$. Figure 5.8 shows a velocity difference PDF that has experimentally been determined with the Structurator from

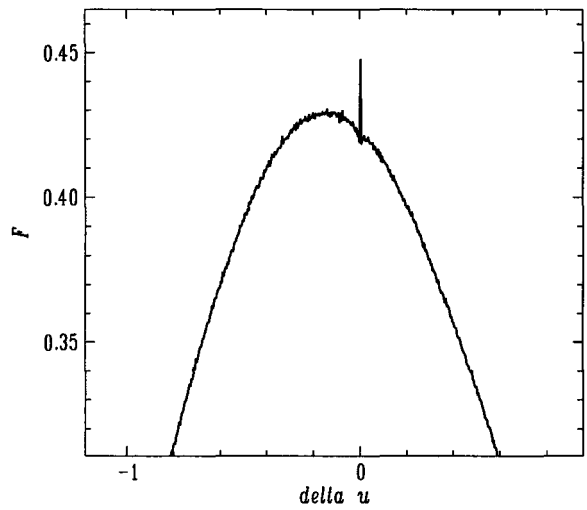


Figure 5.8: *Measured velocity difference distribution function after taking* 10^9 *samples. Shown is the enlarged section around* $\Delta u = 0$.

a turbulent signal. Shown is the range around $\Delta u = 0$ and from the figure it can clearly be seen that there is a large peak (compared with the rest of the PDF) at the point $\Delta u = 0$.

The effect becomes insignificant in the tails of the PDF, since in an experimentally determined PDF the tails suffer most from statistical noise. When we are interested in high order structure functions then the error caused by the differential non-linearity is not important, because these are determined by the tails of the PDF. But if low-order structure functions are to be measured then the differential non-linearity will no longer be an unsignificant error.

5.6 The rectification problem

When using a single hot-wire probe for the measurement of the mean and fluctuating turbulent velocity component in *one* direction, namely in the direction of the mean flow, a type of error can occur which is usually referred to as the *rectification* error. Due to the fact that the hot-wire has rotational symmetry, it is sensitive both for velocity pertubations in the transversal direction (i.e. perpendicular to the main flow) as for pertubations in the direction of the mean flow. Since a single cylindrical hot-wire is also sensitive for flow in the u' direction as for flow in the v' direction this means that an error can be made, see Figure 5.9.

This error is usually ignored in the recent turbulence literature. However, the suspicion arises that this error could effect the value of measured scaling exponents. Therefore it is important to try to quantify this error.

In the section covering the calibration of the hot-wire probes (section 4.3) it was assumed that only components perpendicular to the hot-wire affect its cooling. This means that the z-component w' does not contribute to the cooling of the wire. Thus, when the x-component u' is the turbulent velocity component in the direction of the main flow $\bar{U}$, and v' is the lateral turbulent velocity component perpendicular to the wire in the y-direction, we can write, conform Eq. (4.5),

$$\tilde{U} = \left((\bar{U} + u')^2 + v'^2\right)^{\frac{1}{2}}, \tag{5.50}$$

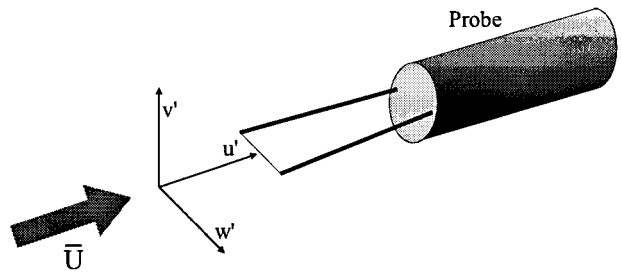


Figure 5.9: *Illustration of the hot-wire probe with the flow components u', v' and w' in the x, y and z direction respectivelly.*

where $\tilde{U}$ denotes the measured resultant velocity and u' and v' are turbulent velocity fluctuations in the x and y direction. It was shown in section 4.3 that to the first order the hot-wire probe was only sensitive to the component u' in the direction of the mean flow. But, is this really the case? First we take a look at the influence on the energy spectrum and after that we try to investigate the influence on the velocity PDF.

5.6.1 Influence on the energy spectrum

Lets consider Eq. (5.50) and rewrite it in the following form,

$$\tilde{U}(t) = \left((\bar{U} + u'(t))^2 + (v'(t))^2\right)^{\frac{1}{2}} \tag{5.51}$$

$$= (\bar{U} + u'(t)) \left(1 + \frac{(v'(t))^2}{(\bar{U} + u'(t))^2}\right)^{\frac{1}{2}}. \tag{5.52}$$

Then we take the first order expansion of the last term,

$$\tilde{U}(t) \approx (\bar{U} + u'(t)) \left(1 + \frac{(v'(t))^2}{2\bar{U}^2}\right), \tag{5.53}$$

then the measured resultant velocity is,

$$\tilde{U}(t) = \bar{U} + u'(t) + \frac{(v'(t))^2}{2\bar{U}} + \mathcal{O}\left(\frac{u'^3, v'^3}{\bar{U}^2}\right). \tag{5.54}$$

The Fourier spectrum of the measured velocity is,

$$\tilde{U}(\omega) = \int_{-\infty}^{\infty} \tilde{U}(t) e^{-i\omega t} dt \tag{5.55}$$

$$= \int_{-\infty}^{\infty} \left[\bar{U} + u'(t) + \frac{(v'(t))^2}{2\bar{U}}\right] e^{-i\omega t} dt \tag{5.56}$$

$$= u'(\omega) + \frac{1}{2\bar{U}} \int_{-\infty}^{\infty} (v'(t))^2 e^{-i\omega t} dt \tag{5.57}$$

For the last term in Eq. (5.57) we take a look at the convolution, which is defined as,

$$\mathcal{F}(f * g) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(p) g(x-p) e^{-i\omega x} dp dx, \tag{5.58}$$

where $\mathcal{F}$ denotes the Fourier transform. Using Eq. (5.58) in Eq. (5.57) gives the following result for the measured energy spectrum

$$\tilde{U}(\omega) = u'(\omega) + \frac{1}{2\bar{U}} \frac{1}{2\pi} \int_{-\infty}^{\infty} v'(\omega') v'(\omega - \omega') d\omega'. \tag{5.59}$$

The modified energy spectrum then becomes,

$$\begin{aligned}
\tilde{E}(\omega) &= \langle \left| \tilde{U}(\omega) \right|^2 \rangle & (5.60)\\
&= \langle \left| u'(\omega) + \frac{1}{4\pi\bar{U}} \int_{-\infty}^{\infty} v'(\omega') v'(\omega - \omega') d\omega' \right|^2 \rangle & (5.61)\\
&= \langle |u'(\omega)|^2 \rangle + \frac{2}{4\pi\bar{U}} \mathrm{Re} \left\{ u'(\omega) \int_{-\infty}^{\infty} d\omega' v'(\omega') v'(\omega - \omega') d\omega' \right\} + \cdots & (5.62)
\end{aligned}$$

Clearly, the energy spectrum obtains an *extra term* which can be written in terms of 3-point correlations between u' and v',

$$\frac{1}{2\pi\bar{U}} \mathrm{Re} \int_{-\infty}^{\infty} d\omega' \langle u'(\omega) v'(\omega') v'(\omega - \omega') \rangle \tag{5.63}$$

It is very hard to say something about this quantity, because it envolves 3-point correlations between velocities of which the relations are unknown. In order to further quantify this influence on the scaling of the spectrum more experimental work is necessary.

5.6.2 Influence on velocity PDFs

In Figure 5.10 an illustration is given of the velocity probability distribution function (velocity PDF) as is measured in grid turbulence. Figure 5.10a is given with linear axes and Figure 5.10b with a logarithmic scale of the y-axis. In figure 5.10b it can be seen that the velocity PDF is non-symmetrical.

We take a look at the influence of the rectification error on the velocity PDFs and try to investigate if the measured non-symmetrical behavior of these PDFs can be explained in this way. The measured velocity is defined according to Eq. (5.50)

$$\tilde{U}(t) = \left((\bar{U} + u'(t))^2 + v'(t)^2 \right)^{\frac{1}{2}} = f(u', v'). \tag{5.64}$$

The probability distribution function of the turbulence flow velocity components are for the u' and v' direction defined as $P(u')$ and $P(v')$. We want to investigate the PDF of $P(\tilde{U})$,

$$P(\tilde{U}) = \int_{-\infty}^{\infty} \delta(\tilde{U} - f(u', v')) P(u', v') du' dv'. \tag{5.65}$$

If we make the assumption that $P(u')$ and $P(v')$ are independent, then $P(u', v') = P(u')P(v')$.

Let's first look at a simplified case, in which the measured velocity is only depending on *one* variable; $\tilde{U} = f(u')$, then according to the fundamental transformation law of probabilities the probability distribution of $\tilde{U}$, denoted by $P(\tilde{U})d\tilde{U}$, is given by

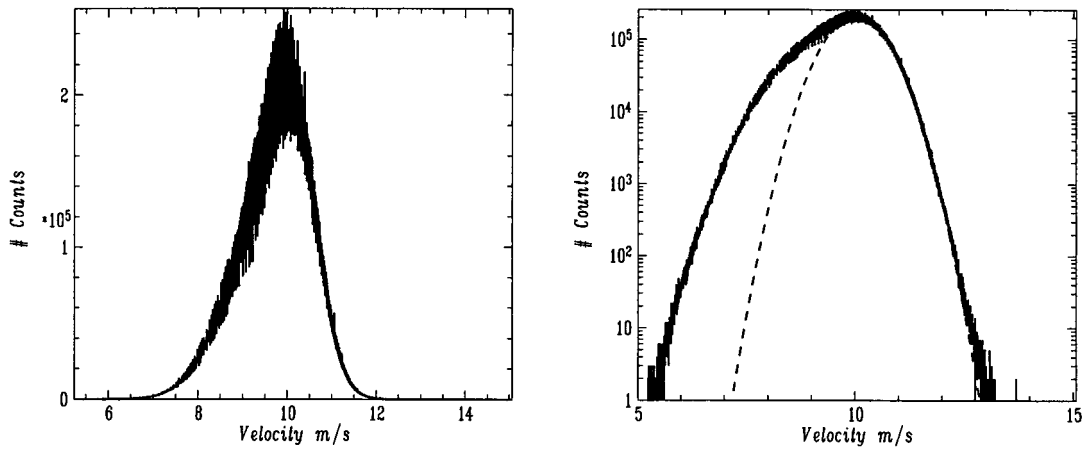


Figure 5.10: *Measured velocity probability distribution function in the windtunnel. (a) Velocity PDF with linear axes. (b) Same velocity PDF but with logarithmic scale of the y-axis; the PDF is clearly non-symmetrical.*

$$|P(\tilde{U})d\tilde{U}| = |P(u')du'|, \tag{5.66}$$

or

$$P(\tilde{U}) = P(u')\left|\frac{du'}{d\tilde{U}}\right|, \tag{5.67}$$

and the following relation holds

$$P(\tilde{U}) = \int_{-\infty}^{\infty} \delta(\tilde{U} - f(u'))P(u')du'. \tag{5.68}$$

The expression for $\tilde{U} = f(u')$ can be expanded into a Taylor series to the first order around the point $u' = \tilde{U}$

$$f(u') = \tilde{U} + (u' - \tilde{U})\left.\frac{df(u')}{du'}\right|_{\tilde{U}} + \cdots. \tag{5.69}$$

Inserting Eq. (5.69) into Eq. (5.68), gives the following relation for the probability distribution of $\tilde{U}$,

$$P(\tilde{U}) = \int_{-\infty}^{\infty} \delta\left((\tilde{U} - u')\left.\frac{df(u')}{du'}\right|_{\tilde{U}}\right) P(u')du', \tag{5.70}$$

and using that $\delta(ax) = \frac{1}{|a|}\delta(x)$ in Eq. (5.70) gives

$$P(\tilde{U}) = \int_{-\infty}^{\infty} \frac{\delta(\tilde{U} - u')}{\left|\frac{df(u')}{du'}\right|_{\tilde{U}}} P(u')du'. \tag{5.71}$$

Taking the inverse of the relation $\tilde{U} = f(u')$ gives the relation of $u' = f^{-1}(\tilde{U})$ and will lead to the following expression for the probability distribution of the measured velocity $\tilde{U}$,

$$P(\tilde{U}) = \frac{1}{\left|\frac{df(u')}{du'}\right|_{\tilde{U}}} P\left(f^{-1}(\tilde{U})\right). \tag{5.72}$$

Now, we do the same for the *two* variable situation of Eq. (5.65) under the assumption that the probability distributions for u' and v' are independent so that $P(u', v') = P(u')P(v')$,

$$P(\tilde{U}) = \int_{-\infty}^{\infty} \delta(\tilde{U} - f(u', v'))P(u')P(v')du'dv' \tag{5.73}$$

$$= \int_{-\infty}^{\infty} \frac{\delta(\tilde{U} - u')}{\left|\frac{\partial f(u', v')}{\partial u'}\right|_{\tilde{U}}} P(u')P(v')dv', \tag{5.74}$$

with,

$$f(u', v') = \left((\bar{U} + u')^2 + (v')^2\right)^{\frac{1}{2}} = \tilde{U}, \tag{5.75}$$

which can be rewritten for u',

$$u' = -\bar{U} \pm \sqrt{\tilde{U} + (v')^2}, \tag{5.76}$$

which has two solutions for u'. If we substitute this in the same manner as is done with the one variable situation, then we get,

$$P(\tilde{U}) = \frac{1}{\left|\frac{\partial(u', v')}{\partial u'}\right|} P(\pm\sqrt{\tilde{U} + (v')^2} - \bar{U})P(v') \tag{5.77}$$

Equation (5.77), which is the probability distribution of the measured velocities, is composed of *two* probability distributions which result from the velocity pertubations u' and v' in the parallel and in the lateral direction relative to the mean velocity of the flow.

In this way the non-symmetrical behavior of the velocity PDF could be explained, but it is not decisive. It is very hard to say something about the total distribution function of the velocities. The assumption that the velocity distributions $P(u')$ and $P(v')$ are independent can also be questioned.

Further research should be carried out to investigate this rectification error more. An idea might be to compare a single wire probe and an X-probe or multiple probes in 3 perpendicular directions.

Chapter 6

Experimental results

In this section some experimental results are given. The experiments were done in the windtunnel and the velocities were measured with the hot-wire anemometer and the signal conditioning unit, as described in chapter 4. The experiments were mainly done to check the new instrumentation that was developed in this project. The results agree with those of earlier experiments. The improved statistics of energy spectra, however, led to verification of 'a novel' phenomenon. Presented here are: *turbulence energy spectra*, *velocity PDFs*, *velocity difference PDFs* and *structure functions*.

6.1 Turbulence energy spectrum

Figure 6.1 shows a measured energy spectrum of turbulence flow generated with a grid in the windtunnel.

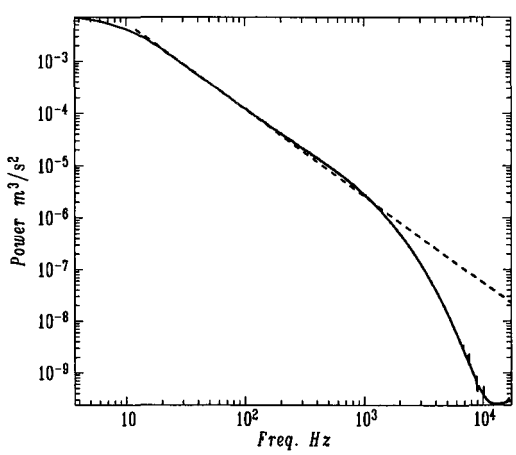


Figure 6.1: *Measured energy spectrum of turbulence generated with a grid, compared with the* $-\frac{5}{3}$ *Kolmogorov energy spectrum.*

When compared with the Kolmogorov minus five-third law (K41), see section 2.1.1, we see clearly that the law holds for a certain inertial range. The inertial range spans a frequency range from approximately 20 till 1000 Hz, until the energy drops off due to viscous dissipation. The energy spectrum stops at approximately 10 KHz and any signal beyond that frequency is

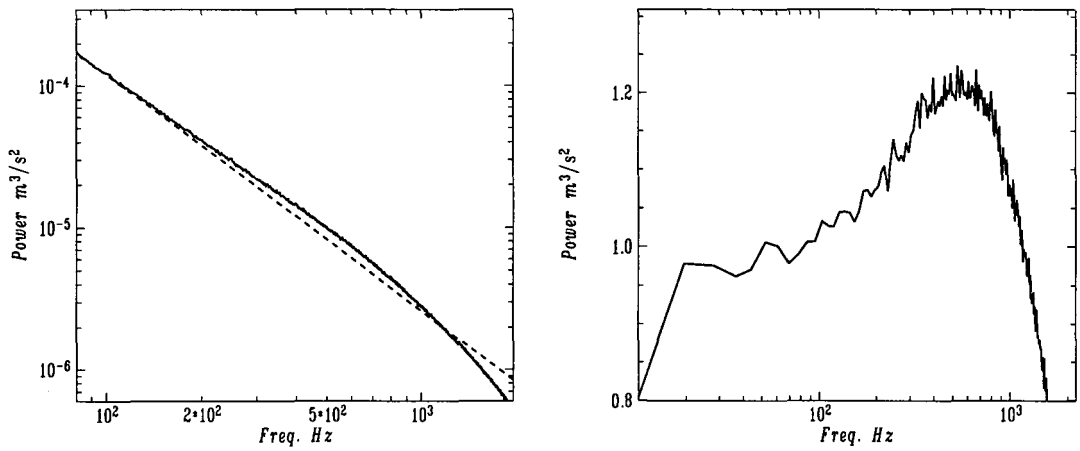


Figure 6.2: *Measured energy spectrum of turbulence generated with a grid. (a) Enlarged section of Fig. 6.1, showing clearly the energy pileup when compared with the classical* $-\frac{5}{3}$ *energy spectrum. (c) Residu plot of the spectrum when compared with the classical* $-\frac{5}{3}$ *slope.*

ADC noise due to LSB fluctuations. The total turbulence energy range relative to this ADC noise spans almost 8 decades which is a good performance for a 12-bit ADC.

