AUTOMATIC EDVS EVENT RATE CONTROL

eingereichtes
Projektpraktikum
von

Ming Xue, Kaspar Hitzelberger

Lehrstuhl für
STEUERUNGS- UND REGELUNGSTECHNIK
Technische Universität München
Univ.-Prof. Dr.-Ing../Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Betreuer/-in:	Mohsen Firouzi
Beginn:	01.05.2013
Abgabe:	18.06.2013

Abstract

The project “eDVS event rate control” has been concerned with a first approach for an automatic event rate control for the eDVS sensor. In this project, we have studied the eDVS optics and chip setup, then try to develop 2 controllers according to the relationships of bias setting and event rate, later we set up different test environment to evaluate our controllers in different situations.

Contents

1 Introduction ---- 4

2 eDVS optics and chip setup ---- 5

2.1 eDVS sensor ---- 5
2.1.1 Description ---- 5
2.1.2 Functionality ---- 6
2.2 eDVS board ---- 6
2.2.1 Description ---- 6
2.2.2 Connection and communication with computer ---- 7

3 Controlling the event rate ---- 9

3.1 Measuring of the event rate ---- 9
3.1.1 Buffer properties of serial interface ---- 9
3.1.2 Method to measure events ---- 9
3.2 Bias current generators ---- 10
3.2.1 Values and correlations ---- 10
3.2.2 Study of bias values ---- 10
3.3 Displaying of the event rate ---- 14
3.3.1 Format of events ---- 14
3.3.2 Display events ---- 14
3.4 Designed controllers ---- 16
3.4.1 Modeled relationship between bias values and event rate ---- 16
3.4.2 P-controller ---- 16
3.4.3 Fuzzy PID Controller ---- 17

4 Evaluation ---- 21

4.1 Evaluation environment ---- 21
4.2 Evaluation method ---- 21
4.3 Evaluation of controllers ---- 21
4.3.1 P-controller ---- 21
4.3.2 Fuzzy PID Controller ---- 23
4.4 Conclusion ---- 26

5 Summary ---- 27

5.1 Main results ---- 27
5.2 Problems ---- 27
5.3 Future Work ---- 27

List of Figures ---- 30

Bibliography ----------31

1 Introduction

The student project “eDVS event rate control” has been concerned with a first approach for an automatic event rate control for the eDVS sensor. At the beginning, the tasks and target will shortly be described.

At first it’s important to understand the functionality and the setup of the eDVS sensor e.g. modifying biases. A short brief about the DVS sensor and the eDVS board the sensor is located on will be given. Afterwards a controller on PC will be designed which keeps the emitted event-rate roughly constant by automatically modifying bias settings based on sensed events. Afterwards the evaluation of the designed controllers will be done.

2 eDVS optics and chip setup

The dynamic vision sensor (DVS) used in this project is an address-event silicon retina that responds to temporal contrast (Figure 1.1). Each output spike address represents a quantized change of log intensity at a particular pixel since the last event from that pixel. The address includes a sign bit to distinguish positive from negative changes. All 128x128 pixels operate asynchronously and signal illumination changes within a few microseconds after occurrence. No frames of "complete" images exist in the system, but instead only individual events that signal changes at a particular spatial position denoted by a particular pixel's address [2].

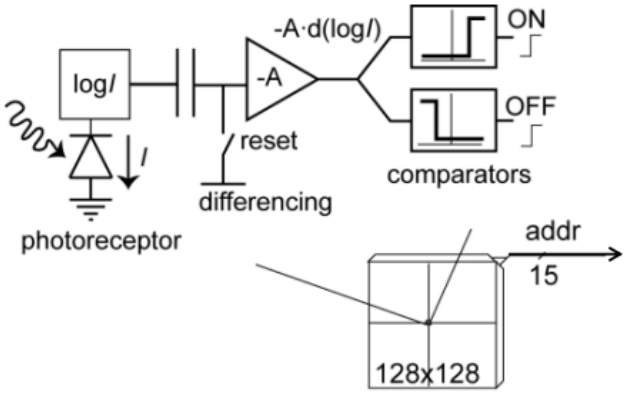

Figure 2.1: dynamic vision sensor (DVS)

The standard DVS contains an on-board digital logic chip with high-speed USB 2.0 interface that takes addresses and delivers time-stamped address events with a resolution of 1us to a host PC for off-line processing.

2.1 eDVS sensor

2.1.1 Description

Conventional vision sensors see the world as a series of frames. Successive frames contain enormous amounts of redundant information, wasting memory access, RAM, disk space, energy, computational power and time. In addition, each frame imposes the same exposure time on every pixel, making it difficult to deal with scenes containing very dark and very bright regions.

The Dynamic Vision Sensor (DVS) solves these problems by using patented technology that works like your own retina. Instead of wastefully sending entire images at fixed frame rates, only the local pixel-level changes caused by movement in a scene are transmitted – at the time they occur. The result is a stream of events at microsecond time resolution, equivalent to or better than conventional high-speed vision sensors running at thousands of frames per second. Power, data storage and computational requirements are also drastically reduced, and sensor dynamic range is increased by orders of magnitude

due to the local processing. Application Areas are for example 1) Motion analysis, e.g. human or animal, Fast Robotics: mobile, Factory automation, Microscopy and so on.

2.1.2 Functionality

The DVS functionality is achieved by having pixels that respond with precisely-timed events to temporal contrast. Movement of the scene or of an object with constant reflectance and illumination causes relative intensity change; thus the pixels are intrinsically invariant to scene illumination and directly encode scene reflectance change.

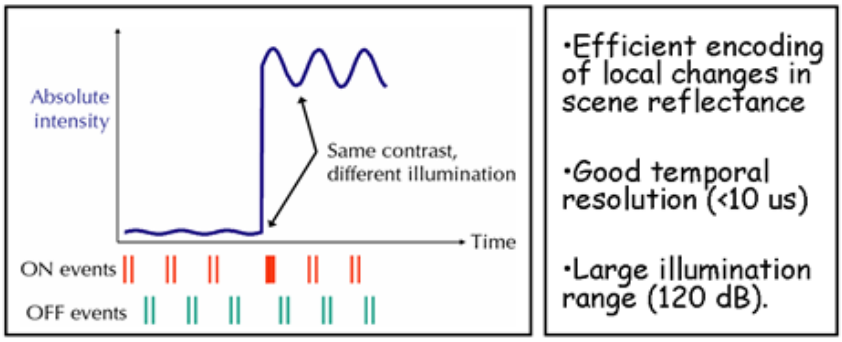

Figure 2.2: Functionality of eDVS

The events are output asynchronously and nearly instantaneously on an Address-Event bus, so they have much higher timing precision than the frame rate of a frame-based imager. Measurements show that people often achieve a timing precision of 1 us and a latency of 15 us with bright illumination. The low latency is very useful for robotic systems, such as the pencil balancing robot.

Because the pixels locally respond to relative change of intensity, the device has a large intra-scene dynamic range. This wide dynamic range is demonstrated by the Edmund gray scale chart, which is differentially illuminated by a ratio of 135:1 – a 42dB illumination ratio.

2.2 eDVS board

2.2.1 Description

The eDVS sensor used in the project is a small embedded DVS system (Figure 1.3) composed of a DVS chip directly connected to a 64MHz 32bit microcontroller (NXP LPC2106/01) with 256kbyte on-board program flash memory and 64kbyte on-board RAM. This processer initializes the DVS chip and captures events for immediate processing, following a simple handshaking protocol: Each occurring event from the DVS is transmitted as a 15-bit address (7 bits x-position, 7 bits y-position and 1 bit polarity), together with a request signal.
The LPC2106 reads the address and acknowledges reception of that event, after which the DVS removes the request signal. The LPC2106/01 microcontroller offers several communication ports and the UART port facilitates connections to a PC for setup, debug output, and reprogramming.

