Development of a controller and a synchronization-algorithm for a light tracker

Development of a controller and a synchronization-algorithm for a light tracker

D.J.H. Bruijnen
DCT-report 2002.47

Eindhoven, October 2002

D.J.H. Bruijnen, student id. 458350
External traineeship in Denmark

supervisors:
Jakob Stoustrup, Anders la Cour-Harbo
Aalborg University
Department of Control Engineering

Abstract

There are a lot of applications where following a light spot is the objective. Every application has its own specification. In this case a infrared light source has to be followed. The tracker consists of two light sensors with a screen between. This can be controlled using the intensity difference of both sensors. The signal to noise ratio will be very bad. Possible disturbances are artificial light, sunlight and maximally 15 other similar infrared light sources which also have to be tracked by other devices. The intensity of the disturbances will vary a lot and sometimes the infrared light source will even be blocked. The goal is to track the source as good as possible and also it must be possible to send 19bit of data from the emitter to the receiver within 0,1s.

To achieve this there will be made use of the Rudin-Shapiro Transform (RST). This has some very nice properties which are useful here. The receiver application will be able to distinguish specific Rudin-Shapiro sequences (RSS's). These are made by transforming a vector with one one at a specific place (the RS-point) and the rest zeros. The length of this vector is a power of 2. The RST has the property to spread the data over the whole spectrum. When adding disturbances and then transforming it back they will still be spread over the whole range. Only the RS-point of the present RSS will show up. This method can be used in bad signal to noise ratio environments.

A controller has been made to get a good tracking of the light source. A standard PID-controller was not able to provide a high accuracy together with a sufficient bandwidth. Some nonlinear elements had to be included to improve this.

Also there is a problem that the receiver and emitter are not synchronized. The applications run at different clocks so there will be frequency difference and frequency drift. An algorithm has been developed to find and keep on tracking the present Rudin-Shapiro sequence. Data can be sent by changing the RS-point at the emitter side. The receiver application will notice those changes and it will recognize if a full 19bit command has been sent.

Contents

Introduction and problem formulation

Pointing towards a moving object has a lot of applications. One of them is the tracking of performers by stage lights. An illustration of this can be seen in figure 1. Some possible disturbances are artificial light, sunlight, light blocking, sensor noise and frequency drift of the emitter and receiver. Light blocking happens when a performer walks in front of another. The problem of frequency drift arises because the emitter and receiver can not be synchronized using a cable between them. The applications of the emitter and receiver run on different clocks which are never exactly the same. Because most of the time there are more performers on the stage it must be possible to use several similar pointing devices in the same room at the same time. Specific sequences will be emitted to make the light sources unique so they can be distinguished from each other. Finally it must be possible to sent a 19bit command with a maximum reaction time of 0,1s. To overcome all these design problems the Rudin Shapiro Transform will be used to get a robust tracking. An experimental setup has been made to check the feasibility.

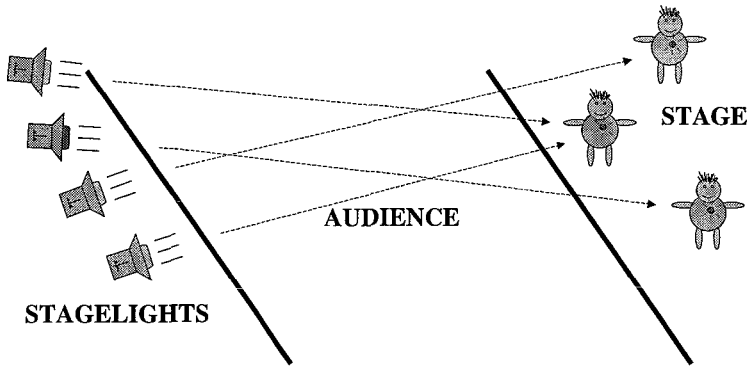


Figure 1: The application

The objective consists of two parts. Firstly a controller has to be designed for the light tracker to achieve an as high as possible bandwidth. Secondly the specific sequence has to be found in the signal. To achieve this an algorithm has to be developed which will be implemented in the light tracker application. If the light emitter and the light tracker are not synchronized it is impossible for the light tracker to follow the light or receive a 19bit command. If the whole application is properly working it will be implemented in a DSP, but that's not a part of this research.

First the experimental setup will be examined and a model of the system will be made. Next a controller will be designed and implemented in the application (written in C++). The second part is about the synchronization. First an algorithm will be developed in Matlab and after that it will be implemented in the tracker application. The emitter application must also be changed so it can send commands.

Chapter 1

The experimental set-up

1.1 Overview

The experimental set-up consists of two units; the light tracker and the light source. In figure 1.2 some photos of the experimental set-up are shown. Globally the electromechanical system consists of an electromotor, a gear reduction and a platform which can rotate around the vertical axis. At the platform two light sensors are mounted with a screen in between. These sensors are sensitive to infrared light. A change of light fall will result in a little resistance change. By amplifying this it can be observed. If the light tracker is not pointing straight to the light source one light sensor will receive less light than the other one. This feature will be used to control the motor so it will follow the light source. In figure 1.1 a schematic view of the set-up is shown.

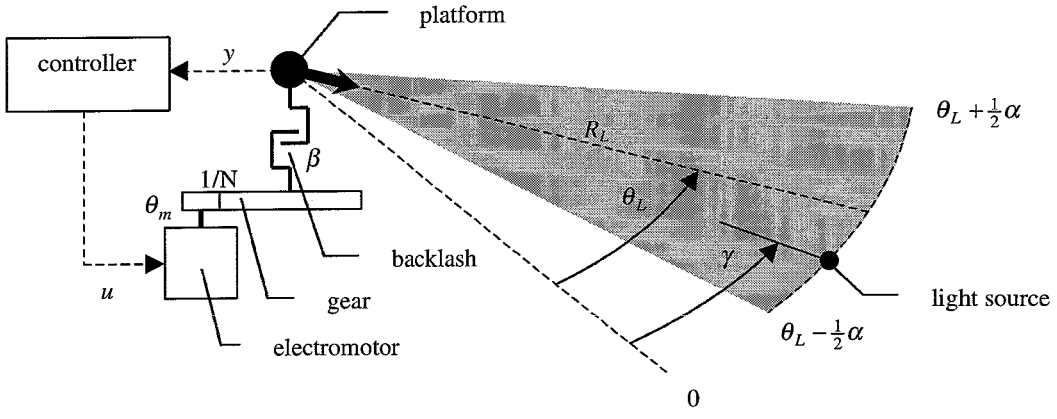


Figure 1.1: Visual illustration of the model

Further a laser is mounted at the platform to visualize what direction the platform is pointing. A schematic view is drawn in figure 1.3.

The laser diode is fed with 5V by a power supply. On top of the platform there are three connections. The middle is fed with 2V by another power supply. The outside connections are the outputs of both amplified sensor signals. The changes induced by the light sensors are amplified with transistor amplifiers. All these electronics are gathered together on a print.

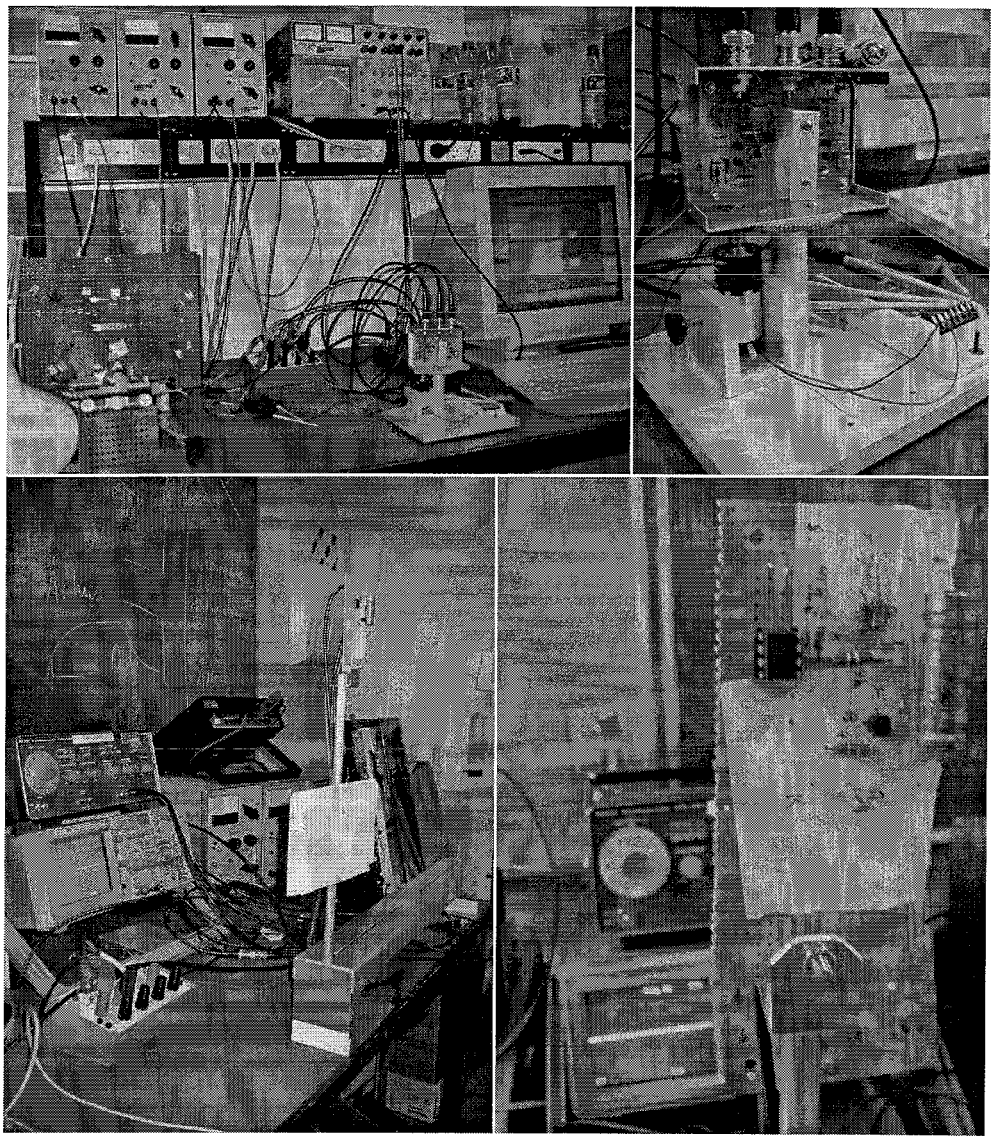


Figure 1.2: Pictures of the experimental setup

The output signals go to a data acquisition device which is connected to a PC. At the PC there's running an application made in Microsoft Visual C++ 6. The maximum sample rate of the data acquisition device is 1MHz. An amount of samples will be read out each time. The PC will decode the coded signal using the Rudin-Shapiro Transform (see Section 4.1) and returns a value related to the light-intensity. Unfortunately there are a lot of disturbances coming from other light sources such as lamps, the sun and possible other Rudin-Shapiro sequences. The Rudin-Shapiro Transform already eliminates a lot of disturbances. There will also be some denoising using an algorithm for removing polynomial contents. This is very useful to get rid of an offset and disturbances like a 100Hz sine created by artificial light.

After all filtering a value is obtained related to the light intensity received by both sensors. The difference between these signals will be used to determine an input for the motor. Also

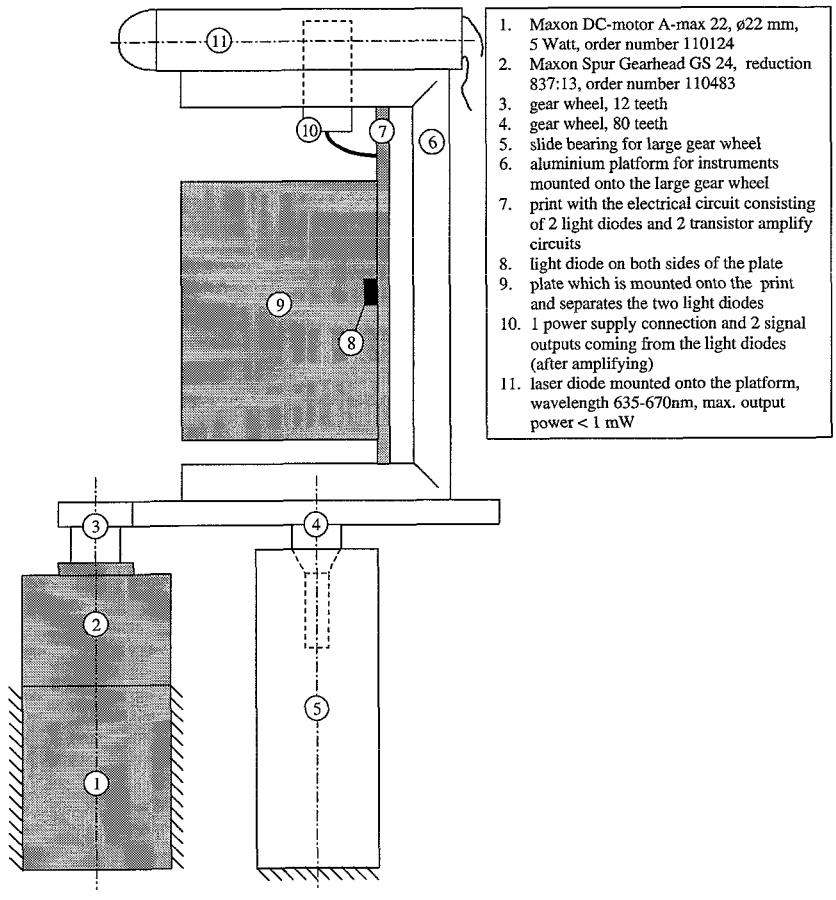


Figure 1.3: Schematic view of the light tracker

here an amplifier is used to provide the motor with enough power. It is an amplifier which controls the voltage and does have a current limiter. Now the loop is closed and a controller has to be designed to get a good tracking performance.

Chapter 2

Modelling of the system

2.1 System overview

The schematic view of the setup shown in figure 1.1 has been used to make a model. This has been implemented in simulink which can be seen in figure 2.1. Choices of the parameters will be discussed in the next sections.

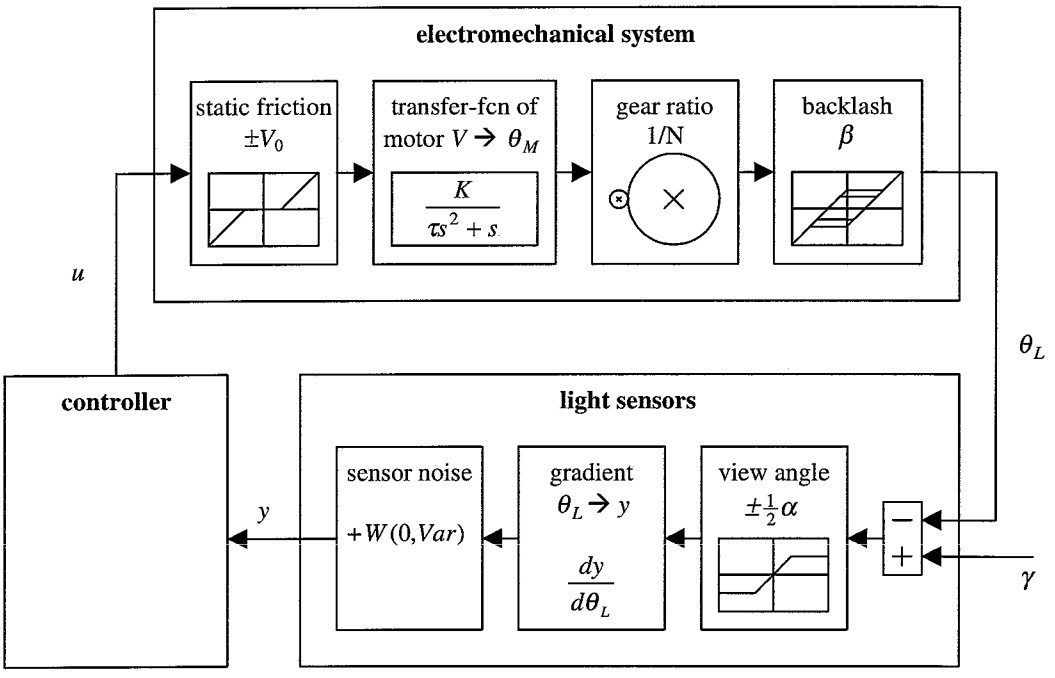


Figure 2.1: Developed model of the system

2.2 Using data from the data sheets

At first the setup will be modelled in Simulink. The data sheets of the electromotor (order number: 110124) and the gear (order number: 110483) can be found in appendix A. The

small and the big gear wheel have respectively 12 and 80 teeth. Together with the gear box mounted on the motor the total gear reduction is: $837/13 \cdot 20/3 = 429,23$. The total backlash of the platform has been measured by using the laser pointing to a screen at 3 meters distance. The backlash results in a 0,025 m movement. This corresponds to 8,3 mrad backlash of the platform. Using this and the data from the data sheet a model was made in the manner of described in [1], the usual way to make a model of a DC motor. The non-linearities (backlash, current limiter, static friction) were also included. When comparing the results of the simulation with experiments they did not agree. The problem was that the resistance changes with the angular velocity. Using only the terminal resistance a much too high current showed up in the simulation. Further there were some numerical problems because of the fast electrical pole. Then it was decided to simplify the model. The electrical pole is much faster than the mechanical pole so it is removed. This is allowed because the electrical pole is much faster than the maximal achievable bandwidth. This pole will only be of importance for higher frequencies. Secondly the model will be totally based on experiments, so the simulation will approximate the non-ideal setup quite good. The structure of the model can be seen in figure 2.1 and a visualization of this can be seen in figure 1.1.

2.3 Using measurement data

Now all parameters have to be determined with experiments. Unfortunately the setup does not have an encoder so a Bode plot is not easily made. Also because of the non-linearities another way has to be found. So it was chosen to put a constant voltage and count the amount of revolutions and keeping up the time using a timer. A very primitive way, but the result of this is quite good. In figure 2.2 you see the angular velocity as a function of the input voltage to the amplifier. The dead zone is caused by the breakaway torque which is a function of cogging (changes in magnetic circuit reluctance), brush friction and bearing friction. This all results in a dead zone of 0,977 V and will be used in the model to include these features. After that dead zone the relation between angular velocity and voltage is linear. The gradient K is equal to 94,8 rad/s/V. In the second plot of figure 2.2 you see the generated torque by the motor as a function of the angular velocity. This was measured simultaneously by measuring the voltage drop over a sequentially connected resistor of 1 Ohm. The obtained current was multiplied by the torque constant which can be found in appendix A. Because the resistance is very low when standing still a high current is needed to get the motor in motion. According to the data sheet this is 1320 mA, but during experiments this can not be seen, because this peak is very short. The resistance increases rapidly after breaking away. The rest of the plot shows the total viscous friction of the system which is $1,28 \cdot 10^{-7}$. This number is not used because the amplifier controls the voltage and not the current. This will not be needed if the current will not be saturated. Since there is a large gear reduction and no external torque is applied to increase the load, a current saturation will not show up.

The gained data of figure 2.2 is only for constant angular velocities. The system has also inertia. This has been determined by applying a block function as an input for the motor. It looks like a first order step responds with a time constant of 0,1 s. This step responds can be seen in section 2.4 figure 2.9.