In section 2.4 the Batchelor parametrisation was discussed and its related Batchelor energy spectrum. It was mentioned that the Batchelor energy spectrum has a so-called energy pileup region just at the end of the scaling range before the energy drops off due to viscous dissipation. This energy pileup is ascribed to the bottleneck phenomenon and is most clearly seen in the energy spectrum of fully developed turbulence.

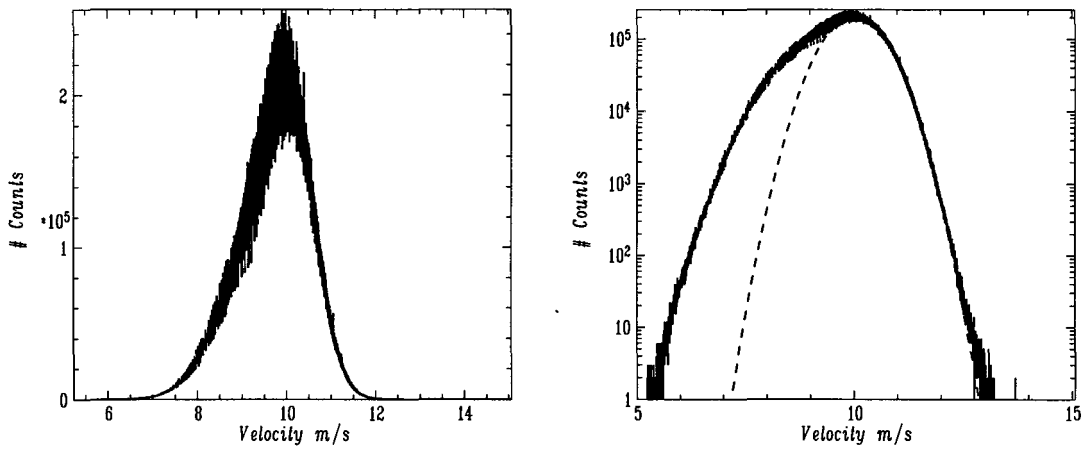
From comparison with the experimentally measured energy-spectra of Figure 6.1 it can be seen that the energy pileup is a physical phenomenon. Figure 6.2a shows the same energy pileup, as is predicted by the Batchelor energy spectrum. Figure 6.2b shows the residues of the $-\frac{5}{3}$ fit to the energy spectrum. The maximum of the pileup occurs at approximately 550 Hz.

We are able to measure this energy pileup due to the ability to sample for long periods of time without having to store vast amounts of data. This long integration time allows us to improve the signal-to-noise ratio of the energy spectrum by averaging over a longer period of time, thus removing the noise which is inherent of measured turbulence energy spectra.

6.2 Velocity PDF

The velocity probability PDF can also be measured for long periods of time without having to store vast amounts of data. From it, several statistical characteristics can be calculated, such as the *mean velocity* and the *rms velocity.* In section 5.5 we discussed the differential non-linearity which affects these velocity PDFs as can be seen in Figure 6.3a.

The rectification problem discussed in section 5.6 could also influence these velocity PDFs and from Figure 6.3b it can be seen that the velocity PDF is clearly non-symmetrical. It even seems to consist out of a summation of two probability distributions as is denoted by



ex

Figure 6.3: *Velocity probability distribution function measured in turbulence generated with a grid. (a) Linear vertical and horizontal scale. (b) Logarithmic vertical scale*

the Gaussian fit in Figure 6.3.

It is not known how these errors in the velocity PDFs affect the calculated statistics from these distribution functions. The rectification error which causes the non-symmetry of the PDF might seem to cause the *mean velocity* to be to high. How the *rms velocity* is influenced is not known.

6.3 Velocity difference PDFs

Research on small-scale structures of turbulence is often done via the scaling behahavior of structure functions. The structure function of order p is the pth-order moment of the distribution functions of velocity differences. Interesting phenomenon may possibly be detected at large p, but the statistical accuracy of high-order moments is a problem. Therefore, also the velocity difference distribution functions are studied directly. This avoids the problems associated with determining high-order moments, [11].

Figure 6.4 shows 4 velocity difference (Δu) probability distribution functions $P(\Delta u)$, measured in real-time using the Structurator routine described in section 3.3. The velocity difference PDFs represent distributions each at different relative distances. The PDFs are normalised such that $\int P(\Delta u)d(\Delta u) = 1$ and $\int P(\Delta u)(\Delta u)^2 d(\Delta u) = 1$. From the figures it can be seen that at small scales $P(\Delta u)$ has wide, almost exponential tails. At large scales $P(\Delta u)$ tends to a Gaussian.

The measured longitudinal distribution functions are asymmetric. They have their maximum at a value $\Delta u < 0$. This asymmetry is a key phenomenon in turbulence; it is related to the cascade of energy from large to small scales and is called the *skewness*.

From the PDFs it can be seen that the tails are not accurate. These tails correspond to rare events of high velocity differences and an even longer sampling time is necessary to improve their statistics. But the sampling time is a compromise between the ability to sample

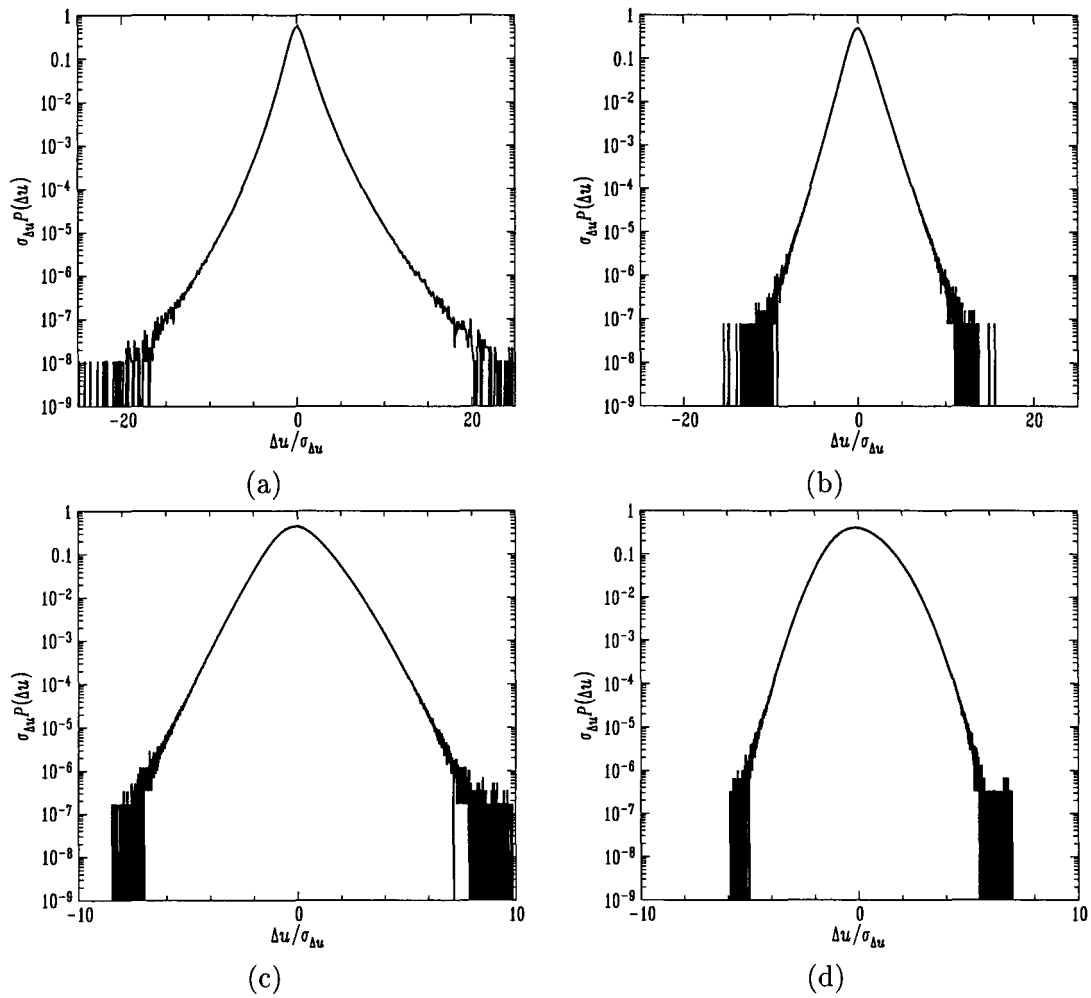
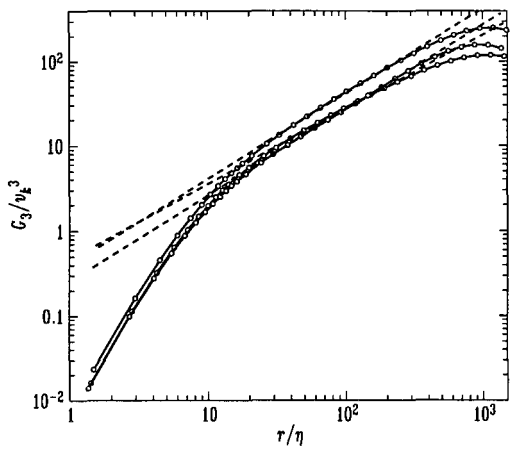


Figure 6.4: *Some velocity difference probability distribution functions measured with the real-time Structurator in turbulence flow. (a) Δu PDF for relative distance $r/\eta = 1.4$. (b) Δu PDF for relative distance $r/\eta = 13.6$. (c) Δu PDF for relative distance $r/\eta = 92.2$. (d) Δu PDF for relative distance $r/\eta = 867.3$.*

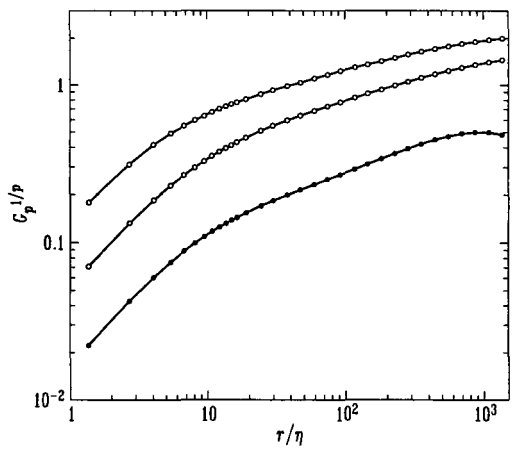
for long periods and the stability of the hot-wire probe. The characteristics of the hot-wire are not stable over a long period of time and recalibration is then necessary, which of course limits the longest possible integration time of the Structurator.

6.4 Structure functions

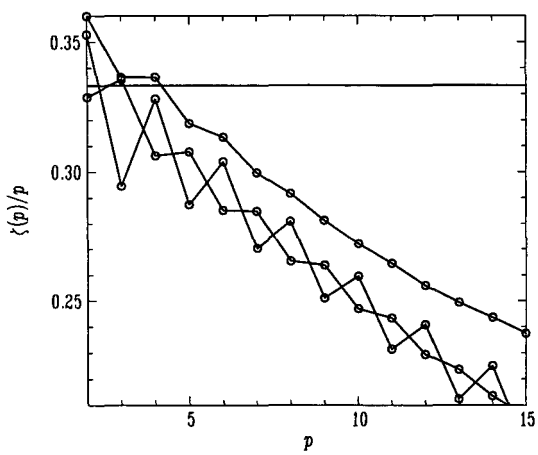
Structure functions are important quantities in the research of the behavior of small-scale structure of turbulence. They show the scaling properties of turbulence. From the 32 velocity difference PDFs measured with the Structurator, of which 4 are shown in Figure 6.4, we can calculate longitudinal structure functions $G_p^L(r)$ of arbitrary order p, using Eq. (3.1). Figure 6.5a shows 3 third-order structure functions measured at 3 different distances from the wall of the windtunnel. The third order structure functions in Figure 6.5a are normalised



(a)



(b)



(c)

Figure 6.5: *Calculated structure functions from the Δu PDFs measured with the stucturator. (a) Third order structure functions normalised to v_K, measured at three different distances from the wall. Distance from the wall is 5cm, 10cm and 19cm for the upper, middle and lower curve respectively (b) Structure functions $Gp^{1/p}$ of order p = 3, 6 and 12, lower, middle and upper curve respectively; the slope $\zeta(p)/p$ of the scaling range decreases with higher orders. (c) Scaling exponents ζ_p/p, taken from the structure functions measured at three different distances from the wall. The horizontal line depicts the Kolmogorov K41 scaling exponents $\zeta_p = \frac{p}{3}$*

to the Kolmogorov velocity v_K in a manner as was described in section 2.3. The third order structure function can be written as Eq. (2.13) resulting in, when plotted on a log-log scale, a slope equal to unity and an offset at log(4/5) in the scaling range.

The third order structure functions in Figure 6.5a were measured with the hot-wire probe at 5cm, 10cm and 19cm distance from the wall, for the upper, middle and lower curve respectively. In this figure, the structure function nearest to the top shows a slope in the scaling range that is equal to 1.01, very close to unity. The middle curve, measured at 10cm from the wall, has a slope of 0.98 and the lowest structure function, measured at 19 cm from the wall, has a slope in the scaling range that is equal to 0.87. On the other hand, the upper curve, which has the best slope nearest to unity, has the worst offset at 0.26, which should be 0.80 (4/5), the middle curve has an offset of 0.45 and the lowest structure function has an offset at 0.57, closer to 0.80 but still not enough.

In grid-turbulence, the situation of isotropic turbulence is most closely approached at the

centerline of the grid. However, it is common practice to find a compromise between isotropy and the value of the Reynolds number Re_λ and to place the probes away from the centerline. The result is that due to the higher Reynolds number, the structure function measured nearest to the wall shows the best scaling exponent but the offset of the structure function (in the log-log graph) differs from 4/5, this discrepancy in the offset is due to the non-isotropy of the flow. The behavior of the structure function $G_3(r)$ as a function of the distance of the detector to the wall is caused by the fact that grid-turbulence is decaying. The presence of the wall makes the flow non-isotropic and non-homogenous. A question is, therefore, the applicability of scaling concepts. On the other hand one could argue that the influence of the particular flow configuration disappears at scales that are small enough, [11].

The structure functions have scaling behavior with a scaling exponent ζ_p. This scaling exponent can be found by plotting $G_p(r)$ in a log-log plot, and measuring the slope of the resulting straight line. A standard procedure is to select in a log-log plot of $G_3(r)$ the interval bounds such that a straight line is fitted to the $\log G_3(r)$. In this interval the slope of $G_3(r)$ should have $\zeta_3 = 1$, as predicted by the Kolmogorov theory. The bounds of this interval are taken to be the bounds of the inertial range for all structure functions of order p.

Figure 6.5b shows a 3th, 6th and 12th order structure functions $G_p^{1/p}$, measured at 5cm from the wall. From this figure it can be seen that the slope in the scaling range on a log-log plot, $\zeta(p)/p$, decreases with higher order. In the case of the Kolmogorov K41-theory, the slope of the lines in the log-log plot would be 1/3 for all p. Instead, the slope of the structure function decreases with increasing p, which is called *anomalous scaling.*

Figure 6.5c shows scaling exponents taken from the structure functions measured at three distances from the wall. The upper curve represents the scaling exponents from the measurement at 5cm from the wall. The middle curve is measured at 10 cm from the wall and the lowest (the most oscillating curve) is the measurement at 19 cm from the wall. Recalling that for the K41-theory the scaling exponent $\zeta(p)/p$ is equal to 1/3, which is denoted by the horizontal line in Figure 6.4c. The scaling exponents of the 2nd, 3rd and 4th order structure functions are close if not equal to this 1/3 line, but the deviation from the K41 scaling theory becomes clear at higher orders. This is why the accurate meassurement of higher order structure functions is so important.

Chapter 7

Recommendations for future research

In this chapter some recommendations are given regarding future developments. A possible setup is given for a multi-channel structurator for the measurement of transverse structure functions. Also some recommendations are given regarding the Analog-to-Digital conversion used for the turbulence measurements.

7.1 Setup for a multi-channel structurator

In experiments involving structure functions measured with several probes, transverse measurements for instance, the velocity differences are calculated between the different probe-signals. Therefore, a number of time-signals (for example eight) is sampled in parallel and the Structurator program for this kind of measurements has to be changed.

Taylor's hypothesis is not used in multiple probe measurements and hence gaps in the time series are not vital. Transverse measurements are becoming more important and longer measurements are desirable. So also for multi-channel experiments it becomes desirable to generate distribution functions in real-time.

In the following we assume that there are eight probes placed parallel in the turbulence flow. Ref. [11] gives several possible strategies for a multi-channel Structurator. The calibration and difference calculations can be implemented in a serial algorithm, in a parallel algorithm, or in a combination of these two. The kind of algorithm forms the basis of the possible performance, it defines in how many machine-cycles the desired velocity differences between a set of eight probe signals can be calculated.