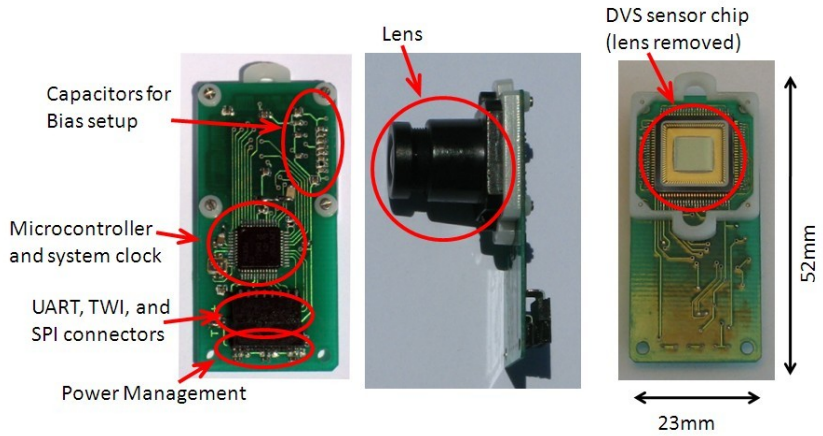

Figure 2.3: Embedded-DVS board showing functional elements

After programming, the eDVS board can run applications that process the sensor events in real time. Sensor maintenance (such as adjusting bias values or acknowledging events) causes only marginal overhead. However, the microcontroller can adjust sensor settings during operation, e.g. to limit the event rate by dynamic control of the pixel contrast threshold.

2.2.2 Connection and communication with computer

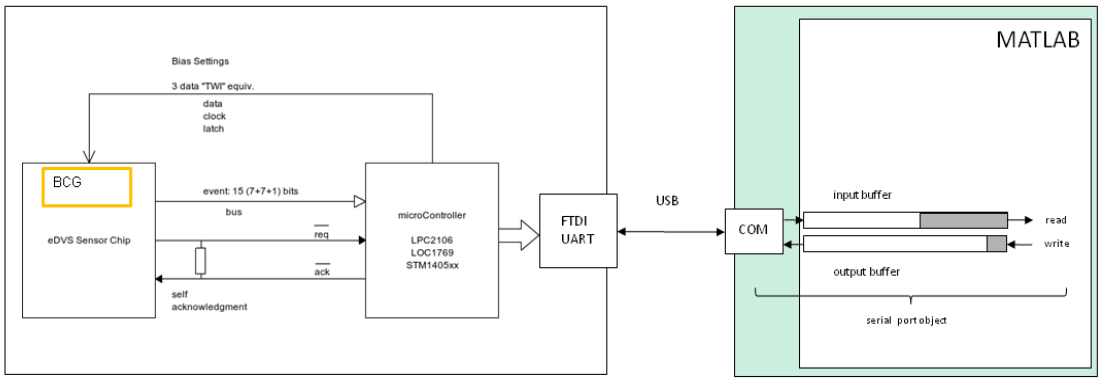

Figure 2.4: connection MATLAB® and eDVS

The eDVS board is linked to the Computer over an USB cord. MATLAB® provides a serial port interface for 'Data and File Management'. The connection is realized over a COM interface which is used by a serial port object. The serial port object supports functions and properties that allow you among others to configure serial port communication, write and read data, use events and callbacks.

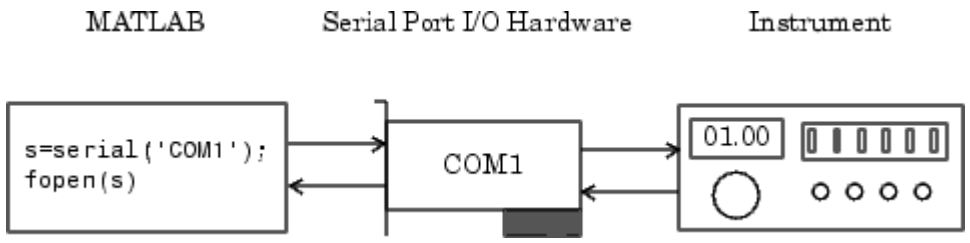

Figure 2.5: connection between the serial port object and the device (www.mathworks.com)

The properties of the serial port object have been set as in the provided MATLAB® scripts from the LSR-wiki[1].

On the serial object the communication with the sensor is provided over the command:

```
fwrite(s, 'command','unit8')
```

The following commands are of interest to communicate with the microcontroller on the board and thereby indirectly with the sensor.

command	description
r	Set the sensor to his default settings
!EX	Specify the data format, here the default setting X=0 is used: 0yyyyyyy pxxxxxxx binary 16 bits = 2 bytes = 1 event) p stands for the polarity bit, denoting polarity of event (ON or OFF event)
!E+	enable sending events
!E-	stop sending events
?BX	read out value of bias register [X], response: ?BX=xyz
!BX=xyz	set value of bias register [X] to xyz mentioned bias current bits: 22 bit result in a range from 0… 4194304 for possible bias values [1]
!BF	write the bias values set to the sensor remark: !BX=xyz doesn't contains this operation

[1] https://wiki.lsr.ei.tum.de/nst/programming/edvsgettingstarted

3 Controlling the event rate

3.1 Measuring of the event rate

The task to control the event rate defines it as controlled variable. Therefore we continued to discuss a method to measure the rate.

3.1.1 Buffer properties of serial interface

To measure the event rate it's necessary to have a closer look on the properties of the MATLAB® serial interface. The events (binary data) from the eDVS board (instrument) are collected in the input buffer of the serial connection object. We set the size of the input buffer to 1024 bytes (1MByte). By using the `read(s,n)` command, n bytes are read out of the input buffer of s. Amount n of bytes inside the buffer can be easily requested.

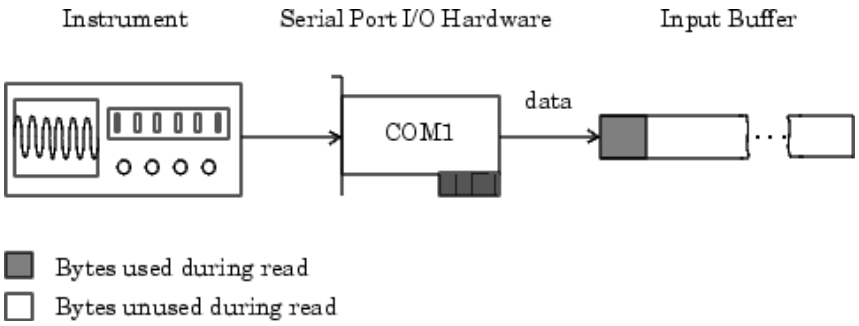

Figure 3.1: input buffer of serial port object (www.mathworks.com)

3.1.2 Method to measure events

To measure the occurring events we discussed different options and decided to following method:

First the input buffer is flushed by reading out all bytes in it, if there are any from possible responses of the '!BX=xyz' commands. They're not of interest anymore. Secondly, the ‚!E+' command enables sending of events from the sensor to the input buffer. The program pauses for 0.1 seconds to collect the events occurring in this time period. After the pause, ‚!E-' stops the sensor to send events. At last, the events occurred can be read out, counted, and characterized. The input buffer is empty afterwards.

```
readOutAllBytes(s)
!E+
pause(0.1)
```

```
!E-
readOutAllBytes(s)
```

There have been two reasons which motivated this method: On the one hand, the later following control operation has to be done in the same iteration step over the total running time and before the next measurement of the rate. On the other hand it was not possible to measure the time of the sending period within the needed accuracy for a reasonable calculation of a rate/per second with the provided MATLAB® commands.

A disadvantage surely is the not continuous measurement of the event rate at any time. Some short occurring events in a time period the sensor is not enabled to send events may be lost.

3.2 Bias current generators

3.2.1 Values and correlations

To influence the thresholds of the sensor the programmable bias current generator[3], which is configuring the sensitivity of the sensor, is regarded. For the control task the bias values can be seen as the actuating variables. There are three bias values which are mainly responsible for the thresholds of the sensor.

register	name	default value
4	*diffOff*	60
8	*diffON*	567391
9	*diff*	19187

They can be set, as mentioned under 2.2.2 by the '!BX=xyz' command. For example to set diffON to 450 000 the command is:

```
'B8=450000'
```

The thresholds for ON and OFF events show the following correlations:

- Sliding *diffOn* to left towards *diff* decreases ON event threshold.
- Sliding *diffOFF* to right towards *diff* decreases OFF event threshold.