Next the sensor noise has to be determined. This is strongly dependent on the update frequency of the motor input. In figure 2.3 you see for three frequencies the sensor noise. The blue and green data are respectively rotated the platform 30 cm to the right and 30 cm to

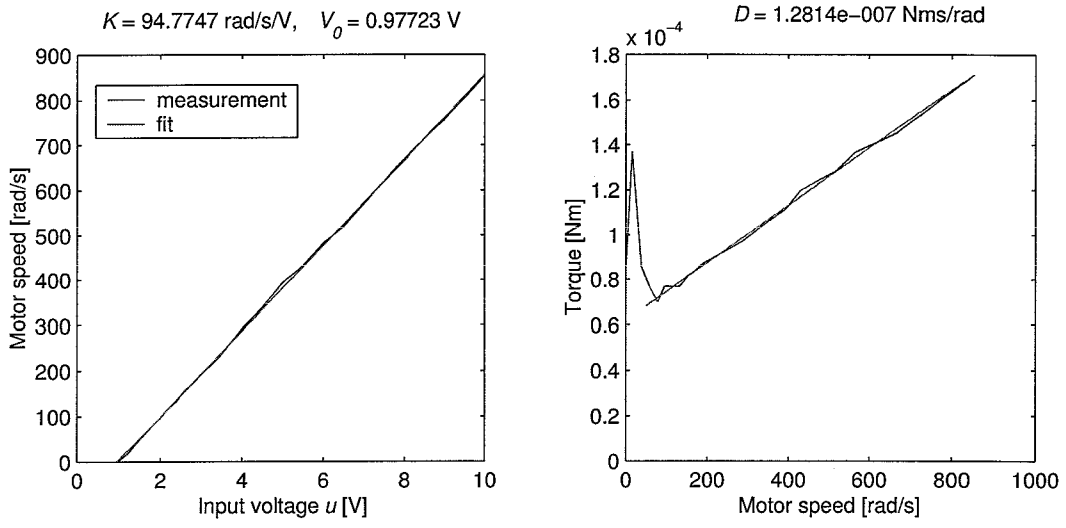


Figure 2.2: Measurements of the electromotor

the left with respect to the light source. (This is outside the view angle) As you can see, the variance of the noise increases a lot with increasing frequency. The cause of this is that the width of the light pulses decrease by increasing frequency. The pulses contain a charge curve. When decreasing the pulse width the maximum amplitude of a pulse will be smaller so the signal to noise ratio will be worse. But decreasing the frequency results in less bandwidth so there has to made a trade of between these two.

Finally the maximum view angle of the light tracker has to be determined. By moving the light source with a constant speed of 0,048 m/s (again using a timer) and the light tracker pointed straight to the light source the plot in figure 2.4 was created. (at a distance of 3 m) The slope covers about 6,5 s so this results in a view angle α of 0,1 rad.

2.4 Model validation

The created model in the previous section has to be validated now. This will be done by applying some same situations to the model and the real setup. The settings for the controller are: 85 Hz, P-action: 37, I-action: 35, D-action:, F-action (static friction compensation): 106. These control values have to be set in the application. These are in fact all scaled.

The light source will be oscillated as a sine with a frequency of 0,22 Hz and an amplitude of 0,2 m (= 0,067 rad at 3 m distance) by hand. (The frequency was determined afterwards when looking at the obtained data) In figure 2.5 you see the light intensity difference y of the experiment and the simulation. The shape and order of magnitudes are quite similar. The type of noise of the experiment data is a bit different because of the simplification of the model.

In figure 2.6 the controller output u has been plotted. Also here the data has similarities such as order of magnitudes, jumps of 2 V caused by the dead zone compensation and similar noise. In the controller is also a mechanism included if the position error is small enough the controller will be shut down. This can also be seen in both experiment and simulation. Of

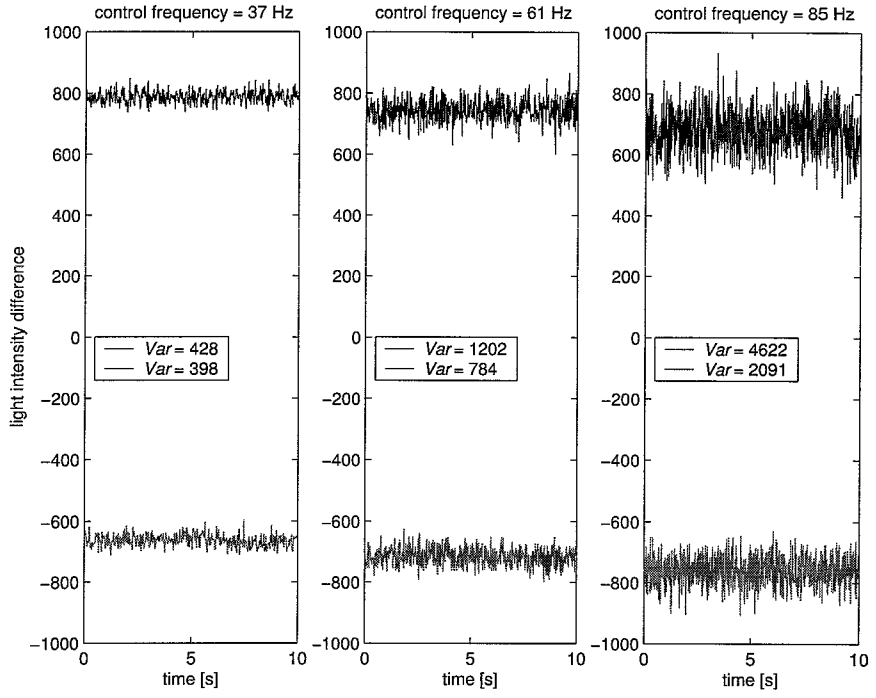


Figure 2.3: Measurements of the sensor noise

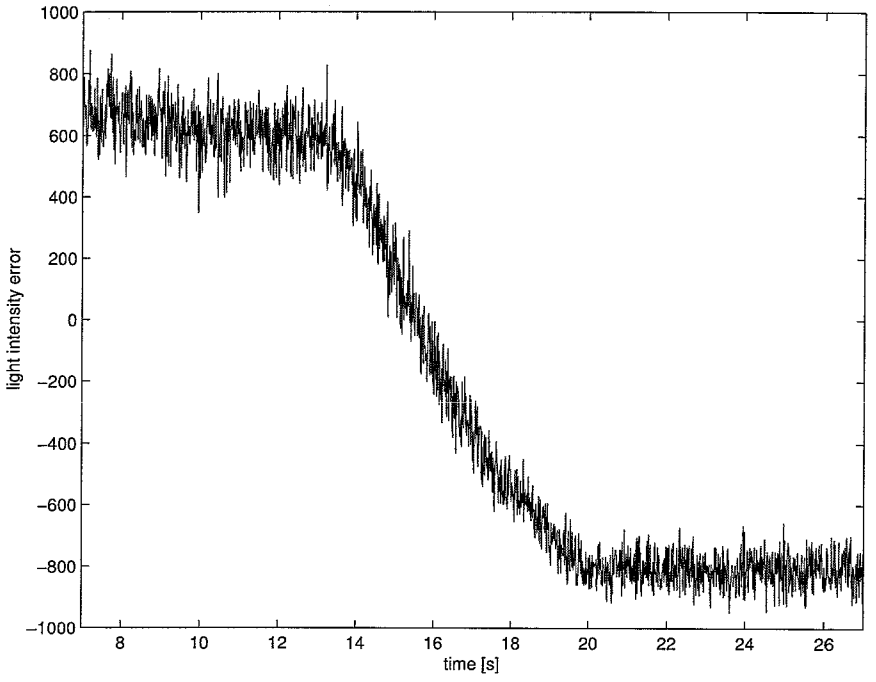


Figure 2.4: Light source moves with a constant speed of 0,016rad/s, the light tracker stands still

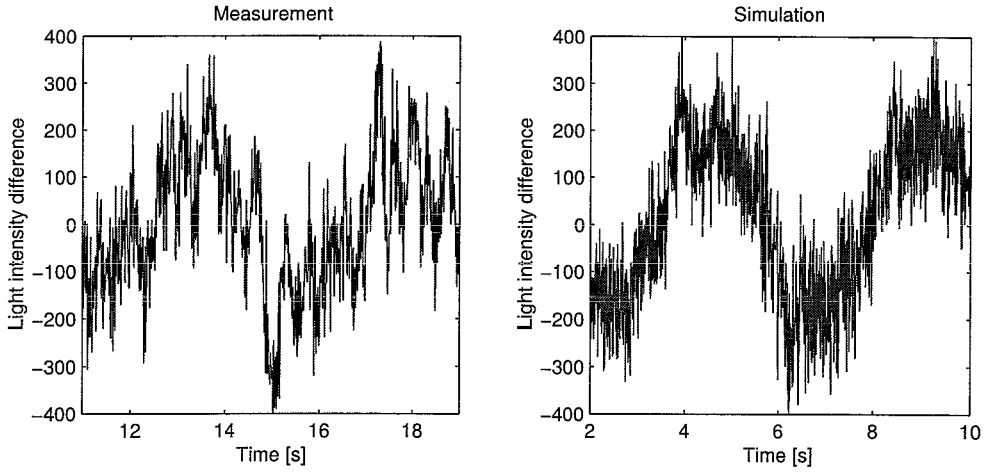


Figure 2.5: Comparing the Light intensity difference

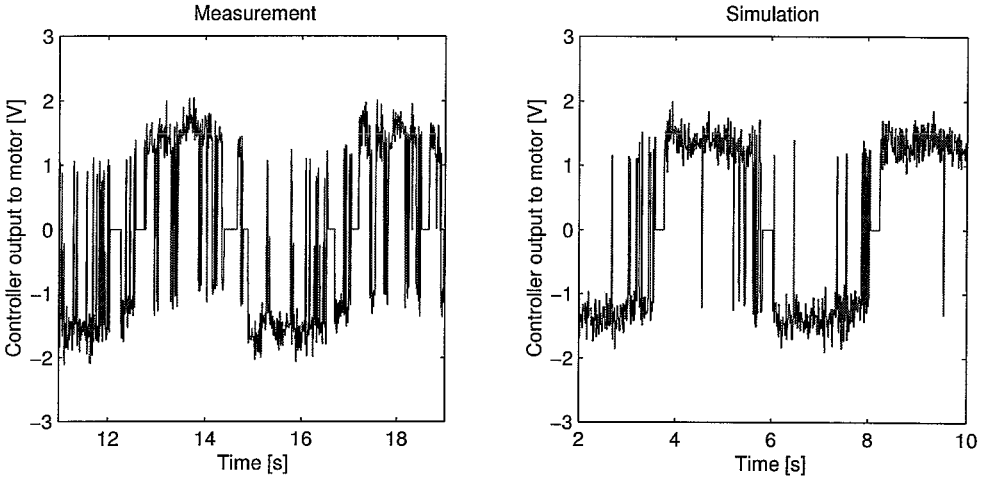


Figure 2.6: Comparing the controller output to the motor

course the peaks do not take place at the same time because the noise is nondeterministic. The amount of jumps in the simulation data is less. The reason of this is the different type of noise of the light intensity difference. If that signal changes its sign a jump will be seen in the controller output signal.

In figure 2.7 the tracking results of the simulation can be seen. This picture is not available for the experiment because there are no instruments to measure this. A maximum error of 50 mm occurs when changing direction because of the static friction.

Next a block function will be set as an input for the motor without using a controller. In figure 2.8 the result can be seen. Because the resulting angle movement caused by the input plotted in this figure is much bigger than the view angle α, there will be a saturation of the light intensity error. During the experiment the laser spot approaches the light source and then the voltage is shut down. The difference of the static position is not important because

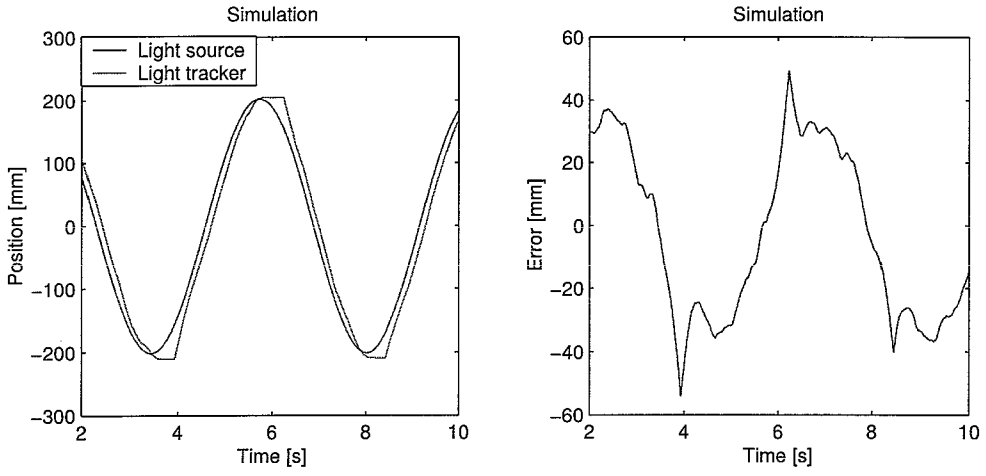


Figure 2.7: Comparing the position of the light source and the light tracker in the simulation

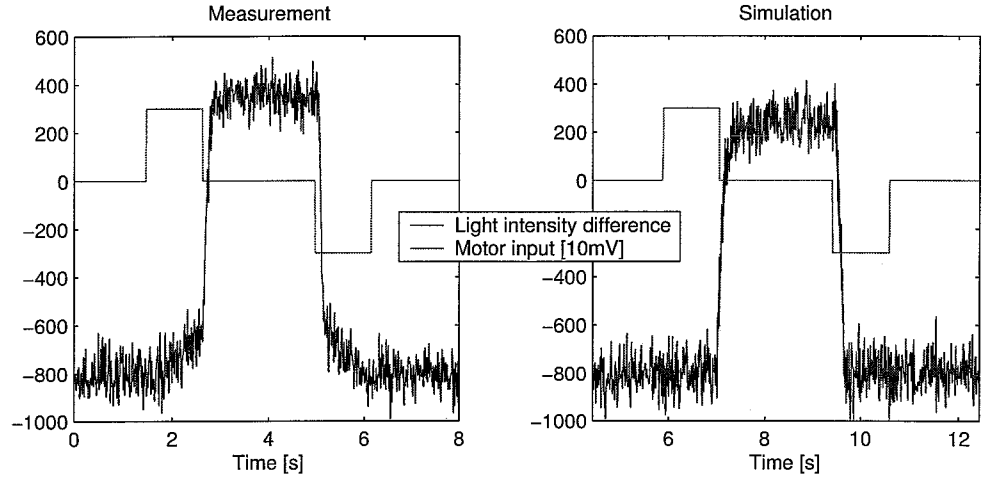


Figure 2.8: Comparing the light intensity difference

you do not know where it started outside the view angle. Only the time it takes to stop is of importance. This time is related to the amount of inertia the system contains. A blow-up of this can be seen in figure 2.9. As you can see the model results approximates the measured data.

According to the previous experiments and simulations the model shows similar characteristic properties so the model can be used for simulation and optimization of different types of controllers.

2.5 Summary

Characteristics of the experimental setup which are included in the model are: inertia, backlash, static friction, current limiter, sensor noise and the update frequency of the motor input

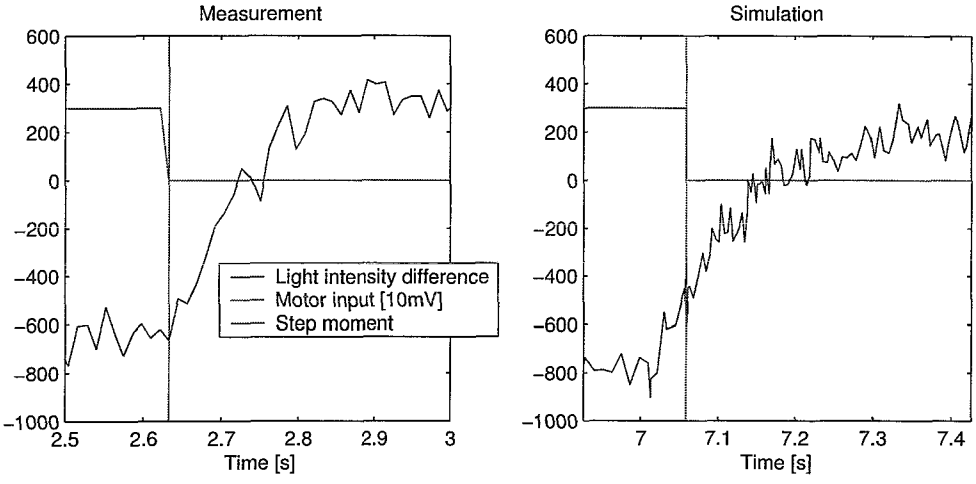


Figure 2.9: Figure 2.8 zoomed in

u. This can be seen in figure 2.1 and figure 1.1.

The following constants of the model have been derived by experiments:

$K = 94.7747\frac{rad}{sV}$	Gain between input voltage and angular velocity of the electromotor
$N = 429.23$	Total gear ratio
$V_0 = 0,97723V$	Static friction of the motor represented as a dead zone of the input voltage
$Var = 5000V^2$	Variance of the sensor noise W
$\alpha = 0,1rad$	Maximum view angle where movements of the light source are noticeable
$\beta = 8,3 \cdot 10^{-3}rad$	Amount of backlash caused by the gears
$\tau = 0,1s$	Mechanical time constant of the system
$\frac{dy}{d\theta_L} = 16000rad^{-1}$	Gradient of the light intensity difference and the angular position

Some simplifications have been made and some features have not been included in the model:

- The electrical pole has been neglected because it is much faster than the mechanical pole. The behavior of the system is practically the same below a frequency of 500 rad/s. That's much higher than the maximum achievable bandwidth. The advantage of this is that there will be no calculation problems. There were a lot of singularity problems when the fast electrical pole was included which even resulted in totally wrong solutions.

- The stationary transfer function has been used to determine the relation between input voltage and angular velocity of the motor. In fact this is a lot more complicated, but for low frequencies this approach is sufficient. With this also the behavior of the amplifier and the static friction is included.

- The inertia of the whole system is concentrated in the motor. This has been determined using experiments. So the linear dynamics are reduced to a second order system. At the end is only a backlash block without a mass at the other end. This is allowed because

the most inertia is located in the motor and gear box. Because of the large reduction the contribution of the platform to the inertia is low.

- It is assumed that the two light sensors have the same behavior, but according to figure 2.3 this is not exactly the case. One contains some more noise and has an offset with respect to the other. This could also be partially caused by asymmetric light disturbances.
- It is assumed that the distance between the light source and the light tracker is always 3m. If this would decrease the intensity would quadratically rise.
- It is assumed that the light source always points directly to the light source. When this is not the case the intensity will be less, because of the properties of the diode. The intensity will decrease if the diode will turn away.

Chapter 3

Examination of control strategies

All controllers have been tested in simulink. After that they have been implemented in the application of the tracker. Parameters had to be changed a little bit because the dynamics of the setup have been simplified in the model. Because the real-time performance is most important only the performance of the implemented controllers will be shown in the next sections.

3.1 PID controller

The controller to be developed has to minimize the tracking error. Further it has to look nice and smooth. There may not be to much overshoot and fast oscillations are not desirable. It is very annoying for the audience when stagelights have such behavior. A first attempt to achieve this is to implement a very basic controller, a PID controller. The performance of this was bad. When moving the light emitter the tracking was very bad. The phase loss of the tracker was almost 180° when moving with an oscillation of 1 Hz, so the tracker was pointing totally to the wrong side. When the light emitter stopped moving, the tracker did not point accurate enough to the emitter. It kept on moving irregularly because of the integrator action. If the proportional action would be increased the system became unstable, it kept on oscillating around the position of the emitter. So it was not possible to set the PID parameters to get the desired behavior. The two major problems are:

- The coulomb friction of the motor. Between -1 V and 1 V the motor stops moving. Because of this it is not possible to make the error zero. The P-action is not able to overcome the 1 V with a relative small error. The I-action is building up but when it passes 1 V the tracker will suddenly move and it will overshoot. This will keep on going all the time. So the tracker will never stand still in the exact direction.

- The measurement noise. Although there is a lot of denoising of the signal, still there will remain a lot of measurement noise, a ratio of about 1 to 10. Because of this the derivative is very bad. It looks more like white noise so the D-action can not be set to the value which is desirable. The lack of knowing the velocity decreases the bandwidth of the system.