Suppose that we have n probes for which we want to calculate all relevant mutual differences. When we just sequentially calculate all possible combinations, n^2 arithmetic operations are needed for each set of signals. Only $\frac{n(n-1)}{2}$ calculations correspond to different, non-zero physical distances r. For an 8 probe multi-wire setup there are thus 28 relevant velocity differences. On the other hand, all the (relevant) calculations can be done in parallel and only one cycle of $\frac{n(n-1)}{2}$ arithmetic operations is needed. The upper and lower boundaries of the number of cycles (N_c) are thus n^2 and 1. A sequential algorithm is slowest but most flexible; adding a detector in the experiments only means expanding the number of arithmetic operations. A parallel algorithm is the fastest option, but obviously not the most flexible. The choise of an *optimal* algorithm is, however, not the only requirement of a design. It also

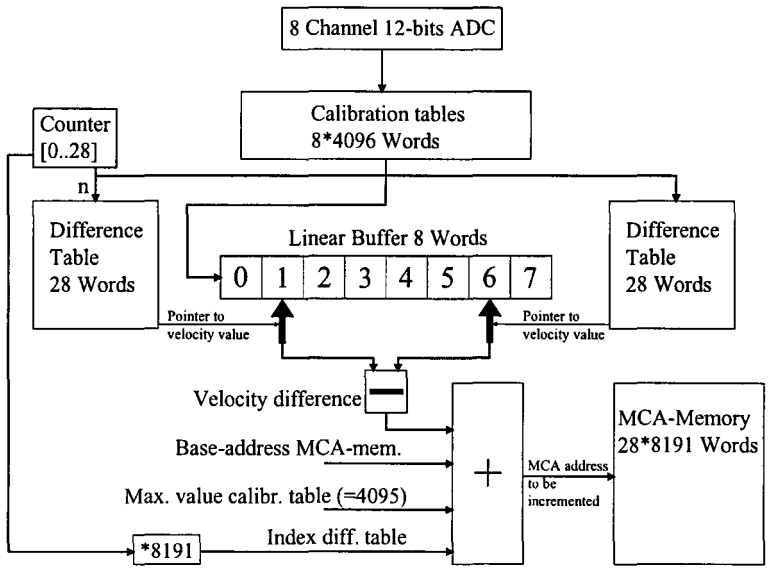


Figure 7.1: *Schematic view of a serial design for the multi-channel Structurator routine, based on the idea of the one-channel Structurator.*

has to fit in the required environment, in our case the Digital Signal Processor.

When implementing a multi-channel Structurator on the DSP system it is most efficiently done in assembler language, this is the best way of implementing the multi-channel Structurator using the full potential of the DSP processor for example for pipelining, for processing velocity difference calculations interleaved (as is done in the one-channel Structurator) and using parallel instructions. Ref. [11] reports of a *serial* Structurator-like multi-channel design being implemented in compiled C-code and run on the Texas Instruments C40 simulator. One cycle of 28 velocity difference calculations and 28 memory increments took about 1400 DSP clock-cycles, which corresponds to a real-time throughput of about 28 KHz for a 40 MHz DSP. Knowing that the performance of the C40's C-compiler can not match a routine written in assembler, it is clear that a *serial* multi-channel Structurator setup written in assembler can be done, achieving a substantial higher throughput.

For comparison let me point out that the one-channel Structurator described in section 3.3, has also been implemented in compiled C-code on the C40 for comparison. The real-time throughput was experimentally checked by increasing the sampling frequency and checking for an ADC FIFO buffer overflow, that is when the Structurator routine (in C) could not keep up with the input stream of new samples. In this way a throughput of about 25 KHz was possible. For comparison, the one-channel Structurator assembler routine can reach a throughput of about 120 KHz, as was measured experimentally. Knowing this, then in the same way the multi-channel Structurator implemented in assembler can also achieve a substantially higher throughput, comparable with the one-channel Structurator.

A serial design for the multi-channel Structurator is based on the already existing one-channel design, so this can be used as a basis to start from. Figure 7.1 gives a schematic view of a possible serial design based on the one-channel Structurator.

For eight probes there are 28 relevant velocity differences. When calculating differences, the velocity difference of two velocities stored in a buffer have to be calcuted. Hence, we can store the numbers of the probes being compared in two difference tables. The multi-

probe velocity data has to be read in parallel in some way, we talk about that later, and be calibrated. For each of the eight velocity data, the integer voltage code has to be transformed into a integer velocity code. The eight input velocity codes are stored into a 8 word long buffer. For each of the 28 difference calculations, the difference tables move two pointers along the buffer and the velocity values at these pointers in the buffer are substracted to give the velocity difference. Again the MCA memory to be incremented consists of this velocity difference, added with the maximum value of the calibration table (=4095 for a 12-bits ADC) to prevent a negative result (sign extend), added with the base address of the MCA memory block and added with $(n * 8191)$, where n is the counter for the difference tables. The resulting MCA memory can than be incremented. These 28 velocity differences can be processed interleaved in $7 * 4 = 28$ differences as is done in the one-channel structurator for $8 * 4 = 32$ offsets.

A problem for this serial design or any other design in that case, can be the actual data-acquisition itself. The eight velocity probes in the turbulence flow have to be sampled *simultaneously* and the DSP processor running the Structurator routine has to read these eight channels simultaneously. If there would be an ADC board available that has for example 8 parallel (not multiplexed!) analog inputs and 8 ADC converters each having a FIFO buffer, than that would be an option. For implementing this on the Transtech DSP PC-board, it would also be necessary for this parallel 8 channel ADC-card to be able to communicate with the C40's comports to get the data to the DSP processor and the ADC-card would have to be implemented on a TIM40 size card (for details about the Transtech DSP PC-board and the TIM40 standard see Appendix A). If this kind of multi-channel ADC is not available, then the whole setup for implementing the multi-channel Structurator on the TMS320C40 as it is now, would be obsolete. Then another way has to be found to get the desired dataflow to the DSP processor.

If such an ADC is available and if the multi-channel Structurator is to be used in transverse experiments, then it should be mentioned that the calibration procedure for the hot-wires should be speeded up. The calibration of 8 parallel hot-wires otherwise consumes to much time.

7.2 Analog to Digital conversion

7.2.1 Higher resolution ADCs

The longitudinal measurements of the turbulence characteristics such as the energy spectrum and the structure functions can be further improved by using a higher resolution for the Analog-to-Digital converters. If, for example an ADC of 16-bit resolution could be obtained then the resolution of these measurements would be much improved. Also, if 64 instead of 32 velocity differences would be used for the Structurator, then the structure functions would gain in accuracy. However, the setup as it is now would then have to be changed regarding the available memory.

The DSP system currently has two blocks of 1Mbyte (262 KWord) SRAM available. For the current setup of the Structurator, the size of the memory block required for storing the 32 velocity difference PDFs when using a 12-bit ADC, is now 32x8192*4 = 1Mbyte, thus exactly enough for filling one SRAM memory block. However, if a 16-bit ADC is to be used, then for storing the same 32 velocity difference PDFs we would require 32*131072*4 = 16Mbyte! This option in SRAM memory for the Transtech C40 module (Transtech TDM407, see Appendix

A) is not available. For the memory required using different ADC resolutions and for using a 32 or 64 size offset table see Table 7.1.

n-bits ADC	# offsets	Mbyte Memory
12	32	1
12	64	2
14	32	4
14	64	8
16	32	16
16	64	32

Table 7.1: *Required DSP processor memory for several Structurator setups.*

The largest memory available for each memory block on the C40 module (TDM407) is 4Mbyte, but that is for the case of slower DRAM. In that case the best possible setup for the Structurator will be an 14-bit ADC with 64 velocity difference PDFs. However, using slower DRAM will slow down the Structurator routine because memory access is slower, so the incrementing of the MCA memory locations will slow down. But on the other hand, we currently only use the Structurator with a 40 KHz ADC sampling frequency for the structure functions measurements, and a throughput of 120 KHz is possible, so there is considerable overhead available.

7.2.2 ADC differential non-linearity

In section 5.5.2 the differential non-linearity (DNL) of the ADC problem was discussed and the statistical error it has on the velocity PDFs. This differential non-linearity is an artifact of the used ADC type (BurrBrown ADS7810), which is a successive approximation ADC. This Analog-to-digital conversion principle is a poor performer where DNL is of importance.

A known cure of ADC differential non-linearity in repeated measurements of the same signal is the sliding scale principle. The principle is that before digitization of the signal a random voltage is added. This voltage corresponds to a known (random) binary number. This number is substracted from the digitized result, giving the digitized voltage of interest, [11]. This will smear the signal out over an appreciable fraction of the ADC's range, and should reduce the effect of the intrinsic differential non-linearity of the ADC.

Where DNL is of prime interest there are other techniques for Analog-to-digital conversion that cope with the differential non-linearity problem. The commonly used ADC type is the *Wilkinson ADC*. Conceptually, the measured quantity is deposited as a charge on a capacitor. A counter then measures the time needed to discharge the capacitor with a linear ramp. These type of ADCs are used intensively in high-energy physics. Other types of ADC converters also have low DNL error, such as sigma-delta ADCs or integrating ADCs.

If the measurement of velocity PDFs are considered of major importance than another type of ADC or the sliding scale principle, should be considered.

7.3 Hot-wire probe

Another point of concern is the rectification error described in section 5.6. Due to the fact that the hot-wire has rotational symmetry, it is both sensitive for velocity pertubations in the transversal direction, that is perpendicular to the main flow, as for pertubations in the direction of the mean flow. In section 5.6 possible errors due to this rectification problem were investigated. It seems that there is an influence on the energy spectrum in the form of an extra scaling term, and the velocity PDFs also seem to be influenced by a second PDF term in the resulting velocity PDF.

Therefore, if the measurements are to be improved and also if investigation of this rectification problem for a single hot-wire setup is required, it would be necessary to compare the single wire setup with a multiple wire or a X-wire setup. Multiple wires in the three directions or an X-wire also sensitive for 3-dimensional velocities, can then be used to filter out the single velocity direction of interest.

Chapter 8

Conclusions

The real-time turbulence measurements have succesfully been implemented on a TMS320C40 Digital Signal Processor. The developed software uses some characteristics of the Digital Signal Processor which are beneficial for our requirements, such as the fast processing capability, provisions for Fast Fourier transformations and circular buffers. Turbulence characteristics can now be measured with high accuracy.

This thesis looked into the digital signal analysis. Because the signal is discretized in both time and signal value, an investigation is done on how this affects the measurements and the statistical turbulence characteristics. The discrete approximation of derivatives is one of the most important issues in this matter. The representation of discrete differencing algorithms in the frequency domain, turns out to have some kind of filtering capability. On first sight it seems that a simple second order discrete differencing formula is less accurate then the higher order versions. But if a high enough sampling frequency is used, then this lower accuracy can be used to filter out the noise resulting from the Analog-to-Digital conversion and other noise from electronc equipment.

Another error, related to the discretization of the turbulence signal, is the differential non-linearity (DNL). It is composed out of two contributions, one resulting from the non-linear calibration table and one from the differential non-linearity of the Analog-to-Digital converter (ADC). The DNL is mainly visible in measured velocity probability distribution functions (PDFs). The influence of the non-linear calibration table has successfully been removed. The influence of the ADC remains still present, and several techniques to counter this problem have been discussed. It is recommended that a different type of Analog-to-digital conversion is necessary to remove this DNL contribution.

The windtunnel measuments have been carried out using a hot-wire anemometer system consisting of a single hot-wire positioned in the turbulent airflow and a constant temperature anemometer (CTA). Due to the rotational symmetry of the hot-wire, it is sensitive for both velocity pertubations in the transversal direction (i.e. perpendicular to the main flow) as for pertubations in the direction of the mean flow. The suspicion arises that this socalled rectification error effects the value of measured scaling exponents and the velocity PDFs. In case of scaling behavior it seems that the energy spectrum obtains an extra term, which can be related to 3-point velocity correlations. In case of the velocity PDFs, it seems that these consist of a summation of two velocity distributions. However, these errors are not completely understood and have to be investigated further.

Some experimental results are presented in this thesis. The experiments were mainly done

to check the new instrumentation that was developed in this project. Experimental results which are presented are: *energy spectra*, *velocity PDFs*, *velocity difference PDFs* and *Structure functions*. The results agree very well with those of earlier experiments.

The highly improved statistics of the energy spectra, however, led to the verification of a new turbulence phenomenon. In the energy spectra, a socalled energy pileup becomes visible that is ascribed to the bottleneck phenomenon. It is most clearly seen in a region in the spectrum just before the energy drops off, due to viscous dissipation. Some theoretical results also show the same energy pileup as the experiments. It is of high importance to further investigate this energy pileup.

Some recommendations are given for a multi-channel Structurator. Due to the succesful implementation of the one-channel Structurator on the Digital Signal Processor, it must be straightforward to implement a multi-channel Structurator for the measurement of transversal structure functions in the same way, provided that a suitable multi-channel ADC is available. Also for the improvement of the accuracy of the current instrumentation, another ADC is an option, provided that the necessary memory configuration of the Digital Signal Processor can be obtained.

Appendix A

Digital signal processing hardware and software

This appendix describes the hardware and software that is used for the implementation of the real-time measurements of turbulence characteristics. The first part deals with digital signal processors on the whole and their special design which makes them ideal for this kind of work. The second part describes the used digital signal processor, the Texas Instruments TMS320C40 DSP processor and the last part describes the actually used DSP PC-board of Transtech Parallel Systems.

A.1 The architecture of Digital signal processors

Digital Signal Processors (DSP's) are special designed integrated circuits for doing fast digital signal processing. There are two kinds of digital signal processors; the ones which are general purpose and freely programmable and the ones which have a fixed function specific for the application. Both these DSP processor types can have two kinds of arithmetic; fixed point or floating point arithmetic. The common known digital signal processors are the ones which are programmable (general purpose). For the application described in this thesis we use a programmable general purpose digital signal processor with floating point arithmetic.

On the whole, digital signal processors are in fact still common CPU's, but the architecture is optimized for fast input, fast processing and fast output of continious data at high data rates. If a DSP processor has to work in real-time then the architecture has to be specially designed and optimized to do real-time digital signal processing tasks. A hardware architecture which is optimized for real-time DSP is often characterized by the following items:

- Multiple bus structure with separate memory space for data and program instructions.
- The I/O port deals with the transport of data of external devices such as an ADC or a DAC and deals with the transport of digital data to other processors. Direct memory acces (DMA) (if applicable), deals with fast transport of data-blocks direct to or from RAM.
- Arithmetic units for logical and arithmetic operations, like an ALU (Arithmetic Logic Unit) and a hardware multiplier-accumulator (MAC).

Why is such an architecture necessary? Most DSP algorithms such as filtering, correlation and the Fast Fourier Transform (FFT), involve repetitive arithmetic operations such as multiply, add, memory accesses and heavy data flow through the CPU. The architecture of standard microprocessors is not suited for this type of activity. An important aspect of DSP processor hardware design is the optimization of both the hardware architecture and the instruction set for DSP operations. In digital signal processors, this is achieved by making extensive use of the concepts of parallelism. In particular, the following techniques are used, [13]:

- *Harvard architecture*;
- *Pipelining*;
- *Fast, dedicated hardware multiplier-accumulator (MAC)*;
- *Special instructions dedicated to DSP*;
- *Replication*;
- *On-chip memory/cache*;

For successful DSP design, it is important to understand these key architectural features and they are discussed below.

Harvard architecture: The architecture of a digital signal processor is usually characterized by the Harvard architecture. In this setup the access to data and instruction memory is done with seperate channels so that the two information streams do not interfere. Data and instructions are placed in seperate memory spaces. This in contrast to the Von Neumann architecture which can be found in general purpose CPU's like in a PC. The Von Neumann architecture does not destinguish between instructions and data. The same memory can be used for the storage of data and instructions. A CPU executes an instruction in a socalled Von Neumann cyclus:

- *Instruction Fetch* (IF); the instruction is fetched from memory and the program counter is updated,
- *Instruction Decode* (ID); the instruction is decoded and operand address generation is performed,
- *Operand Fetch* (OF); if necessary, the operands are read from memory,
- *Execute* (EX); performs the requested operation and stores the result.

In a standard microprocessor, without a Harvard architecture, the program instructions (program code) and the data (operands) are held in one memory space. Thus the fetching of the next instruction while the current one is executing is not allowed, because fetch and execution phases each require memory access. The use of a Harvard architecture permits the full overlap of instruction fetch and execution, because seperate memory spaces are used. Strict Harvard archtecture is used by some digital signal processors (for example, Motorola DSP56000), but most use a modified Harvard archtecture (for example the TMS320 family of Texas Instruments). In the modified architecture used by the TMS320C40, for example,

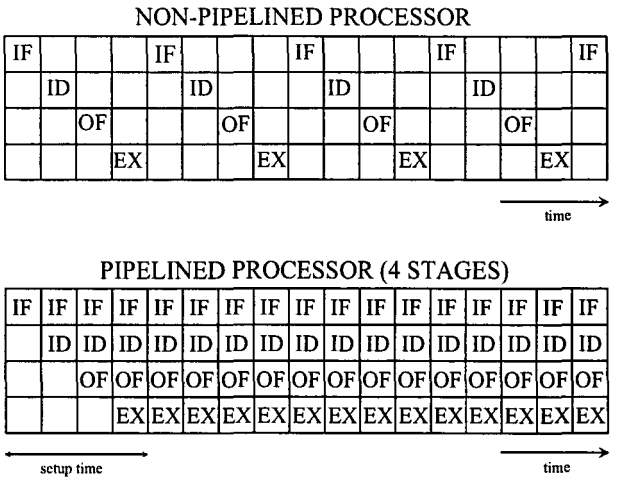


Figure A.1: *Schematic representation of a non-pipelined and a (4 stage) pipelined processor. After an initial setup time the pipelined processor clearly processes instructions faster than a non-pipelined processor.*

separate program and data memory spaces are still maintained, but communication between the two memory spaces is permissible, unlike the strict Harvard architecture.

