Ingenieurspraxis

Institute of Automatic Control Engineering (LSR)
Neuroscientific System Theory (NST)

Simon Bremer

Adviser: Cristian Axenie

Onboard event based processing algorithms for Embedded Dynamic Vision Sensors (eDVS)

Proposal

Problem description

The ability to perceive the surrounding environment visually opens a path to solving many robotics perception problems such as object detection, obstacle avoidance etc. Typical algorithms are targeting standard cameras and they are based on well established engineered methods of processing data.
Opposite from standard cameras which provide a time sequenced stream of frames the dynamic vision sensor provides only relative changes in a scene, given by individual events at the pixel level. Using this different encoding scheme the dynamic vision sensor brings advantages over standard cameras. First, there is no redundancy in the data received from the sensor, only changes are reported. Second, as only events are considered the eDVS data rate is high. Third, the power consumption of the overall system is small, as just a low-end microcontroller is used to fetch events from the sensor and forward them to a PC for processing.
Recently a new version of the dynamic vision sensor ApsDVS was developed. The new sensor provides a higher resolution of 240x180 pixels and an additional grayscale image in parallel to the pixel events, which extends the usage also for static scenes. Opposite to the first setup (eDVS sensor + PC), here all processing should be performed locally on the DVS interface microcontroller, to create a standalone system.

Task

The goal of this 4-week Ingenieurspraxis is to extend the basic interface code and to develop additional algorithms for the new eDVS sensor on a new and powerful 32-bit ARM Cortex-M4 microcontroller. A test board has already been developed for the new sensor in [1] and the basic sensor setup is done. The basic interface manages event fetching from the DVS sensor and buffering. Starting from this basic setup the grayscale image readout must be implemented and more complex algorithms (e.g. optic flow, line tracking, object tracking) should be developed on board of the new microcontroller, in order to keep the system compact. Although initial algorithms have been developed, they need to be optimized (e.g. that the microcontroller can accommodate both event fetching and event processing).

Time schedule

- 1st week: get familiar with eDVS, interfaces and event based processing algorithms;
- 2nd week: optimize event fetching on the new eDVS test system and implement image readout;
- 3rd week: implement event based algorithms on new eDVS test system (e.g.: optic flow, line tracking);
- 4th week: algorithm tuning and result analysis.

[1] Microcontroller Interface for Embedded Vision Sensor - Razvan Tiganu, Roman Ursu, PP Report, WSS2012-13

Problem Analysis

Currently the microcontroller is only used to fetch events from the sensor and send it to the PC via UART/Virtual Com-Port. The method of fetching is implemented using the microcontroller's CPU as the main resource which is therefore occupied.
In order to use the controller's CPU power for other tasks such as event analysis algorithms, the actual event fetching needs to be outsourced to the microcontroller's periphery to free up CPU time. One main task to accomplish this is to set up a DMA controller to transfer the parallel data from GPIO inputs into the microcontroller's memory. In addition the sensor's request needs to be acknowledged properly for the sensor to continue sending events.

Also the grayscale functionality is to be added to the microcontroller's program code. Additionally the visualization tool running on the PC needs to support grayscale data.

General Technical Information

ApsDVS (embedded)

Resolution: 240 * 180 pixel

Functionality:

Bias Setup
The sensor is connected to a number of capacitors which store analog bias voltages. In order to set these analog settings, the sensor itself has the functionality of reading settings via a digital sequential bus, producing the analog voltage and storing it to the desired capacitor.

In the main loop the program checks if the flag “enable_bias” is set. In that case the setup function “Bias()” is called. In order to ensure that the sensor is set up before any usage, this flag is set at initialization before entering the main loop.
To set this flag again even after startup a “B” can be send over UART.

Event based sensor readout
To use the event based readout the DVS needs to be properly setup. The sensor will send out events as soon as the SensorReset signal is set to high. This is done as soon as a “+” is received over UART. A “-” command will reset the signal.

Events are sent in data packages over a 10bit parallel bus. As soon as an event is captured by the sensor the N_Req signal be set to low (low active Request, from sensor to uC). The parallel bus will then be set to the data package. After these 10 bit of data are stored the N_Ack line (low active Acknowledge, from uC to sensor) needs to be set to low. The sensor will then take back the request (N_Req to high) and after N_ack is also set back to high a new request can be sent (N_Req to low again).

Data packages can contain a y-address or x-addresses and polarity of an actual event.