3.2 Non-linear controller

To overcome the problem of the coulomb friction in the motor a sign function for the error is added to the controller law. This eliminates the dead zone when the sign of the error changes. This makes the tracker reacting to the P-action and the I-action almost immediately. Using this the tracking is improved a lot. But still there is a problem when the emitter is not moving. The tracker will keep on moving near to the position of the emitter. To eliminate these movements all parameters have to be decreased what is not good for the tracking performance. In the next section a solution for this is presented.

3.3 Switching controller

The only problem now is the error when the emitter is not moving. The controller has to be shut of somehow. Something like shutting of when the error is smaller than some value does not work, because there is a lot of measurement noise. The tracker has inertia so it will not stop immediately. Further it is constructed very cheap so it has some unwanted movements. That's why the amount of overshoot is always different. How is it possible to determine if the tracker is pointing to the emitter accurate enough with so much noise? Suppose the minimum and maximum value of the scaled difference is about -800 respectively 800 and the noise has a maximum amplitude of ± 100. (a realistic situation) The range corresponds to the maximum view angle of $0,1°$. At a distance of 3 m this is about 30 cm. The noise will then correspond to a width of 4 cm. The objective is about 1 cm of accuracy, so the task is controlling better than you can measure. With standard linear methods this is not possible, but using some other technics it will be possible.

The controller has to be switched off at the right moment. The signal difference will be filtered with a low-pass filter of 10Hz to get rid of some noise. If this value is lower than a certain constant for a short determined period the controller will be shut off. For the boundaries an amplitude of 80 is taken and for the time period is 40 samples taken. Why does this work? The fact is used that the noise is approximately white noise with zero average. If this noise band is between the boundaries -80 and 80 the average will be near zero. (because of the low-pass filter the amplitude will be about 60 instead of 100) The delay time to shut off the controller after 40 samples is used because the huge oscillation of the tracker first have to be weakened. So if the controller is switched off the tracker will stop almost immediately close to the position of the emitter. The low-pass filter is necessary to decrease the noise. Also the time delay caused by it is no problem because the filtered signal is not used for tracking, only for shutting of the controller.

Using this switching controller all parameters of the PIDF-controller can be increased what will improve the bandwidth without affecting the performance when the emitter stops moving.

3.4 Implementation and performance

In appendix C the implementation of the controller in the application in the so called "call-backfunction" can be found. First the error signal is scaled with the standard deviation of the amplitudes of both sides. Next the integrator-variable is updated. Then the low-pass

filter is applied. After that the controller output is calculated including the PIDF actions and the switching controller. Also if something is blocking the light it will detect that the amplitude of both channels is low and will shut down the controller temporarily. To prevent the output of getting to big the controller output is set to a maximum of 1 corresponding with the maximum voltage supplied to the amplifier of the motor. (5 V)

Measurement data to show the performance is not available, because the motor does not have an encoder and the position of the light emitter is not measured. The performance will be checked manually. When oscillating the light emitter by hand the tracker will follow with some phase lag. There will always be phase lag because it needs an position error to control with. The bandwidth is about 2Hz. When oscillating it at 2 Hz with a amplitude of 30 cm the maximum error will be about 8 cm. This is about 30° of phase lag. The static error when stopping the oscillation is maximally 1cm. The maximum overshoot that can occur is about 3cm, but that does not happen always because it is a non-linear controller. Increasing the frequency will give more phase lag. When moving the light emitter slowly sometimes stick-slip behavior is the result. The cause of this is probably a combination of the static friction of the motor and the switch-off mechanism of the controller. To reduce this is possible by decreasing the control-parameters, especially the P-action. Unfortunately this will decrease the performance for higher frequencies (2 Hz). In the final application this has to be optimized according to the desired behavior.

Chapter 4

Synchronization of the transmitted data

First the Rudin-Shapiro Transform will be explained. This is used to code and decode a transmitted signal. Because of this more lights can be tracked simultaneously and robust by different trackers. After that there will be explained what synchronization problem is. Finally choices will be made for the emitted signal.

4.1 Rudin-Shapiro Transform

The (symmetric) Rudin-Shapiro Transform (RST) has some very nice properties which can be used for data transmission in an environment with a lot of disturbances. The C-code of the used dll-file in Matlab can be found in appendix C. You can find an article about the use of the Rudin-Shapiro sequence in [2]. A more detailed mathematical description is given in chapter 12 of [3]. The RST is a linear transformation. Actually it is just a matrix multiplication consisting of 1's and -1's (and for example a scaling factor of $\sqrt{2}^{^2log(n)}$ to preserve the energy of the signal. This is actually not really important because it is a linear transformation so you could choose any suitable scaling). A nxn-matrix multiplication with a n-vector requires n^2 operations. This can be reduced to $n \cdot ^2 log(n)$ by applying a recursive operation showed in equation 4.1. A visualization of this mechanism is shown in figure 4.1. j starts with $^2log(n)$ on top and the bottom is reach when $j = 1$.

$$\begin{bmatrix} y_k \\ y_{k+2^{j-1}} \end{bmatrix} = \frac{(-1)^{mk}}{\sqrt{2}} \begin{bmatrix} 1 & (-1)^k \\ (-1)^m & -(-1)^{k+m} \end{bmatrix} \begin{bmatrix} x_{2k} \\ x_{2k+1} \end{bmatrix} \tag{4.1}$$

Some nice properties of the Rudin-Shapiro Transform are: (suppose $y = RST(x)$)

- The RST is equal to it is inverse. So applying it twice, the same vector will return. $x = RST(RST(x))$
- The RST spreads the data over the whole frequency range. This gives a pseudo-random vector with positive and negative numbers of the same amplitude. (when applying it to an unity vector like $[10000000]^T$)

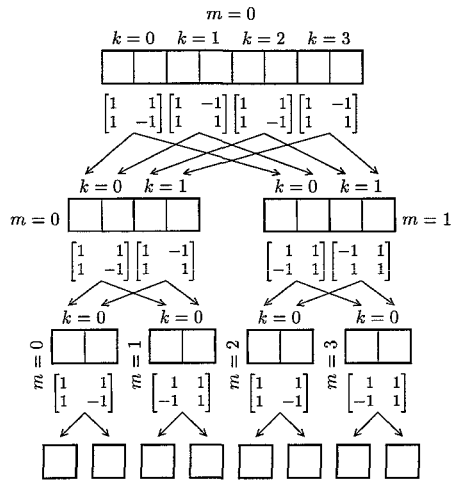


Figure 4.1: The changes of variables in the fast implementation of a symmetric RST. Here applied to a vector in $\mathbb{R}^8$.

- The RST is a linear transformation. ($y = \frac{1}{a}RST(ax), RST(x_1)+RST(x_2) = RST(x_1+x_2)$) If you investigate it further you'll discover that in each position of y each position of x is exactly one time present. (either - or +) You can interpret this as spreading data over the whole frequency range. So if you change one arbitrary position of x all position of y will change.
- If x_1 and x_2 are orthogonal vectors then $RST(x_1)$ and $RST(x_2)$ are also orthogonal.
- The auto-correlation of y looks like that of white noise. When white noise is added to y and then the RST is applied the first vector x will be found back with a little bit of noise spread out over the whole vector.
- The RST is a numerical stable transformation and is not high computational demanding. It is only adding or subtracting of the numbers in vector x. The fast algorithm described before uses $n \cdot {}^2log(n)$ additions or subtractions with n the length of vector x. A necessity is that the length of vector x has to be a power of two.

4.2 Problem formulation

During the experiments there was the problem that the light tracker and the light source were synchronized by a cable. The same application gets the received signal from the light tracker and sends the signal to the light source to be emitted. The application has to find a specific Rudin-Shapiro sequence in the signal. Since the signals are synchronized, the time of the first point of the Rudin-Shapiro sequence (grid offset) is known and the width between the points (grid width) is known. In the final application this will not be the case, because there can not be a cable between the emitter and the receiver. The timing of the emitter and the receiver will be realized by separate crystals. The start moments are not synchronized so the receiver application will not know what the grid offset is. Further the generated frequencies

of both crystals will not be exactly the same because they have to be as cheap as possible. The frequencies will also drift because of for example sensitivity to temperature.

Beside of these problems there will be noise at the received signal and 100 Hz disturbances by other light sources. And last but not least, there will not be only one Rudin-Shapiro sequence but there could be 16 sequences in the signal simultaneously with different intensities. And further it must be possible to send 19 bits of data in each channel by altering the Rudin-Shapiro sequence in the signal.

Keep in mind that only the light intensity difference (amplitude) of both signals will be used to track the light emitter. A visualization of the signal-transfer from the emitter to the receiver is shown in figure 4.2. The red part represents the emitter side and the green part the receiver side. A RSS-length of 16 is used in this example. The RS-point is set to 0. Transforming it with the Rudin-Shapiro Transform you'll get the matching Rudin-Shapiro sequence. This is emitted with a grid width of 4 samples, a pulse width of 1 sample and a samplefrequency of f_s. Below that you see the continuous time, infrared light-signal generated by the diode. This will be received by the sensors of the receiver with a frequency of f_r. Because f_s and f_r are different and not synchronized the received data is not totally the same. To select data from the received signal a grid has been introduced. The algorithm described in chapter 5 will optimize the grid offset (o) and width (w) so the quality of the reconstructed RSS will be as good as possible. When transforming it back the right RS-point will show up. Because the RST is a linear transformation and there are used two sensors, the magnitude of the RS-point of both sensor-signals can be used to determine the direction of the location of the emitter.

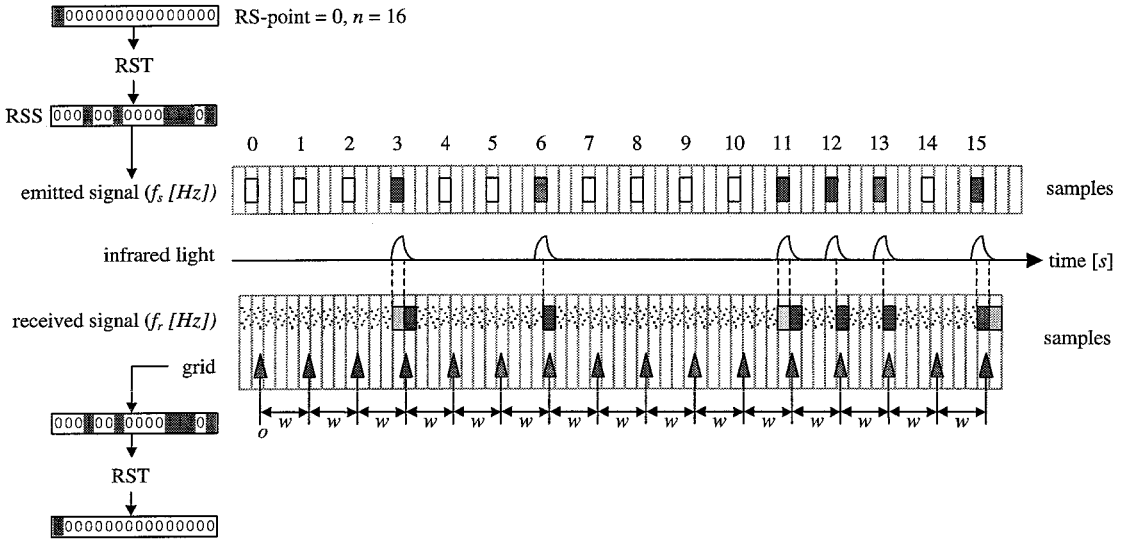


Figure 4.2: Example of a signal-transfer of a RSS with length 16.

4.3 Choices of the transferred signal

There will be used a 48 kHz crystal to emit the infrared light. The main reason is that they are cheap because they are used a lot in the music industry. Crystals with higher frequencies

will dramatically increase costs. The length of the Rudin-Shapiro sequence is still to be chosen. The maximum amount of channels is 16 and each channel must be able to send a 19 bit command. But when there is no command the tracker must keep on tracking so there is a need for a null-signal. These boundaries sets the minimum length of the emitted sequences to 64. So for each channel there are $64/16 = 4$ Rudin-Shapiro sequences available. With 4 different sequences 2bit of data can be send each time. (00, 01, 10 or 11) 00 will be used as the null-signal. Then only three possible RSS's are available. The 19 bit command will be converted to a ternary base with a length of 12. This can be send sequentially and at the receiver side it can be converted back to the binary base to get the original command.

Now it has to be decided what the peak distance will be. (The amount of samples between the points of the Rudin-Shapiro Sequence) The maximum time to emit 12 sequences of length 64 is 0,1s. There will be some time delay at the emitter side because of the buffering of the received data. This will be maximally the length of 2 RSS's. Further there has to be included some zero space between every emitted sequence. The problem occurred that a RSS shifted by a specific amount is another RSS. The maximum amount of zeros or ones in a row in a sequence of length 64 is 5. So when putting more than 5 zeros between the sequences the founded sequences will be unique. Unfortunately if only 70% of a sequence is correct and the rest is wrong data, it is possible it will be tracked. Finding such "ghost"-sequence can cause problems like tracking of the wrong light source or even receiving wrong commands. Increasing the zero space between the sequences decreases this chance but it will slow down the reaction time when a command is send. So this can be used as a tuning parameter to balance speed with robustness.

Suppose the zero space is 10. The time delay between the sending and receiving of the command is 14 sequences of length 64. The maximum allowable peak distance will than be: $\frac{48000 \cdot 0,1}{(64+10) \cdot 14} = 4,63$ samples. So a maximum of 4 samples may be used. The pulse width will be set to 1 sample. This has to be kept as low as possible to save life time of the diode, save energy and reduce interference between different channels. When more speed is demanded the only way to achieve that easily is decreasing the peak distance. A peak distance of 2 samples would mean a increase of reaction speed by factor 2. The level of interference when using this has to be examined in a later stage.

Chapter 5

Development of the GetRSS-algorithm

The problem will be divided into two parts. Firstly finding the null-signal in the form of a Rudin-Shapiro sequence from a particular channel in the measured signal. Secondly adjust the grid offset and width every next step to keep up tracking it robust enough. This specific Rudin-Shapiro sequence is generated by applying the Rudin-Shapiro Transform at a vector of zeros with one 1. The place of the one will be called the RS-point.

The signal contains a sine because of artificial light and the Rudin-Shapiro sequence has an unknown offset (that's the case when it is not synchronized). The light signal can only be positive because zero signal means dark. Sensor noise can be modelled as white noise with a zero average. The signal will be denoised using an algorithm to remove polynomial contents in the signal. For the algorithm there has to be specified the polynomial order and the amount of parts to be evaluated. With the right choice only the sine will be removed and the average will be set to zero. The rest of the signal-content will be mostly unchanged. Applying this denoising will improve the results of the next operations. Actually there is some effect of this algorithm on the amplitude of the Rudin-Shapiro sequence. This factor is constant for a specific Rudin-Shapiro sequence. This can be determined and can be applied to the RS-point every time the algorithm is applied.

The grid offset can be found by determining the cross-correlation of the measured signal with the specific Rudin-Shapiro sequence. This can be done for variations of the grid offset and width. The initialization-phase has to be continued until it finds a sufficient good sequence. The amplitude of the RS-point and the maximum amplitude of the other points can be used to determine if the founded sequence is good enough. In figure 5.1 you see an example how the maximum amount of available data is related to the grid offset error and the grid width. The emitter frequency is 47900 Hz, the receiver frequency is 48000Hz, the grid width of the emitter is 4 samples, the Rudin-Shapiro sequence length is 1024 and the pulse width is 1 sample. On the right you see how much correct data of the Rudin-Shapiro sequence is available. When the algorithm adjusts the grid offset and grid width keep this figure in mind.

Figure 5.1 changes for different settings. The pulse width approximates the height of the 50% band in the figure. The width of the 50% band is about two times the pulse width divided by the Rudin-Shapiro sequence length. Just draw it and it will become clear. So in this case it is $2/1024 = 0,002$ samples. If the emitter frequency is higher than the receiver frequency the

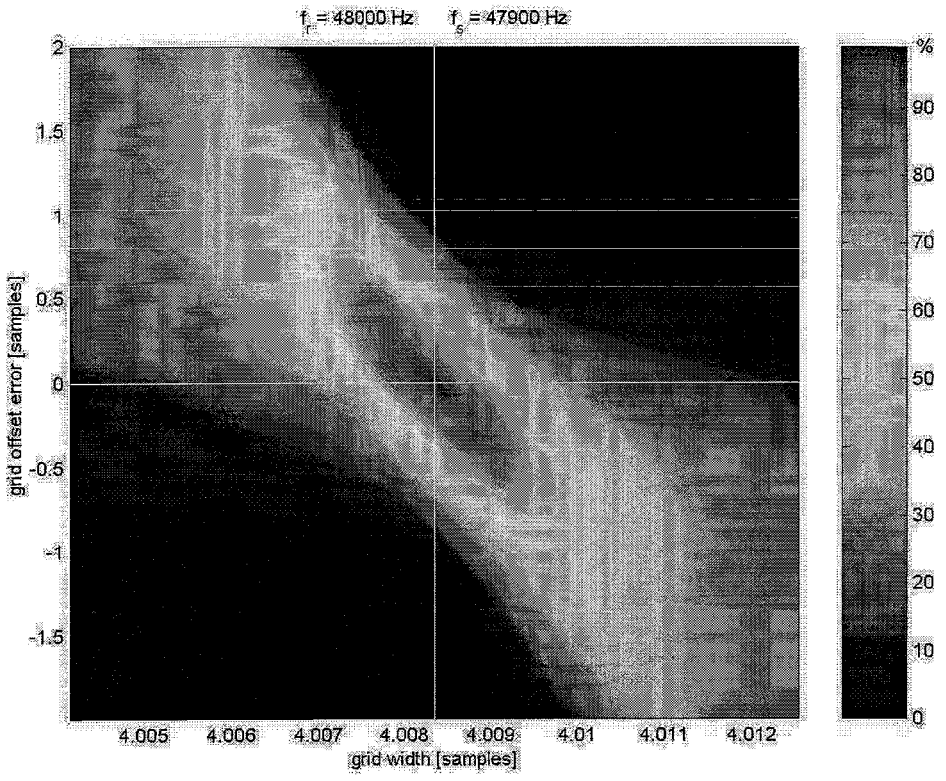


Figure 5.1: Maximum amount of available data depending on the grid offset and width.

maximum amount of available data for the best grid offset and width will decrease because of under sampling. When the frequencies are almost the same the surface of the top of the "mountain" is more flat so there is a bigger region with values near 100%. The top is more rounded when the frequencies get more different. Furthermore the angle of the oval rotates slowly clockwise with decreasing emitter frequency or increasing receiver frequency.

According to the previous information the find-chance P of the RSS every attempt can be approximated by formula 5.1.

$$P = \frac{p}{w} \cdot 100\% \quad if \quad |\Delta w| < \frac{p}{n} \tag{5.1}$$

When the sequence has been found the grid offset and width is known. Next the grid offset and width have to stay near the optimum. This will be done by looking only at a part of the sequence (to decrease computational time) and shifting the offset a little bit. A part of the Rudin-Shapiro sequence is such sequence itself with another RS-point. This also has to be determined. Now only the grid offset will be varied for a few points. The grid width will be estimated by looking at the offset correction, because the main cause of a grid offset deviation each step is a grid width deviation. This process can be repeated and will iteratively track the optimum grid offset and width. In figure 5.2 can be seen that if the grid width is

not exactly the same as the optimum there will be a grid offset correction needed every step. This means that the offset correction is related to the grid width error so it can be used to adjust the grid width and iteratively follow slow changes of the real width between the pulses caused by frequency drift.