3.2.2 Study of bias values

3.2.2.1 Description

In order to design our controller, we have to get further information about the 3 bias values. Three groups of tests had been conducted to study the relationship between bias value and events rate.

3.2.2.2 Relationships

We have set up the test environment the same as descript in chapter *Evaluation*, for the controller, we have found that, diffON has big influence for events rate, so the tests mainly focus on diffOFF and diff parameters. 3 groups tests are divided as follows, for reference are the default bias values given as well.

diffOFF=60, diff=19187, diffON=567391.

1) by changing only diffOFF from 0 to 40k (about 2* diff default)

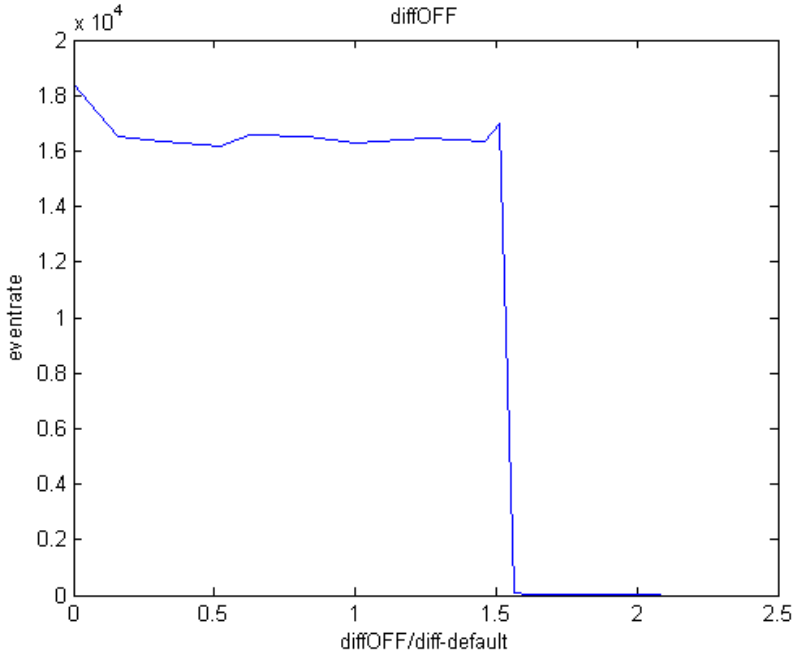

Figure 3.2: properties of diffOFF

In this case, we got the properties of diffOFF shown as Figure 3.2. From the picture, we can see that the average event rate seems like a step signal and threshold appears in about 30k (about 1.5*diff default), for bigger diffOFF, the event rate is almost 0.

2) by changing only diff from 10k to 200k (about 10*diff default)

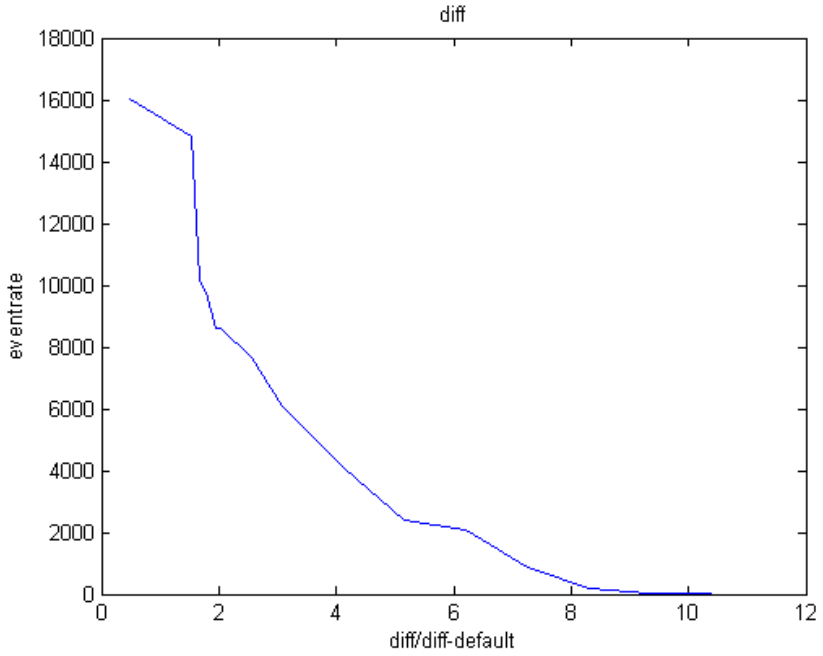

Figure 3.3: properties of diff

In this case, we got the properties of diff shown as Figure 3.3. For bigger diff, the event rate is almost 0.

3) by fixing diff in a big value (about 100k), change diffOFF from 1k to 230k(about 12*diff default)

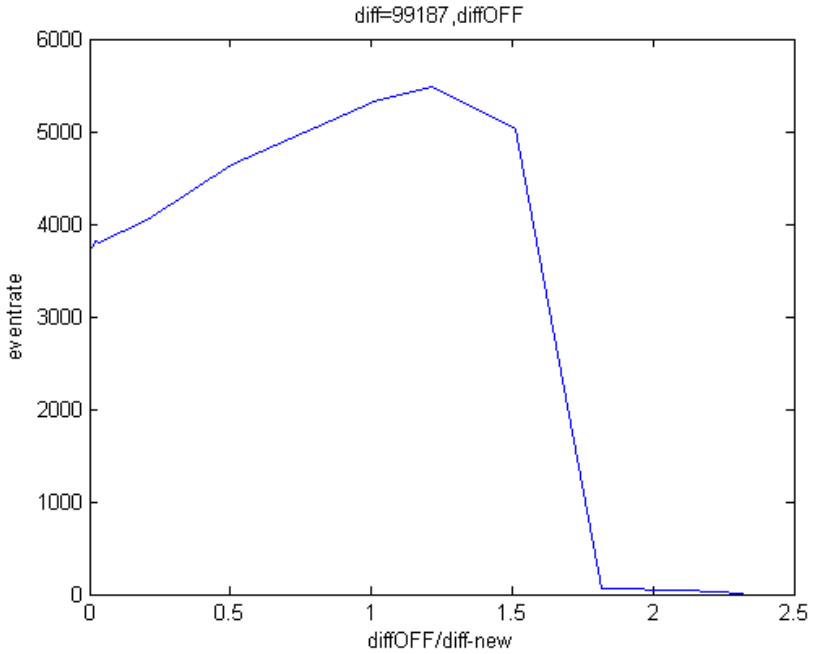

Figure 3.4: diff=99187, properties of diffOFF

In this case, we got the properties of diffOFF shown as Figure 3.4. From the picture, we can see that the plot is a little different from case 1), there is still a threshold which appears in about 180k (about 1.8*diff new), for bigger diffOFF, the event rate is almost 0.

3.2.2.3 Conclusiones

- According to case 1 and 3, we can see that diffOFF has only little influence to the events rate in the range from 0 to diff (which is practically used by eDVS).
- According to case 2, we can see that diff value has some influence for events rate, but later we find out that it's hard by changing diff value, because there is a long time delay (1s to 7s) for the events rate to reach a stable value (Figure 3.5), which makes it difficult to realize a real-time controller.

