ArDVS: A DVS Arduino shield

Final Report

submitted by
Michael Dirix,
Jonas Timmermann

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universität München

Prof. Dr Jörg Conradt

Supervisor: Dr. Christoph Richter
Final Submission: 07.07.2015

Abstract

Arduino is a popular open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the Arduino boards. Given the low computational power of a typical Arduino microcontroller, processing live image data of even a low-resolution camera is, often not realizable. Dynamic Vision Sensors (DVS) naturally reduce the data stream by orders of magnitude at a chip level. Instead of sending whole frames of pixel data at fixed time intervals the DVS chip emits single events that represent pixel-local changes of light intensity.
The topic of this practical course was to develop a DVS Arduino shield and basic libraries to use and control the DVS data stream with Arduino. This final report describes the development steps, specification, design and manufacturing of two hardware prototypes. Additionally, the specification of the software, software design, implementation and finally the documentation of an ArDVS example project. In order to test and debug the DVS library during development and to offer the ArDVS users an intuitive graphical interface, an Android application for the ArDVS shield was developed additionally.
In summary, the Arduino shield, library, pan-tilt system and the Android application developed in this practical course can be perfectly used as plug and play solution to get familiar with dynamic vision sensors. However, for professional use, the low computational power of Arduino boards limits the applicability.

Contents

Chapter 1

Introduction

Arduino is an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the arduiono boards. Lots of people prefer to realize their hardware and software projects with Arduino, because it is inexpensive and has a simple and clear programming environment. Given the low computational power of a typical Arduino microcontroller, processing live image data of even a low-resolution camera is, unfortunately, close to impossible. While workarounds in the form of dedicated preprocessing engines do exist, a general approach to vision on Arduino is lacking.
Dynamic Vision Sensors (DVS) naturally reduce the data stream by orders of magnitude at a chip level. Instead of sending whole frames of pixel data at fixed time intervals the DVS chip emits single events that represent pixel-local changes of light intensity. As this resembles a part of our retinal functionality, the technology is also known as silicon retina.
The topic of this practical course was to develop a DVS Arduino shield (ArDVS) and basic libraries to use and control the DVS data stream with Arduino. This final report describes the development steps, specification, design and manufacturing of two hardware prototypes. Additionally the specification of the software, software design, implementation and finally the documentation of an ArDVS example project. In order to test and debug the DVS library during development and to offer the ArDVS users an intuitive graphical interface, an Android application for the ArDVS was developed additionally.

Chapter 2

Hardware

The main goal of the hardware was to create a universally useful board for the Arduino community. Therefore, the focus was on flexibility, ensuring compatibility to different Arduino boards, offering various mounting options for the eDVS board, providing different interfacing channels and extension possibilities.

2.1 Components

In this section the main components of the hardware solution are discussed. Figure 2.1 shows the fully assembled ArDVS shield.