Pipeling: Pipelining is a technique which allows two or more operations to overlap during execution. In pipelining, a task is broken down into a number of distinct subtasks which are overlapped during execution. It is used extensively in digital signal processors to increase speed. Figure A.1 shows a schematic representation of a 4 stage pipeline processor in comparison with a non-pipelined processor. Each step in the pipeline takes 1 machine cycle (*not* one clock cycle). Thus during a given cycle up to four different instructions may be active at the same time, although each will be at a different stage of completion. Complications which can occur if a pipelined structure is used are:

- The maximal speed-up is reached only after a certain setup time, because the pipeline must be filled at startup.
- The former also applies when after each branch (Branch, Branch from subroutine) to a different part of the program, the program returns (Branch from subroutine). Then again the pipeline has to be filled.
- Not all stages of the pipeline have the same execution time. Decoding sometimes costs a lot of time; then the rest of the pipeline has to wait.
- Memory conflicts can occur at IF, OF and EX if instruction and operand are in the same memory block.

The last complication can be avoided if the Harvard architecture is used. Thus, this architecture is well suited to use with the pipeline structure of a CPU, because memory conflicts are avoided. On the whole, pipelining can give a significant reduction in execution time per instruction. The throughput of a pipelined machine is determined by the number of instructions through the pipe per unit time. All stages of the pipeline are to be synchronized. The time for moving an instruction from one step to another within the pipe is one cycle and depends

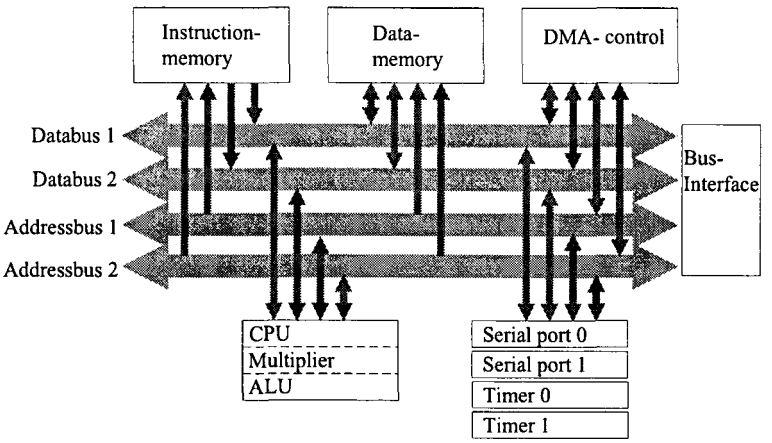


Figure A.2: *Schematic representation of the buildup of the internal DSP processor architecture. DSP processors usually have more than one seperate data- and instruction-busses and multipliers-accumulators (MACs) in hardware.*

on the slowest stage of the pipeline. In a perfect pipeline, the average time per instruction is given by, [13]

$$\frac{\text{time per instruction}}{\text{number of pipeline stages}}$$

In the ideal case, the speed increase is equal to the number of pipeline stages. In practice the speed increase will be less because of the overheads in setting up the pipeline, and delays in the pipeline registers, and so on.

Hardware multiplier-accumulator: Other basic elements of a digital signal processor are also used to let the processor do more than one operations per clockcycle. For this the DSP processor has often one or more hardware multipliers-accumulators (MACs) and the complete internal processing is captured in a pipeline structure. The basic numerical operations in DSP are multiplications and additions. To make real-time DSP possible, a fast dedicated hardware multiplier-accumulator using fixed or floating point arithmetic is mandatory. Further the troughput capacity is increased by using multiple data and/or addressbusses in the architecture, which makes it possible to read more than one words per cycle and by which better I/O-functionallity is reached. One or more serial commmunication channels increase further the I/O capability, see Figure A.2

Special instructions: Digital signal processing involves the use of special data structures such as can be found in FFT-operations and working with circular buffers. This has lead to some specific addressing modes. Some of them can also be found with regular processors:

- *immediate*; operand is part of instruction.
- *direct*; address of operand is part of instruction.
- *indirect*; address is in register, which is increased or decreased after execution.
- *modulo*; for addressing circular buffers.
- *index*; basic address with offset.

- *bit reverse*; for FFT operations.

Also DSP processors are equiped with special instructions which can be of use when dealing with special DSP operations. The benefits of having special instructions are twofold: they lead to a more compact code which takes up less space in memory and they lead to an increase in the speed of execution of DSP algorithms. Special instructions provided by DSP IC's include (**i**) instructions that support general DSP operations (such as digital filtering and correlations), (**ii**) instructions that reduce the overhead in instruction loops (for example zero overhead loops and single cycle branching instructions) and (**iii**) application-oriented instructions (for example for updating of LMS-algorithm adaptive filters, FFT operations).

Replication: In DSP processors, replication involves using two or more basic units, for example using more than one ALU, multiplier or memory unit. Often the units are arranged to work simultaneously, see also Figure A.2 where more than one data and addressbus makes the concurrent use of basic units possible. However, full-blown parallel processing concepts where for example a number of independent processors work on a given task, or several processors under one control unit work simultaneously on a single problem, are being extended to DSP. An examples of a DSP processors with parallel processing capability is the Texas Instruments TMS320C40.

On-chip memory/cache: In most cases, DSP chips operate so fast that slow inexpensive memories are unable to keep up. The common practice is to slow the processor down by adding wait states. In some processors, wait states are software programmable, but in others a piece of external hardware is necessary to slow the processor down. Wait states mean of course that the processor cannot operate at full speed. To alleviate this problem many DSP chips contain fast on-chip data RAMs and/or ROMs. In such processors, slow external memories may be used to hold program code. At initialization, the code may be transferred to the fast, internal memory for full-speed execution. Fast on-chip EPROMs are useful for real-time development and for final prototyping. Some DSP ICs provide an on-chip program cache which may be used to hold often repeated sections of a program. Execution of codes in the cache avoids further memory fetches and speeds up program execution.

A.2 The Texas Instruments TMS320C40 Digital signal processor

The general purpose digital signal processor used for the implementation of the real-time turbulence measurements, is a DSP processor manufactured by Texas Instruments, type TMS320C40. Often this DSP processor is simply called the 'C40'. It is a high performance digital signal processor, especially suitable for applications in parallel processing systems and real-time embedded applications. The C40 provides vast I/O bandwidth through the implementation of two external interface ports and six additional communication ports. The on-chip DMA coprocessor supports concurrent I/O and CPU operations, alleviating the CPU from time consuming I/O operations. Due to an on-chip floating point/integer multiplier and an arithmetic and logic unit the C40 is capable of executing computationally intensive algorithms.

In order to become acquainted with the architectural concepts of the TMS320C40 this section presents a brief summary of its key features. Many of the techniques discussed in the former section are used in this DSP processor. Figure A.3 depicts the internal architecture

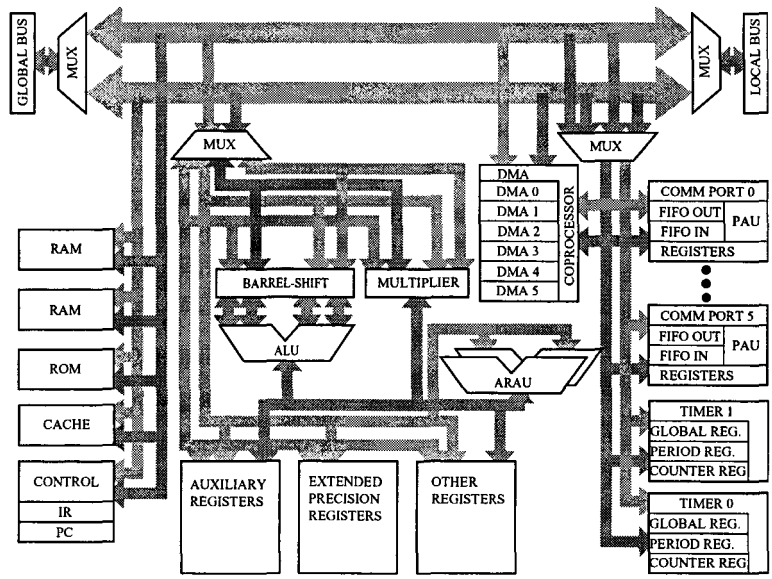


Figure A.3: *Schematic overview of the internal architecture of the Texas Instruments TMS320C40 DSP processor.*

of the C40.

The central processing unit: The Central Processing Unit (CPU) of the C40 is capable of performing both floating point and integer operations. In order to do so, the CPU comprises a *multiplier* and an *Arithmetic Logic Unit* (ALU). The multiplier of the C40 performs single-cycle multiplications on both 32-bit integer and 40-bit floating point values. These operations include integer and floating point conversions. Additional to the ALU and the multiplier, the C40 CPU comprises a 32-bit *barrel shifter*. This barrel shifter is used to shift an operand up to 32 bits left or right in a single cycle. To augment efficiency in the hardware implementation, the floating point format of the C40 differs from the IEEE floating point standard. To overcome this problem the C40 provides two special instructions which convert numbers from C40 to IEEE format and vice versa. The C40 provides 32 registers in its *primary register file*, this primary register file is tightly coupled to the C40 and all of its registers can be manipulated by the multiplier and the ALU. The names and the functions of the registers in the primary register file are tabulated in Table A.2. In addition to the 32 registers in the primary register file, the *expansion register file* provides two registers representing pointers to the *trap vector table* and the *interrupt vector table*. The registers in the expansion register file can not be modified by the multiplier or ALU, however the C40 provides the instructions LDEP and LDPE to copy their contents to and from the primary register file. The registers in the expansion register file are tabulated in Table A.2. A special case is the *program counter* (PC), a 32-bit register. Although the PC is not contained in the primary register file, its contents can be modified by instructions that control the program flow.

Memory organisation: In contrast to other members of the TMS family of DSP processors, the C40 is completely memory mapped, i.e. there are no separate program, data or I/O strobes. The total memory span of the C40 is 4 GigaBytes. In order to increase performance the C40 is equipped with internal storage facilities such as RAM, ROM and Cache memory. The internal RAM of the C40 is organized into two blocks of 1Kx32bit each. in order to access these memory blocks the C40 is equipped with multiple address, data and

Register	Function
R0..R11	Extended Precision registers 0 to 11
AR0..AR7	Auxiliary registers 0 to 7
DP	Data-page pointer
IR0, IR1	Index register 0 and 1
BK	Block-size register
SP	System stack pointer
ST	Status register
DIE	DMA interrupt enable
IIE	Internal interrupt enable
IIF	IIOF register
RS	Repeat start register
RE	Repeat end address
RC	Repeat counter

Table A.1: *TMS320C40 registers*

Register	Function
IVTP	Interrupt vector table pointer
TVTP	Trap vector table pointer

Table A.2: *TMS320C40 expansion regeisters*

program busses. These seperate busses allow program fetches, data reads or writes and DMA operations to occur parallel. The internal ROM is reserved for an internal boot loader. This boot loader is described later on. The internal cache is a 128x32bit instruction cache capable of storing often repeated sections of program code. Due to the use of internal cache, external storage devices may be slower, without significantly deteriorating processor performance. An additional advantage is the less frequently use of the external busses, thus simplifying DMA-transfers to external devices. External devices can be connected to the C40 through the implementation of two almost identical interfaces: the *Global memory interface* and the *Local memory interface*. Each of these interfaces is made up of a 32bit data bus and a 31bit address bus and two independently functioning sets of control signals. Both busses can be used to access shared or dedicated resources. Therefore, an unlimited number of different multiprocessor configurations may be constructed.

Boot loader: The internal ROM of the C40 is reserved for a factory fitted boot loader. This boot loader enables the processor to load and execute programs from a host computer, external memory or another C40. The boot loader can operate in two modes, depending on the values of the IIOF[1..3] pins during boot phase.

These modes are:

- *Memory boot load,*
- *Communication port boot load.*

During the memory boot load the C40 normally loads the program from relatively slow ROM to fast RAM. If the load phase is completed the program is executed. The C40 is also capable of performing a boot load by using one of its 6 communications ports. The C40 scans each communication port repeatedly for incoming data. If incoming data is detected the associated communications port is used to perform the bootload.

Communication Ports: A main feature of the C40 DSP processor is the implementation of 6 *communication ports.* These communication ports enable direct, *Glueless*, interprocessor communications without the need for external hardware. Although many multiprocessor systems use a shared bus structure to provide a datapath between processors this method limits the processor communication bandwith. In many high performance multiprocessing systems processor to processor communication is critical. The six communication ports of the C40 consists of four control lines and eight data lines. Both control and data lines may be defined as either outputs or inputs by a *Port Arbitration Unit* (PAU). Each PAU determines the direction of its associated communication port by examining the ownership of a transmit-token. To obtain bus ownership a communication port must request the token from another bus or device and await the associated acknowledge that signals a successful token transfer. At reset ports 0, 1 and 2 own the transmit token and are consequently configured as outputs, whereas ports 3, 4 and 5 are configured as inputs. Due to their parallel transmission protocol, the maximum data transfer rate of each communication port is 5Mx32bit words per second. To avoid deceleration of the data transfer each communication port is buffered by a bidirectional *First-In First-Out* buffer (FIFO). Each FIFO is made up of an eight level 32bit storage. To synchronize operations between CPU or DMA and communication ports, the ports report their condition via internal interrupts and internal ready signals.

Direct Memory Acces: To increase the data transfer rate the C40 is equipped with a *Direct Memory Access* (DMA) coprocessor which supports 6 DMA channels. These DMA channels can perform transfers to and from anywhere in the processor's memory map. Transfers can be made to and from external devices, internal memory or communication ports. The DMA coprocessor is especially designed for use with digital signal processing algorithms because it provides special addressing modes for Fast Fourier Transforms (FFT) and matrix operations. An important feature of the C40's DMA coprocessor is its autoinitialisation mode. Due to this autoinitialisation mode the DMA coprocessor can be configured completely autonomous and hence data transfers can be started without any intervention of the CPU. In addition to transferring data the DMA channels may also be programmed to fill a block of memory with a single value. This operation can be used to initialize array's in a CPU independent manner.

Timers: The C40 provides two internal timers (Timer 0 and Timer 1), which can be used to count external events or to signal a device at specified intervals. These timers may be clocked by an internal or an external ckock. By using the internal clock the timer may signal external devices such as A/D converters to start a conversion. If an external clock is used, the timer can interrupt the C40 if a specified number of external events has occurred.

Pipeline Operation: As mentioned before a basic processor may be partioned in four destinguished levels:

- *Fetch*; the instruction is fetched from the memory and the progam counter is updated,
- *Decode*, the instruction is decoded and operand address generation is performed,
- *Read.* if necessary the operands are read from memory,
- *Execute*; performs the requested operation and stores the result.

To augment the processor's performance, the C40 is equipped with a four stage instruction pipeline. This pipeline enables the 4 levels listed above to be executed in parallel. If the pipeline is completely filled, the CPU operates at an execution rate of one execution per cycle. Although the maximum benefit of the pipeline can only be reached by carefull programming, no special precautions are needed to guarantee correct operation and hence the pipeline operation is completely transparent to the user. In order to take full advantage of the instruction pipeline, the C40 provides a number of special instructions. These instructions include delayed branches, delayed calls and parallel instructions.

Instruction Set: The C40 provides a number of different addressing modes, some of them implemented to support signal processing algorithms:

- *Register*; the operand is one of the registers in the primary register file (see table A.2),
- *Immediate*; the operand is a 16bit immediate value contained in the instruction word,
- *Direct*; the data address is formed by the concatenation of the 16 least significant bits of the DP register with the 16 least sifnificant bits of the instruction word,
- *Indirect*; the data address is formed by the addition of the auxiliary register content and an optional displacement,
- *PC-relative*; is used for branch instructions.

JTAG-Interface: The C40 is equipped with a dedicated emulation port. This emulation port is operated according to a superset of the IEEE 1149.1 (JTAG) standard. To access the JTAG port an XDS510 emulator should be used. Multiple processors may be daisy-chained to enable easy debugging of a multiprocessor configuration.

A.3 The Transtech DSP PC-board

For the implementation of the digital signal processing hardware a DSP PC-board from Transtech Parallel Systems was used. This internally mounted PC-board has a PC-AT bus (or ISA-bus) connector for communication with the host personal computer. In the next subsection the hardware is described. After that we describe the Transtech software used to operate this DSP board.

A.3.1 Hardware

The Transtech system consists of the following hardware elements:

- *TDMB412*; Transtech motherboard for PC-AT class host machines,

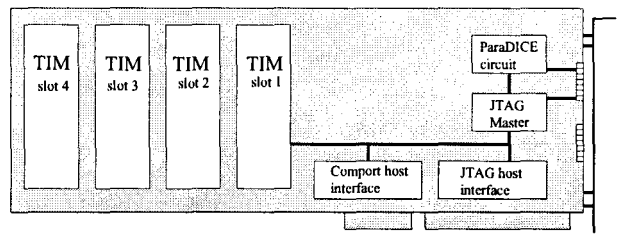


Figure A.4: *Block diagram of the Transtech TDMB412 motherboard. The motherboard has room for 4 TIM size one cards, which can be either a TIM40 module with a TMS320C40 DSP processor or a TIM40 size one 12 bit ADC/DAC card.*

- *TDM407*; Transtech TIM40 module with a Texas Instruments TMS320C40 DSP processor,
- *TDM431*; Transtech two channel 12-bit analog to digital converter with the size of one TIM40 module.