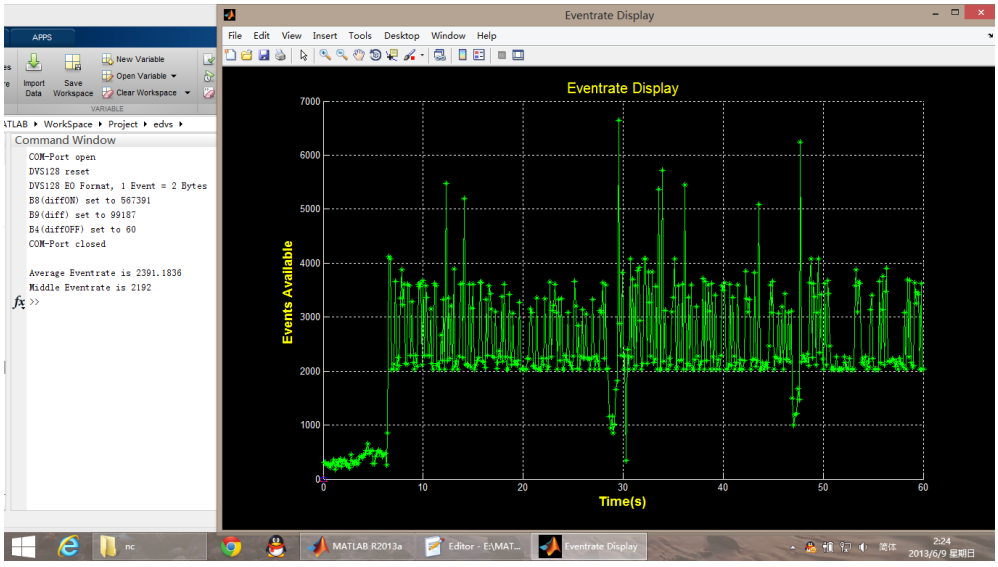

Figure 3.5: properties of diff

- For very big diff and diffOFF values (e.g. 500k and 430k), event rate is almost 0. (Figure 3.6)

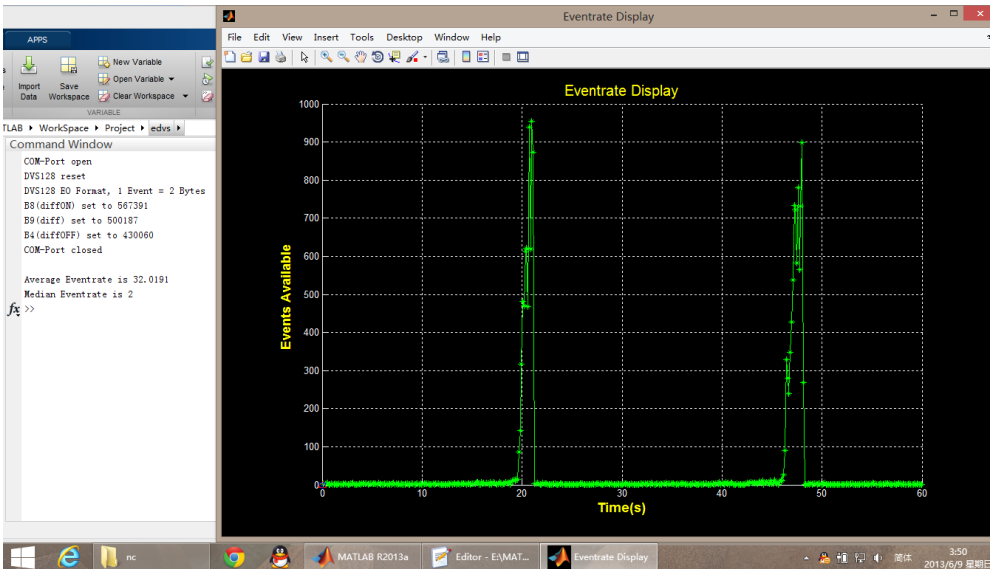

Figure 3.6: diff and diffOFF set to big values

Taking into account the difficulties of MIMO systems and the conclusions above, we decided to develop the controller by changing only diffON value, and later tests show that even by controlling only a bias value, we still get some good results and see the effect on the rate by changing diffON.

3.3 Displaying of the event rate

3.3.1 Format of events

The events which are read out are converted to a matrix with three columns and the appearance:

$$Event_{Vector} = \begin{pmatrix} x_1 & y_1 & p_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & p_n \end{pmatrix} \tag{3.1}$$

The x_n or y_n provides the pixel position. Important for the characteristic of the event is the polarity bit p of an event. This is saved in the 3rd column as p_n. We assumed $p = 1$ corresponding to an ON event and $p = 0$ to an OFF event. To evaluate the number of ON-events, the 3rd column is summed up. The difference of the size n of the $Event_{Vector}$ and the ON vents delivers the number of OFF events.

3.3.2 Display events

In a display window the events are plotted during each measurement.

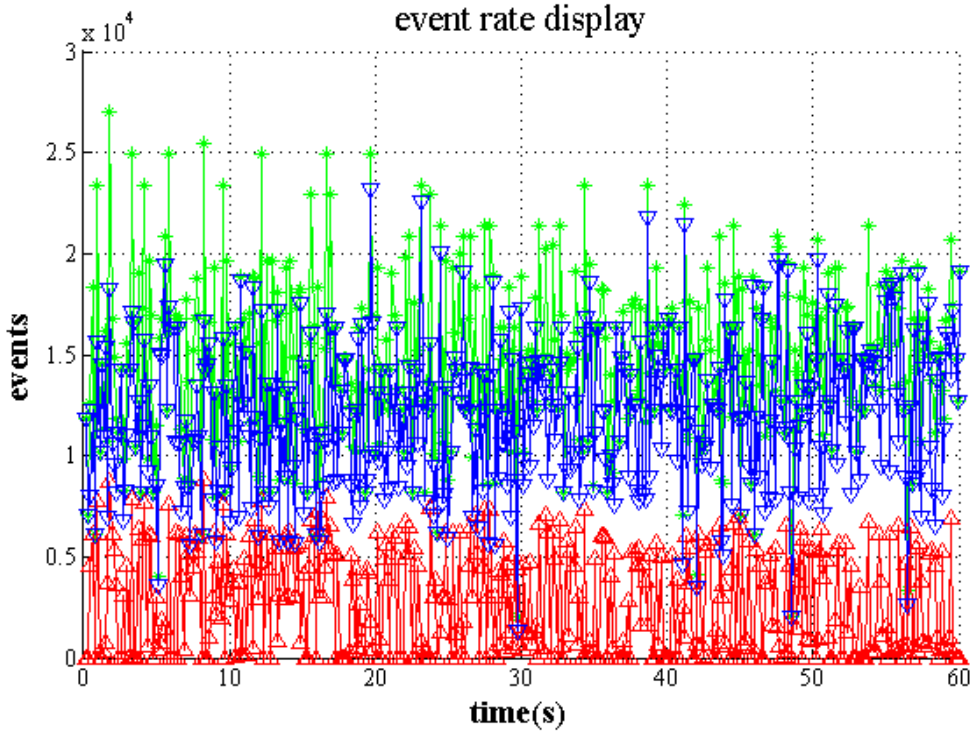

Figure 3.7: event rate display (ON = red, OFF = blue, events_all = green)

The ON and OFF events and also the sum of events can be plotted separately. This allows a more detailed view on the influence of changes to the three values.

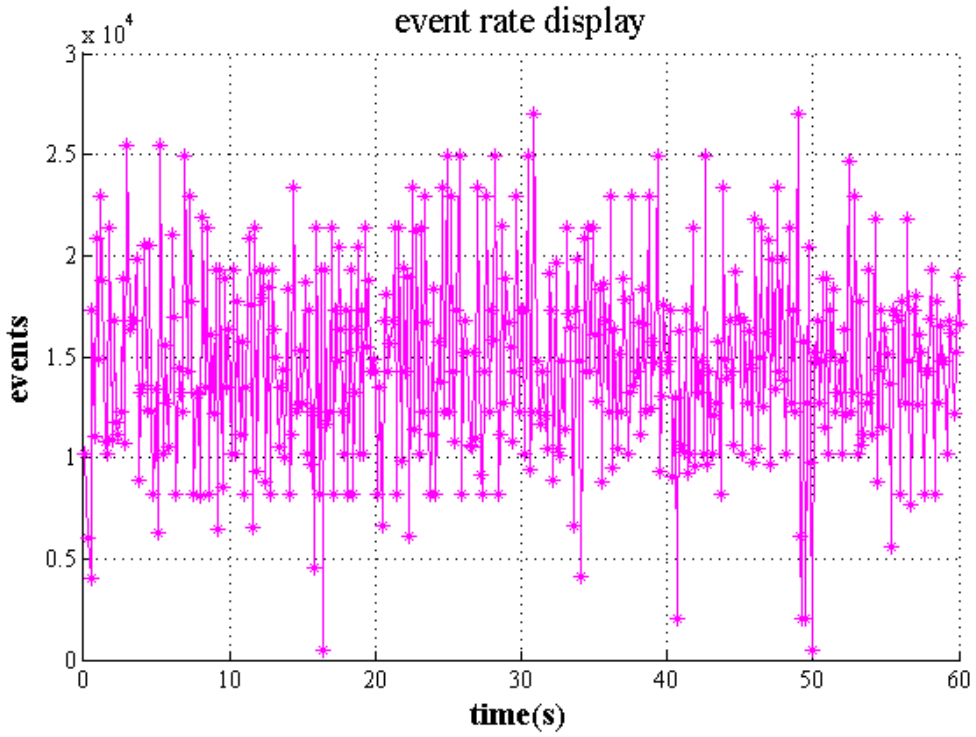

Figure 3.8: event rate display (sum of events = magenta)

3.4 Designed controllers

One of the main tasks has been the design of a controller on PC to keep the emitted event rate roughly constant by automatically modifying the bias settings based on the sensed events. The most challenging thing is the not mathematical described behavior between the values of the bias registers and the sensitivity. We subsequently made different approaches to find and tune a controller which can fulfill the aim of a roughly constant event rate.