Bits shown as MSB first:

AE9	AE8	AE7	AE6	AE5	AE4	AE3	AE2	AE1	AE0

AE9: If this bit is 0 this package contains a y-address else a x-address is sent.

y-address data package (AE9 == 0)

0	unused	AE7 – AE0 8bit event y-address data

AE8: not used
AE0 – AE7: 8bit y-address data (AE7 MSB, AE0 LSB)

x-address data package (AE9 == 1)

1	AE8 – AE1 8bit event x-address data	AE0

AE1 – AE8: 8bit x-address data (AE8 MSB, AE1 LSB)
AE0: polarity of the event (0: off-event; 1: on-event)

One complete event is set together by an x-address package, the corresponding y-address was sent in the last received y-address package

Example

Data packages in chronological order:

y-addr y1	x-addr x1 p1	x-addr x2 p2	x-addr x3 p3	*y-addr y2*	x-addr x4 p4

Events:

Event polarity	x-address	y-address
p1	x1	y1
p2	x2	y1
p3	x3	y1
p4	x4	y2

Currently the sensor has a rather strong background noise of approximately 30000 events per second. Whether this is normal for the new ApsDVS or this can be traced back to the PCB used is unknown.

Grayscale image sensor readout

Contrary to the previous DVS version the ApsDVS is also able to provide a grayscale image. Due to the different architecture compared to a frame based camera and the prototype design there is no digital data output for pixel data.
Pixel coordinates have to be selected manually with shift registers which are 1 bit wide and the amount of pixels per axis deep.
To feed bit data into the registers a value is applied to the bit data inputs (RowBit / ColBit). After a clock signal (raising edge) is sent to the clock inputs (RowClk / ColClk) the input will be stored in the first cell of the register while all other values slide down one cell.
On the y-axis one row can be selected by feeding a “1” into the registers and clocking it to the designated row number. All other values need to be set to “0”. This can be archived by applying a low voltage to RowBit (“0”) and sending 180 (amount of pixels on the y-axis) clocks to RowClk.

The same sequence should also be applied to the x-axis (ColBit / ColClk) with the proper amount of clock signals (240) since values can be random after initialization.
While only one row can be selected on the y-axis, 2 columns can be selected on the x-axis using 2 different 3-bit patterns. The “A” column selector (bit pattern: 110) supports “read” and “reset” while the “B” selector (bit pattern: 010) only supports “read”.
The selected functionality is chosen using the 2 command inputs (ColState0 / ColState1).
Commands:

ColState0	ColState1	Function
0	0	No function selected (Must be selected while shifting/clocking)
1	0	Read pixel on selector "A"
0	1	Read pixel on selector "B"
1	1	Reset pixel on selector "A"

The function “read” will connect apsReadout/adc123_IN0 to the voltage of the pixel selected. The microcontroller's ADC unit can then convert the analog voltage applied to an actual digital value.

In order to read the gray value of a pixel it needs to be reset first. The internal pixel voltage will then be reset to a high analog reset value. Due to the difference of reset voltages, this voltage needs to be read shortly after the reset.
After a certain exposure time ranging from 0,5 to 300ms depending on the light conditions the pixel is read out again. A bright environment will cause a higher drop of pixel voltage while a dark light input will cause no or only a small drop of voltage.
The final gray value is the calculated difference of both readout values.

To capture a whole image selector “A” and “B” will sweep trough all columns while the row selector will go through all rows for every column. Column selector “A” is used for voltage reset and the first read operation. The space between “A” and “B” will act as the exposure time and needs to be equal for all pixels. Selector “B” is then used to capture the drop of voltage with the second read.

See description paper for visualization of the readout process.

This process including sending the resulting data to the PC is right now launched by sending an “i” to the microcontroller via UART.

Because the ADC readout is not yet implemented efficiently, reading one whole image takes about 0.5s. In this period of time the processor of the microcontroller is occupied and will not read or send any events.

Microcontroller (STM32F407ZE)

The controller runs the C program directly without underlying operating system. The provided STM32 library is mainly used to initialize the peripheral units of the controller. Most time critical actions are implemented using direct register manipulation to gain performance.

UART

A connection to a PC is established via a UART microcontroller unit which is connected to a FTDI UART USB interface. The following settings are currently employed for this connection:

Baud rate	8M = 8000000 bits/s
Word Length	8 bit
StopBits	1 bit
Parity	None
Handshake Mode	RTS, CTS

The corresponding Windows driver for the FTDI chip does not properly cope with the rather high data rate. It is possible that the data rate of the virtual Com-Port is set to a much lower value than 8M. Reinstalling the driver of the “USB Serial Port” usually resolves this problem.
A restart of Windows might be required. In addition it can help to delete the previously installed, now unconnected devices in the Device Manager.