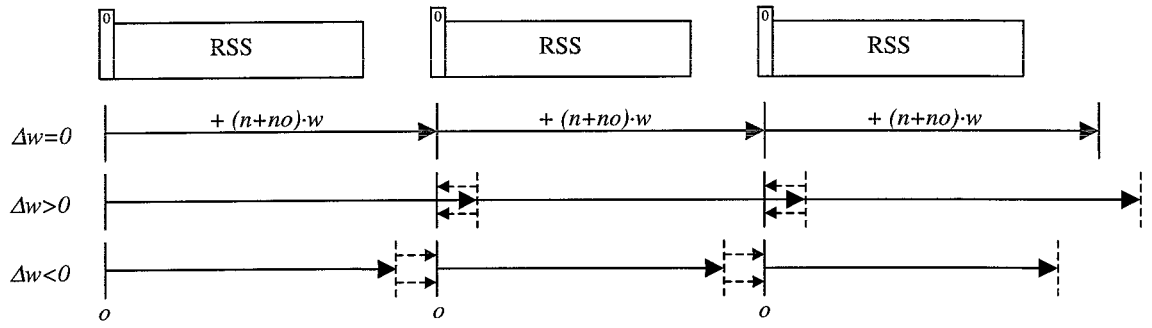


Figure 5.2: Visualization of the grid correction every step

In appendix B makesignal.m and getrss.m are included. These two files where used to develop the algorithm. A detailed description can also be found there. In appendix B.5 you'll see a visualization of the structure of that algorithm. The structure and some settings have been changed when the algorithm was written in C++. In the next chapter the final C++ algorithm will be described.

Chapter 6

Implementation

6.1 C++ algorithm

Receive measured data

When the application is started, the data acquisition device will continuously collect data at a sample rate of 250 kHz and in blocks of 4096 samples. When it has captured all samples it will trigger the callback function written in the code. Here the data will be copied to a global variable. Next the polynomial content is removed so the disturbances caused by artificial light and a possible offset will be removed. Because the position of the tracker is not known it is possible one side does not contain useful data. The average of both light sensor signals will be used. Furthermore the received signals are buffered because there will be some frequency difference between the emitter and receiver and they will also drift a bit. The grid offset indicates the position in this buffer. It can happen that not all data is available for one whole sequence. Than the amount of samples left in the buffer after the grid offset is less than one sequence including the zero space when using the present grid width. If that's the case the algorithm will wait for the next delivering of data. In case there are two sequences available in the buffer it will simply step through both sequences.

At first there was no buffer. Only the 4096 samples where taken and this was shifted in a way that the first sample was the location of the first position of the Rudin-Shapiro sequence. The problem with this is that it is assumed that the Rudin-Shapiro sequence to be tracked is the same all the way. But there has to be send some data what means different Rudin-Shapiro sequences. So the assumption is wrong. Using a buffer this problem is gone, because a sequence is considered only if it is completely in the buffer. When the buffer is large enough every sequence will be at least one time completely in the buffer, so every sequence will be looked at. This is necessary to receive all commands correctly.

Initialization-phase

Then the GetRSS-algorithm is called which tries to find the right grid offset and width by adjusting them randomly. It searches only for the null-sequence of the selected channel in the measured signal. If there would be a command at that moment it is simply not able to track it. But between the commands there will be some null-sequences so than it is able to find the null-sequence.

For the grid width there has to be determined an estimation. Every 10 times the grid width is randomly changed. If the grid width is near enough to the real grid width in the signal it will be able to find the sequence. The grid offset will be randomly changed every time. If the previous RS-point was correct there will be made only a little random change of maximally halve a pulse width in the direction of the biggest amplitude of the RS-point. (according to the previous RS-point) Else the change will be randomly with a range of one grid width. This search will go on until the founded RS-point is correct and the maximum amplitude of the rest of the back transformed signal is smaller than 0,3 times the amplitude of the RS-point. When both conditions are satisfied the grid offset and width are good enough determined and the step-phase will take over the tracking of the sequence.

Step-phase

If the step-phase is activated the grid offset will be changed several times for each sequence to find the optimum. At the moment it is about a maximum change of ± 40 samples with a resolution of 1/3. But setting these parameters that high is not necessary. Increasing the maximum change will increase robustness for frequency drift. The resolution will increase the quality of the founded sequence. In the final application these parameters can be used to optimize for robustness, computational time and tracking performance. The maximum change may not be larger than the grid width because if there is a lot of frequency drift another peak will show up in the range of offset shifts. Most of the time this peak is smaller what would be no problem but sometimes because of disturbances this is larger and then it is possible to lose synchronization.

Only 1/4 of the sequence is used which is a Rudin-Shapiro sequence itself. By collecting the amplitudes of the matching RS-point for each offset shift, the weighted average of the biggest amplitudes will be used. (amplitudes near the peak-value which are bigger than 80% of the peak-value)

This will be done for all 4 Rudin-Shapiro sequences in the present channel, because you do not know if there will be a command or not. This tracking of all sequences is necessary because if suddenly another Rudin-Shapiro sequence shows up only that particular sequence will give the right grid offset correction. If you would use another one it is possible it will lose tracking and it has to start over again with the initialization-phase. For all four attempts the whole sequence will be transformed back using the determined grid offset correction. The one which gives the correct RS-point and has the biggest amplitude will be selected. The offset correction is applied to the previous grid offset and the grid width will be changed according to the grid offset divided by the total sequence length including the zero space and also multiplied by 0,2. The last factor is like a low-pass filter. The grid width will change slowly but it will be more steady so it is less sensitive to disturbances.

Finally there will be a check if the founded RS-point is the correct one. If the RS-point is 4 times not correct in a row the sequence is supposedly lost and the initialization-phase will be activated to try to find the sequence again. The cause of losing the signal could be blocking of the signal so the specific sequence is not present or because of too much disturbances.

6.2 Performance

The performance is dependent on a lot of things. The main issues are:

- Disturbances. Like a 100 Hz sine from artificial light or other light sources. Also there is sensor noise, the received signal is very weak so it has to be amplified a lot. Another problem is blocking of the signal which decreases the signal or even eliminates the signal.

- Sequence appearance. The emitting frequency, the grid width, the pulse width, the zero space between the sequences and the length of the sequence. All these items can be chosen with some limitations and have influence on the performance of the algorithm. The time duration of a pulse may not be to short because the emitter has some charge curve. A smaller time duration will give a smaller amplitude. Further the ratio of the pulse width and the grid width must be as small as possible. This will increase the chance that the initialization-phase will find the grid. The zero space has to be big enough so the change will be very small that a "ghost"-sequence will be found.

- Algorithm parameters. Depending on the circumstances the parameters can be set to optimize the performance. In the function "InitSearch()" most of them are located. For the initialization-phase these are the maximum amplitude of the back transformed signal with regard to the amplitude of the RS-point and the initial estimation of the grid offset and width. The first parameter can be set lower so the grid offset and width have to be better determined when going to the step-phase. This will take some more time but the chance it will find a wrong "ghost"-sequence is reduced.

  For the step-phase the parameters are: The maximum offset shift and resolution, the part of the signal to be examined, the correction factor for the grid width and the maximum amount of wrong determined RS-points in a row. Enlarging the parameters for the offset shift will increase robustness but will be more computational demanding. Decreasing the correction factor for the grid width will give a more robust tracking for disturbances but will slower react to fast frequency drift. Finally the maximum amount of wrong determinations in a row will be useful if the signal is only blocked for a short time or because of noise there was temporarily a track problem. The algorithm will continue with the step-phase if the sequence comes back in time.

6.3 Application

In figure 6.1 you see the application of the receiver side. The two big plots show a part of the measured signal of both sensors. The location of the Rudin-Shapiro Sequence is also visualized in the plot. The first two blue bars indicate the amplitude of the RS-point of each side. The third is the difference of them scaled by some factor. Below these bars you see a small plot which indicates the amplitude when shifting the offset a little bit. On the right of this you see 4 slides to set the controller parameters: K_p, K_i, K_d, K_f. The small plot above indicates the controller output to the motor which is proportional to the voltage applied to the motor. The four light indicators on the right of this plot indicate the controller voltage contribution of each component. (black: no contribution, red: negative contribution, blue: positive contribution) The GetRSS-window shows some data and statistics about the GetRSS-algorithm. The Data Transfer-window shows which channel to track and what the last sent command was in ternary and binary base.

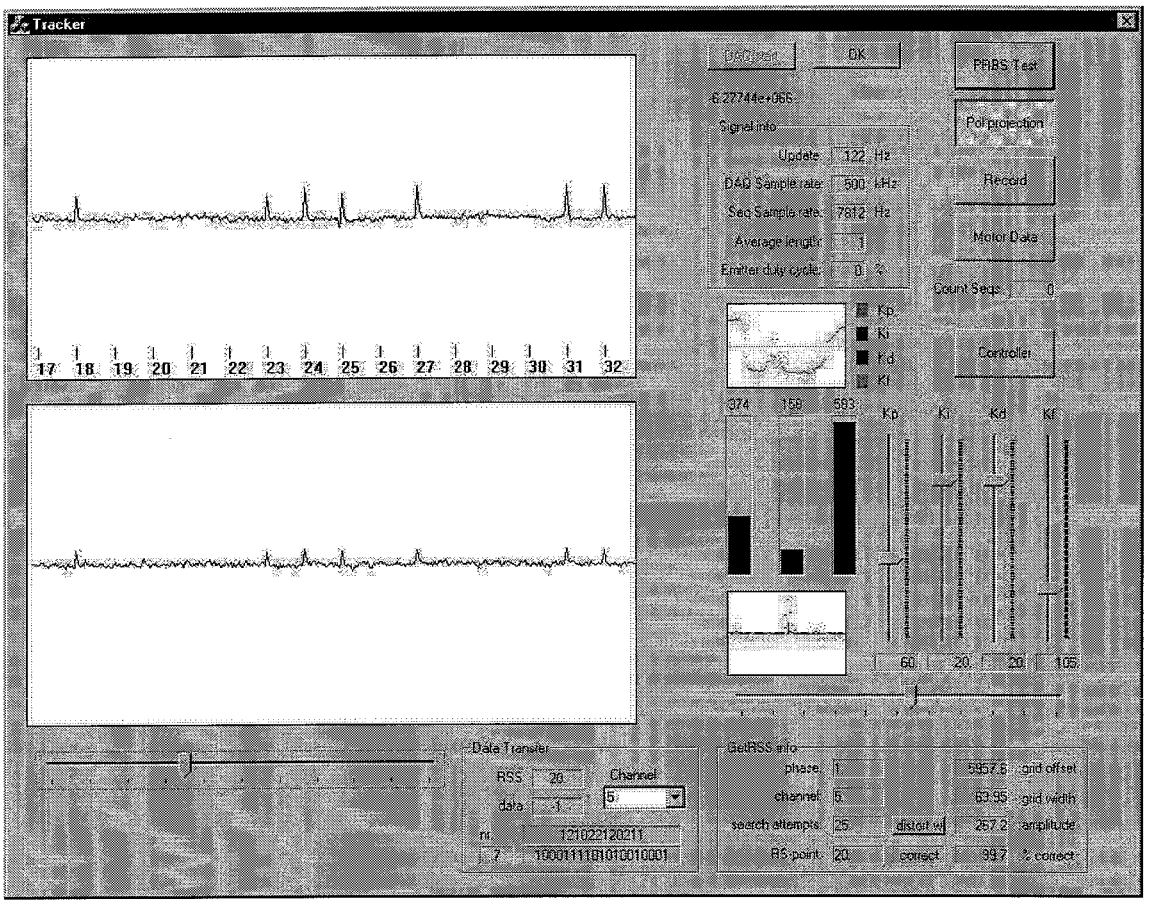


Figure 6.1: Application at the receiver side

In figure 6.2 you see the application of the emitter side. The RSS-length, pulse width, grid width and zero space can be set. With this the duty-cycle is calculated. This is the percentage of how long the emitter is turned on per time unit. If this is too high the diode will burn. The maximum is dependent on the type of diode. For this diode 2% is about the maximum. Further the channel can be set in the range of 0 and 15. The corresponding null-RSS (nodata-sequence) is easily calculated by multiplying the channel by 4. The three next RSS's are the data-sequences corresponding to the ternary base values 0, 1 and 2. A ternary or binary code can be set. With the button "S" this code will be emitted sequentially. After emitting all ternary values the application will emit again the null-RSS.

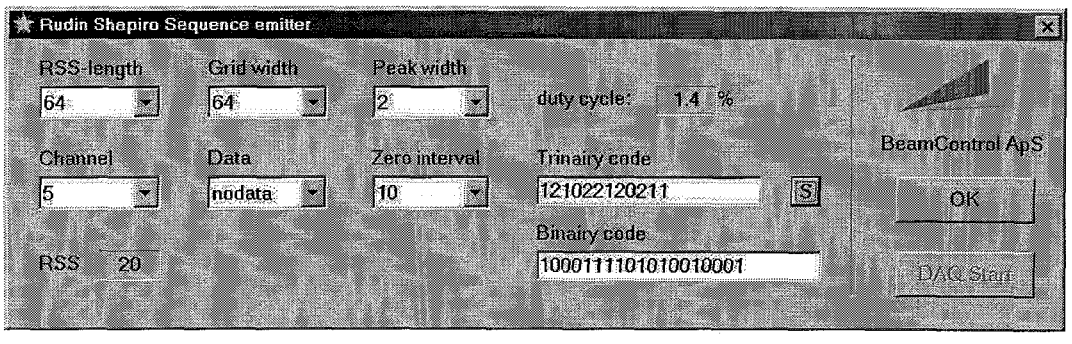


Figure 6.2: Application at the emitter side

Conclusion

Both objectives are successively accomplished. The first objective was to track the light source as good as possible. The controller consists of a standard PID-controller, a sign-function and a switch-off mechanism. The bandwidth which is achieved is about 2 Hz and the final error is smaller than 1 cm at a distance of 3 m. The noise of the input-signal corresponds to about 4 cm.

The second objective was to allocate a specific Rudin-Shapiro sequence in the signal and keep it on track. An algorithm has been developed what does do that. It consists of two phases. The initialization-phase searches for the right grid offset and width and the step-phase will take over if the right sequence is found. It will optimize the grid offset for every sequence and according to the offset correction the grid width will be changed. This way both grid offset and width will be iteratively stay near the optimum value. If the sequence is nevertheless lost because of some disturbances it will return to the initialization-phase trying to find the signal back.

Also data can be transferred. For the application 19 bit commands needed to be sent. It will be sent using a ternary base. (12 numbers) A command will only be recognized if the algorithm is in the step-phase. The algorithm will detect if another Rudin-Shapiro sequence is emitted in the selected channel. If this is the case 12 times in a row it will be seen as a sent command.

All the demanded requirements have been achieved so it can be concluded that this approach for the new application is feasible. The only thing that has to be done is implementing the code in a DSP and optimize it for the final settings.

Recommendations

Probably the code will be implemented in a fixed-point DSP to reduce costs. So when this is the case some code has to be changed which uses floating point variables. Also the code has to be optimized for speed because when using a cheap DSP you'll have to deal with limitations.

At the moment the distribution of the 64 Rudin-Shapiro sequences over the 16 channels is simply done. Multiplying the channel number by 4 you will get the null-RSS and the next three RSS's are the three possible data signals (ternary base). The problem is that a shifted Rudin-Shapiro sequence can be another sequence. This problem was partly solved by inserting a zero interval between every emitted sequence. But the problem is that if the grid only contains 60% of correct sequence data it will recognize it as the right sequence. Because some sequences are more than 75% equal when shifting it for a certain amount there

can still show up some "ghost"-sequences. A way to handle those "ghost"-sequences is to make a smart selection of the 4 Rudin-Shapiro sequences for each channel. The 4 sequences have to be correlated as little as possible for all shifts. When the algorithm is tracking a "ghost"-sequence as being the null-RSS of the channel the algorithm will lose track of the sequence when a command is sent. At that moment there is no fit with the other three expected sequences so it will search again in the initialization-phase for the right null-RSS of the channel. This way it is possible that it sporadically will find the wrong null-RSS in the initialization-phase, but it will never execute a wrong command because it loses track when a command is sent.

Bibliography

[1] VT mechatronics, *PROCEDURE FOR IDENTIFYING PERMANENT MAGNET DC MOTORS*, http://mechatronics.me.vt.edu/book/Section3/motormodelling.html

[2] Anders la Cour-Harbo, Jakob Stoustrup, Lars F.

Villemoes, *FAST AND ROBUST MEASUREMENTS OF OPTICAL CHANNEL GAINS*

[3] Anders la Cour-Harbo, *ROBUST AND LOW-COST ACTIVE SENSORS BY MEANS OF SIGNAL PROCESSING ALGORITHMS*, ISBN 87-90664-13-2, Doc. no. D-4562, August 2002

List of symbols

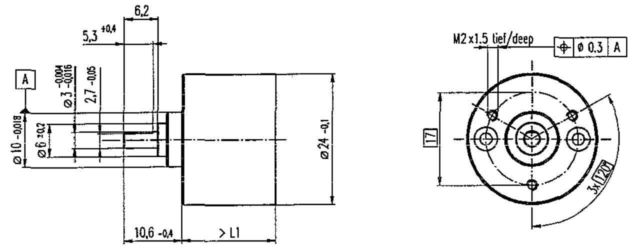
D	$[Nms/rad]$	Damping constant
K	$[rad/s/V]$	Gain between input voltage and angular velocity of the electromotor
n	$[-]$	Rudin-Shapiro sequence length
N	$[-]$	Total gear ratio
no	$[-]$	Zero space length
o	$[samples]$	Grid offset
R_L	$[m]$	Distance between light tracker and light source
u	$[V]$	Output voltage of the controller to the motor
V_0	$[V]$	Static friction of the motor represented as a dead zone of the input voltage
Var	$[V^2]$	Variance of the sensor noise W
w	$[samples]$	Grid width
W	$[-]$	Sensor noise of the light intensity difference
y	$[-]$	Light intensity difference (scaled)
α	$[rad]$	Maximum view angle where movements of the light source are noticeable
β	$[rad]$	Amount of backlash caused by the gears
γ	$[rad]$	Angular position of the light source
τ	$[s]$	Mechanical time constant of the system
θ_m	$[rad]$	Angular position of the electromotor
θ_L	$[rad]$	Pointing direction of the light tracker

List of Figures

Appendix A

Data sheets

Spur Gearhead GS 24 Ø24 mm, 0.1 Nm



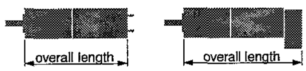
Technical Data

Spur Gearhead	straight teeth
Housing	plastic
Output shaft	stainless steel, hardened
Bearing at output	sleeve bearing
Radial play, 8 mm from flange	max. 0.038 mm
Axial play	0.03 - 0.30 mm
Max. perm. radial load, 8 mm from flange	5 N
Max. permissible axial load	8 N
Max. permissible force for press fits	500 N
Average backlash no load	< 2.5°
Recommended input speed	< 4000 rpm
Recommended temperature range	-20/+65°C

Stock program
Standard program
Special program (on request!)