3.4.1 Modeled relationship between bias values and event rate

To model the relationship between a set bias value and his influence to the event rate we defined a parameter k_{ON}. The relationship can now be modeled as

$$eventrate = f(k_{ON} \cdot diff) \tag{3.2}$$

This is a very simple linear model and not very reasonable, but no good description in the user guide could be found describing this interdependency more detailed.

3.4.2 P-controller

The 1st approach was a single feedback loop with a P-controller. To get a SISO system we decided to keep the *diff* and *diffOFF* values constant at first and only change the *diffON* value. Following P-controller is set up:

$$C = k_P \cdot (eventrate - rate_{des}) = k_P \cdot error \tag{3.3}$$

After the calculation of the error between the desired eventrate (*rate_des*) and the measured eventrate (*eventrate*) and the weighting by the linear parameter k_P it's necessary to calculate a value, which will be set as new diffON value to the sensor.

$$diffON = diffON_{last} - k_{ON} \cdot C \tag{3.4}$$

$$diffON = diffON_{last} - k_{ON} \cdot k_P \cdot error \tag{3.5}$$

The equation (3.5) shows, that $k_{ON} \cdot k_P$ are simply multiplicated. This reduces the degrees of freedom in this very simple case, because it's only one parameter k_{ON} or k_P ·that has to be modified for first studies of the behavior of the controlled sensor.

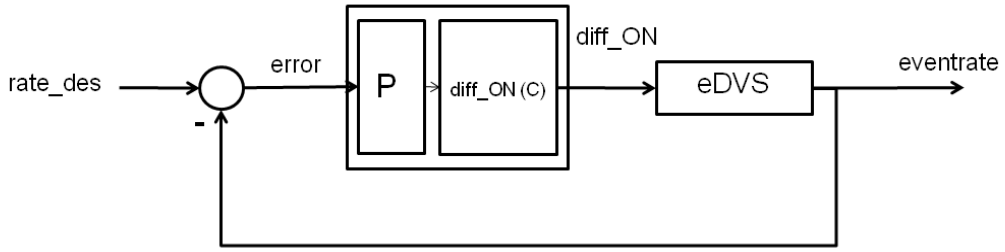

Figure 3.9: feedback loop with P-controller

Due to the chosen method of measuring the data and the seesaw changes of the rate, it was not possible to extend the P-Controller to a PID-controller as it must be to eliminate the steady state error and increase the performance by an I- and D-Block.

3.4.3 Fuzzy PID Controller

3.4.3.1 Description

The 2nd approach is a single feedback loop with a Fuzzy-PID controller. After the development of fuzzy logic, Fuzzy-PID controllers are used in a variety of processes instead of linear PID controllers, due to their advantages imposed by the non-linear behavior. They are converting the error between the measured or controlled variable and the reference variable, into a command, which is applied to the actuator of a process. Here we still keep the diff and diffOFF values constant and only change the diffON value. The control system is set up as Figure 3.10 below:

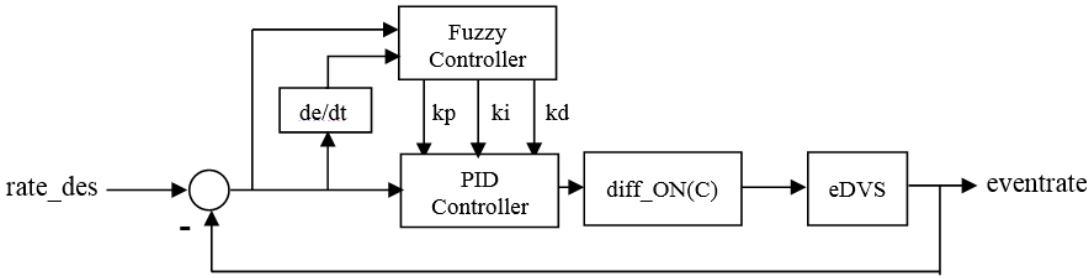

Figure 3.10: feedback loop with Fuzzy-PID controller

In the 2 inputs, 3 outputs Fuzzy-PID controller, we have

$$e = (rate_{des} - eventrate) * Ge \quad (3.6)$$

$$ec = (e_{last} - eventrate) * Gec \quad (3.7)$$

And the outputs of fuzzy controller are kp, ki, kd, according to our fuzzy rules PID controller we have

$$Kp = kp0 + kp * Gp \quad (3.8)$$

$$ki = ki0 + ki * Gi \quad (3.9)$$

$$Kd = kd0 + kd * Gd \tag{3.10}$$

Here Ge, Gec and Gp, Gi, Gd are scaling factors, and kp0, ki0, kd0 are initial parameters, we will set the values later.

Similar to P-controller, we have

$$diffON = diffON_{last} + k_{ON} \cdot U \tag{3.11}$$

And U is output of PID controller.

$$U = Kp * x(1) + Ki \cdot x(2) + Kd * x(3) \tag{3.12}$$

$$x(1) = e \tag{3.13}$$

$$x(2) = x(2)_{last} + e \tag{3.14}$$

$$x(3) = ec \tag{3.15}$$

3.4.3.2 Implementation in Matlab

A fuzzy inference system (FIS) maps in Matlab given inputs to outputs using fuzzy logic[4]. The FIS inputs are the signals of error (e) and change of error (ec). The FIS output is the control action inferred from the fuzzy rules. Fuzzy Logic Toolbox™ provides commands and GUI tools to design a FIS for a desired control surface. By configuring the FIS and selecting scaling factors, we obtain a linear fuzzy PID control.

1) First, configure the fuzzy inference system so that it produces a linear control surface from inputs e and ec to output Kp, Ki, Kd. The FIS settings are summarized below:

```
a=newfis('fuzzypid');
```

2) Define input e, so is ec:

```
a=addvar(a,'input','e',[-3,3]);                    %Parameter e
a=addmf(a,'input',1,'NB','zmf',[-3,-1]);
a=addmf(a,'input',1,'NM','trimf',[-3,-2,0]);
a=addmf(a,'input',1,'NS','trimf',[-3,-1,1]);
a=addmf(a,'input',1,'Z','trimf',[-2,0,2]);
a=addmf(a,'input',1,'PS','trimf',[-1,1,3]);
a=addmf(a,'input',1,'PM','trimf',[0,2,3]);
a=addmf(a,'input',1,'PB','smf',[1,3]);
```

3) Define output kp, so are ki and kd:

```
a=addvar(a,'output','kp',[-0.3,0.3]);              %Parameter kp
a=addmf(a,'output',1,'NB','zmf',[-0.3,-0.1]);
```

```
a=addmf(a,'output',1,'NM','trimf',[-0.3,-0.2,0]);
a=addmf(a,'output',1,'NS','trimf',[-0.3,-0.1,0.1]);
a=addmf(a,'output',1,'Z','trimf',[-0.2,0,0.2]);
a=addmf(a,'output',1,'PS','trimf',[-0.1,0.1,0.3]);
a=addmf(a,'output',1,'PM','trimf',[0,0.2,0.3]);
a=addmf(a,'output',1,'PB','smf',[0.1,0.3]);
```

4) Define the rules:

There are 7*7 = 49 rules in total, and the first 7 are listed below.