Protocols for data transmission from microcontroller to PC in bytes:

<u>Sensor Event</u>

Polarity (0 = off, 1 = on)	X address	Y address

Inside the program's main loop, events from a large circular event buffer are written into an UART output buffer as soon as this buffer is empty.

<u>Grayscale readout pixel data</u>

0xFF	0xFF	Gray value	X address	Y address

A double 0xFF index byte is employed as it turned out to be more reliable.

Commands to the microcontroller sent from the PC:

B	Run Bias initialization
+	Enables Event capturing
-	Disable Event capturing
i	Capture and send one grayscale frame (still slow)

TIM (Timer)

The STM32F4 has a variety of timers build in. To produce a good timestamp for events a 32bit timer should be used. TIM2 and TIM5 meet this criteria.
Timers can be used to listen to events such as raising or falling flanks sent to GPIO pins and can be used to trigger other peripheral devices.
Each timer has a limited set of available input pins.

DMA (Direct Memory Access)

The STM32F4 contains 2 DMA controllers of which only DMA2 is able to access AHB1 peripherals such as GPIO pins.
Like timers DMAs have a limited set of events which can be used as a trigger source. DMA2 for

example can only be triggered by TIM1 and TIM8 of all timers.

Event capturing

Timer Trigger / Software Fetch and Acknowledge (current)

The request signal is hooked up to a timer that writes the current timestamp into a register as soon as a falling flank is received on the request pin. This action is still independent from the state of the main program. The program's main loop then recognizes a change of the timer's register and proceeds by reading the parallel data and sending manual acknowledge signal.
Y-addresses are stored in a variable for later x-addresses.
When an x-addresses is received a new event is created and written into the circular event buffer. If many events are produced by the sensor it is possible that more events are received than sent out. In this case of an overflow old events are overwritten and the boards red LED is turned on.

Timer Trigger / DMA transfer

TIM1 is triggered by a falling flank on the request pin and then triggers DMA2. The DMA is able to transfer parallel data into an array for later processing which is independent from the CPU.

The challenge using this method is that copying the data to memory alone is not enough. An acknowledge signal needs to be sent back to the sensor to receive the following event also.

Two options that do not occupy CPU time are possible:

Sensor self-acknowledgment using a resistor between request and acknowledge line

Implementing an electronic connection in between the request line from sensor to microcontroller and the acknowledge line from microcontroller and sensor causes the sensor to acknowledge itself since a low signal on request will also result in a low signal on acknowledge.
Using this technique the sensor will send out events independent from any action of the microcontroller and at maximum speed.
This method was successfully tested on the old eDVS sensor. However on the new ApsDVS this setting results in a very high sending frequency (~2Mhz). Data fetched by DMA turned out to be faulty and inaccurate.
That might be caused by the rather short time the parallel data input is set to its correct state due to the high frequency.
Analyzing request and acknowledge signal with an oscilloscope the signals did not look like proper digital signals but more triangularly shaped.

Acknowledgment using DMA

To generate a proper digital output signal on the acknowledge line, it should be hooked up to a GPIO pin generating output voltages with steep flanks.
GPIO output ports can be set by writing a value into a control register which can be done by the CPU executing code. This is done in the current fetching method.
DMA can also perform this task by copying a constant variable into the GPIO control register. Since a full acknowledge signal consists of a falling flank and a raising flank after a short time, 2 DMA actions are required.
DMA streams can be run after each other, following a certain priority list. Running both actions one by one the timer trigger signal will result in an extremely short pulse in which the time the signal is actually low is too short to work reliable.

To bypass this problem a second timer trigger listening to a raising flank on the request signal can be employed.

The following schematics show the cycle of events and it's causes:

Trigger	*Caused Event*
Request to low (active)	First timer triggers DMA sequence 1
DMA sequence 1	Copy parallel data into array
	Copy timestamp into array (optimal)
	Set acknowledge signal to low (active)
Sensor receives low acknowledge signal	Sensor sets request to high (inactive)
Request to high (inactive)	2nd timer triggers DMA sequence 2
DMA sequence 2	Set acknowledge signal to high (inactive)
Sensor receives high acknowledge signal	Sensor can send a new event

This method of fetching and acknowledging events would resolve the trouble experiences with self-acknowledgment.
DMA2 can only be triggered by TIM1 and TIM8 and for this method to work both of them need to be in use since one timer can only listen to a falling or a raising flank and each flank is triggering a different action.