Order Number		110480	110481	110482	110483	110484	110485	110486
Gearhead Data								
1 Reduction		7.2 : 1	20 : 1	32 : 1	64 : 1	131 : 1	199 : 1	325 : 1
2 Reduction absolute		93/13	121915/6153	151813/4695	637/13	212629/1625	778642/3909	1093365/3362
3 Number of stages		2	4	4	4	4	6	6
4 Max. continuous torque at gear output	Nm	0.1	0.1	0.1	0.1	0.1	0.1	0.1
5 Intermittently permissible torque at gear output	Nm	0.15	0.15	0.15	0.15	0.15	0.15	0.15
6 Sense of rotation, drive to output		=	=	=	=	=	=	=
7 Max. efficiency	%	81	66	66	66	66	53	53
8 Weight	g	25	28	28	28	28	30	30
9 Gearhead length L1*	mm	13.7	17.4	17.4	17.4	17.4	21.2	21.2

*for A-max 19 is L1 + 2.8 mm

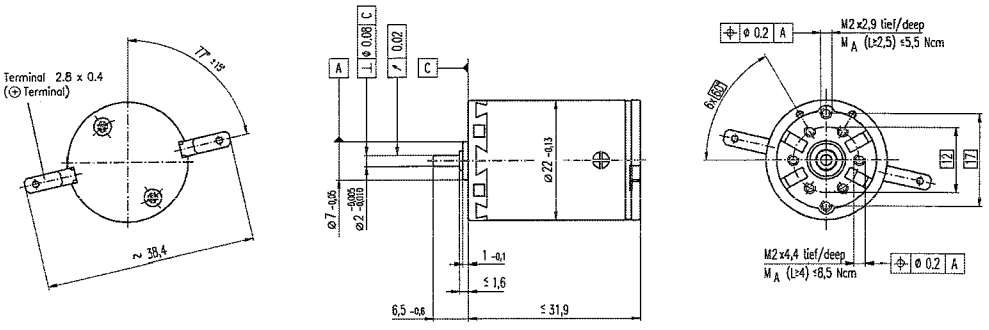


Combination:

+ Motor	Page	+ Tacho / Brake	Page	Overall length [mm]						
A-max 19	109-112			45.4	49.1	49.1	49.1	49.1	52.9	52.9
A-max 19	109/111	Digital Magnetic Encoder 13	212	52.9	56.6	56.6	56.6	56.6	60.4	60.4
A-max 19	110/112	Digital Encoder 22	203	59.9	63.6	63.6	63.6	63.6	67.4	67.4
A-max 22	113-116			45.6	49.3	49.3	49.3	49.3	53.1	53.1
A-max 22	113/115	Digital Magnetic Encoder 13	212	52.7	56.4	56.4	56.4	56.4	60.2	60.2
A-max 22	114/116	Digital Encoder 22	203	60.1	63.8	63.8	63.8	63.8	67.6	67.6
RE-max 21	137-140	MR Encoder on request		45.4	49.1	49.1	49.1	49.1	52.9	52.9

April 2001 edition / subject to change

A-max 22 ∅22 mm, Precious Metal Brushes CLL, 5 Watt, CE approved



Stock program
Standard program
Special program (on request!)

Order Number		110117	110119	110120	110121	110122	110123	110124	110125	110126	110127	110128	110129
Motor Data													
1 Assigned power rating	W	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
2 Nominal voltage	Volt	6.0	9.0	9.0	12.0	12.0	15.0	18.0	24.0	30.0	36.0	48.0	48.0
3 No load speed	rpm	9630	9970	8760	10400	9400	10300	9970	10700	10800	9800	9280	8370
4 Stall torque	mNm	20.7	23.2	20.7	24.6	21.6	23.6	22.6	24.2	24.2	21.5	19.5	17.9
5 Speed / torque gradient	rpm / mNm	469	433	426	428	438	439	444	445	451	460	480	472
6 No load current	mA	30	21	17	17	14	13	10	9	7	5	3	3
7 Starting current	mA	3510	2710	2130	2250	1790	1700	1320	1140	921	617	399	330
8 Terminal resistance	Ohm	1.71	3.32	4.22	5.33	6.71	8.82	13.6	21.0	32.6	58.4	120	145
9 Max. permissible speed	rpm	10600	10600	10600	10600	10600	10600	10600	10600	10600	10600	10600	10600
10 Max. continuous current	mA	840	751	666	593	528	461	371	298	240	179	125	113
11 Max. continuous torque	mNm	4.96	6.43	6.48	6.47	6.39	6.39	6.34	6.33	6.29	6.23	6.10	6.15
12 Max. power output at nominal voltage	mW	5210	6030	4740	6680	5310	6320	5890	6780	6840	5490	4730	3910
13 Max. efficiency	%	83	84	83	84	83	84	83	84	84	83	83	82
14 Torque constant	mNm / A	5.90	8.55	9.73	10.9	12.1	13.9	17.1	21.2	26.3	34.8	49.0	54.3
15 Speed constant	rpm / V	1620	1120	981	875	790	689	558	450	364	274	195	176
16 Mechanical time constant	ms	19	19	19	19	19	18	18	18	18	18	19	19
17 Rotor inertia	gcm²	3.88	4.10	4.16	4.13	4.04	4.03	3.97	3.96	3.91	3.84	3.70	3.75
18 Terminal inductance	mH	0.11	0.22	0.29	0.36	0.45	0.59	0.89	1.37	2.10	3.69	7.30	8.98
19 Thermal resistance housing-ambient	K / W	20	20	20	20	20	20	20	20	20	20	20	20
20 Thermal resistance rotor-housing	K / W	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
21 Thermal time constant winding	s	9	10	10	10	9	9	9	9	9	9	8	9

Specifications

- Axial play 0.05 - 0.15 mm
- Max. sleeve bearing loads
  axial (dynamic) 1.0 N
  radial (5 mm from flange) 2.8 N
  Press-fit force (static) 80 N
- Max. **ball bearing** loads
  axial (dynamic) 3.3 N
  radial (5 mm from flange) 12.3 N
  Press-fit force (static) 45 N
- Radial play sleeve bearing 0.012 mm
- Radial play **ball bearing** 0.025 mm
- Ambient temperature range -30 / +85°C
- Max. rotor temperature +85°C
- Number of commutator segments 9
- Weight of motor 54 g
- Values listed in the table are nominal.
  For applicable tolerances (see page 43).
  For additional details please use the maxon selection program on the enclosed CD-Rom.
- CLL = Capacitor Long Life

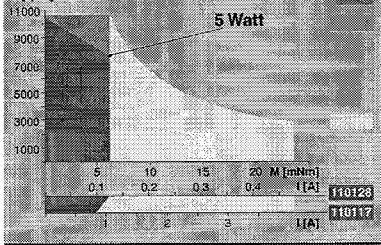
- Options: Ball bearings in place of sleeve bearings and pigtails in place of terminals, without CLL.

April 2002 edition / subject to change

Operating Range

Comments

Details on page 49



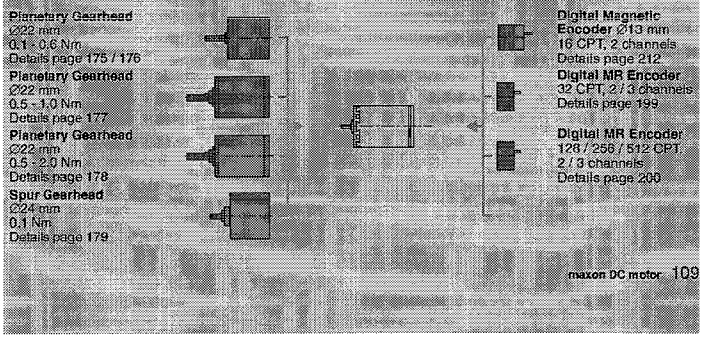
Recommended operating range

Continuous operation
In observation of above listed thermal resistances (lines 19 and 20) the maximum permissible rotor temperature will be reached during continuous operation at 25°C ambient.
= Thermal limit.

Short term operation
The motor may be briefly overloaded (recurring).

110128 Motor with high resistance winding
110117 Motor with low resistance winding

maxon Modular System



Planetary Gearhead
ø22 mm
0.1 - 0.6 Nm
Details page 175 / 176

Planetary Gearhead
ø22 mm
0.5 - 1.0 Nm
Details page 177

Planetary Gearhead
ø22 mm
0.5 - 2.0 Nm
Details page 178

Spur Gearhead
ø24 mm
0.1 Nm
Details page 179

Digital Magnetic Encoder ø13 mm
16 CPT, 2 channels
Details page 212

Digital MR Encoder
32 CPT, 2 / 3 channels
Details page 199

Digital MR Encoder
128 / 256 / 512 CPT,
2 / 3 channels
Details page 200

Appendix B

The GetRSS-algorithm in Matlab

B.1 GETRSS.M

```
%--------------------initialize---------------------%
clear all
load data           %get data saved by makesignal.m: x t w rs n fr shf rsp ls ww A fs drift
load poladjust      %data to adjust the influence of the polynomial removal to the sequence
disp(' ')
sel=input(['Select a channel between 1 and ',int2str(length(w)),': ']); %channel selection
wsl=w(sel);         %grid resolution
rsl=rs(:,sel);      %Rudin-Shapiro-sequence of selected channel
rsps=rsp(sel);      %datapoint of Rudin-Shapiro-sequence
mss=0;              %mss=0: indicator for correctness of founded sequence
pdg=3;              %polynomial degree remove of Gramm-Schmitt-algorithm
nop=16;             %nummer of parts for Gramm-Schmitt-algorithm
poladj=s1024p3n16;  %adjustment of RS-point caused by polynomial removal


%////////////////////////// INITIALIZATION PHASE ///////////////////////////%

fff=flops;          %determine flops
disp(' ') disp('#### INITIALIZATION #### ')
disp(['selected channel = ',int2str(sel)])
disp(['   - RS-point = ',int2str(rsps)])
disp(['   - emitter frequency = ',sprintf('%0.1f',fs(sel)),'Hz'])
disp(['   - frequency error = ',sprintf('%0.1f',drift(sel)),'Hz'])
disp(['   - amplitude = ',sprintf('%0.2f',A(sel))])


GO=rand*wsl;        %randomly select start grid offset

%search options
varo=10;            %amount of adjustments of the grid offset for each grid width
otwg=12;            %amount of optimize steps
tlwa=0.0015;        %adjustment of the grid width
wcr=.0005;          %starting adjustment of the grid width when optimizing
ocr=.75;            %starting adjustment of the grid offset when optimizing
paop=.5;            %maximum amplitude of noiselevel with respect to the amplitude of the RS-point

search=1;ff=0;lf=0;ss=0;ntry=0;crm=0;tmp3=1;
perf=[];Cwsl=[];CGO=[]; CCwsl=[];CCGO=[];CCbsh=[];CC2mi=[];
disp('adjust/optimize grid offset and width...')

%find the specific Rudin-Shapiro Sequence
while search
   ntry=ntry+1;
   ex=ps(x(round(GO+wsl:wsl:GO+ls*wsl)),pdg,nop);
   CCGO(ntry)=GO;
   CCwsl(ntry)=wsl;

   %back-shifting of the sequence
   if ss==0
      Rx=xcorr(ex(1:ls),rsl(1:ls),ls/2,'unbiased');   %cross corrolation
      bsh=mod(find(Rx==max(Rx))-ls/2-1,ls);bsh=bsh(1);
   end
   ex=[ex(ls+1-bsh:ls);ex(1:ls-bsh)];
   CCbsh(ntry)=bsh;

   %get RS-points
   sgnl=RSTfast(ex);        %Rudin-Shapiro-Transformation of extracted sequence
   sgnl(rsps)=sgnl(rsps)/poladj(ceil(rsps/nop/2));
   CC2mi(ntry)=sgnl(rsps)-min(sgnl([1:rsps-1 rsps+1:ls]));
   CCperf(ntry)=sgnl(rsps);

   %check if the founded sequence is correct
   dpt=find(sgnl==min(sgnl));dpt=dpt(1);
```

```
if dpt==rsps & ss==0 & sgnl(dpt)*(1-paop)>CC2mi(ntry)
   ss=1;
   best=1;
   bestperf=sgnl(dpt);
end

%optimizing
if ss>0 %specific sequence found
      Cwsl=[Cwsl wsl];
   CGO=[CGO GO];
   if dpt==rsps
      perf(ss)=sgnl(dpt);
   else
      perf(ss)=0;
   end

   %select the best of three
   if crm==2 | crm==4
      tmp=[bestperf perf(ss-1:ss)];
      tmp2=[best ss-1 ss];
      best=tmp2(find(tmp==min(tmp)));best=best(1);
      bestperf=perf(best);
      wsl=Cwsl(best);
      GO=CGO(best);
   end

   %change search settings for next crm-loop
   if crm==4
         wcr=wcr/2;
         ocr=ocr/2;
      crm=0;
   end

   %adjust search settings in crm-loop
   switch crm
   case 0
      wsl=wsl+wcr;
   case 1
      wsl=wsl-2*wcr;
   case 2
      wsl=wsl+ocr/600;
      GO=GO-ocr;
   case 3
      wsl=wsl-2*ocr/600;
      GO=GO+2*ocr;
   end

   %stop optimizing if...
   if ss>otwg
      wsl=Cwsl(best);
      GO=CGO(best);
      search=0;
      ss=ss-2;
      pb=ntry-otwg+best-2;
   end

   %next round
   crm=crm+1;
   ss=ss+1;
else %if no specific sequence is found
   ff=ff+1;
   if ff<varo %adjust grid offset varo times
      if dpt==rsps
         GO=mod(GO+(rand+.5)*wsl/8,wsl);
         if ntry>1
            if xor(CC2mi(ntry)<CC2mi(ntry-1),tmp3)
```

```
                tmp3=0;
                GO=mod(GO-2*wsl/8,wsl);
             else
                tmp3=1;
             end
          end
       else
          GO=mod(GO+(rand+.5)*wsl/3,wsl);
       end
    else %adjust grid width
       lf=lf+1;
       ff=0;
       if lf==1
          wsl=wsl+tlwa;
       elseif lf==2
          wsl=wsl-tlwa*2;
       else %stop and try the best match found
          search=0;
          pb=find(CC2mi==min(CC2mi));pb=pb(1);
          bsh=CCbsh(pb);
          wsl=CCwsl(pb);
          GO=CCGO(pb);
          ff=ff-1;
       end
    end
  end
end
ntry=ntry-1;
disp(['changed grid offset/width ',int2str(lf*varo+ff),' time(s)   optimized ',int2str(ss),
' times total: ',int2str(ntry)])
disp(['total flops needed for initialization = ',int2str(flops-fff)]) %determine flops

%get the signal
ex=ps(x(round(GO+wsl:wsl:GO+wsl*ls)),pdg,nop);
ex=[ex(ls+1-bsh:ls);ex(1:ls-bsh)];
sgnl=RSTfast(ex);
sgnl(rsps)=sgnl(rsps)/poladj(ceil(rsps/nop/2));
dpt=find(sgnl==min(sgnl));dpt=dpt(1);
if dpt==rsps
  mss=1;
end
err=(A(sel)*ls/2+sgnl(rsps))/ls*200;




%check if the founded signal is correct
if mss==1
  mes='correct';
  disp(['The initialization was succesful with an error of: ',int2str(err),'%'])
  disp(' ');disp(['#### LOOPING THROUGH SEQUENCES IN CHANNEL ',int2str(sel),' (of ',int2str(length(w)),') ####'])
else
  mes='wrong';
  disp('The sequence could not be found')
end
b1=GO+(ls+1-bsh)*wsl;       %first position of the first whole RSS [# samples]
b2=b1+(ls-1)*wsl;           %last position of the first whole RSS [# samples]




%--------------------plot---------------------%
figure(1),clf subplot(311),plot(0:ntry,CCwsl,'.-'),hold on
plot([lf*varo+ff lf*varo+ff],[min(CCwsl) max(CCwsl)],'r:',ntry,ww*fr/fs(sel),'k*',[pb pb],
```

```
[min(CCwsl) max(CCwsl)],'g:')
title('search for grid offset and width'),ylabel('grid width [samples]'),axis tight

subplot(312),plot(0:ntry,CCGO,'.-'),hold on
plot([lf*varo+ff lf*varo+ff],[min(CCGO) max(CCGO)],'r:',[pb pb],[min(CCGO) max(CCGO)],'g:')
ylabel('grid offset [samples]')
axis tight

perferr=(A(sel)*ls/2+CCperf)/ls*200;
subplot(313),plot(0:ntry,perferr,'.-'),hold on
plot([lf*varo+ff lf*varo+ff],[min(perferr) max(perferr)],'r:',[pb pb],[min(perferr) max(perferr)],'g:')
xlabel('search number'),ylabel('amplitude error %'), axis([0,ntry,min([0 perferr]),max([0 perferr])])


figure(2),clf
subplot(3,1,1),plot(fr*t(1:round(GO+wsl*ls)),x(1:round(GO+wsl*ls))),axis
tight title('examined part of signal')
xlabel('samples'),ylabel('values')

subplot(3,1,2),plot(-ls/2:ls/2,Rx),axis tight
title(['finding shift of sequence using ',int2str(ls),' datapoints'])
xlabel('sample difference'),ylabel('cross corrolation E[x(k)x(k+\tau)]')

subplot(3,1,3),plot(sgnl),axis tight
title(['point = ',int2str(rsps),'    min.point = ',int2str(dpt),'  (',mes,') fs=',int2str(ww/wsl*fr),' Hz'])
xlabel('Rudin-Shapiro points'),ylabel('intensity')




%////////////////////////// STEP PHASE ////////////////////////////%

%--------------------initialize---------------------%

%search options
ac=3;                           %adjust resolution [1/(# samples)]
ma=2;                           %maximum adjustment + 1/ac [# samples]
maxwait=3;wait=0;cntwait=0;     %amount of skips when wrong signal is found
ww2=ls/8;                       %amount of samples to examine for adjustments
pdg2=2;                         %polynomial degree
nop2=2;                         %number of parts
poladj=s128p2n2;                %adjustment of RS-point caused by polynomial removal
YY=zeros(ma*ac*2-1,1);          %size of RS-p
wcrf=1/6;                       %grid width correction filter
CCC=[];ttttt=0;                 %temp data collection variables

%getting the RS-point of a part of the total sequence
tmp=RSTfast(rsl(1:ww2));        %Rudin-Shapiro-transformation of part
sgn=sign(max(tmp)+min(tmp));    %determine the direction of the peak
tmp=tmp*sgn;                    %adjusting tmp
zzz=find(tmp==max(tmp));        %RS-point of part


%------------loop until the end of the signal or until lost of signal----------%

while b2<=n & mss
   fff=flops;                   %determine flops

   %changing offset of sequence
   CCtmp=[];
   for i=1:ma*ac*2-1
      tmp=sgn*RSTfast(ps(x(round( [b1:wsl:b1+wsl*(ww2-1)] +i/ac-ma )),pdg2,nop2))/poladj(ceil(zzz/nop2/2));
      YY(i)=tmp(zzz);
```

```
   CCtmp=[CCtmp;tmp(zzz) max(tmp([1:zzz-1 zzz+1:length(tmp)]))];
end


%determine the best correction
maxYY=max(YY);
crmn=mean(find(YY==maxYY));
cr=crmn/ac-ma;
crmn=round(crmn);
if crmn==1
   crmn=[crmn; crmn+1];
elseif crmn==ma*ac*2-1
   crmn=[crmn-1; crmn];
else
   crmn=[crmn-1; crmn; crmn+1];
end
crmean=[crmn/ac-ma]'*YY(crmn)/sum(YY(crmn));
tmp=sgn*RSTfast(ps(x(round([b1:wsl:b1+wsl*(ww2-1)] + crmean)),pdg2,nop2))/poladj(ceil(zzz/nop2/2));
if tmp(zzz)>=maxYY
   cr=crmean;
end
 b1=b1+cr;                                    %make offset correction
nexex=x(round(b1:wsl:b1+wsl*(ls-1)));         %get next sequence out of signal x
nexex=ps(nexex,pdg,nop);                      %polynomial removal
nexsgnl=RSTfast(nexex);                       %Rudin-Shapiro-Transformation
nexsgnl(rsps)=nexsgnl(rsps)/poladj(ceil(rsps/nop/2));
err=(A(sel)*ls/2+nexsgnl(rsps))/ls*200;       %determine error

CCC=[CCC;size(CCC,1)+1 err cr ttttt ww/wsl*fr-fs(sel)];
if size(CCC,1)==1
   disp(['flops needed for one loop: ',int2str(flops-fff)])
end
disp([int2str(CCC(end,1)),'.   ampl.error = ',int2str(err),'%   offset-cr = ',sprintf('%0.3f',CCC(end,3)),
                              '   fs-error = ',sprintf('%0.2f',CCC(end,5)),'Hz']); %determine flops




%check if the founded sequence is correct
dpt=find(nexsgnl==min(nexsgnl));dpt=dpt(1);
if dpt~=rsps
   mes='wrong';
   wait=wait+1;
   cntwait=cntwait+1;
   if wait>maxwait
      mss=0;
      disp('The sequence is totally lost')
   else
      disp('The peak-value is not the correct RS-point, try next sequence')
   end
else
   if wait>0
      wait=0;
      mes='correct';
      disp('The correct RS-point has been relocated')
   end
end



%--------------------plot----------------------%

figure(3),clf,subplot(2,1,1),plot(t,x),hold on,axis tight
plot([t(round(b1)) t(round(b1))],[min(x) max(x)],'r')
```

```
   plot([t(round(b2)) t(round(b2))],[min(x) max(x)],'r')
   plot([t(round(b1+wsl*(ww2-1))) t(round(b1+wsl*(ww2-1)))],[min(x) max(x)],'r:')
   title(['stepping through sequences in channel ',int2str(sel),' (of ',int2str(length(w)),')'])
   xlabel('time [s]')

   RRtmp=[1:ma*ac*2-1]'/ac-ma;
   subplot(2,2,3)
   plot(RRtmp,CCtmp(:,1),'.b-',RRtmp,CCtmp(:,2),'.r:',[RRtmp RRtmp]',[zeros(size(RRtmp)) CCtmp(:,1)]','b:')
   hold on
   plot([cr cr],[0 max([tmp(zzz);CCtmp(:,1)])],'r')
   title(['cr = ',sprintf('%0.3f',cr),', fs=',sprintf('%0.1f',ww/wsl*fr),
                  'Hz(err.',sprintf('%0.1f',ww/wsl*fr-fs(sel)),'Hz)'])
   xlabel('correction [samples]'),ylabel('intensity'),axis tight

   subplot(2,2,4),plot(nexsgnl),axis tight
   title(['RS-point = ',int2str(dpt),'  (',mes,')    ampl.error= ',int2str(err),'%'])
   xlabel('Rudin-Shapiro points'),ylabel('intensity')

   %--------------prepare for next round--------------%

   if mss==1
        wsl=wsl+cr/ls*wcrf;     %correction of the grid width according to the founded offset correction
      b1=b1+ls*wsl;             %first location of next sequence
        b2=b1+(ls-1)*wsl;       %last location of next sequence
      pause                     %wait for user to continu
   end
   ttttt=cr/ls*wcrf;            %temp variable
end

if b2>n & mss & cntwait==0
   disp('The total search was succesful')
elseif b2>n & mss & wait==0
   disp(['The sequence has been ',int2str(cntwait),' time(s) lost, but every time it was relocated'])
elseif b2>n & mss
   disp('The peak-value is not the right RS-point, no next sequences available')
end

ovalpathplot;
```