- If (e is NB) and (ec is NB) then (kp is PB)(ki is NB)(kd is PS)
- If (e is NB) and (ec is NM) then (kp is PB)(ki is NB)(kd is NS)
- If (e is NB) and (ec is NS) then (kp is PM)(ki is NM)(kd is NB)
- If (e is NB) and (ec is Z) then (kp is PM)(ki is NM)(kd is NB)
- If (e is NB) and (ec is PS) then (kp is PS)(ki is NS)(kd is NB)
- If (e is NB) and (ec is PM) then (kp is Z)(ki is Z)(kd is NM)
- If (e is NB) and (ec is PB) then (kp is Z)(ki is Z)(kd is PS)

```
rulelist=[1 1 7 1 5 1 1;
          1 2 7 1 3 1 1;
          1 3 6 2 1 1 1;
          1 4 6 2 1 1 1;
          1 5 5 3 1 1 1;
          1 6 4 4 2 1 1;
          1 7 4 4 5 1 1;
        …
a=addrule(a,rulelist);
```

k_p ec / e	NB	NM	NS	ZO	PS	PM	PB
NB	PB	PB	PM	PM	PS	ZO	ZO
NM	PB	PB	PM	PS	PS	ZO	NS
NS	PM	PM	PM	PS	ZO	NS	NS
ZO	PM	PM	PS	ZO	NS	NM	NM
PS	PS	PS	ZO	NS	NS	NM	NM
PM	PS	ZO	NS	NM	NM	NM	NB
PB	ZO	ZO	NM	NM	NM	NB	NB

Figure 3.11: fuzzy rules table for kp

Figure 3.11 shows the whole fuzzy rules table for kp, For more rules of parameters, please refer to the fpid.m program.

5) Finally, we determine scaling factors Ge, Gec, Gp, Gi, Gd from the Kp, Ki, Kd gains. Since the input range of e is [-3,3], ec is [-3,3], the output range of kp is [-0.3,0.3], ki is [-0.06,0.06], kd is [-3,3], the variables are related as:

```
if abs(x(1))>1200 x(1)=sign(x(1))*1200; end
e = x(1) / 400;
if abs(x(3))>3900 x(3)=sign(x(2))*3900; end
ec = x(3) / 1300;

Kp =kp0 + kp *10;
Ki =ki0 + ki *1;
Kd =kd0 + kd *0.3;
```

The FIS chart is displayed below.

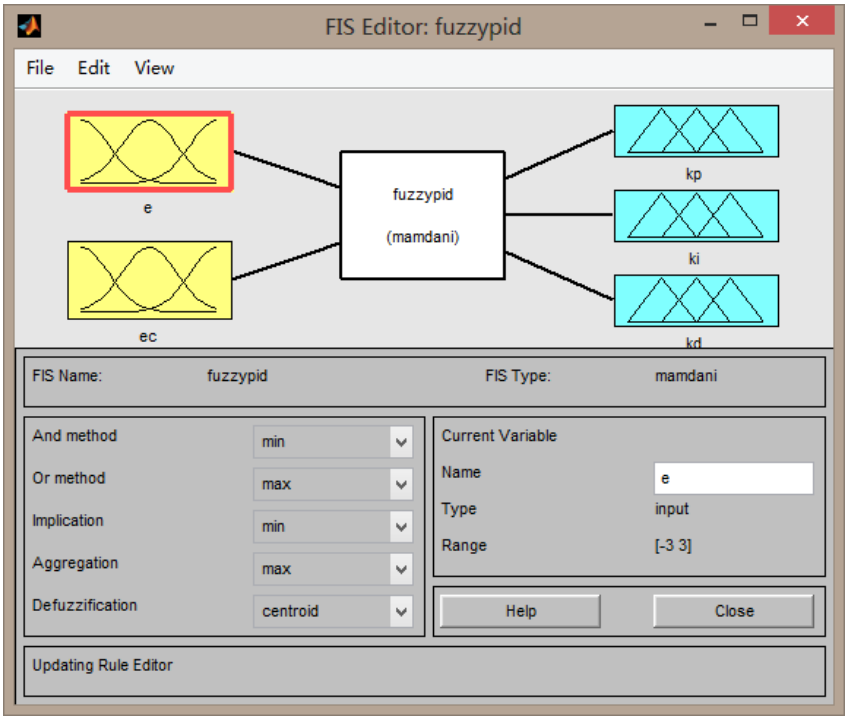

Figure 3.12: FIS chart

4 Evaluation

4.1 Evaluation environment

After the design steps of the controllers, we build up a test setting to have a closer look and evaluation of the chosen parameters. As the sensor doesn't sends the required events (e.g. if no movement and minimal reflection rates of the environment) if he is steady without event sources in the environment, we searched for an easy available stimulus. The sensor in this scenario can be steady, but events are created and were perceived by the sensor. As stimulus we have chosen a standard LED monitor. With a light background (e.g. the opened MATLAB® window) on the screen, the monitor is creating a certain event rate.

4.2 Evaluation method

In order to evaluate the performance of the 2 designed controllers in different situations, we use 3 parameters as follows:

- mean(): average value of events rate
- median(): median value in the events rate
- std/rate_des: Standard Deviation divided by rate_des

The discontinuity of the measured event rate motivated a comparison of the desired rate_des to the average and the median of the events recorded at all timestamps in the recording time. This allows a first conclusion, if over a time the rate is coming to a desired average value. The std/rate_des can be used to quantify the fluctuation of events rate.

4.3 Evaluation of controllers

In this part we will test our controllers separately in the same setting environment to see their dynamic performance.

4.3.1 P-controller

The P-controller was the first tested controller and lead to the following evaluation. The environment without any active control is comparable to Figure 3.8. With an active control by the P-controller over 60 seconds, the event rate could be influenced and the

average and median values are indicators for the influence on the rate. Also the variace is calculated to show the size of "overshoots" over and below the desired rate.

As time period 60 seconds are chosen.

- control status = false, 60s

rate_des	*diffON*	*diff*	*diffOFF*	mean()	median()
---	567391	19187	60	14674	14427

- control status = true, 1 rate_des, 60s

rate_des	*diffON*	*diff*	*diffOFF*	mean()	median()	standard variation()
5000	2263831	19187	60	5397	5040	1900
6000	2144021	19187	60	6403	6136	1860
8000	2263831	19187	60	8427	8172	3520
9000	2125531	19187	60	9366	9695	2340

The corresponding plots can be found in the appendix.

- control status = true, changing rate_des after 30s, 60s: A significant change in *diffON* value and event rate can be seen if after 30s the rate_des is reduced from 8000 to 5000. A plot is existing in the appendix on CD.

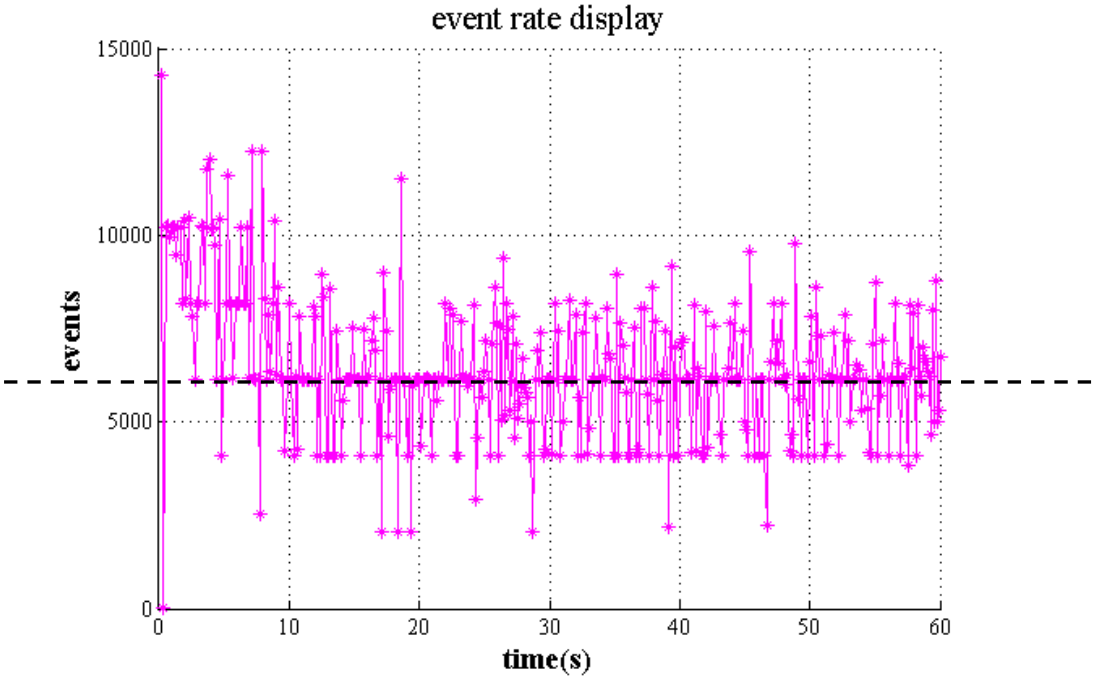

Figure 4.1: event rate for P-controlled sensor (rate_des = 6000)