Timers though have a very limited choice of possible input ports. The current PCB design does not allow the use of both timers simultaneously because all possible input ports for TIM8 are either taken by the parallel input bus (GPIO Port C) or not accessible with this version of microcontroller.

PC Visualization Tool (UART_Comm)

UART_Comm is written in C# and implements the functionality of displaying a constant flow of events and single grayscale images.
A total of 3 displaying modes are available:

Events	Events are displayed acording to their polarity as green or red pixels and fade away after a short time.
B/W	The pixel of a received event will constantly set to white or black acording to the events polarity.
Grayscale	Displays grayscale frames. Select the mode first, then send an "i" command.

Like the previous Java program commands can be sent via UART.
The code is well commented and structured. Information on the exact functionality can be retrieved from comments and code.

Progress

Grayscale functionality

I added support for grayscale functionality of the sensor to the code, a single frame can be captured

by sending the “i” command now.
The PC visualization tool UART_Comm supports displaying grayscale images, the correct mode has to be selected.

Event fetching

I cleaned the previous version of the code and optimized a few commands. This version fetches events software based and sends them out via UART. Since the CPU is occupied running algorithms would be very hard in this version.
To free up the CPU I decided to implement DMA based fetching.
As the board layout of the PCB was designed, the request input is connected to a pin which supports TIM2. In order to copy data automatically using DMA, DMA2 is necessary. Since DMA2 can only be triggered by TIM1 and TIM8 the request line had to be soldered to another pin also which supports TIM1. I experienced some difficulties with DMA especially running multiple DMA tasks triggered by one event. After a new version of the code is flashed to the microcontroller, it should be reset once to run DMA properly.
After the DMA data transfer itself worked I tried using a resistor to have the sensor self-acknowledge itself. This turned out to be unreliable since the retrieved data was faulty often. General shapes shown in the event visualization were still recognizable but the signal quality was far to weak to be acceptable.
An attempt to send both the acknowledge low and high signal at the falling request signal trigger failed because of unreliability. The acknowledge signal was too short to be always recognized as such.
I was not able to implement the idea to use TIM1 and TIM8 simultaneously and listen to both request flanks to send the acknowledge signal one by one because in the current circuit layout all GPIOs supporting TIM8 are blocked. Since the parallel input bus blocks there pins, the whole bus would need to be relocated in order to free one of the pins.

Event based algorithms

I was not able to implement or optimize any event based algorithms since I was not able to successfully outsource event fetching to the microcontroller's periphery.

Future Work

Grayscale functionality

The current use of the ADC unit is designed for temporary use only and is therefore not optimized and slow. ADC can run in triple mode using all 3 ADC controllers. In addition DMA could be used to store values after the ADC process is finished.
Right now both, event based fetching and grayscale readout, need CPU time therefore they can not be run simultaneously.

Event fetching

To use the microcontroller running algorithms the event fetching needs to be outsourced. It might be possible to use self-acknowledgment, a different electronic connection or layout could resolve the problem.
If another new PCB is created it should be made sure that the request signal can be accessed by both TIM1 and TIM8 in order to use DMA acknowledgment.
Using a second smaller microcontroller/CPLD that only deals with fetching and assembling events to then send them out could also be an approach.

The rather high noise level of the ApsDVS needs to be checked.

New board

The board can be programmed and will run while no sensor is plugged into the socket. As soon as a ApsDVS is connected an electronic fault occurs. This needs to be resolved.

Some pins had to be changed to connect the SRAM to the STM32. Pin changes have been applied to the code in “ApsDVS_Firmware3”. SRAM initialization has been implemented and shortly tested in “ApsDVS_NewBoard”. The program still needs to be setup to actively use the SRAM.

Bibliography

Weikersdorfer D., Conradt J. (2012), Event-based Particle Filtering for Robot Self-Localization, Proceedings of the IEEE International Conference on Robotics and Biomimetics (IEEE-ROBIO), pages: 866-870, Guangzhou, China.

Weikersdorfer D., Hoffmann R., and Conradt J. (2013) Simultaneous Localization and Mapping for event-based Vision Systems, International Conference on Computer Vision Systems (ICVS), preprint.