The TIM40 standard: TIM40 is a recognized industry standard developed by a consortium lead by Texas Instruments for C40 and C44 modules. The specification allows (for example) for a processor and memory or other circuitry on a small daughter board. The TDMB412 motherboard conforms to the standard of a TIM40 motherboard and will accept any modules which have been produced to the TIM40 specification.

The TDMB412 motherboard: The TDMB412 is a four slot TIM40 motherboard for PC-AT class machines. The TDMB412 has both an 8 and a 16 bit host interface which drive a single C40 communications link (comport). The comport from the interface can be connected either to the first on-board TIM40 slot or to the board's edge connector (see figure A.4). This allows the TDMB412 to be used as a PC-to-comport interface card for connecting to C40 equipment outside the PC. It is possible to disable the interface circuitry completely, allowing the TDMB412 to be used as a slave board to another TDMB412. There can be up to four TIM40 modules installed on a TDMB412 allowing up to eight C40 or C44 processors per board. Some of the TIM40 comports are hardwired by the motherboard to connect between certain of the slots. Most remaining TIM40 comports are brought to cable headers along the top edge of the board. This allows customization of the C40 topology by the use of comport cables. The TDMB412 contains an XDS510 compliant JTAG PC interface. This is designed primarelly for use with C40 JTAG based debuggers which access the JTAG bus via an XDS510 interface card (such as the Transtech's C40 debugger). The JTAG bus can be easily daisy chained between any number of motherboards via the ParaDICE port on the TDMB412. The TDMB412 can be used as a component in a larger C40 system. Normally, one C40 in the whole network is regarded as the *root* C40, and has the priviledge status in that one of its comports is connected via a comport adapter to the host computer. The motherboard which has the root C40 installed is referred to as the *master*board in the system and will have its host interface enabled. It will be responsible for coordinating other motherboards in the system at program boot time. All boards other than the masterboard are called *slave*boards and they will have their host interface disabled.

The TDM407 TIM40 module: The TDM407 is a Transtech TIM40 module which can be put into one of the four slots of the TDMB412 motherboard. The TDM407 integrates a 40, 50 or in our case a 60 MHz TMS320C40 DSP processor with a choise of memory configurations

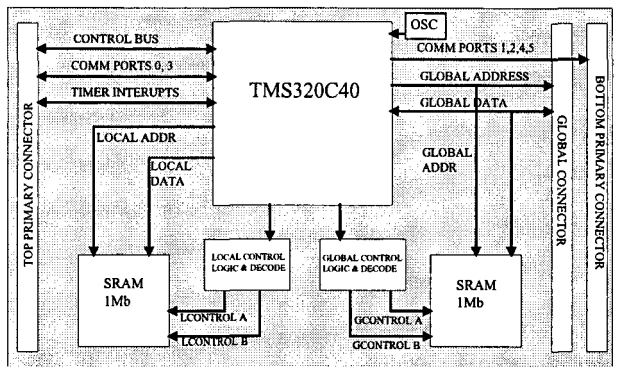


Figure A.5: *Block diagram of the TDM407 TIM40 size one card. The TDM407 integrates a 60 MHz TMS320C40 DSP processor with a choise of memory configurations. In our case the memory blocks are devided into 2 blocks of 1 Mb SRAM split between the local and global busses of the C40 DSP processor.*

that may allow up to 8Mb of DRAM or 2Mb of SRAM. The TDM407 module conforms to the specification for a size one TIM40 module. The memory is split between the global and local busses of the C40 DSP processor. In our case the memory configurations are 1 MB SRAM local memory and 1 Mb SRAM global memory. The SRAM is configured as two 512 Kb pages of memory within each local or global block. The SRAM memory has zero wait state access within a page, but an additional cycle is inserted when successive accesses cross the page boundary. The TDM407 has the additional feature of a global connector, allowing additional flexibility with memory configurations. This global connector allows the TDM407 to access any global memory available on the motherboard. The clock for the module may be derived locally or taken from the motherboard. The block diagram for the TDM407 is shown in figure A.5.

The TDM431 12-bit ADC: The Transtech TDM431 is a two channel 12-bit analog Input/Output TIM40 size one module capable of analog to digital and digital to analog conversion at sampling rates of up to 250 KHz with FIFO buffering to two C40 comports, one for analog input and the other for analog output. In our case only the analog to digital part of the TDM431 was supplied. Each input channel accepts bipolar analog signals with a range of $\pm$10Volt. The inputs are digitised using analog to digital converters (ADCs) with a resolution of 12 bits and the data is routed via FIFO buffers to either comport 1 or 2. The analog input impedance is approximately 3 KΩ. 4 Kbyte FIFOs are used to buffer data between each of the ADCs and DACs and the input and output comports. Transfers of one or two samples per word are supported with block transfers of up to 1024 32bit words between the comports and the FIFOs. The sampling rate is identical for all channels and is software adjustable from 39 Khz to 250 KHz. Alternatively an external TTL clock up to 250 KHz may be used. When using the software adjustable sampling rate option, the sample interval counter is clocked by a 10MHz internal clock, so this has a resolution of 100ns. Because of the fact that the sample interval is chosen by a socalled commandbyte (= 8 bits) in the command word, the lowest possible sampling rate is 39.1KHz. Because of the fact that the TDM431 ADC-card is compatible with the TIM40 standard, the card can be put into one of the free slots of the TDMB412 motherboard and can be connected to one of the comports of the C40. In this way we can use the high transfer speed of the C40 comport to acquire data from the ADC

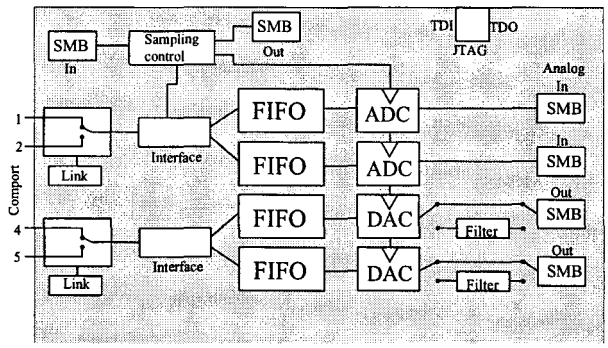


Figure A.6: *Block diagram of the TDM431, a two channel 12bit analog Input/Output TIM-40 size one module capable of analog to digital and digital to analog conversion at sampling rates of up to 250 KHz with FIFO buffering to two C40 comports, one for analog input and the other for analog output.*

into the DSP processor. Figure A.6 shows the block diagram of the TDM431 The TDM431 uses 12bit converters, so the -10V to +10V input range is mapped on to 2^{12} or 4096 codes, corresponding to 4.833mV per step. The ADC data encoding is in two's complement format, so values from ADC-code 0 to code 2047 correspond to *positive* input voltages while ADC codes from 2048 to 4095 correspond to *negative* input voltages.

A.3.2 Software

The Transtech system is supplied with the Transtech Parallel C40 Environment (PaCE). PaCE enables the development of parallel programs on networks of TI's TMS320C40s. PaCE consists of a number of components:

- *Compiler*; the Texas Instruments C40 optimizing compiler and assembler/linker tools,
- *loadc40*; the Transtech C40 network loader,
- *TOPS*; Transtech Open Parallel Server,
- *Emulator*; Transtech C40 Emulator under Microsoft Windows.

The PaCE development process by which a network of C40s can be loaded with an application consisting of a program or programs written in C is summarized in Figure A.7.

A PaCE application consists of programs written in C to run on each C40 in a network of C40s. The nodes all run the same program or run different programs as required. Each individual C program is compiled and linked using the Texas Instruments C compiler and linker. Given a network description file (extension .nd), TOPS is used to load the application onto a network of TIM40 modules. This network description file containes information for each processor in the network specifying the number of the processor from which the node is booted, the number for this node starting at 1 for the root processor, the booting node's comport down which the processor is booted, the name of the program to load onto the node and the command line arguments to give to the program. By using the TOPS server and library, the program running on the C40 connected to the PC can perform I/O on the PC-host such as printing to the screen and file accesses. User programs on the root processor can access PC host I/O facilities using standard ANSI C calls, such as *printf(), fopen(), fwrite(), etc..*,

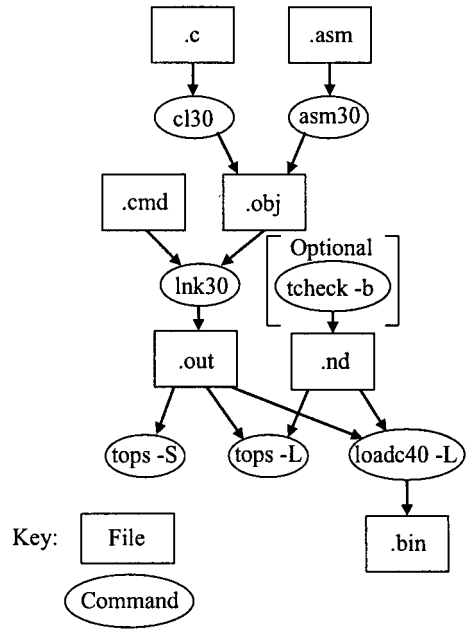


Figure A.7: *The PaCE development process. Assembler programs are assembled with* asm30 *and C programs are compiled with* cl30 *which both produce an* .obj *file. This can be linked with object libraries using a* linker command file (.cmd) *and the* lnk30 *linker. The produced* .out *file can be loaded onto a network of C40s using the* tops *server program and a* network discription file (.nd) *or each individual C40 DSP can be loaded with the* loadc40 *bootloader.*

which have been implemented in the TOPS library. The TOPS server uses the C40 Network Loader (*loadc40*) to load user's applications onto a network of C40s. PaCE applications can be debugged from MS Windows running on the host PC by loading the PaCE application using TOPS and controlling the execution of the individual C40 programs using the C40 Emulator.

Appendix B

Program listings

In this appendix the programs are listed, which were developed for the turbulence measurements. The first program, the *spectrum program* measures and calculates the turbulence energy spectrum and the turbulence characteristic quantities. The second program, the *Structurator program*, is for the real-time measurement of structure functions.

B.1 Spectrum program

The program *spectrum.c* takes a number of blocks of samples from the ADC with a given sample frequency. The number of blocks, the block size and the sample frequency can freely be chosen. On each block of data a series of processes are performed. The voltage codes are transformed into velocity codes via the calibration table. From this velocity time-block the velocity probability distribution (PDF) is calculated and the mean PDF is updated. Some statistical quantities are calculated from the velocity time-block among which are the *mean velocity*, the *rms velocity* and the *rms derivative velocity*. From these quantities some turbulence quaracteristics are calculated such as the *viscous dissipation rate*, the *Kolomorov -lenght, -velocity, -time-scale* and *-frequency*, the *Taylor microscale*, the *distance scale* and the *Reynolds number*. The velocity time-block is windowed using a Hanning window and the windowed data is transformed to the frequency domain using an assembler FFT routine. The resulting frequency record is converted into a power spectrum and the mean power spectrum is updated. The used FFT routine is called *fffft_rl*, a real radix-2 DIF FFT for the TMS320C40, and is described in ref. [27]. The program *spectrum.c* is listed below as first.

The second file is the linker command file *spectrum.cmd* and is needed for linking the compiled *spectrum.c* program with the object libraries. The linker command file declares the available DSP processor memory and specifies the allocation of the different program sections into the available memory.