B.2 MAKESIGNAL.M

```
%--------------------initialize---------------------%

close all

%values to be set
val={{4096, 4, 64,    1, 600000,   9,    0, .1, .2,  0,  0},       %experimental setup
     {4096, 1, 1024, 16,  48000,  15, 1000, .1,  3, .5, 40},       %final application
     {4096, 1, 1024,  1,  48000,  15,    0, .0, 10, .8, 40},       %ideal situation
     {4096, 1, 1024,  1,  48000,   5,    0,  0,  0,  0, 0}};       %testing
slv=4;                      %selection of data

%setting data
ns=val{slv}{1};             %number of samples for one Rudin-Shapiro-Sequence
ws=val{slv}{2};             %width of pulses [# samples]
ls=val{slv}{3};             %length of Rudin-Shapiro-Sequence
nn=val{slv}{4};             %number of sequences in the signal
fr=val{slv}{5};             %sample frequency of receiver [Hz]
tl=val{slv}{6};             %ratio of size of measured signal and largest sequence length
sdr=val{slv}{7};            %SD of the sample frequencies of the light emitters [Hz]
noi=val{slv}{8};            %SD of added noise to the signal
amp=val{slv}{9};            %amplitude of the 100Hz sine disturbance
vi=val{slv}{10};            %variation of intensities
unk=val{slv}{11};           %maximum unknown frequency drift at the start

%random features
fs=fr-sdr*(rand(nn,1)-.5);                              %sample frequency of light emitters
rsp=1+floor(ls/nn*(rand(nn,1)+[0:nn-1]'));rsp=1;        %place of '1' in the Rudin-Shapiro-Sequence
shf=rand(nn,1)*ls;shf=0;                                %amount of shift to the right
A=1+vi*(rand(nn,1)-.5);                                 %intensity of sequences

%setup other stuff
ww=ns/ls;                   %grid resolution of sequences of the emitters
drift=unk*(rand(nn,1)-.5);  %drift
w=ww*fr./(fs+drift);        %grid resolution of sequences in measured sequence
n=round(tl*fr*ns/min(fs));  %length of measured sequence
xs=zeros(ns,nn);            %transformed and shifted sequences
x=zeros(n,1);               %measured signal
rs=zeros(ls,nn);            %Rudin-Shapiro unity-vector

%-----------------create signal--------------------%

%adding each channel to the receiver signal x
for i=1:nn
   rs(:,i)=RSTfast([zeros(rsp(i)-1,1);eye(ls-rsp(i)+1,1)])<0;   %Rudin-Shapiro-Transformation
   for j=1:ws
      xs(ww*[1:ls]+j-1,i)=A(i)*rs(:,i);                         %emitter signal
   end
   x=x+xs(mod(floor([(1:n)+shf(i)]'*fs(i)/fr)-1,ns)+1,i);       %receiver signal
end

%adding disturbances
t=[0:1/fr:(n-1)/fr]';           %time range
x=x+noi*randn(n,1);             %adding noise
x=x+amp*(1+sin(2*pi*100*t));    %adding 100Hz sine

%save data
save data x t w rs n fr shf rsp ls ww A fs drift

%--------------------plot-----------------------%

figure(1) clf plot(t,x)
axis([0,n/fr,min(x),max(x)])
xlabel('t[s]')
ylabel('measured signal x')
```

B.3 Description of getrss.m

Initialization-phase

The created measured signal by makesignal.m will be loaded and the user specifies the channel to observe. Using a random grid offset and the estimated grid width (provided by makesignal.m) 1024 samples of the first part will be taken. The polynomial contents will be removed by another algorithm. The options for this are the maximum polynomial degree to remove and the amount of parts to divide the signal into. With a sample frequency of 48000Hz, 1024 samples and a pulse width of about 4 the 100Hz disturbance results in about 8,5 sample times in the extracted sequence. When using only a 2nd order polynomial removal the signal must be divided into 16 parts, for 3th order about 8 parts etc. Trying some combinations led to the choice 3th order with 16 parts. (It is necessary that this is a power of 2 to prevent unknown side-effects with the Rudin-Shapiro Transform)

Now the full cross-correlation between this signal and the specific Rudin-Shapiro sequence can be made. if the grid offset and width are near the correct value a nice peak will show up. This peak will be used to shift the extracted sequence. Then this is transformed back with the Rudin-Shapiro Transform. If the correct peak shows up with an amplitude greater than twice the maximum amplitude of the other points the optimize loop will be started. If this is not the case the grid offset will be changed. If the peak was the correct one, then the grid offset will be changed only a little because the correct offset has to be close. (a change of (rand+0,5)*width/8 will be made, with "rand" an uniform distribution $0 <= x < 1$). When the peak was wrong there will be made a big change: (rand+0,5)*width/3 because the optimum offset is not nearby. After changing the grid offset 8 times and still no good solution is found the grid width will be changed (+0.0015) and again 8 times will be tried to find a good offset. After that a last attempt to change the grid width is done (-0.0015). If the optimize loop still is not triggered then the best result will be used. With the best is meant, the maximum difference between the amplitude of the RS-point and the rest of the data.

If there has been found a good enough grid offset and width combination the optimize loop will start. In 12 steps it will go in the direction of the best combination. Now the best is defined as the greatest amplitude of the RS-point. You can see this as finding the top of a mountain with the grid offset and width as the X en Y-coordinates. In figure B.1 you see this landscape together with a simulation of the iterate steps. The landscape represents the maximum available data of the original sequence in the measured sequence. Of course in real it is not as smooth, because of all other disturbances is a lot less smooth. The dotted line is the first search for a suitable grid offset and width pair. The solid line represents the optimize loop.

First it will start with determining the amplitude for $\pm 0,0005$. The best of these three is chosen and used for the next iteration. Now the offset will be changed $\pm 0,5$ and the width will change according to the direction of the main direction of the visual oval. (gradient about -600) The next round all adjustments are reduced by a factor two. This will go on until 12 steps are made. Because of the disturbances this will not always converge to the real grid offset and width but to some local maximum. But this is no problem, the sequence has been found and the grid offset and width are close enough to keep the sequence in track.

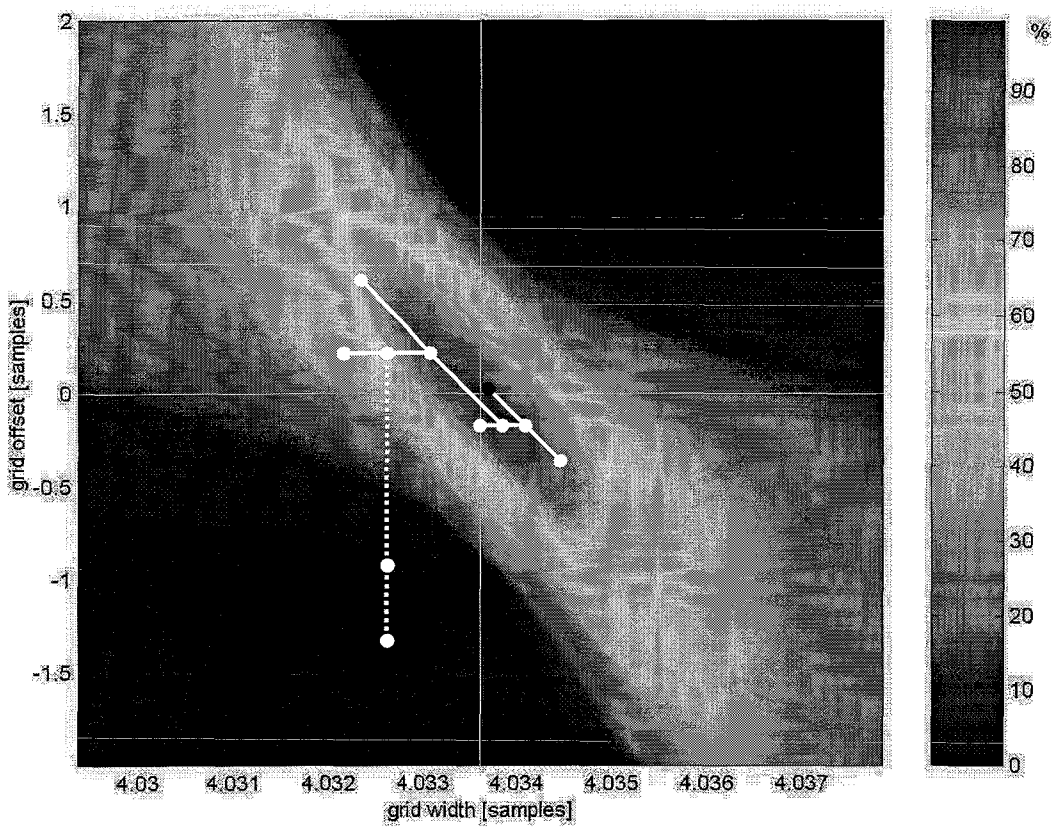


Figure B.1: Maximum available data landscape with a emitter frequency of 47600Hz and a receiver frequency of 48000Hz.

Step-phase

The initialization-phase will return the amount of shift of the sequence, a grid offset and a grid width which can be used to find the first whole sequence in the signal. Then only the first 1/8 part of the specific Rudin-Shapiro sequence and the extracted sequence points is taken. Because of a shorter sequence the algorithm to remove polynomial contents has to be adjusted. Now only 2nd order with 2 parts will be used. The fact that a part of a Rudin-Shapiro sequence is again a Rudin-Shapiro sequence is used here. There is a defined relation between the RS-point of the full sequence and the RS-point and it is amplitudes sign of the 1/8 part. The amplitude of the RS-point in the extracted sequence points will be collected for several shifts of the grid. How many shifts is a trade-off between robustness and calculation-speed. Here it is chosen -5/3:1/3:5/3 sample-shift. The maximum amplitude is determined and also the amplitude of the mean of the maximum plus and minus one shift resolution. The best of these two will be used for the correction of the grid offset. Also a check is made for the whole sequence if the RS-point is correct. The shift in grid offset can be caused by disturbances and a slightly wrong grid width. So according to the grid offset correction the grid width will be corrected. If there would be no disturbances the right correction could be made in one time, namely correction/1024. But this leads to jumpy effects of the grid width caused by disturbances so the correction is reduced by factor 6. It is like a low-pass filter. The

grid width will go more slowly to the right value but it will stay more near it when it get's there. Now the next sequence can be examined. If the end of the signal has been reached or 3 times the founded RS-point was incorrect then stop the step-phase.

B.4 Simulation with getrss.m

When getrss.m is started you'll be asked to select a channel. After that some information of the initialization-phase will appear. Next the step-phase will appear with every loop a pause and a update of the figure. An example of the results in the command window you can see below:

```
Select a channel between 1 and 16: 8

#### INITIALIZATION ####
selected channel = 8
   - RS-point = 510
   - emitter frequency = 47723.6Hz
   - frequency error = -16.4Hz
   - amplitude = 0.93
adjust/optimize grid offset and width...
changed grid offset/width 5 time(s)   optimized 12 times   total: 17
total flops needed for initialization = 7960106
The initialization was succesful with an error of: -10%

#### LOOPING THROUGH SEQUENCES IN CHANNEL 8 (of 16) ####
flops needed for one loop: 347397
1.   ampl.error = 8%   offset-cr = -0.001   fs-error = -0.88Hz
2.   ampl.error = 13%   offset-cr = 0.009   fs-error = -0.88Hz
3.   ampl.error = 20%   offset-cr = -0.328   fs-error = -0.89Hz
4.   ampl.error = 24%   offset-cr = 0.334   fs-error = -0.26Hz
5.   ampl.error = 1%   offset-cr = -0.333   fs-error = -0.90Hz
6.   ampl.error = 7%   offset-cr = 0.000   fs-error = -0.26Hz
7.   ampl.error = 4%   offset-cr = -0.000   fs-error = -0.26Hz
8.   ampl.error = 3%   offset-cr = -0.064   fs-error = -0.26Hz
9.   ampl.error = 6%   offset-cr = 0.008   fs-error = -0.14Hz
10.   ampl.error = 2%   offset-cr = 0.000   fs-error = -0.15Hz
11.   ampl.error = -4%   offset-cr = -0.002   fs-error = -0.15Hz
12.   ampl.error = 33%   offset-cr = 0.148   fs-error = -0.15Hz
13.   ampl.error = -1%   offset-cr = -0.317   fs-error = -0.43Hz
14.   ampl.error = 49%   offset-cr = 0.667   fs-error = 0.18Hz
The total search was succesful
```

In the initialization-phase the grid offset and width will be adjusted. These adjustments are visualized in figure B.2. The first plot reflects the grid width adjustment. The second plot reflects the grid offset and the last plot shows the error of the amplitude of the RS-point. The red dotted line indicates where the optimize loop starts and the green dotted line indicates which grid offset and width is chosen as the best. The error is minimal there. The black star in the first plot is the real grid width.

In figure B.3 you see the adjustments of the grid again but then plotted in a simulated landscape of how much data is maximally available for each combination. The white cross indicates the exact position of the top of the mountain. The white dotted line represents the randomly changing of the grid offset and/or width (in this case only the offset). The white solid line represents the optimization loop. As expected it converges near the optimum. The accuracy is good enough, because the step-phase has some robustness.

The best choice indicated by the green dotted line in figure B.2 is presented in figure B.4. The first plot is the first part of the measured signal including the whole grid. The next plot is the autocorrelation of the extracted sequence points with the specific Rudin-Shapiro

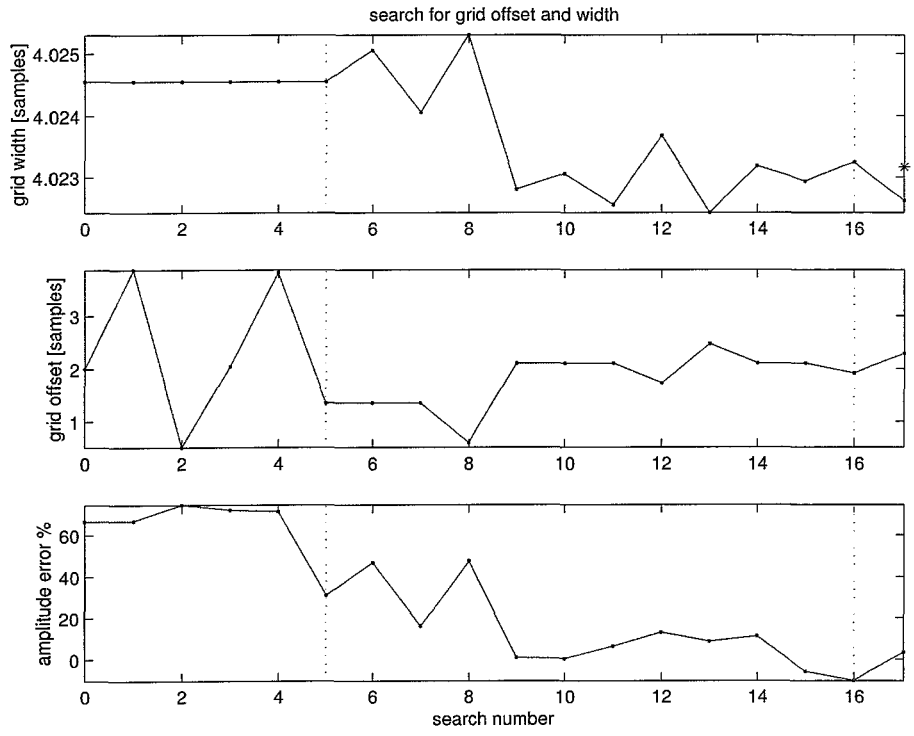


Figure B.2: Adjustment of grid offset and width during the initialization-phase

sequence. There is a peak at point 241 so the data has to be shifted by that amount to line up the sequence. The last plot is the sequence transformed back. The correct RS-point appears. Compared to the amplitude the noise level is quite low. Keep in mind that there are 15 other sequences, a sine and noise in the measured signal.

Then the step-sequence starts working. The first step of this can be seen in figure B.5. In the first plot you see the entire measured signal. The red solid lines indicate the current Rudin-Shapiro sequence in the signal. The red dotted line indicates the 1/8 part of that sequence. The second plot shows the amplitude of the RS-point of the 1/8 part when shifting the sequence a little bit. (blue line) The red dotted line is the maximum amplitude of the rest of the 1/8 part. Because grid offset/width errors or noise the maximum can shift. The grid offset will be adjusted according to the maximum value. Also the grid width will be adjusted if the grid offset changes, because this is the main cause of the grid shift every step. When stepping to the next sequence the sample-position will be enlarged by the current width multiplied by 1024. A small error of the grid width leads to a change in the next grid offset. Using this fact the grid width can be corrected and it will converge to the value it should be. Also if there are changes of the emitter frequency this can be handled. (if the changes are not to large)

The algorithm is also tested with much more disturbances. Like a sine with an amplitude of 30, noise with a standard deviation of 0,5, 32 channels, variation of amplitudes of the emitted signals, accuracy of the start grid width and variation of the emitter frequencies.