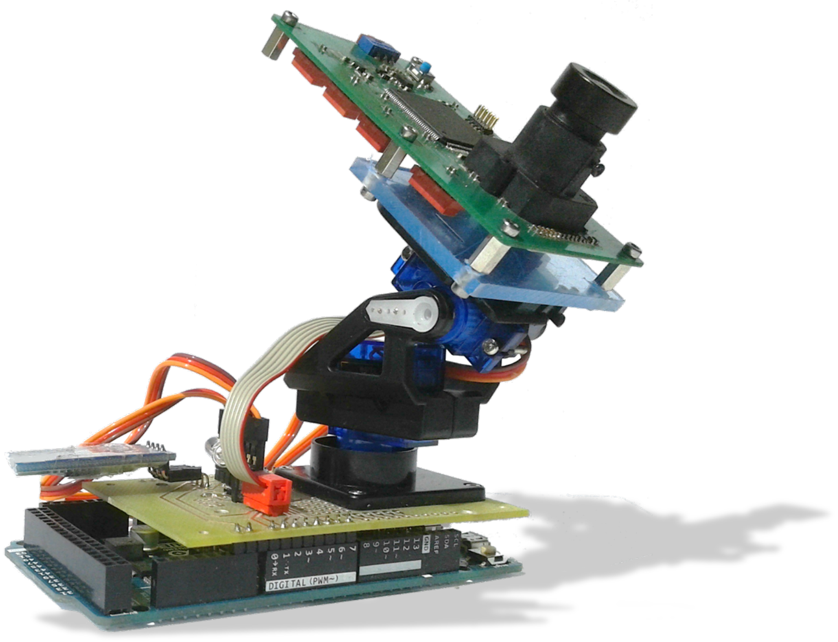


Figure 2.1: ArDVS shield

2.1.1 Arduino Due

The Arduino Due board, shown in figure 2.2, was chosen for its superior performance compared to the other Arduino controllers. All the other boards are based on 8-bit Atmel AVR ATmega microcontrollers which are clocked at up to 16 MHz and feature very limited RAM and flash memory. The Arduino Due in contrast features a 32-bit ARM Cortex M3 from AVR with a clock rate of 84 MHz and with 96 kB of SRAM 12 times as much main memory compared to the second largest Arduino.
While the eDVS reliefs the host controller from much of the processing, a microcontroller with a decent CPU is still required to handle and store the incoming event stream.
Another benefit of the Cortex M3 is the great selection of peripherals. Especially, a second UART port for communicating with the eDVS and the Bluetooth module at the same time is essential and most other Arduino boards only offer one port.

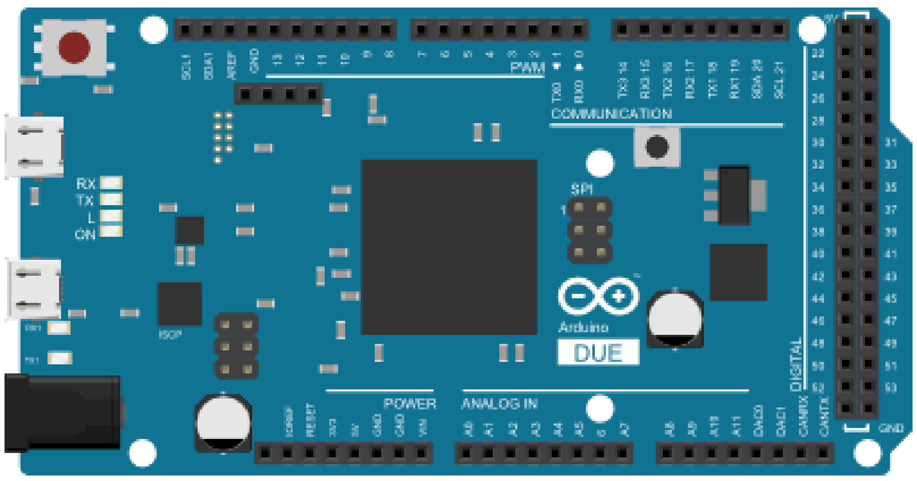


Figure 2.2: Arduino Due[ard15]

2.1.2 eDVS4337

The eDVS4337 board by iniLabs, depicted in figure 2.3, is used as a vision system. The board features iniLabs' DVS128 sensor chip as well as a powerful LPC4337 ARM Cortex based microcontroller by NXP. The microcontroller's M0 coprocessor handles the communication with the sensor while the 204 MHz clocked M4 main core takes care of the data processing and interfacing to other systems. The communication with the Arduino microcontroller is facilitated through a UART connection with a protocol detailed in the software section.
The eDVS4337 board offers many additional features like a battery charging circuit, an inertial measurement unit, a microSD card slot and motor drivers. With this rich configuration it can act as a standalone robot controller. The ArDVS shield

does not use any of these features, so that the shield is also compatible to more economical eDVS boards in the future. As the board is connected to circuit board of the Arduino shield by a 6 pin Micro-MaTch connector it is easy to switch the eDVS board to newer revisions or other DVS modules that offer the same connector.