The last file is an example of a configuration file *spectrum.cnf* used to pass some variables and filenames to the *spectrum.c* program. With this file the user can chose the amount of data blocks to be sampled, the size of each data blocks, the sample frequency and the energy spectrum cutoff frequency when calculating the rms- and rms derivative velocity from the energy spectrum. The last option can be used to filter out the ADC noise level.

```
/*
*
* Program: spectrum.c
*
* Author: Marcel Donker
*
* Last Update: 101296
*
* This program uses the TI C40 DSP and a 12-bits ADC to
* sample one input-channel. A FFT routine is used to transform
* the real input data into a turbulence spectrum. A number of
* power-spectra measured to give an average power-spectrum.
*
* The turbulence characteristics are calculated from the time-series
* and from the power-spectrum.
*
* This program uses the Transtech TDMB412 motherboard, with one TIM-40
* module (C40) in slot 1 and a TDM431 12-bits ADC in slot 3. Comport 4 of
* the C40 is connected to comport 1 of the ADC.
* Data is sampled from ADC channel 1.
*
* Some constants like the sample frequency, sample size, number of blocks
* and some file names are read from the configuration file:
* c:\marcel\spectrum\spectrum.cnf
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <link.h>
#include <values.h>
#include <math.h>
#include <fcntl.h>
#include <string.h>

#define TRUE                    1
#define FALSE                   0
#define CALIBR_SIZE             4096    /* size of calibration table */
#define BITREV                  1       /* Enable bitreversing for FFT */
#define DEFAULT_ADC_COMPORT     4       /* Comport ADC */
#define ADC_READ_DATA           0x00
#define ADC_READ_STATUS         0x01
#define ADC_BLOCK_READ          0x04
#define ADC_CHANNEL_1           0x02
#define ADC_CHANNEL_2           0x00
#define ADC_TWO_SAMPLES_PER_WORD        0x08
#define ADC_SET_SAMPLE_INTERVAL         0x10
#define ADC_EXTERNAL_SAMPLE_MODE        0x20
#define ADC_STOP                0x40
#define ADC_CONTINUOUS_MODE     0x80
#define ADC_FIFO_NOT_EMPTY      0x1
#define ADC_FIFO_NOT_HALF_FULL  0x2
#define ADC_FIFO_NOT_FULL       0x4
#define MAX_BLOCK_SIZE          1024
#define TIMEOUT                 100

#define visc    1.5e-5

int *sample_data;
int *calibr;
float *veloc;
float *fourier;
float *power;
float *power_average;
float *sintable;
float *fourier_align;
float *hanning;
int *hist;

float vmul,vmin,vmax;
float theta;
float ugem,urms;
float uugem,uurms;
double s1du,s2du;
double s1duu,s2duu;
double dugem,durms;
float urms_spec,durms_spec;
float eps,eta,vK,tK,fK,lambda,reynolds,scale;

int LOG2_FFT_SIZE;
int NUM_AVERAGE;
int MAX_SIZE;
int sample_size;
int sample_freq;
float sample_time;
int spectr_cutoff;
int adc_comport = DEFAULT_ADC_COMPORT;
unsigned adc_channel = ADC_CHANNEL_1;
int hist_size = CALIBR_SIZE;
int fft_size;
int half_fft_size;
unsigned sample_interval;
```

```
char prefix[16];
char filename_1[13];
char filename_2[13];
char filename_3[13];
char filename_4[13];
char filename_5[13];

void read_cnf_file(void)
{
int i;
char dummy[60];
FILE *CNFFile;

/* Reading the configuration file */

CNFFile = fopen ("c:/marcel/spectrum/spectrum.cnf","r");
if (CNFFile == NULL)
{
printf ("Unable to read configuration-file!\n");
printf ("Error-message: %s \n",strerror(errno));
exit(0);
}
fscanf(CNFFile,"%d",&NUM_AVERAGE);     /* # of blocks to sample */
fgets (dummy,60,CNFFile);
fscanf(CNFFile,"%d",&sample_freq);     /* sample frequency */
fgets (dummy,60,CNFFile);
fscanf(CNFFile,"%d",&sample_interval); /* hex number for adc */
fgets (dummy,60,CNFFile);
fscanf(CNFFile,"%d",&sample_size);     /* # of samples per block */
fgets (dummy,60,CNFFile);
fscanf(CNFFile,"%d",&LOG2_FFT_SIZE);   /* Log 2 sample size */
fgets (dummy,60,CNFFile);
fscanf(CNFFile,"%d",&spectr_cutoff);   /* Cutoff frequency for spectrum
 turbulence characteristics */
fgets (dummy,60,CNFFile);
fgets (prefix,16,CNFFile);             /* prefix for file names */
fgets (dummy,60,CNFFile);
fgets (filename_1,13,CNFFile);         /* filename calibration
 input file */
fgets (dummy,60,CNFFile);
fgets (filename_2,13,CNFFile);         /* filename power spectrum
 output file (binary) */
fgets (dummy,60,CNFFile);
fgets (filename_3,13,CNFFile);         /* filename histogram
 output file */
fgets (dummy,60,CNFFile);
fgets (filename_4,13,CNFFile);         /* filename turbulence
 characteristics output file */
fgets (dummy,60,CNFFile);
fclose (CNFFile);
return;
}

void flush_adc_fifo(void)
{
int value;
unsigned status;

/* Stop previous sampling */
while(TRUE)
{
/* Get status */
link_out_word(adc_channel|ADC_READ_STATUS|ADC_STOP,adc_comport);
link_in_word( &value,adc_comport);
status = value & 7;
if (adc_channel == ADC_CHANNEL_2)
value >>= 4;
/* Test status */
if (!(status & ADC_FIFO_NOT_EMPTY))
{
/* Empty */
return;
}
else if (!(status & ADC_FIFO_NOT_FULL))
{
/* Read 2048 samples in 1024 words */
link_out_word( (1024<<8)|adc_channel|ADC_STOP|ADC_READ_DATA|
ADC_TWO_SAMPLES_PER_WORD|ADC_BLOCK_READ,adc_comport);
link_in(1024, (int *) sample_data, adc_comport);
}
else if (!(status & ADC_FIFO_NOT_HALF_FULL))
{
/* Read 1024 samples in 512 words */
link_out_word ((512<<8)|adc_channel|ADC_STOP|ADC_READ_DATA|
ADC_TWO_SAMPLES_PER_WORD|ADC_BLOCK_READ,adc_comport);
link_in(512, (int *) sample_data, adc_comport);
}
else
{
/* Read one sample in one word */
link_out_word(adc_channel|ADC_STOP|ADC_READ_DATA|
ADC_TWO_SAMPLES_PER_WORD|ADC_BLOCK_READ,adc_comport);
```

```
link_in_word(&value, adc_comport);
}
}
return;
}

void init_vars(void)
{
int i;
float *ptr1, *ptr2;
float delta_t, t_trunc;

delta_t = (float)(1.0/sample_freq);
sample_time = delta_t;
t_trunc = (float)(sample_size*delta_t);
theta = 2*PI/fft_size;

/* set some vars to zero */

ugem = urms = dugem = durms = 0.0;
uugem = uurms = 0.0;
s1du = s2du = 0.0;
s1duu = s2duu = 0.0;
urms_spec = durms_spec = 0.0;
eps = eta = vK = tK = fK = lambda = reynolds = scale = 0.0;

/* making hanning memory table */

ptr1 = hanning;
for (i=0;i<sample_size;i++)
*(ptr1)++ = (float)(0.5 -0.5*cos(2.0*PI*i*delta_t/t_trunc));

/* Fill sinus table in memory: */

ptr2 = sintable;
for (i=0;i<half_fft_size;i++)
*(ptr2)++ = sin(i*theta);
}

void make_calibr_table(void)
{
int i,k;
float coef[6];
float rw,ry,rg,V_offset,temp,press,ampl,V_tegen;
float corr,volt;
float vcal[CALIBR_SIZE];
char dummy[40];
char filename[28];
int *ptr;
FILE *DataFile;
FILE *CalibrFile;


/* Reading calibration characteristics file and making
   calibration table and writing it to file */

strcpy (filename,prefix);
strcat (filename,filename_1);
DataFile = fopen(filename,"r");
if (DataFile == NULL)
{
printf("Unable to read calibration data file\n");
printf("Error-message: %s \n",strerror(errno));
exit(0);
}
fscanf (DataFile,"%f",&temp);       /* temperature */
fgets (dummy,40,DataFile);
fscanf (DataFile,"%f\n",&press);    /* air pressure */
fgets (dummy,40,DataFile);
fscanf (DataFile,"%f\n",&rw);       /* hot resistance Rw at calibration*/
fgets (dummy,40,DataFile);
fscanf (DataFile,"%f\n",&ry);       /* calibration resistance Ry */
fgets (dummy,40,DataFile);
fscanf (DataFile,"%f\n",&V_offset); /* offset voltage at calibration*/
fgets (dummy,40,DataFile);
for (i=1;i<=5;i++)                  /* polynoom coefficients */
{
fscanf (DataFile,"%f\n",&coef[i]);
fgets (dummy,40,DataFile);
}
fscanf (DataFile,"%f\n",&rg);       /* cold resistance at measurement*/
fgets (dummy,40,DataFile);
fscanf (DataFile,"%f\n",&V_tegen);  /* 'tegen spanning' at measurement*/
fgets (dummy,40,DataFile);
fscanf (DataFile,"%f\n",&ampl);     /* amplification factor at measurement*/
fgets (dummy,40,DataFile);

fclose(DataFile);

corr = sqrt((rw-ry)/(rw-rg));   /* correction factor */

/* Make velocity-table for all possible voltages */
```

```
for (k=0;k<CALIBR_SIZE;k++)
{
if (k <= 2047)
volt = (float)(V_tegen + (10.0*k)/(2048.0*ampl));
else
volt = (float)(V_tegen + (10.0*(k-4096.0)/(2048.0*ampl)));
volt = (float)(volt*corr - V_offset);
vcal[k] = (float)(coef[5]);
i = 4;
while (i>0)
{
vcal[k] = (float)(coef[i] + vcal[k]*volt);
i--;
}
}
vmin = (float)(vcal[2048]);
vmax = (float)(vcal[2047]);
vmul = (float)(4095.0/(vmax-vmin));

for (i=0;i<CALIBR_SIZE;i++)
{
calibr[i] = (int)((vcal[i]-vmin)*vmul);
if (calibr[i] <0 || calibr[i] >4095)
{
printf("Giant fuckup in calibration table!!!!\n");
exit(0);
}
}
write_calibr_file();
return;
}

void write_calibr_file(void)
{
int i;
int *ptr;
FILE *CalibrFile;

/* Writing calibration table to file */

CalibrFile = fopen("c:/marcel/data/calibr.dat","w");
if (CalibrFile == NULL)
{
printf ("Unable to write calibration table to file!\n");
printf ("Error-message: %s \n",strerror(errno));
exit(0);
}
for (i=0;i<CALIBR_SIZE;i++)
fprintf (CalibrFile,"%d %d\n",i,calibr[i]);
fclose(CalibrFile);
return;
}

float calibration (int adc_code)
{
float velocity;
int *ptr;

/* Calibrating to integer velocities (note: veloc is already
   an float array, this is needed later for kolmog and fft) */

ptr = calibr + adc_code;
velocity = (float)(*ptr);
return velocity;
}

void read_samples (void)
{
int i,count,value,part;
int *ptr;
float *dest;

count = sample_size/2;
ptr = (int *) sample_data;

/* Start sampling data */

link_out_word(adc_channel|ADC_SET_SAMPLE_INTERVAL|ADC_READ_STATUS|
(sample_interval<<24),adc_comport);
link_in_word(&value, adc_comport);

/* Get data, read 4* block of 1024 32-bit words = 8192 samples */

while (count >0)
{
part = (count>MAX_BLOCK_SIZE) ? MAX_BLOCK_SIZE : count;
link_out_word((part<<8)|adc_channel|ADC_READ_DATA|
ADC_TWO_SAMPLES_PER_WORD|ADC_BLOCK_READ, adc_comport);
link_in(part,ptr,adc_comport);
ptr += part;
count -= part;
```

```
}

/* Calibrating adc-codes to (integer) velocities */

ptr = (int *) sample_data;
dest = (float *) veloc;

for (count=sample_size/2; count>0; count--)
{
*(dest++) = calibration((int)((*ptr) & 0x0fff));
*(dest++) = calibration((int)(((*ptr) >> 16) & 0x0fff));
ptr++;
}
return;
}

void histogram()
{
int i;
int index;

/* Making histogram of recorded integer velocities. */

for (i=0;i<sample_size;i++)
{
index = (int)(veloc[i]);
hist[index] += 1;
}
return;
}

void int_to_veloc(void)
{
int i;
float *ptr;

/* Turning integer velocities into real velocities */

ptr = veloc;
for (i=0;i<sample_size;i++)
{
*ptr = (float)(*ptr/vmul + vmin);
ptr++;
}
return;
}
```



```
void mean()
{
int i;
float j;
double mean,sum,sum_sqr;


/* Calculate mean and rms-value of a series of n velocity values.
 * The mean and rms are calculated via the current histogram of
 * recorded velocities.
 */

mean = 0.0;
sum = 0.0;
sum_sqr = 0.0;
for (i=0;i<hist_size;i++)
{
j = (float)(i*1.0);
sum += (float)(hist[i]*1.0);
mean += (float)(j*hist[i]*1.0);
sum_sqr += (float)(j*j*hist[i]*1.0);
}
uugem = (float)(mean/sum);
uurms = (float)(sqrt((sum_sqr/sum) - pow(uugem,2.0)));

return;
}

void mderiv(int ndif,float dx)
{
int i;
int nderiv;
double s1,s2,dydx;

/* Calculates the mean and rms values of derivatives dy/dx of a
 * series of points y(1..n), which are assumed to be on equidistant
 * x'es, with distance dx. This routine uses a differencing formula
 * of order ndif (ndif = 2,4,6,8 or 10).
 */

/* nderiv is the number of derivs
 * s1 is the sum of dydx
 * s2 is the sum of squares.
 */
```

```
nderiv = sample_size - ndif;
s1 = 0.0;
s2 = 0.0;

if (ndif == 2)
{
for (i=1;i<=(sample_size-1);i++)
{
dydx = (veloc[i+1] - veloc[i-1])/(2.0*dx);
s1 += dydx;
s2 += dydx*dydx;
}
}

else if (ndif == 4)
{
for (i=2;i<=(sample_size-2);i++)
{
dydx = ((veloc[i-2] - veloc[i+2])
- 8.0*(veloc[i-1] - veloc[i+1]))/(12.0*dx);
s1 += dydx;
s2 += dydx*dydx;
}
}

else if (ndif == 6)
{
for (i=3;i<=(sample_size-3);i++)
{
dydx = (-1.0*(veloc[i-3] - veloc[i+3])
+ 9.0*(veloc[i-2] - veloc[i+2])
- 45.0*(veloc[i-1] - veloc[i+1]))/(60.0*dx);
s1 += dydx;
s2 += dydx*dydx;
}
}

else if (ndif == 8)
{
for (i=4;i<=(sample_size-4);i++)
{
dydx = (3.0*(veloc[i-4] - veloc[i+4])
- 32.0*(veloc[i-3] - veloc[i+3])
+ 168.0*(veloc[i-2] - veloc[i+2])
- 672.0*(veloc[i-1] - veloc[i+1]))/(840.0*dx);
s1 += dydx;
s2 += dydx*dydx;
}
}

else if (ndif == 10)
{
for (i=5;i<=(sample_size-5);i++)
{
dydx = (-2.0*(veloc[i-5]-veloc[i+5])
+ 25.0*(veloc[i-4] - veloc[i+4])
- 150.0*(veloc[i-3] - veloc[i+3])
+ 600.0*(veloc[i-2] - veloc[i+2])
- 2100.0*(veloc[i-1] - veloc[i+1]))/(2520.0*dx);
s1 += dydx;
s2 += dydx*dydx;
}
}

s1duu = (double)(s1/nderiv);
s2duu = (double)(s2/nderiv);

return;

}

void kolmog (int iblock)
{
int i;
int ndif;
float du;

/* Computes characteristic quantities from the
 * integer velocity signal for the current
 * data block.
 */

mean();           /* calculating uugem and uurms from histogram */

du = sample_time;
ndif = 6;
mderiv (ndif,du); /* calculating s1duu and s2duu from velocity array */

/* Turn integer into real velocities */

uugem = (float)(uugem/vmul + vmin);
uurms = (float)(uurms/vmul);
```

```
/* Update new ugem, urms, dugem and durms.
 * Because durms is fluctuating, we update s1du and s2du.
 */

ugem = (float)uugem;
urms = (float)uurms;
s1du = (double)(((iblock-1)*s1du + s1duu)/(iblock));
s2du = (double)(((iblock-1)*s2du + s2duu)/(iblock));

dugem = s1du;
durms = sqrt(s2du - s1du*s1du);

/* Calculate some turbulence characteristics: */

/* Viscous dissipation rate */
eps = (float)((15.0 * visc * pow(durms,2.0))/(pow(ugem,2.0)));
/* Kolmogorov length [m] */
eta = (float)(pow((pow(visc,3.0)/eps),0.25));
/* Kolmogorov velocity [m/s] */
vK = (float)(pow((visc*eps),0.25));
/* Kolmogorov time-scale [s] */
tK = (float)(pow((visc/eps),0.5));
/* Kolmogorov frequency [Hz] */
fK = (float)(ugem/(2.0*PI*eta));
/* Taylor micro scale [m] */
lambda = (float)(ugem*urms/durms);
/* Lenght of one sample
 * interval in units of eta:
 * (distance scale)
 */
scale = (float)((ugem*sample_time)/eta);
/* The Reynolds number */
reynolds = (float)((urms*lambda)/visc);

return;
}

void hanning_window()
{
int i, cntr;
float *ptr1,*ptr2;

/* multiply (real) velocity data with hanning window */

ptr1 = veloc;
ptr2 = hanning;
for (cntr=0;cntr<sample_size;cntr++)
{
*ptr1 = (float)((*ptr1) * (*ptr2++));
ptr1++;
}
return;
}

void make_power(void)
{
int i;
float norm;

norm = (float)(1.0/(sample_size*1.0));

power[0] = (float)(     norm*fourier_align[0]*
norm*fourier_align[0]);
power[fft_size/2] = (float)(    norm*fourier_align[fft_size/2]*
norm*fourier_align[fft_size/2]);
for (i=1;i<half_fft_size;i++)
power[i] = (float)(     norm*fourier_align[i]*
norm*fourier_align[i] +
norm*fourier_align[fft_size-i]*
norm*fourier_align[fft_size-i]);
return;
}

void update_average(int count)
{
int i;

for (i=0;i<=half_fft_size;i++)
power_average[i]= (float) ( (power[i] +
(count-1)*power_average[i]) / count);
return;
}

void spectr_charact(int spectr_cutoff)
{
int i,ifirst,ilast;
float f;
float spec_norm;
float df;

/* Compute spectral averaged quantities */
```

```
spec_norm = 1.0;
df = (float)(sample_freq*1.0/fft_size*1.0);

/* Forget 1st 'ifirst' points of spectrum and stop at
   spectrum cutoff frequency */

ifirst = 3;
ilast = (int)(spectr_cutoff*fft_size*1.0/(sample_freq*1.0));

for (i=ifirst;i<(ilast+1);i++)
{
urms_spec += power_average[i];
durms_spec += power_average[i]*i*i;
}
urms_spec = sqrt(2.0*urms_spec*df*spec_norm);
durms_spec = sqrt(2.0*durms_spec*(4*PI*PI*df*df*df*spec_norm));

return;
}

void turbdata_to_files(void)
{
int i;
float f,freq;
float velocity;
int *ptr;
float *ptr1;
char filename[28];
int append_file,file_size;
FILE *PowerFile;
FILE *KolmoFile;
FILE *HistFile;

/* Writing the calculated turbulence data to files */

/* Averaged power spectrum (binary) */

strcpy (filename,prefix);
strcat (filename,filename_2);
PowerFile = fopen(filename,"wb");
if (PowerFile == NULL)
{
printf("Unable to create binary power file\n");
printf ("Error-message: %s \n",strerror(errno));
exit(0);
}
fwrite((void *)&sample_size,sizeof(int),1,PowerFile);
freq = (float)(sample_freq*1.0);
fwrite((void *)&freq,sizeof(float),1,PowerFile);
append_file = 0;
if (sample_size == 32768)
{
file_size = 8192;
append_file = 1;
}
else
file_size = half_fft_size;
fwrite((void *)power_average,sizeof(float),file_size,PowerFile);
if (append_file == 1)
fwrite((void *)(power_average+file_size),sizeof(float),file_size,PowerFile);
fclose (PowerFile);

/* Histogram of recorded velocities */

strcpy (filename,prefix);
strcat (filename,filename_3);
HistFile = fopen (filename,"w");
if (HistFile == NULL)
{
printf("Unable to create histogram file\n");
printf ("Error-message: %s \n",strerror(errno));
exit(0);
}
ptr = hist;
for (i=0;i<hist_size;i++)
{
velocity = (float)(i/vmul + vmin);
fprintf(HistFile,"%f %d\n",velocity,*(ptr)++);
}
fclose (HistFile);

/* Turbulence characteristics (ascii) */

strcpy (filename,prefix);
strcat (filename,filename_4);
KolmoFile = fopen (filename,"w");
if (KolmoFile == NULL)
{
printf("Unable to create turbulence data file\n");
printf ("Error-message: %s \n",strerror(errno));
exit(0);
```

```
}
fprintf(KolmoFile," \n"); /* Garbage line */
fprintf(KolmoFile,"%e sample time \n",sample_time);
fprintf(KolmoFile,"%f vmin \n",vmin);
fprintf(KolmoFile,"%f vmax \n",vmax);
fprintf(KolmoFile,"%f mean velocity \n",ugem);
fprintf(KolmoFile,"%f rms velocity \n",urms);
fprintf(KolmoFile," \n"); /* another garbage line */
fprintf(KolmoFile,"%e Kolmogorov length \n",eta);
fprintf(KolmoFile,"%e Kolmogorov velocity \n",vK);
fprintf(KolmoFile,"%e Kolmogorov time scale \n",tK);
fprintf(KolmoFile,"%f Kolmogorov frequency \n",fK);
fprintf(KolmoFile,"%f Viscous dissipation rate \n",eps);
fprintf(KolmoFile,"%e Taylor micro scale  \n",lambda);
fprintf(KolmoFile,"%f Reynolds number (lambda) \n",reynolds);
fprintf(KolmoFile,"%f Distance scale \n",scale);
fprintf(KolmoFile,"%f rms derivative velocity \n",durms);
fprintf(KolmoFile,"\nKolmog file created by spectrum.c (TMS320C40)\n");
fprintf(KolmoFile,"%f rms velocity (spectrum, cutoff = %dHz)\n",
urms_spec,spectr_cutoff);
fprintf(KolmoFile,"%f rms derivative velocity (spectrum, cutoff = %dHz)\n",
durms_spec,spectr_cutoff);
fclose(KolmoFile);

return;
}

void main()
{
int i,count;
int align;
int w;

printf ("Reading configuration file \n");
read_cnf_file();

fft_size = sample_size;
half_fft_size = fft_size/2;
MAX_SIZE = 2*fft_size;

/* Allocate arrays from the heap */

sample_data = calloc (sample_size, sizeof (int));
calibr = calloc(CALIBR_SIZE,sizeof(int));
veloc = calloc (sample_size, sizeof(float));
fourier = calloc (MAX_SIZE, sizeof(float));
power = calloc(half_fft_size, sizeof(float));
power_average = calloc (half_fft_size, sizeof(float));
sintable = calloc(half_fft_size, sizeof (float));
hanning = calloc(sample_size, sizeof(float));
hist = calloc (hist_size, sizeof(int));

/* Align the start address of the fourier-array
   so that the first LOG2_FFT_SIZE bits are zero
   This is a requirement of the used FFT-algorithm ! */

align = pow(2,LOG2_FFT_SIZE);
fourier_align = (float *)((fourier + align) -
(float *)((long)fourier & (align-1)));
/* Cache is on: */
asm (" OR 0800h,ST");

flush_adc_fifo();
init_vars();

printf("Making calibration table\n");
make_calibr_table();

printf("Sampling and calculating %d FFT's.\n",NUM_AVERAGE);
count = 1;
while (count<NUM_AVERAGE)
{
read_samples();
histogram();
int_to_veloc();
kolmog(count);
hanning_window();
ffft_rl(fft_size,LOG2_FFT_SIZE,veloc,fourier_align,sintable,BITREV);
make_power();
update_average(count);
count++;
}
spectr_charact(spectr_cutoff);
printf("Writing turbulence data to files.\n");

turbdata_to_files();
exit(0);
}
```