A sine with a bigger amplitude is not much of a problem because the polynomial content

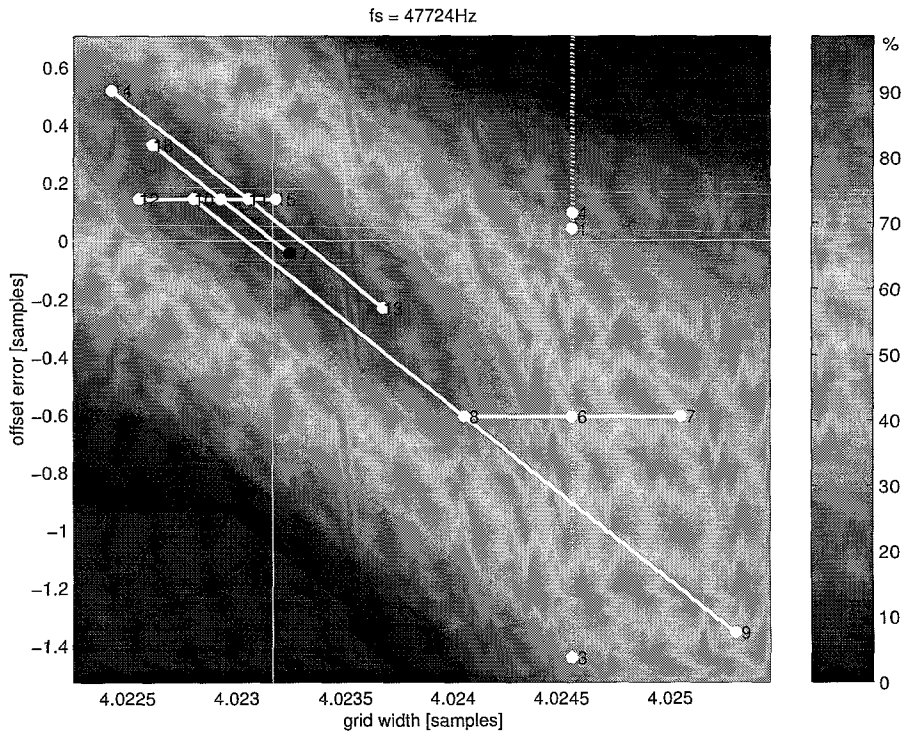


Figure B.3: Adjustment of grid offset and width during the initialization-phase plotted over the maximum amount of available data landscape

will be removed. Adding more noise will give a worse signal to noise ratio. This decreases the performance and sometimes with weak signals the sequence can be lost. Using 32 channels does not give a lot of problems. The sequences are orthogonal so they have a low level of correlation. When the amplitude is quite low this could become a problem in some cases. The sensitivity to disturbances changes in time, when having a small amplitude of the emitted sequence the disturbances will be sometimes to large to determine the best adjustment. When the variation of the emitter frequencies are low there is more correlation between the Rudin-Shapiro sequences. Sometimes another peak appears beside of the correct RS-point. This is no problem as long the correct RS-point is still present.

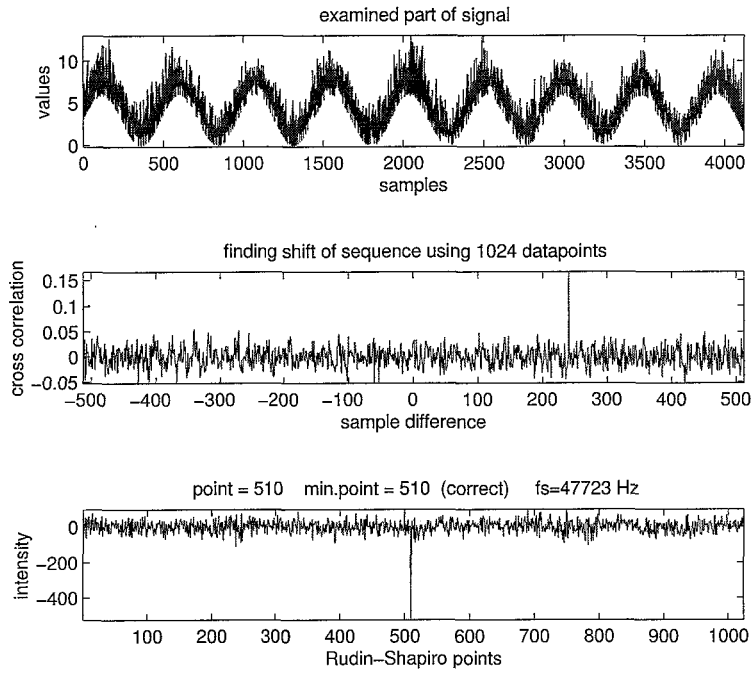


Figure B.4: Properties of the best match in the initialization-phase

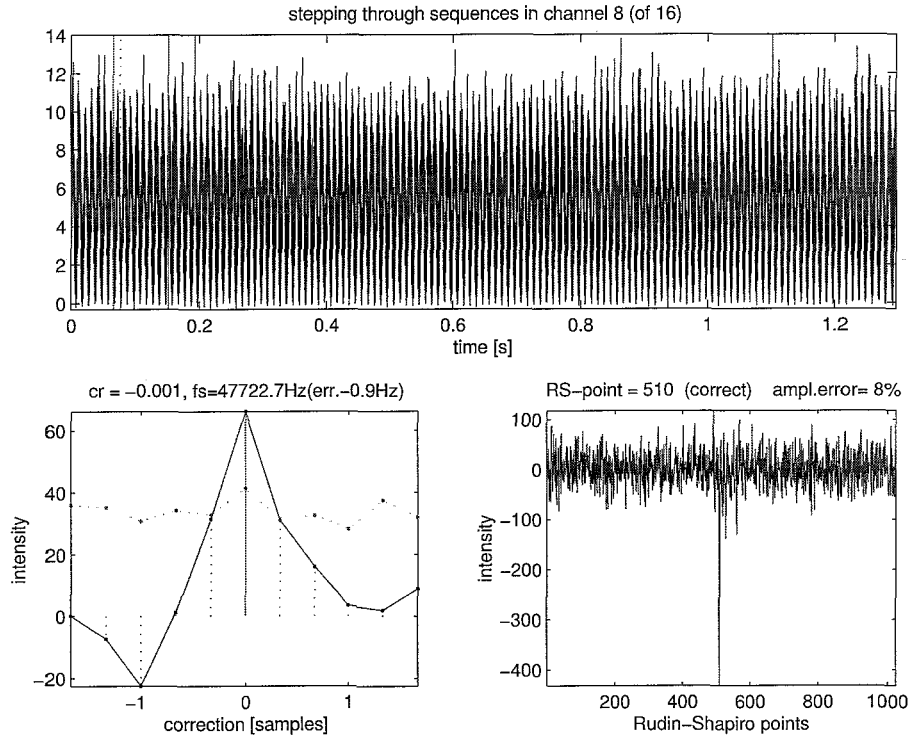
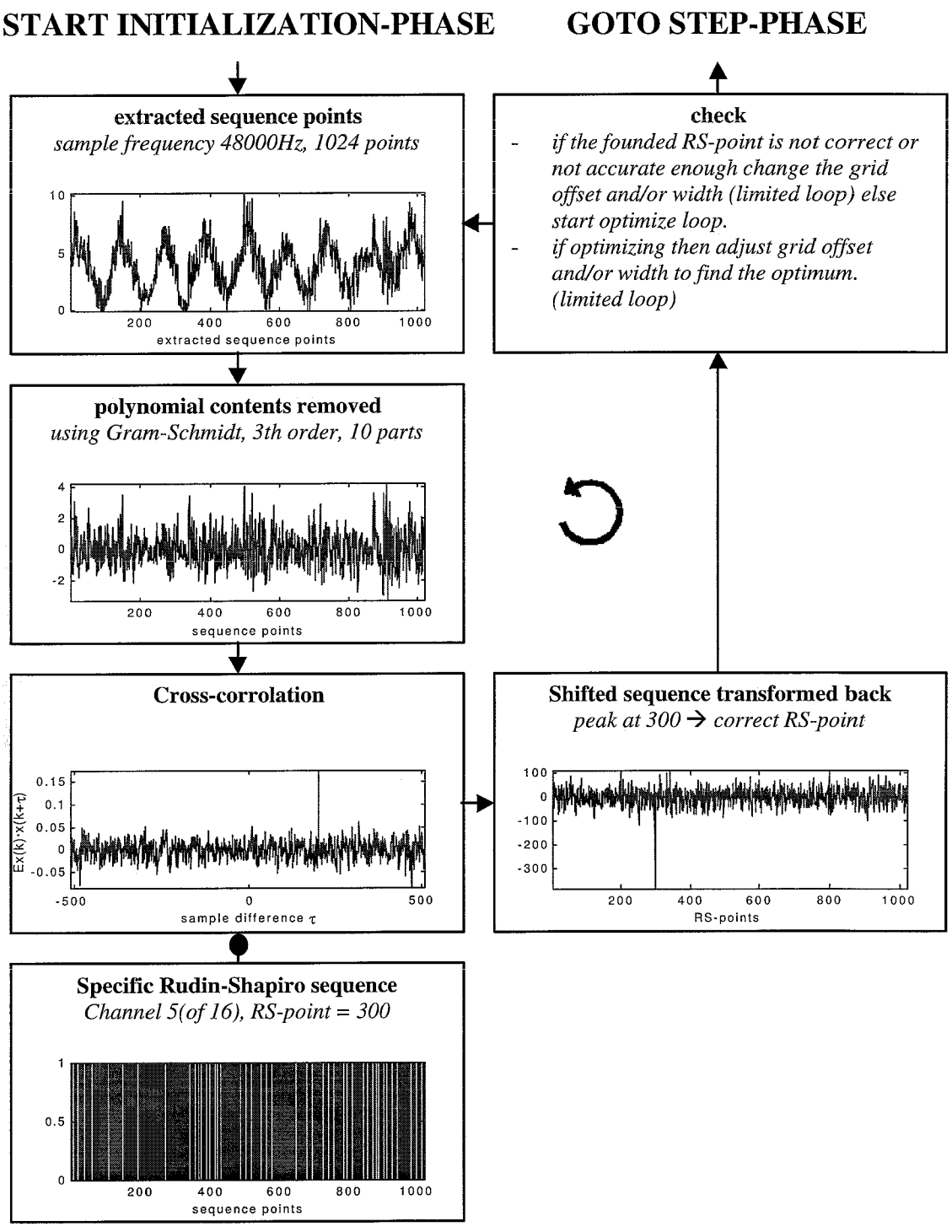
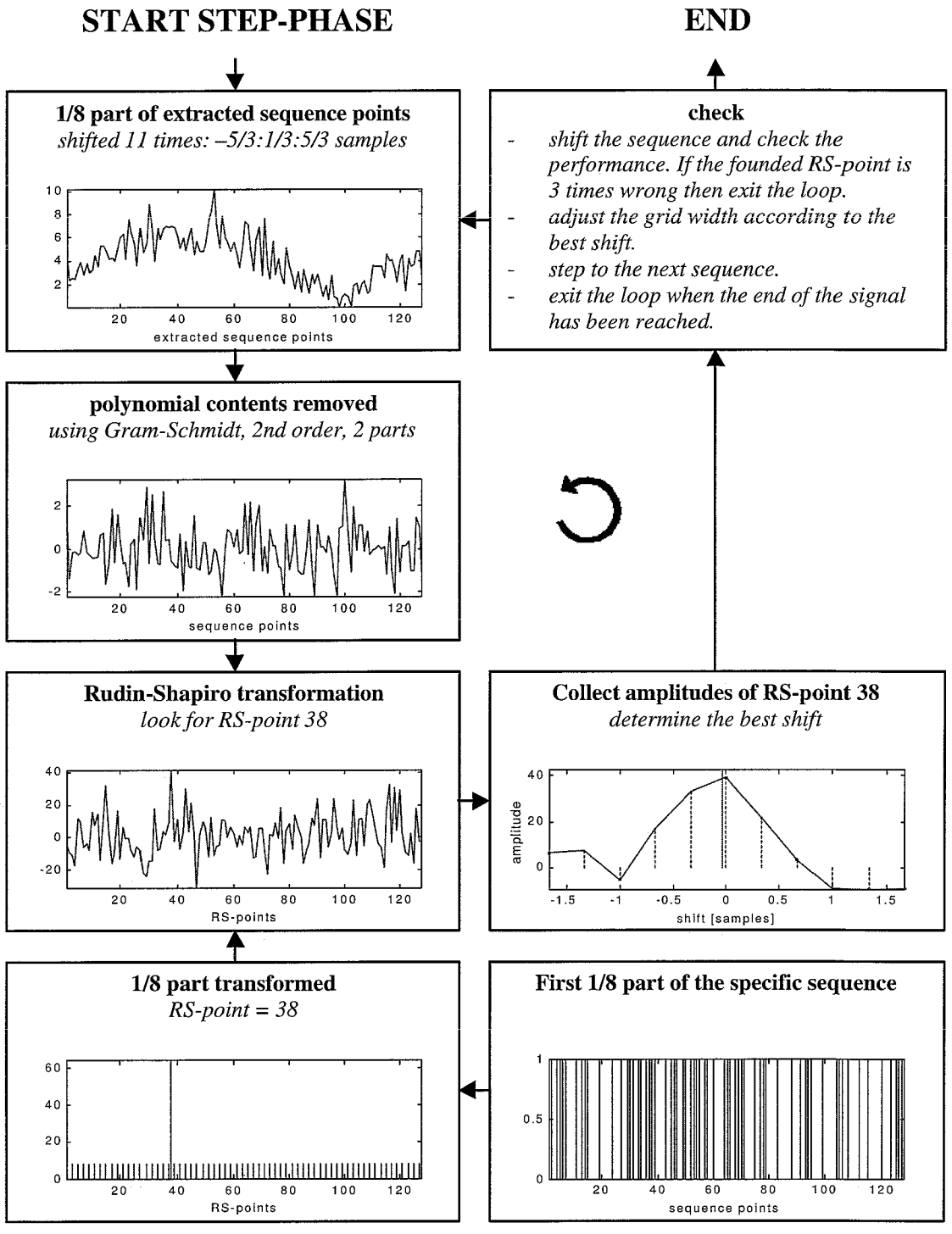


Figure B.5: Grid adjustment during step-phase using 1/8 part of a sequence

B.5 Visualization of getrss.m




START STEP-PHASE
END
1/8 part of extracted sequence points
shifted 11 times: –5/3:1/3:5/3 samples
extracted sequence points
check
- shift the sequence and check the performance. If the founded RS-point is 3 times wrong then exit the loop.
- adjust the grid width according to the best shift.
- step to the next sequence.
- exit the loop when the end of the signal has been reached.
polynomial contents removed
using Gram-Schmidt, 2nd order, 2 parts
sequence points
Rudin-Shapiro transformation
look for RS-point 38
RS-points
Collect amplitudes of RS-point 38
determine the best shift
amplitude
shift [samples]
1/8 part transformed
RS-point = 38
RS-points
First 1/8 part of the specific sequence
sequence points

Appendix C

C-code

C.1 Code of the Rudin-Shapiro Transform

```
/*********************************************/ /* Fixed
algorithms from this point and beyond. */
/*********************************************/

void RST(double *Signal, double *RetSig, int SigLen, int
TransSteps) {
  /* The original signal is not changed.
     The argument TransStep must be N, where 2^N = SigLen. */

    double Scale = (double)(1/sq2);

  int SigParts, SigPartLen, RetSigPartLen;
  double *SigPtr1, *SigPtr2;
  double *RetSigPtr1, *RetSigPtr2;
  double *FromSignal;
  double *ToSignal;

    int k, m, n;

  /* The very first transform step. Only change in time counter. */
  if (fmod(TransSteps,2)) ToSignal = RetSig; else ToSignal = RSTaux;

  SigPtr1 = Signal;
  SigPtr2 = &Signal[1];
  RetSigPtr1 = ToSignal;
  RetSigPtr2 = &ToSignal[SigLen/2];

  for (n = 0; n < SigLen/4; n++) {

    *RetSigPtr1++ = Scale * (*SigPtr1 + *SigPtr2);
    *RetSigPtr2++ = Scale * (*SigPtr1 - *SigPtr2);
    SigPtr1 += 2;
    SigPtr2 += 2;

    *RetSigPtr1++ = Scale * (*SigPtr1 - *SigPtr2);
    *RetSigPtr2++ = Scale * (*SigPtr1 + *SigPtr2);
    SigPtr1 += 2;
    SigPtr2 += 2;
  }

  SigPartLen = SigLen/2;
  RetSigPartLen = SigLen/4;
  SigParts = 2;

  /* The following transform steps, except the very last one,
     change in both time and frequency counters. */

  for (k = 1; k < TransSteps - 1; k++) {

    /* Determine proper To- and From signals. */
    if (fmod(TransSteps-k,2)) {
      FromSignal = RSTaux;
      ToSignal = RetSig;
    } else {
      FromSignal = RetSig;
      ToSignal = RSTaux;
    }

    SigPtr1 = FromSignal;
    SigPtr2 = &FromSignal[1];

    /* Frequency counter. */
    for (m = 0; m < SigParts/2; m++) {
```

```
      RetSigPtr1 = &ToSignal[2*m*SigPartLen];
      RetSigPtr2 = &ToSignal[2*m*SigPartLen + RetSigPartLen];

      /* Time counter. */
      for (n = 0; n < RetSigPartLen/2; n++) {
        *RetSigPtr1++ = Scale * (*SigPtr1 + *SigPtr2);
        *RetSigPtr2++ = Scale * (*SigPtr1 - *SigPtr2);
        SigPtr1 += 2;
        SigPtr2 += 2;

        *RetSigPtr1++ = Scale * (*SigPtr1 - *SigPtr2);
        *RetSigPtr2++ = Scale * (*SigPtr1 + *SigPtr2);
        SigPtr1 += 2;
        SigPtr2 += 2;
      }

      RetSigPtr1 = &ToSignal[(2*m+1)*SigPartLen];
      RetSigPtr2 = &ToSignal[(2*m+1)*SigPartLen + RetSigPartLen];

      /* Time counter. */
      for (n = 0; n < RetSigPartLen/2; n++) {
        *RetSigPtr1++ = Scale * (*SigPtr2 + *SigPtr1);
        *RetSigPtr2++ = Scale * (*SigPtr2 - *SigPtr1);
        SigPtr1 += 2;
        SigPtr2 += 2;

        *RetSigPtr1++ = Scale * (*SigPtr2 - *SigPtr1);
        *RetSigPtr2++ = Scale * (*SigPtr2 + *SigPtr1);
        SigPtr1 += 2;
        SigPtr2 += 2;
      }
    }

    SigPartLen /= 2;
    RetSigPartLen /= 2;
    SigParts *= 2;
  }

    /* The very last transform step. Only change in frequency counter. */
    RetSigPtr1 = RetSig;
    RetSigPtr2 = &RetSig[1];
  SigPtr1 = RSTaux;
  SigPtr2 = &RSTaux[1];

  for (m = 0; m < SigLen/4; m++) {
    *RetSigPtr1 = Scale * (*SigPtr1 + *SigPtr2);
    *RetSigPtr2 = Scale * (*SigPtr1 - *SigPtr2);
    SigPtr1 += 2;
    SigPtr2 += 2;
    RetSigPtr1 += 2;
    RetSigPtr2 += 2;

    *RetSigPtr1 =  Scale * (*SigPtr2 + *SigPtr1);  /* 3rd matrix. */
    *RetSigPtr2 =  Scale * (*SigPtr2 - *SigPtr1);
    SigPtr1 += 2;
    SigPtr2 += 2;
    RetSigPtr1 += 2;
    RetSigPtr2 += 2;
  }

} /* RudinShapiroTransform() */
```

This code is included in dwte.dll. To use it as the symmetric Rudin-Shapiro Transform a function RSTfast.m is made:

```
function y=RSTfast(x)

T = [1 -1 -1 1; 1 1 1 1; 1 1 1 1 ; 1 -1 -1 1];

y=dwte(x,T);
```

The T-matrix is necessary to use the symmetric form of the Rudin-Shapiro Transform. The length of the data must be a power of 2.