The setting of higher rates than 10000 for rate_des didn't led to a apparent in- or decreaisng of the rate in setting 1. An assumption that came over time for lower rates than 5000 was, that only the noise of the sensor will be controlled. This could be supported by setting the last *diffON* value in the java programm from the lsr-wiki. The necessary events to recognize the LED screen were not visible any longer for human eyes. Only noise of the environment was responsible for the rate.

The median values corresponded better to the desired rates as the average, but this can be a reason of the changing rate in the first 10 to 15 seconds. The median value in this example is near or lies on the marked line in Figure 3.1. It's a rate that is occurring many times over the time period. Because of the unknown transfer behavior, it was not possible to calculate a steady state error and to compare it to the difference between rate_des and the average or median of the measured event rate. The overshoots by trend are higher for high rates and vice versa.

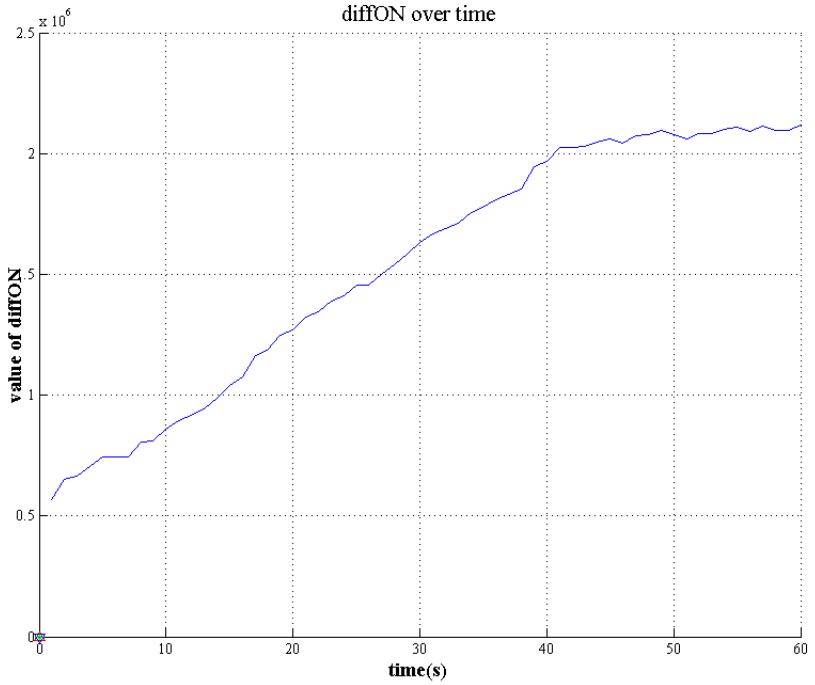

Figure 4.2: development of the *diffON* value (rate_des = 6000)

4.3.2 Fuzzy PID Controller

With the test environment set up as above, we have some tests results in the table. As time period 60 seconds are chosen.

- Control status = false, 60s, all bias parameters are set to default values. The Performance is shown below.

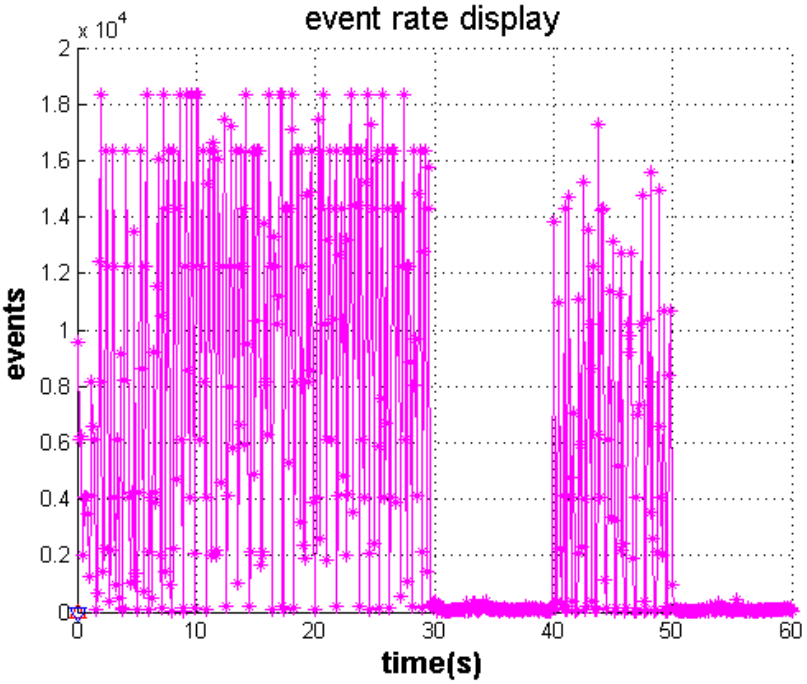

Figure 4.3: without controller

- Control status = true, 60s, all bias parameters are set to default values.

rate_des	mean()	median()	*std/rate_des*	std
2000	2001	2058	0.64	1280
5000	4935	4245	0.38	1900
6000	5971	6120	0.27	1620
8000	7891	8160	0.17	1360
9000	8312	7159	0.39	3510

For rate_des = 6000, the corresponding plots can be found in the appendix.

- Control status = true, changing rate_des after 30s, 60s: A significant change in diffON value and event rate can be seen if after 30s the rate_des is reduced from 6000 to 4000. A plot is shown in Figure 4.4.
- Control status = true, move the sensor slowly during 60 seconds. A plot is shown in Figure 4.5, the table below shows the difference compared to sensor in static status.

status	rate_des	mean()	median()	*std/rate_des*	std
static	6000	5971	6120	0.27	1620
moving	6000	5960	6120	0.32	1920