The code below shows the linker command file *spectrum.cmd*, which is needed to link the compiled code from *spectrum.c* with the object libraries.

```
spectrum.obj                /* Object-file name(s) */
forc40.obj
-c                          /* LINK USING C CONVENTIONS(ROM-model)*/
-v40                        /* Create C4x code */
-x                          /* Re-reading of libraries */
-stack 0x2000               /* 8 KWord STACK */
-heap  0x038000             /* 224 KWord HEAP */
-o spectrum.out             /* output file */
-m spectrum.map             /* linker map file */
-l tops.lib                 /* Link with object libraries */
-l comport.lib
-l pace.lib
-l threel.lib
-l rts40r.lib               /* Get Run-Time support (register-based
                               argument passing) */
```

```
/* SPECIFY THE SYSTEM MEMORY MAP */

MEMORY
{
   ROM:       org = 0x000000  len = 0x1000      /* 4 KWord INTERNAL ROM */
   RAM0:      org = 0x2FF800  len = 0x400       /* 1 KWord RAM BLOCK 0 */
   RAM1:      org = 0x2FFC00  len = 0x400       /* 1 KWord RAM BLOCK 1 */
   LocalRAM:  org = 0x300000  len = 0x0040000   /* 256KWord LOCAL SRAM  */
   GlobalRAM: org = 0x80000000  len = 0x0040000 /* 256KWord GLOBAL SRAM */
}
```

```
/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */

SECTIONS
{
   .text:   load = LocalRAM  /* Executable code and float-point consts */
   .cinit:  load = LocalRAM  /* Tables with values for init. var's and consts */
   .const:  load = LocalRAM  /* float-point consts and switch tables */
   .bss:    load = LocalRAM, block 0x10000
                              /* Reserve space for global and static var's */
   .stack:  load = LocalRAM  /* Allocate mem. for system stack */
   .sysmem: load = GlobalRAM /* memory heap used by dynamic memory funct's */
   .data:   load = LocalRAM  /* Assembly-code data section */
   .fftdat: load = LocalRAM  /* FFT routine data section */
   .ffttxt: load = LocalRAM  /* FFT routine code section */
}
```

And the code below is an example of a configuration file *spectrum.cnf*, which passes some constants and filenames to the main program, *spectrum.c*:

```
5000                    # of blocks to sample
40000                   sample frequency
0xF9                    sample rate (hex number for ADC)
32768                   sample size = fft size (max. 32768)
15                      log2 block size = log2 fft size
20000                   spectrum cutoff freq for turb.char.
c:/marcel/data/         prefix for all data filenames
c40disa8.dat            file name calibration input data
spe11812.bin            file name average power-spectrum output data
his11812.dat            file name velocity histogram output data
kol11812.cnf            file name turbulence characteristics output data
```

B.2 Structurator program

The C40 assembler routine for the real-time measurement of velocity difference probability distribution functions (PDFs) is called the Structurator. The assembler routine is named *c40struc.asm* and the call sequence from C environment is:

int c40struc(MCATBL,RELOFS,CALIBR,BUFFER,SIZE)

with the following arguments:

int *MCATBL:	pointer to location of MCA memory block
int *RELOFS:	pointer to table with relative offsets
int *CALIBR:	pointer to location of calibration table
int *BUFFER:	pointer to 1024-size circular buffer
int SIZE:	total number of samples to take before program end
On return:	int R0 = status of ADC FIFO buffer

A detailed description of the Structurator routine is given in section 3.3. The assembler routine *c40struc.asm* is the first file given below.

The Structurator routine is called within the C-program *structur.c*, where the allocation of the arrays is done. The memory positions of these allocated arrays are passed to the Structurator program by means of register-based argument passing, however, stack-based argument passing is optional. After the Structurator is called and has returned to the C-environment, the ADC FIFO buffer status is available to the user and can be checked. In case of FIFO buffer overflow, the sampling frequency has been too high for the Structurator routine and samples were lost. In that case the available MCA memory block is useless and is not saved. In case of no overflow, the MCA memory holding the velocity difference PDFs is saved to disk via binairy transfer, to be processed further.

The next file is the linker command file for linking the assembler routine *c40struc.asm*, the compiled code from *structur.c* and the object libraries. In this file the available DSP processor memory is declared and the allocation of the different program sections into this available memory is specified.

The last file is an example of a configuration file *structur.cnf* and is used to pass some arguments to the *structur.c* program, such as the number af samples to take for the Structurator routine, the externally set sample frequency and some file names.

```
*************************************************************************
*
* Program: c40struc.asm
*
* What?: C40 Assembler routine for C40 Structurator implementation
*
* Authors:
* - Klaas Kopinga (routine for the TI C40 Simulator)
* - Marcel Donker (Transtech C40 implemention and C-callable function)
*
* Last update: 030197
*
* Description:
*
* This routine uses the Transtech TDMB412 DSP motherboard with one TIM-40
* module (TMS320C40) in slot 1 and a TDM431 12-bits ADC in slot 3.
* Comport 4 of the C40 is connected to comport 1 of the ADC. Data is
* sampled from ADC channel 1.
*
* This routine is the DSP implementation of the 'Structurator' at the
* turbulence research group at the Eindhoven Technological University.
*
* Usage: int c40struc(MCATBL,RELOFS,CALIBR,BUFFER,SIZE)
*                      AR2    R2     R3     RC     RS
*
*       int *MCATBL: points to location of MCA memory table
*       int *RELOFS: points to table with relative offsets
*       int *CALIBR: points to location of calibration table
*       int *BUFFER: points to 1024-size buffer
*       int SIZE   : number of samples to take
*
* On return: on return to the C-environment the ADC FIFO status is read
*            and stored into register R0.
*
* The MCA memory locations are calculated by addition of:
*
* - the index in the offset-table (by incrementing R4 with 8191)
* - the (velocity)difference between the moving calibrated sample and
*   the calibrated sample that is located the value of the offset
*   earlier in the circular buffer.
*   (this difference is added with the maximum value of a sample (=4095)
*   to prevent a negative result (sign extend))
* - the startaddress of the MCA memory block.
*
* To avoid register conflicts, groups of 4 offset-values are more or
* less processed interleaved.
*
* Registers in use:
*
* - R0/R3       temporary storing of contents of MCA memory locations
* - R4          value from cal.-table, value + index offset table
* - R5          startaddress MCA memory block
* - R6          startaddress MCA memory block + max value cal.-table
* - R7          value 1 for increment MCA-memory and sample counter
* - R8          # of samples to take before return to C environment
* - R9          counter for the # of samples sofar
* - R10         temporary storing of ADC input value
* - R11         ADC command word
* - AR0/AR3     temporary storing MCA-memory(sub)addresses
* - AR1         startaddress cal.-table
* - AR2         address max value from cal.-table
* - AR4         startaddress circular buffer
* - AR5         index table with relative offsets
* - AR6         index sample in buffer conform offset-table
* - AR7         address input-samples from ADC
* - IR0         relative offset
* - IR1         index cal.-table, relative offset
* - DP, BK, RC
*************************************************************************

                .file   "c40struc.asm"

FP              .set    AR3

                .global _c40struc       ; ENTRY EXECUTION POINT

                .global INIT01,FILL,NXTSMP,STRUCT,END

MCATBL:         .usect  ".strdat",1     ; RESERVE MEMORY FOR ARGUMENTS
RELOFS:         .usect  ".strdat",1
CALIBR:         .usect  ".strdat",1
BUFFER:         .usect  ".strdat",1
SIZE:           .usect  ".strdat",1

                .data

adc             .word   000100080h      ; CPCR4 TO ADC-COMPORT
cmd1            .word   000000033h      ; ADC COMMAND WORD 1 =
                                        ; SETUP EXTERNAL TRIGGERING BY
                                        ; SENDING A DUMMY VALUE TO THE
                                        ; SAMPLING INTERVAL COUNTER AND
                                        ; READING THE STATUS OF THE FIFO
```

```
                                        ; THIS COMMAND WILL ALSO EMPTY
                                        ; THE ADC FIFO!
cmd2            .word   000000032h      ; ADC COMMAND WORD 2 = EXTERNAL
                                        ; TRIGGER, 1 SPW, SINGLE WORD
                                        ; READ, CHANNEL 1
stop            .word   000000041h      ; COMMAND WORD FOR STOP SAMPLING
maxval          .word   4095            ; MAX. VALUE IN CALIBR.TABLE
                                        ; (ADC = 12BITS)

                .asg    AR7,ADC         ; REGISTER ALLOCATIONS FOR
                                        ; ADC I/O
                .asg    *+AR7(1),ADC_IN
                .asg    *+AR7(2),ADC_OUT
                .asg    R11,CMD

;
; INITIALISE C FUNCTION
;
                .sect   ".strtxt"       ; LOCATE INTO .strtxt

_c40struc:      PUSH    FP              ; PRESERVE C ENVIRONMENT
                LDI     SP,FP
                PUSH    R4
                PUSH    R5
                PUSH    R6
                PUSHF   R6
                PUSH    R7
                PUSHF   R7
                PUSH    R8
                PUSH    AR4
                PUSH    AR5
                PUSH    AR6
                PUSH    AR7
                PUSH    DP

                LDP     MCATBL          ; DATAPAGE -> .strdat SECTION

                .if .REGPARM == 0       ; ARGUMENTS PASSED IN STACK
                LDA     *-FP(2),AR2
                LDI     *-FP(3),R2
                LDI     *-FP(4),R3
                LDI     *-FP(5),RC
                LDI     *-FP(6),RS
                .endif
```

```
                STI     AR2,@MCATBL     ; STORE PASSED ARGUMENTS IN VAR'S
                STI     R2,@RELOFS      ; FOR REGISTER BASED MODEL
                STI     R3,@CALIBR
                STI     RC,@BUFFER
                STI     RS,@SIZE

INIT01:         LDA     1024,BK         ; 1024 IN BLOCKSIZE REGISTER
                LDI     @MCATBL,R5      ; STARTADDRESS MCA MEM. BLOCK
                LDA     @RELOFS,AR5     ; STARTADDRESS OFFSET TABLE
                LDA     @CALIBR,AR1     ; STARTADDRESS CALIBR.TABLE
                LDA     @BUFFER,AR4     ; STARTADDRESS CIRCULAR BUFFER
                LDI     @SIZE,R8        ; # OF SAMPLES TO TAKE IN R8
                LDI     @maxval,R1      ; MAXVAL INTO R1;
                LDI     0,R9            ; SET SAMPLE COUNTER TO 0
                LDI     1,R7            ; INCREMENT FOR MCA AND COUNTER
                ADDI3   R1,R5,R6        ; MAX. CALIBR.TABLE+MCAMEM IN R6
                LDI     @adc,ADC        ; INIT. REGISTER FOR ADC-COMPORT
                LDI     @cmd1,CMD       ; INIT. REGISTER FOR CMD-WORD 1
                STI     CMD,ADC_OUT     ; SEND COMMAND WORD 1 TO ADC
                                        ; (ENABLE EXT. TRIGGER + EMPTY
                                        ; FIFO)
                LDI     ADC_IN,R10      ; READ BACK STATUS OF FIFO IN R10
                LDI     *AR4--%,R0      ; AR4 POINTS TO OLDEST VALUE IN
                                        ; CIRCULAR BUFFER (R0 = DUMMY)
                LDI     @cmd2,CMD       ; INITIALIZE REGISTER FOR
                                        ; CMD-WORD 2
                LDI     1023,RC         ; LOAD REPEAT COUNTER = 1024
                RPTB    FILL            ; REPEAT BLOCK 'FILL'
                STI     CMD,ADC_OUT     ; SEND COMMAND WORD 2 TO ADC
                                        ; (SINGLE WORD READ)
                LDI     ADC_IN,R10      ; LOAD VALUE FROM ADC INTO R10
                AND     0FFFFh,R10      ; 1 SPW -> LOGICAL AND WITH FFFFh
                LDA     R10,IR1         ; ADC VALUE IN IR1, SAMPLE IS
                                        ; USED AS INDEX FOR THE CAL-TABLE
                LDI     *+AR1(IR1),R4   ; VALUE CALTABLE CALIBR[SAMPLE]
                                        ; IN R4, THIS IS VELOCITY VALUE
FILL:           STI     R4,*AR4--%      ; STORE CALIBR[SAMPLE] INTO
                                        ; CIRCULAR BUFFER
                ADDI    1024,R9         ; 1024 INPUT VALUES ALREADY TAKEN
                STI     CMD,ADC_OUT     ; SEND COMMAND WORD 2 TO ADC
                                        ; (SINGLE WORD READ)
                LDI     ADC_IN,R10      ; THIS IS THE FIRST SAMPLE
                AND     0FFFFh,R10      ; 1 SPW -> LOGICAL AND WITH FFFFh
                LDA     R10,IR1         ; ADC-VALUE IN IR1, SAMPLE-VALUE
                                        ; IS USED AS INDEX FOR CAL.TABLE
                ADDI    R7,R9           ; INCREMENT SAMPLE-COUNTER WITH 1
```

```
NXTSMP:     LDA     *AR5,IR0        ; 1ST INDEX TABLE WITH RELATIVE
                                    ; OFFSETS ([0]) IN IR0
            LDA     AR4,AR6         ; AR6 POINTS (AGAIN) TO RUNNING
                                    ; INDEX IN CIRCULAR BUFFER
            LDI     7,RC            ; 8 DIFFERENT BLOCKS OF 4 OFFSET
                                    ; VALUES
            LDI     *+AR1(IR1),R4   ; VALUE CALTABLE CALIBR[SAMPLE]
                                    ; IN R4, THIS IS VELOCITY-VALUE
||          LDI     *AR6++(IR0)%,R0 ; AR6 POINTS TO SAMPLE WITH 1ST
                                    ; RELATIVE OFFSETS ([0]) IN THE
                                    ; CIRCULAR BUFFER, %=CIRCULAR
                                    ; ADDRESSING, R0 = DUMMY REGISTER
            RPTBD   STRUCT          ; REPEAT BLOCK (STRUCT) DELAYED
                                    ; THE 1ST 3 INSTRUCTIONS ARE
                                    ; NOT(!) PART OF THE LOOP,
                                    ; (PIPELINE FLUSH)
            STI     R4,*AR4--%      ; PUT CALIBR[SAMPLE] INTO
                                    ; CIRCULAR BUFFER, OLDEST SAMPLE
                                    ; IS OVERWRITTEN
            LDA     *++AR5,IR0      ; NEXT RELATIVE OFFSET([1+4*J])
                                    ; IN IR0
            ADDI    R6,R4           ; CALMAX+CALIBR[SAMPLE]+MCAMEM
                                    ; IN R4 = MCA MEMORY LOCATION
                                    ; LOOP BEGINS AT NEXT INSTRUCTION
            LDA     *++AR5,IR1      ; NEXT RELATIVE OFFSET([2+4*J])
                                    ; IN IR1
            SUBI3   *AR6++(IR0)%,R4,AR0
                                    ; DECREASE R4 WITH VALUE OF
                                    ; SAMPLE FROM CIRCULAR BUFFER,
                                    ; STORE RESULT IN AR0 AND SET AR6
                                    ; ONTO NEXT SAMPLE IN CIRCULAR
                                    ; BUFFER CONFORM THE OFFSET-TABLE
            LDA     *++AR5,IR0      ; RELATIVE OFFSET ([3+4*J]) IN
                                    ; IR0
            ADDI    8191,R4         ; R4+=8191 (INDEX OFFSET-TABLE+1)
            ADDI3   R7,*AR0,R0      ; READ MCA MEMORY LOCATION,
                                    ; INCREMENT WITH 1, STORE INTO R0
            SUBI3   *AR6++(IR1)%,R4,AR1
                                    ; DECREASE R4 WITH VALUE OF
                                    ; SAMPLE FROM CIRCULAR BUFFER,
                                    ; STORE RESULT IN AR1 AND SET AR6
                                    ; ONTO NEXT SAMPLE IN CIRCULAR
                                    ; BUFFER CONFORM THE OFFSET TABLE
            LDA     *++AR5,IR1      ; RELATIVE OFFSET ([4+4*J]) IN
                                    ; IR1
            ADDI    8191,R4         ; R4+=8191(INDEX OFFSET-TABLE+1)
            ADDI3   R7,*AR1,R1      ; READ MCA MEMORY LOCATION,
                                    ; INCREMENT WITH 1, STORE INTO R1
||          STI     R0,*AR0         ; UPDATE THE LAST MCA LOCATION
            SUBI3   *AR6++(IR0)%,R4,AR2
                                    ; DECREASE R4 WITH VALUE OF
                                    ; SAMPLE FROM CIRCULAR BUFFER,
                                    ; STORE RESULT IN AR0 AND SET AR6
                                    ; ONTO NEXT SAMPLE IN CIRCULAR
                                    ; BUFFER CONFORM THE OFFSET-TABLE
            ADDI    8191,R4         ; R4+=8191(INDEX OFFSET-TABLE+1)
            SUBI3   *AR6++(IR1)%,R4,AR3
                                    ; DECREASE R4 WITH VALUE OF
                                    ; SAMPLE FROM CIRCULAR BUFFER,
                                    ; STORE RESULT IN AR0 AND SET AR6
                                    ; ONTO NEXT SAMPLE IN CIRCULAR
                                    ; BUFFER CONFORM THE OFFSET-TABLE
            ADDI3   R7,*AR2,R2      ; READ MCA MEMORY LOCATION,
                                    ; INCREMENT WITH 1, STORE INTO R2
||          STI     R1,*AR1         ; UPDATE THE LAST MCA LOCATION
            ADDI    8191,R4         ; R4+=8191(INDEX OFFSET-TABLE+1)
            LDA     *++AR5,IR0      ; RELATIVE OFFSET ([1+4*(J+1)])
                                    ; IN IR0
            ADDI3   R7,*AR3,R3      ; READ MCA MEMORY LOCATION,
                                    ; INCREMENT WITH 1, STORE INTO R3
            STI     R2,*AR2         ; UPDATE THE LAST MCA LOCATION
STRUCT:     STI     R3,*AR3         ; UPDATE THE LAST MCA LOCATION
            LDA     @CALIBR,AR1     ; RELOAD AR1 WITH STARTADDR
                                    ; CALIBR.TABLE
            LDA     @RELOFS,AR5     ; RELOAD AR5 WITH BEGIN
                                    ; OFFSET-TABLE
            CMPI    R8,R9           ; CHECK FOR END OF PROGRAM
            BZ      END             ; GOTO END OF PROGRAM
            STI     CMD,ADC_OUT     ; SEND COMMAND WORD 2 TO ADC
                                    ; (SINGLE WORD READ)
            LDI     ADC_IN,R10      ; THIS IS THE NEXT SAMPLE
            BD      NXTSMP          ; BRANCH DELAYED TO NXTSMP,
                                    ; BUT FIRST DO 3 INSTRUCTIONS
            AND     0FFFFh,R10      ; 1 SPW -> LOGICAL AND WITH FFFFh
            LDA     R10,IR1         ; ADC-VALUE IN IR1, SAMPLE-VALUE
                                    ; IS USED AS INDEX FOR CAL.TABLE
            ADDI    R7,R9           ; INCREMENT SAMPLE-COUNTER WITH 1
;
; RETURN TO C ENVIRONMENT
;
```

```
END:            LDI     @stop,CMD       ; INITIALIZE REGISTER FOR STOP
                                        ; COMMAND WORD
                STI     CMD,ADC_OUT     ; SEND COMMAND WORD TO ADC (STOP
                                        ; SAMPLING)
                LDI     ADC_IN,R0       ; READ BACK THE FIFO STATUS AND
                                        ; RETURN VALUE TO C ENVIRONMENT
                POP     DP              ; RESTORE C ENVIRONMENT VARIABLES
                POP     AR7
                POP     AR6
                POP     AR5
                POP     AR4
                POP     R8
                POPF    R7
                POP     R7
                POPF    R6
                POP     R6
                POP     R5
                POP     R4
                POP     FP
                RETS

                .end

*
* no more
*
*******************************************************************************
```