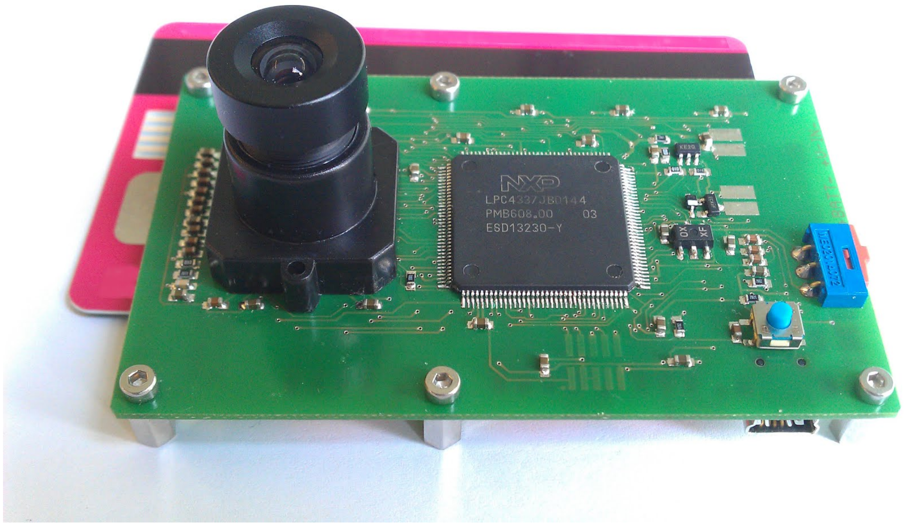


Figure 2.3: eDVS4337[ini15]

2.1.3 Bluetooth module HC-06

For the wireless communication from the ArDVS to a host system a Bluetooth module port was included in the shield design. The Bluetooth link basically serves as a wireless bridge for a serial connection. The HC-06 module by Guangzhou HC Information Technology, pictured in figure 2.4, was chosen, mainly because of the low price point of 2USD. The device operates in slave mode and achieves a baud rate of 230,000 bit/s, which is not enough for high throughput tasks but sufficient for many monitoring and configuration applications.

2.1.4 Pan-tilt-system

For a flexible mounting solution of the eDVS board on the ArDVS shield the mini pan-tilt kit available from Adafruit was chosen. As seen in figure 2.5, the system consists of two small servos and some plastic parts that allow the sensor chip to point in any direction within about 180° for panning and tilting. The whole system is only about 40mm x 40mm x 70mm in size and 37g in weight. The servos operate from 3 to 6V and are controlled with the standard servo protocol. The positioning flexibility and the possibility of readjusting during operation enable many interesting use-cases, which are impossible with a static construction.

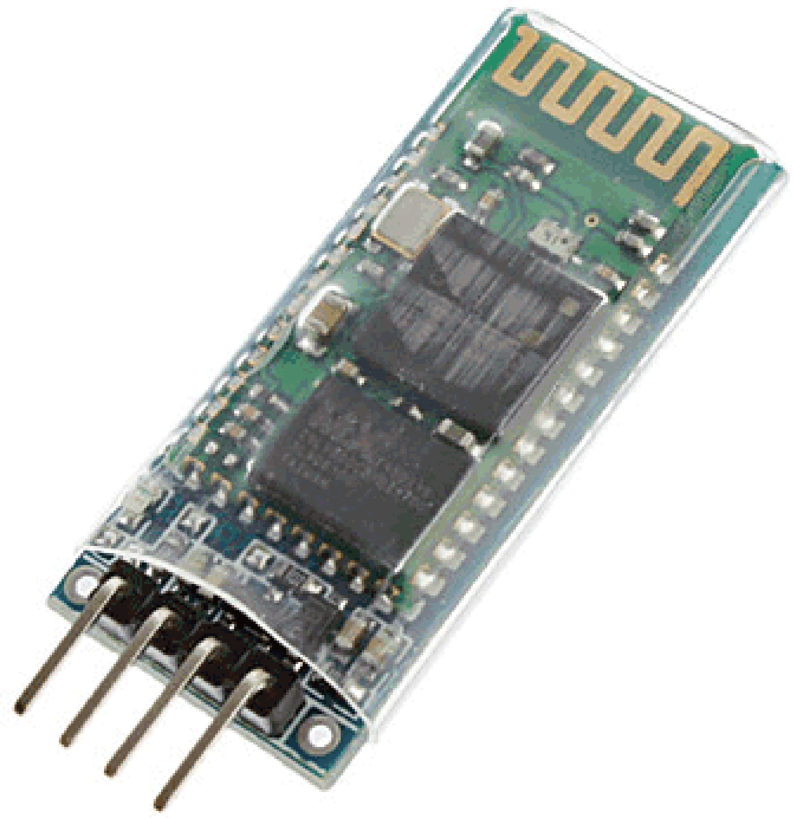


Figure 2.4: HC-06[Pas15]

2.2 Arduino shield

The foundation for interconnecting all the previously mentioned components is a self designed circuit board.