C.2 The GetRSS-algorithm in the C++ application

The GetRSS function is called within the callback function. The callback function can be found after the functions related to the GetRSS function. The rest of the C-code of the application is related to setting up the application, the data acquisition device and visualization purposes.

```
/*********************************************/
/*              RSS-track algorithm          */
/*********************************************/

void GetRss(double *x,double *ggo, double *ggw) {
    double G[n+no],G2[n+no];
    int k;


    if (P.phase==0) {   ////////////////////INITIALIZATION-PHASE//////////////////
        //get data using grid and remove the polynomial content
        GetGrid(x,G,P.o,P.w,n+no,1);

        //determine shift of data
        k=xcorr(G,&S.rss[da][0],n+no,G2);
        P.o-=k*P.w;
        //while (P.o<0) P.o+=(n+no)*P.w;
        if (P.o+P.w*n<sl*nbuf) {

            //back transformation of the shifted data
            GetGrid(x,G,P.o,P.w,n,1);
            RST(G, G2, n, n2);
            minmeanmm(G2,n,2./sqrt(n));
            P.amp=-G2[S.rsp];
            *ggo=P.o;
            *ggw=P.w;

            //check if the founded sequence is good enough
            P.dpt=fmin(G2,n);
            if ((P.dpt==S.rsp) && (-G2[fmin2(G2,n,S.rsp)]<P.paop*P.amp)) P.phase=1;
            else { //adjust grid offset/width
                P.w=P.wstart+.4*rand()/0x7fff-.2;
                if (P.dpt!=S.rsp) P.o+=P.w*rand()/0x7fff-P.w/2;
                else P.o+=sign(P.awo-P.amp)*pw/2*rand()/0x7fff;
                P.awo=P.amp;
            }
        }
        P.ntry++;
    }
    else { ////////////////////STEP-PHASE//////////////////////

        //adjust grid offset
        AdjustGridOffset(x);
        *ggo=P.o;
        *ggw=P.w;

        //change grid width according to the correction of the grid offset
        P.w+=P.cr/(n+no)*P.wcrf;
    }

    //check if the founded peak is the RS-Point
    if (P.dpt==S.rsp) {
        P.mss=true;
        for(k=0;k<9;k++) bb[k]=bb[k+1];
        bb[9]=P.w;
    }   else P.mss=false;
    if (P.dpt==S.rsp && P.phase==1) P.af=0;
```

```
    else {
        P.af++;
        if (P.af>P.maxaf) {
            P.af=P.maxaf;
            if (P.phase==1) P.wstart=bb[0];
            da=0;
            S.rsp=ch*4;
            P.phase=0;
        }
    }
}



/***********************************************/
/*          signal related functions           */
/***********************************************/


void GetGrid(double *a,double *b,double go,double gw,int nn,int sgn) {
    for (int i=0;i<nn && go<0;i++,go+=gw) b[i]=0;
    for (i;i<nn && go<nbuf*sl;i++,go+=gw) b[i]=sgn*a[(int)go];
    for (i;i<nn;i++) b[i]=0;
}


void InitSearch() {
    //adjustable settings
    //initialization-phase (P.phase==0)
    P.paop =.3;     //maximum amplitude of rest of transformed signal relative to the RS-point
    P.w=ww;         //starting grid width
    P.o=0;          //starting grid offset

    //step-phase (P.phase==1)
    P.ac = (int)ceil(6./pw);    //[samples] search resolution
    P.ma = 20;                  //[samples] maximum adjustment of grid offset
    P.np = n/2;                 //part of signal to be examined
    P.wcrf=1./4;  //factor for grid width adjustment according to the grid offset adjustment
    P.maxaf=6;    //maximum amount of wrong RS-points in a row


    //non-adjustable settings
    P.np2 = (int)floor(log(P.np)/log(2)+.5); //P.np = pow (2, P.np2)
    P.vo=P.ac*P.ma*2+1;         //amount of offset-shifts in the step-phase
    P.phase=0;                  //current phase
    P.wstart=P.w;               //start value of grid width
    P.ntry=0;                   //counter for searches in initialization-phase
    P.cr=0;                     //applied correction in step-phase
    P.af=0;                     //amount of wrong RS-points in a row
    P.awo=0;                    //previous amplitude of RS-point
    P.cnt=0;                    //overall loop counter
    P.perc=0;                   //correct RS-point counter
    P.mamp=0;                   //amplitude of RS-point with a low-pass filter applied
    da=0;                       //starting data point in channel

    //get the sequences of the specified channel
    GetRssWhole();
    GetRssPart();
}


void GetRssWhole() {
    double rs[n];
    S.rsp=ch*4;
    for (int z=0;z<4;z++) {
        for (int i=0;i<n;i++) rs[i]=0;
```

```
        for (i=0;i<n+no;i++) S.rss[z][i]=0;
        rs[ch*4+z]=1;
        RST(&rs[0], &S.rss[z][0], n, n2);
        for (i=0;i<n+no;i++)
            if (S.rss[z][i]<0 && i<n) S.rss[z][i]=1; else S.rss[z][i]=0;
    }
}

void GetRssPart() {
    double *tmp;
    int tmpmax,tmpmin;
    P.op=n/P.np/2;
    tmp=(double *)malloc(P.np*sizeof(double));

    for (int z=0;z<4;z++) {
        RST(&S.rss[z][P.op*P.np],tmp,P.np,P.np2);
        tmpmax=fmax(tmp,P.np);
        tmpmin=fmin(tmp,P.np);
        if (tmp[tmpmax]+tmp[tmpmin]>0) {
            rspp[z]=tmpmax;
            sgn[z]=-1;
        }
        else {
            rspp[z]=tmpmin;
            sgn[z]=1;
        }
    }
    free(tmp);
}

void AdjustGridOffset(double *x) {
    double G[n],G2[n],*YY,*PP,YYY[4];
    int i,iL,iH,YYmax,tmp=0,DPT[4];
    double YYtot,imean[4],AMP[4];

    YY=(double *)malloc(P.vo*sizeof(double));
    PP=(double *)malloc(P.vo*sizeof(double));

    //check all 4 possible RSS's in the channel
    for (int z=0;z<4;z++) {
        //variation of the grid offset
        for (i=0;i<P.vo;i++) {
            GetGrid(x,G,P.o+(double)i/(double)P.ac-P.ma+P.op*P.np*P.w,P.w,P.np,sgn[z]);
            RST(G, G2, P.np, P.np2); //get RS-points
            minmeanmm(G2,P.np,1);
            YY[i]=-fabs(G2[rspp[z]]);
            PP[i]=-fabs(G2[fmin2(G2,P.np,rspp[z])]);
        }

        //get the weighted average near the peak
        YYmax=fmin(YY,P.vo);
        iL=YYmax; iH=YYmax;
        while (YY[iL]<YY[YYmax]*.8 && iL>0) iL--;
        while (YY[iH]<YY[YYmax]*.8 && iH<P.vo-1) iH++;
        imean[z]=0;YYtot=0;if (iH-YYmax>YYmax-iL) iH=2*YYmax-iL; else iL=2*YYmax-iH;
        for (i=iL;i<=iH;i++) {
            imean[z]+=i*min(YY[i],0);
            YYtot+=min(YY[i],0);
        }
        if (YYtot!=0) imean[z]/=YYtot; else imean[z]=P.ma*P.ac;
        P.cr=imean[z]/(double)P.ac-P.ma;

        //evaluate new grid offset
        GetGrid(x,G,P.o+P.cr,P.w,n,1);
```

```
        RST(G, G2, n, n2);
        minmeanmm(G2,n,2./sqrt(n));
        DPT[z]=fmin(G2,n);
        AMP[z]=-G2[ch*4+z];
        if (DPT[z]==ch*4+z) YYY[z]=AMP[z]; else {YYY[z]=0; imean[z]=P.ma*P.ac;}
        if (YYY[z]>YYY[tmp] || z==0) {
            tmp=z;
            if (P.vo<50) for (i=0;i<P.vo;i++) { CYY[i]=YY[i]; CPP[i]=PP[i];}
            else for (i=0;i<50;i++) { CYY[i]=YY[i*P.vo/50]; CPP[i]=PP[i*P.vo/50];}
        }
    }

    //determine which RSS was present in the sequence
    free(YY);
    free(PP);
    if (YYY[0]+YYY[1]+YYY[2]+YYY[3]!=0) da=fmax(YYY,4);
    P.dpt=DPT[da];
    P.amp=AMP[da];
    S.rsp=da+ch*4;
    CIM=imean[da];
    P.cr=imean[da]/(double)P.ac-P.ma;
    P.o+=P.cr;
}


/***********************************************/
/*                functions                    */
/***********************************************/


int fmax(double *xx,int nn) {
    double maxvalue=*xx;
    int maxpos=0;
    for (int i=1;i<nn;i++)
    {
        if (*(xx+i)>maxvalue)
        {
            maxvalue=*(xx+i);
          maxpos=i;
        }
    }
    return maxpos;
}


int fmin(double *xx,int nn) {
    double minvalue=*xx;
    int minpos=0;
    for (int i=1;i<nn;i++)
    {
        if (*(xx+i)<minvalue)
        {
            minvalue=*(xx+i);
          minpos=i;
        }
    }
    return minpos;
}


int fmin2(double *xx,int nn,int zzz) {
    double minvalue=*xx++;
    int minpos=0;
    if (zzz==0) {
        minvalue=*xx;
        minpos=1;
```

```
    }
    for (int i=1;i<nn;i++,xx++) {
        if (*xx<minvalue && zzz!=i) {
            minvalue=*xx;
          minpos=i;
        }
    }
    return minpos;
}


int xcorr(double *xx, double *yy,int nn,double *zz) {
    double *zzz;
    zzz=zz;
    for (int i=0;i<nn;i++,zz++) {
        *zz=0;
        for (int j=0;j<nn;j++) *zz+=*(xx+j) * *(yy+(j+i)%nn);
    }
    return fmax(zzz,n);
}


void minmeanmm(double *xx,int nn,double mult) {
    double mn=0;
    int i;
    for (i=0;i<nn;i++) mn+=xx[i]/nn;
    for (i=0;i<nn;i++) xx[i]=(xx[i]-mn)*mult;
}

int sign (double xx) {
    if (xx>0) return 1;
    else if (xx<0) return -1;
    else return 0;
}


void BaseConv(int *y,int ny,int F2,int *x,int nx,int F1) {
    int R=0;
    for (int k=0;k<nx;k++) R=R+(int)pow(F1,k)*(*x++);
    if (R==0) for (k=0;k<ny;k++) *y++=0;
    else {
        int L=max((int)ceil(log(R+.5)/log(F2)),ny);
        for (k=0;k<L;k++) y[k]=0;
        for (k=0;k<L;k++) {
            y[L-k-1]=(int)(R/floor(pow(F2,L-k-1)));
            R=R-y[L-k-1]*(int)pow(F2,L-k-1);
        }
    }
}
```

```
void CallBackFunction(HWND handle, UINT message, WPARAM wParam,
LPARAM lParam) {
    double TmpArray[8192];
  int u, j, m;
    short *sPtr;
    double *dSProc1, *dSProc2, *dTA;
        double *dPtr,*dPtr2,*dPtrL,*dPtrR;
```

```
    double ggo,ggw,G[n+no],G2[n+no];

  DAQ.Status = DAQ_DB_HalfReady(DAQ.DeviceIn, &DAQ.HalfReady, &DAQ.Stopped);
if (DAQ.HalfReady != 1) {
      strcpy(MessageString, "Halfbuffer not ready!");
      DAQ.Status = 0;
      return;
  } else {
      strcpy(MessageString, "");
  }

  /* Grab half-buffer of data. */
  DAQ.Status = DAQ_DB_Transfer(DAQ.DeviceIn, Signal.Input, &DAQ.PtsTfr, &DAQ.Stopped);
  if (DAQ.Status == -10846 || DAQ.Status == 10846 ) {
      strcpy(MessageString, "Halfbuffer corrupted!");
      DAQ.Status = 0;
  } else {
      NIDAQErrorHandler(DAQ.Status, "DAQ_DB_Transfer", 0);
      strcpy(MessageString, "");
  }



  /* Copy signal into the sequential Proc array. */

  sPtr = Signal.Input;
  dSProc1 = &Signal.Proc[0];
  dSProc2 = &Signal.Proc[Signal.Length];
  for (u = 0; u < Signal.Length; u++) {
      *dSProc1++ = *sPtr++;
      *dSProc2++ = *sPtr++;
  }


  /* Remove polynomials content. */
  if (Signal.RemovePoly) {
    Pol.RemovePoly = FALSE;
    for (m = 0; m < 2; m++) {

      /* Construct the signal to be approximated. */
      dSProc1 = &Signal.Proc[m*Signal.Length];
      dTA = TmpArray;
      for (u = 0; u < Signal.Length; u++) *dTA++ = *dSProc1++;

      for (u = 0; u < Signal.SeqLength; u++)
        for (j = 0; j < Signal.PeakWidth+2; j++)
          TmpArray[u*Signal.PeakDistance+j] = TmpArray[u*Signal.PeakDistance+10];

      RemovePolynomialContent(TmpArray, Signal.Length);

      /* And subtract it from the real signal. */
      dSProc1 = &Signal.Proc[m*Signal.Length];
      dTA = TmpArray;
      for (u = 0; u < Signal.Length; u++) *dSProc1++ -= *dTA++;
    }
  }

  /* Copy signal to display buffer. */
  if (!Signal.CurrentlyDisplaying)
    for (u = 0; u < 2*Signal.Length; u++) Signal.Display[u] = (short)Signal.Proc[u];

  /* Update CGM index. */
  Signal.CurrentCGM++;
  if (Signal.CurrentCGM >= Signal.AverageLength) Signal.CurrentCGM = 0;
```

```
//search for a specific Rudin Shapiro Sequence
if (AGR) {
  //get the middled signal of both light receivers
  dPtr = &data[(nbuf-1)*sl];
  dPtrL= &dataL[(nbuf-1)*sl];
  dPtrR= &dataR[(nbuf-1)*sl];
  dSProc1 = &Signal.Proc[0];
  dSProc2 = &Signal.Proc[Signal.Length];
  for (u = 0; u < sl; u++) {
    *dPtrL++ = *dSProc1;
    *dPtrR++ = *dSProc2;
    *dPtr++ = (*dSProc1++++*dSProc2++)/2.;
  }

  //step everytime to next sequence until the end of the buffer has been reached
  while (P.o+P.w*(n+no)+P.ma<=sl*nbuf) {

    //THE algorithm to find and keep on track with the sequence
    /*******************************/
    /***/GetRss(data,&ggo,&ggw);/***/
    /*******************************/

    //apply the new grid offset and width
    if (P.mss && P.phase==1) {
      GetGrid(&dataL[0],G,ggo,ggw,n,1);
      RST(G, G2, n, n2);
      minmeanmm(G2,n,2./sqrt(n));
      Signal.CGMAverage[0]=-G2[S.rsp];
      GetGrid(&dataR[0],G,ggo,ggw,n,1);
      RST(G, G2, n, n2);
      minmeanmm(G2,n,2./sqrt(n));
      Signal.CGMAverage[1]=-G2[S.rsp];
      if (P.phase==1 && da!=0) {
        cmd[Ccmd]=da-1;
        Ccmd+=1;
        if (Ccmd==12) {
          Ccmd=0;
          ncf++;
          for (int zzz=0;zzz<12;zzz++) sprintf(&Scmd[zzz],"%d",cmd[zzz]);
          BaseConv(Di,19,2,cmd,12,3);
          for ( zzz=0;zzz<19;zzz++) sprintf(&DI[zzz],"%d",Di[zzz]);
        }
      } else Ccmd=0;
      P.perc++;
    }
    else {
      Signal.CGMAverage[0]=0;
      Signal.CGMAverage[1]=0;
    }
    P.o+=P.w*(n+no);
    P.cnt++;
    P.mamp=.98*P.mamp+.02*P.amp;
  }

  //shift all data in the buffer so the next data can be added at the end
  P.o-=sl;
  dPtr = &data[0];
  dPtr2 = &data[sl];
  for (u = 0; u < (nbuf-1)*sl; u++) *dPtr++ = *dPtr2++;
  dPtr = &dataL[0];
  dPtr2 = &dataL[sl];
  for (u = 0; u < (nbuf-1)*sl; u++) *dPtr++ = *dPtr2++;
```

```
      dPtr = &dataR[0];
      dPtr2 = &dataR[sl];
      for (u = 0; u < (nbuf-1)*sl; u++) *dPtr++ = *dPtr2++;
    }
////////////controller////////////

    if (Signal.CGMAverage[0]!=0 || Signal.CGMAverage[1]!=0)
    Motor.eDisplay = (Signal.CGMAverage[0]-Signal.CGMAverage[1])
                   /  sqrt(Signal.CGMAverage[0]*Signal.CGMAverage[0]
                        +Signal.CGMAverage[1]*Signal.CGMAverage[1])*1024
                   + (double)Signal.Offset;
    else Motor.eDisplay=0;

    Motor.e = Motor.eDisplay;

    Motor.Sum += Motor.e;
    if (Motor.Ki==0) Motor.Sum=0;
    Motor.Kfsign=sign(Motor.e);

    //low-pass filter used for noise band
    Motor.fs=DAQ.SampRate/Signal.Length/2; //=146,5Hz
    Motor.tau=.1;
    Motor.filt=(Motor.filtrec-Motor.e)*pow(EE,-1/Motor.fs/Motor.tau)+Motor.e;

    //determine motor input u
    if (P.phase==1) {
      Motor.u = Motor.e * Motor.Kp
              + Motor.Ki/100 * Motor.Sum
              + Motor.Kfsign*Motor.Kf/5.;
      if (fabs(Motor.filt)<80) Motor.cnt++;
      else {
        Motor.u+=Motor.Kd * (Motor.e-Motor.erec);
        Motor.cnt=0;
      }
      if (fabs(Signal.CGMAverage[0])<40 && fabs(Signal.CGMAverage[1])<40 || Motor.cnt>40){
        Motor.u = 0;
        Motor.Sum = 0;
      }
    } else Motor.u=0;

    t++;
    if (Motor.u < -1) Motor.u = -1;
    if (Motor.u >  1) Motor.u =  1;

    r[0]=Motor.e * Motor.Kp;
    r[1]=Motor.Ki/100. * Motor.Sum;
    r[2]=Motor.Kd * (Motor.e-Motor.erec);
    r[3]=Motor.Kfsign*Motor.Kf/5.;

    if (MotorData.CountSeqs > 0) {
      MotorData.Data[MotorData.NumOfVariables*(MotorData.NumOfDataSeqs-MotorData.CountSeqs)] = (float)Motor.e;
      MotorData.Data[MotorData.NumOfVariables*(MotorData.NumOfDataSeqs-MotorData.CountSeqs)+1] = (float)Motor.u;
      MotorData.Data[MotorData.NumOfVariables*(MotorData.NumOfDataSeqs-MotorData.CountSeqs)+2] = (float)r[0];
      MotorData.Data[MotorData.NumOfVariables*(MotorData.NumOfDataSeqs-MotorData.CountSeqs)+3] = (float)r[1];
      MotorData.Data[MotorData.NumOfVariables*(MotorData.NumOfDataSeqs-MotorData.CountSeqs)+4] = (float)r[2];
      MotorData.Data[MotorData.NumOfVariables*(MotorData.NumOfDataSeqs-MotorData.CountSeqs)+5] = (float)r[3];
      MotorData.Data[MotorData.NumOfVariables*(MotorData.NumOfDataSeqs-MotorData.CountSeqs)+6] = (float)Motor.filt;
      MotorData.Data[MotorData.NumOfVariables*(MotorData.NumOfDataSeqs-MotorData.CountSeqs)+7] = (float)Motor.cnt;
      MotorData.CountSeqs--;
      if (MotorData.CountSeqs == 0) MotorData.SavePending = TRUE;
    }
    Motor.erec = Motor.e;
    Motor.urec = Motor.u;
    Motor.filtrec = Motor.filt;

    for (j = 0; j < Signal.Length; j++) Signal.Output[2*j+1] =
```

```
      (int)floor(-Motor.u*2047);// + int(floor(sin(j/4096*Pi*2)*2047/5*1));
    if (!uoet) DAQ.Status = WFM_DB_Transfer(DAQ.DeviceOut, DAQ.NumChansOut,
      DAQ.ChanVectOut, Signal.Output, 2 * Signal.Length);
}
```