Main program to control the flow of data to and from the Structurator assembler routine and the PC: *structur.c*.

```
/*
*
* Program: structur.c
*
* Author: Marcel Donker
*
* Last update: 030197
*
* Description:
*
* C40 program in C-language for the implementation of the structurator.
* In this program the offset table and the calibration table are read
* from file and it uses the C-callable routine 'c40struc.asm' to start
* the structurator routine.
* After return to the C environment the ADC FIFO is checked for overflow,
* and the MCA table is written to file.
*
* The file names and the # of samples to take are read from the
* configuration file structur.cnf
*/

#include <stdio.h>
#include <stdlib.h>
#include <link.h>
#include <tops.h>
#include <errno.h>
#include <string.h>
#include <ctype.h>

#define MCAMEM_SIZE     8192*32         /* MCA table size in words */
#define OFFSET_SIZE     32              /* Offset table size in words */
#define IJKTBL_SIZE     4096            /* Calibr. table size in words */
#define BUFTBL_SIZE     1024            /* Buffer size in words */
#define ALIGN_SIZE      2500            /* Length needed for declaring
                                         * enough space to align the
                                         * buffer startaddress to an
                                         * 11 bit boundary */

#define ADC_FIFO_NOT_FULL               0x4

#pragma DATA_SECTION (mca_ptr,".mcasect") /* Data section .mcasect
                                           * allocated in linker */

int mca_ptr;
int *mcamem = (void *)&mca_ptr;
int *offset;
int *ijktbl;
int *buftbl;
int *buftbl_align;
int INPUT_SIZE;
int sample_freq;
char prefix[80];
char filename_1[80];
char filename_2[80];
char filename_3[80];
char filename[80];

void read_cnf_file(void)
{
```

```
int i;
char dummy[60];
FILE *CNFFile;

printf ("Reading the configuration file\n");
CNFFile = fopen ("c:/marcel/struct/structur.cnf","r");
if (CNFFile == NULL)
{
printf("Unable to read configuration file!\n");
printf("Error-message = %s \n",strerror(errno));
exit(0);
}
fscanf(CNFFile,"%d",&INPUT_SIZE);
fgets(dummy,60,CNFFile);
fscanf(CNFFile,"%d",&sample_freq);
fgets(dummy,60,CNFFile);
fgets(prefix,16,CNFFile);
fgets(dummy,60,CNFFile);
fgets(filename_1,13,CNFFile);
fgets(dummy,60,CNFFile);
fgets(filename_2,13,CNFFile);
fgets(dummy,60,CNFFile);
fgets(filename_3,13,CNFFile);
fgets(dummy,60,CNFFile);

fclose(CNFFile);
return;
}

void empty_arrays(void)
{
int i;
int *ptr;

/* Empty some arrays to be sure, before we continue */


ptr = mcamem;
for (i=0;i<MCAMEM_SIZE;i++)
*ptr++ = (int)0;
ptr = offset;
for (i=0;i<OFFSET_SIZE;i++)
*ptr++ = (int)0;
ptr = ijktbl;
for (i=0;i<IJKTBL_SIZE;i++)
*ptr++ = (int)0;
ptr = buftbl;
for (i=0;i<BUFTBL_SIZE;i++)
*ptr++ = (int)0;
}

void read_offset_table(void)
{
int count;
int *ptr;
FILE *OffsetFile;

printf ("Reading offset-table from file.\n");

strcpy(filename,prefix);
strcat(filename,filename_1);
OffsetFile = fopen (filename,"r");
if (OffsetFile == NULL)
{
printf("Unable to read offset file\n");
printf("Error-message: %s \n",strerror(errno));
exit(0);
}
ptr = offset;
for (count=0;count<OFFSET_SIZE;count++)
{
fscanf(OffsetFile,"%d",&(*ptr));
ptr++;
}
fclose (OffsetFile);
return;
}

void read_calibr_table(void)
{
int count,i;
int *ptr;
int dummy;
FILE *CalibrFile;

printf ("Reading calibration-table from file.\n");

strcpy(filename,prefix);
strcat(filename,filename_2);
CalibrFile = fopen (filename,"r");
if (CalibrFile == NULL)
{
```

```
printf("Unable to read calibration file\n");
printf("Error-message: %s \n",strerror(errno));
exit(0);
}
ptr = ijktbl;
for (count=0;count<IJKTBL_SIZE;count++)
{
fscanf(CalibrFile,"%d %d",&dummy,&(*ptr));
ptr++;
}
fclose (CalibrFile);
return;
}

void mcatbl_to_file(void)
{
int i;
int *ptr;
FILE *MCAFile;

/* Write the structurator MCA table and the
 * offset tabel to file (binary).
 */

printf("Writing MCA-table to file.\n");

strcpy(filename,prefix);
strcat(filename,filename_3);
MCAFile = fopen(filename,"wb");
if (MCAFile == NULL)
{
printf("Unable to write MCA table to file\n");
printf("Error-message: %s \n",strerror(errno));
exit(0);
}
fwrite((void *)offset,sizeof(int),OFFSET_SIZE,MCAFile);
ptr = mcamem;
for (i=0;i<32;i++)
{
fwrite((void *)ptr,sizeof(int),(MCAMEM_SIZE/32-1),MCAFile);
ptr += (MCAMEM_SIZE/32-1);
}
fclose (MCAFile);
return;
}
```

```
void main()
{
int i;
int value;
int status;
char a;
float time;

printf ("Program start. \n");

/* Allocate 3 array's from the heap: */

offset = calloc (OFFSET_SIZE, sizeof (int));
ijktbl = calloc (IJKTBL_SIZE, sizeof (int));
buftbl = calloc (ALIGN_SIZE, sizeof (int));

if ((offset == NULL)||(ijktbl == 0)||(buftbl == NULL))
printf ("Not enough heap memory! \n");

/* Align the address of the circular buffer so that the first
 * 11 bits of the address are zero.
 * This is needed for circular addressing.
 * (see TI TMS320C4x UG ,pp 5-25)
 */

buftbl_align = (int *)((buftbl + 0x800) - (int *)((long)buftbl &
  0x7ff));
read_cnf_file();
empty_arrays();
read_offset_table();
read_calibr_table();

/* Cache is on: */
asm (" OR 0800h,ST");

time = (float)((INPUT_SIZE*1.0)/(sample_freq*60.0));
printf ("Running structurator routine (time = %5.1f min @ %dKHz). \n",time,sample_freq/1000);
value = c40struc(mcamem,offset,ijktbl,buftbl_align,INPUT_SIZE);
status = value & 7;
if (!(status & ADC_FIFO_NOT_FULL))
printf("WARNING: adc fifo full!, NOT writing mca-table
  to file. \n");
else
{
mcatbl_to_file();
printf("Program done.\n");
```

```
}
exit(0);
}
```

Linker command file *structur.cmd* needed to link the compiled program *structur.c* and the assembler Structurator routine *c40struc.asm.*

```
structur.obj              /* Object-file name(s) */
c40struc.obj
-c                        /* LINK USING C CONVENTIONS(ROM-model)*/
-v40                      /* Create C4x code */
-x                        /* Rereading of libraries */
-stack 0x8000             /* 32 KWord STACK */
-heap  0x8000             /* 32 KWord HEAP */
-e main                   /* Define entry point */
-o structur.out           /* output file */
-m structur.map           /* Linker map file */
-l tops.lib               /* Link with object libraries */
-l comport.lib
-l pace.lib
-l threel.lib
-l rts40r.lib             /* Run-Time support (register-based
                             argument passing) */


/* SPECIFY THE SYSTEM MEMORY MAP */

MEMORY
{
   ROM:       org = 0x000000  len = 0x001000   /* 4 KWord INTERNAL ROM */
   RAM0:      org = 0x2FF800  len = 0x00400    /* 1 KWord RAM BLOCK 0 */
   RAM1:      org = 0x2FFC00  len = 0x00400    /* 1 KWord RAM BLOCK 1 */
   LocalRAM:  org = 0x300000  len = 0x0040000  /* 256 KWord LOCAL SRAM */
   GlobalRAM: org = 0x80000000  len = 0x0040000/* 256 KWord GLOBAL SRAM*/
}

/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */

SECTIONS
{
   .text:    > LocalRAM    /* Executable C code and float-point consts */
   .cinit:   > LocalRAM    /* Values for init. var's and consts*/
   .const:   > LocalRAM    /* float-point consts and switch tables*/
   .bss:     > LocalRAM    /* Reserve space for C global + static var'*/
   .stack:   > LocalRAM    /* Allocate mem. for system stack */
   .sysmem:  > LocalRAM    /* memory heap used by dynamic memory functions */
   .data:    > LocalRAM    /* data section */
   .strdat:  > LocalRAM    /* Structurator data section */
   .strtxt:  > LocalRAM    /* Structurator code section */
   .mcasect: > GlobalRAM block 40000h
                           /* MCA table section = 8192*32 words */
}
```

Example of a configuration file *structur.cnf*, which passes some constants and some filenames to the program *structur.c.*

```
1008000000          number of samples to take
40000               sample frequency (external)
c:/marcel/data/     prefix for all data filenames
off1000c.dat        file name offset table
calibr.dat          file name calibr. table
mca11612.bin        file name MCA table


That's all folks
```

Bibliography

[1] Batchelor, G.K., *Pressure fluctuations in isotropic turbulence*, Proc.Camb.Philos.Soc **47**, 359, (1951).

[2] Bradshaw, P., *An introduction to turbulence and its measurement*, Pergamon Press, Oxford, (1971).

[3] Brigham, E.O. and Cliffs, E., *The Fast Fourier Transform*, Prentice-Hall, England, (1974).

[4] Borue, V. and Orszag, S.A., *Forced three-dimensional homogeneous turbulence with hyperviscosity*, Europhys.Lett **29**, 687, (1995).

[5] Bruun, H.H., *Hot-wire anemometry; principles and signal analysis*, Oxford University Press, USA, (1995).

[6] Falkovich, G., *Bottleneck phenomenon in developed turbulence*, Phys.Fluids **6**, 1411, (1994).

[7] Frisch, U., *Turbulence: the legacy of A.N. Kolmogorov*, Cambridge University Press, USA, (1995).

[8] Galen, R.A.M.L. van, *Design of a modular multichannel hot-wire anemometer system*, Institute for Continuing Education, Eindhoven University of Technology, (1994).

[9] Grossmann, S., *Asymptotic dissipation range in turbulence*, Phys.Rev.E **51**, 6275, (1995).

[10] Grossmann, S. and Lohse, D., *Universality in fully developed turbulence*, Phys.Rev.E **50**, 2784, (1994).

[11] Herweijer, J.A., *The small scale structure of turbulence* Eindhoven University of Technology, Ph.D Thesis, Eindhoven, The Netherlands (1995).

[12] Herweijer, J.A., van Nijmweegen, F.C., Kopinga, K., Voskamp, J.H. and Water, W., *A digital device for measuring high-order structure functions*, Rev.Sci.Instrum. **65**, 1786, (1994).

[13] Ifeachor, E.C. and Jervis, B.W., *Digital signal processing: a practical approach* (Electronic systems engineering series), Addison-Wesley, Amsterdam, The Netherlands, (1993).

[14] Kolmogorov, A.N., *The local structure of turbulence in incompressible viscous fluid for very large Reynolds numbers*, Dokl.Akad.Nauk SSSR **30**, 4, (1941).

[15] Kolmogorov, A.N., *A refinement of previous hypotheses concerning the local structure of turbulence in a viscous incompressible fluid at high Reynolds number*, J.Fluid Mech. **13**, 82, (1962).

[16] Kopinga, K., Eindhoven University of Technology. Private Communication. Developed the basic Structurator routine for the Texas Instruments TMS320C40 simulator.

[17] Landau, L.D. and Lifshitz, E.M., *Fluid Mechanics*, 2nd ed. (Course of theoretical physics, v. 6), Pergamon Press, Oxford, England (1989).

[18] Lohse, D. and Müller-Groeling, A., *Bottleneck effects in turbulence: Scaling phenomena in r versus p space*, Phys.Rev.Lett. **74**, 1747, (1995).

[19] Lohse, D. and Müller-Groeling, A., *Anisotropy and scaling corrections in turbulence*, Phys.Rev.E **54**, 395, (1996).

[20] Monin, A.S. and Yaglom, A.M., *Statistical fluid mechanics; Mechanics of Turbulence.* Volume 2, MIT Press, Cambridge, Massachusetts, London, England (1975).

[21] Oboukov, A.M., *Some specific features of atmospheric turbulence*, J.Fluid Mech. **13**, 77, (1962).

[22] Richardson, L.F., *Weather prediction by numerical process*, Cambridge University Press, England, (1922).

[23] She, Z.S. and Jackson, E., *On the universal form of energy spectra in fully developed turbulence*, Phys.Fluids A **5**, 1526, (1993).

[24] Sirovich, L., Smith, L. and Yakhot, V., *Energy spectrum of homogeneous and isotropic turbulence in far dissipation range*, Phys.Rev.Lett. **72**, 344, (1994).

[25] Stolovitzky, G., Sreenivasan, K.R. and Juneja, A., *Scaling functions and scaling exponents in turbulence*, Phys.Rev.E **48**, R3217, (1993).

[26] Tennekes, H. and Lumley, J.L., *A first course in turbulence*, MIT Press, Cambridge, London, England (1972).

[27] Texas Instruments, *TMS320C4x User's Guide, Digital Signal Processor*, Texas Instruments, USA (1993).