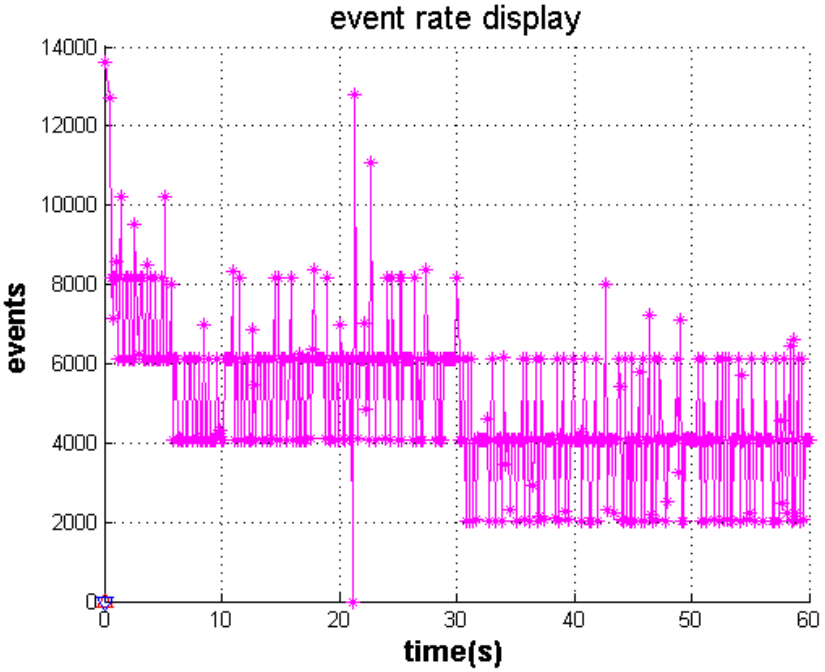

Figure 4.4: rate_des from 6000 to 4000 at 30 seconds

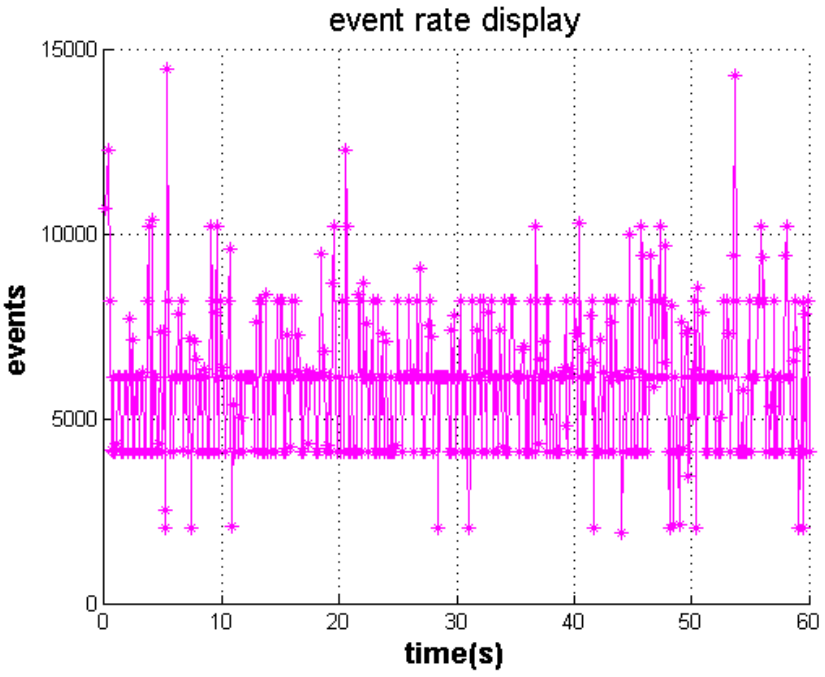

Figure 4.5: moving sensor slowly

4.4 Conclusion

- The P-controller led to first experiences but is not really able to follow the aim of controlling the rate to a roughly constant value. The evaluation depends on the overshoot one is allowing.
- The Fuzzy PID Controller seems to have better performance than P Controller when the rate_des is from 2000 to 8000, from about 9000 on, the performance of Fuzzy PID Controller sinks strongly.
- Both controllers work better with lower rate_des.
- P-controller is easy to tune, but Fuzzy PID Controller is much more difficult.
- A survey on the performance and a performance orientated tuning of the controller parameters are difficult at this moment.
- When moving the sensor, the controller still works fine.

5 Summary

In this project, we have studied the eDVS optics and chip setup, then try to develop 2 controllers according to the relationships of bias setting and event rate, later we set up different test environment to evaluate our controllers in different situations. Some of our results and problems are listed below:

5.1 Main results

The main tasks could be attended but also a few problems and open questions resulted from the work on our topic.

- study existing eDVS optics and chip setup
- evaluate chip tuning options to modify event-rate
- design and implement 2 real-time controllers for bias settings to keep event-rate roughly constant
- design and fabricate test setups for event rate
- characterize and evaluate performance of designed controller

5.2 Problems

- Masurement of the event rate
  - Stored events in the buffer and the pause time for collecting events.
  - Not continuous generated event rate by the sensor.
- No accurate eDVS model and transfer function given.
- Relationship between diffON and event rate is not accurate.
- A more stable test setup is needed.
- Controller considering both diff and diffON difficult to realize.

5.3 Future Work

To further improve the controller, at least two or all the diff parameters should be taken into account and an accurate mathematical model of eDVS should be explored.

An enhanced Controller could probably be improved by following steps: First increase *diffON*, then regulate on *diff* afterwards if a desired rate is not already reached. Another

step can be to reduce control commands in a certain region above and below the desired rate by defining a tolerance. Within this region, the diff value then will not be changed and can reduce the oscillation of the rate around the desired rate by keeping a constant bias setting.

Distribution of written text between students:

Kaspar Hitzelberger / Ming Xue

1 Introduction

2 eDVS optics and chip setup

2.1 eDVS sensor
2.1.1 Description
2.1.2 Functionality
2.2 eDVS board
2.2.1 Description
2.2.2 Connection and communication with computer

3 Controlling the event rate

3.1 Measuring of the event rate
3.1.1 Buffer properties of serial interface
3.1.2 Method to measure events
3.2 Bias current generators
3.2.1 Values and correlations
3.2.2 Study of bias values
3.3 Displaying of the event rate
3.3.1 Format of events
3.3.2 Display events
3.4 Designed controllers
3.4.1 Modeled relationship between bias values and event rate
3.4.2 P-controller
3.4.3 Fuzzy PID Controller

4 Evaluation

4.1 Evaluation environment
4.2 Evaluation method
4.3 Evaluation of controllers
4.3.1 P-controller
4.3.2 Fuzzy PID Controller
4.4 Conclusion

5 Summary

5.1 Main results
5.2 Problems
5.3 Future Work

List of Figures

Figure 2.1: dynamic vision sensor (DVS) 5
Figure 2.2: Functionality of eDVS 6
Figure 2.3: Embedded-DVS board showing functional elements 7
Figure 2.4: connection MATLAB® and eDVS 7
Figure 2.5: connection between the serial port object and the device (www.mathworks.com) 8
Figure 3.1: input buffer of serial port object (www.mathworks.com) 9
Figure 3.2: properties of diffOFF 11
Figure 3.3: properties of diff 12
Figure 3.4: diff=99187, properties of diffOFF 12
Figure 3.5: properties of diff 13
Figure 3.6: diff and diffOFF set to big values 14
Figure 3.7: event rate display (ON = red, OFF = blue, events_all = green) 15
Figure 3.8: event rate display (sum of events = magenta) 15
Figure 3.9: feedback loop with P-controller 17
Figure 3.10: feedback loop with Fuzzy-PID controller 17
Figure 3.11: fuzzy rules table for kp 19
Figure 3.12: FIS chart 20
Figure 4.1: event rate for P-controlled sensor (rate_des = 6000) 22
Figure 4.2: development of the *diffON* value (rate_des = 6000) 23
Figure 4.3: without controller 24
Figure 4.4: rate_des from 6000 to 4000 at 30 seconds 25
Figure 4.5: moving sensor slowly 25

Bibliography

[1] Delbruck T, Berner R, Lichtsteiner P, Dualibe C: 32-bit Configurable bias current generator with sub-off-current capability. In ISCAS; Paris: 2010

[2] Jorg Conradt, Raphael Berner, Matthew Cook, Tobi Delbruck: An Embedded AER Dynamic Vision Sensor for Low-Latency Pole Balancing.

[3] Patrick Lichtsteiner, Christoph Posch, Tobi Delbruck: A 128 128 120 dB 15s Latency Asynchronous Temporal Contrast Vision Sensor.

[4] http://www.mathworks.de/de/help/fuzzy/examples/using-lookup-table-in-simulink-to-implement-fuzzy-pid-controller.html

[5] Denk C., Llobet-Blandino F., Galluppi F., Plana LA., Furber S., and Conradt, J. (2013) Real-Time Interface Board for Closed-Loop Robotic Tasks on the SpiNNaker Neural Computing System, International Conf. on Artificial Neural Networks (ICANN), pre-print.

[6] Müller GR., Conradt J. (2011). A Miniature Low-Power Sensor System for Real Time 2D Visual Tracking of LED Markers, Proceedings of the IEEE International Conference on Robotics and Biomimetics (IEEE-ROBIO), pages 2429-35, Phuket, Thailand.