2.2.1 Design

This so-called shield follows the Arduino Uno R3 layout for the core functionality of interfacing the eDVS board. Yet, for the Bluetooth communication feature an additional UART port is required, which is only offered by Arduino Due, Mega 2560 and Mega ADK. Figures 2.6 and 2.7 show the schematics and layout of the board created with the board design software EAGLE by CadSoft.
The schematics of the board include UART connections for the Bluetooth module and one with flow control for the eDVS. Further, there are motor control circuits for the two servos and an infrared LED control, which is used as a remote control. The board layout additionally features mounting holes for the pan-tilt-system base-

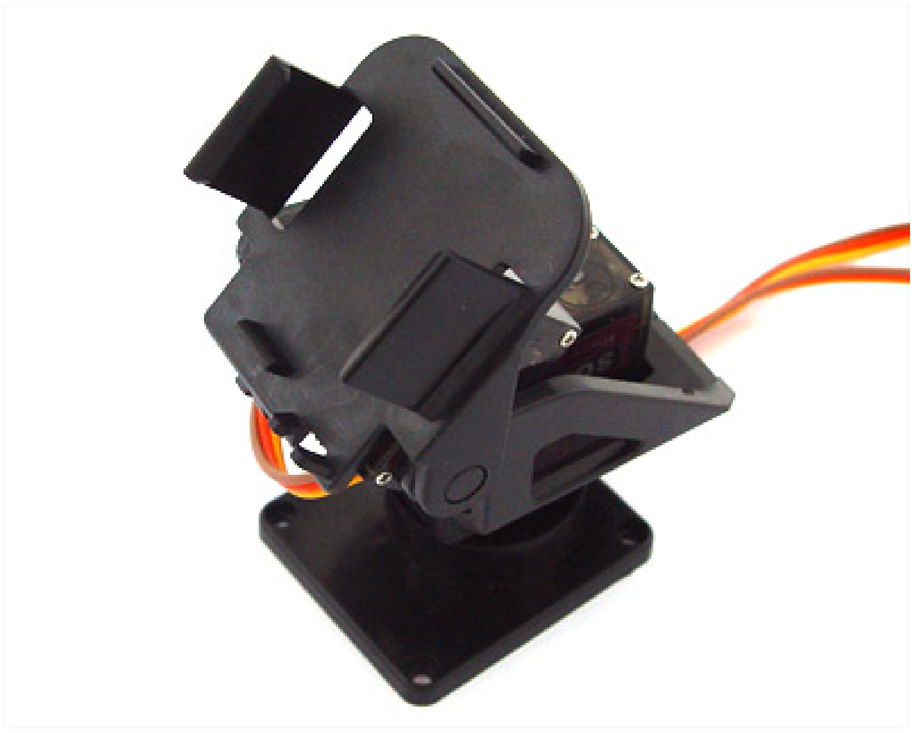


Figure 2.5: Pan-tilt-system[4tr15]

plate with a cutout underneath to fix the servo in the right position without the need to remove the base plate. The four holes in the corners allow for the eDVS to also be mounted flat on the shield without a pan-tilt-unit. By using only two holes with 90° joints it is further possible to tilt the eDVS board to any of the sides. For creating simple additional output circuits a prototyping grid is included next to the eDVS port.

2.2.2 Prototype iterations

The presented board is the second iteration, which benefitted from the learnings of the first prototype. The prototyping area as well as the remote control LED were only added in the second generation. Further, the cutout for the servo was included and the port for the Bluetooth chip was turned by 180° for the possibility of mounting the module flat.
Another learning of the board's copper milling production process was the importance of removing unused copper planes. Due to the lacking solder resist with this production technique it was impossible to solder the components without creating many soldering bridges. The additionally required soldering time led to damaged pads and unreliable connections. By clearing the unconnected copper planes in the production of the second iteration the board was much easier to solder with far superior soldering connections.

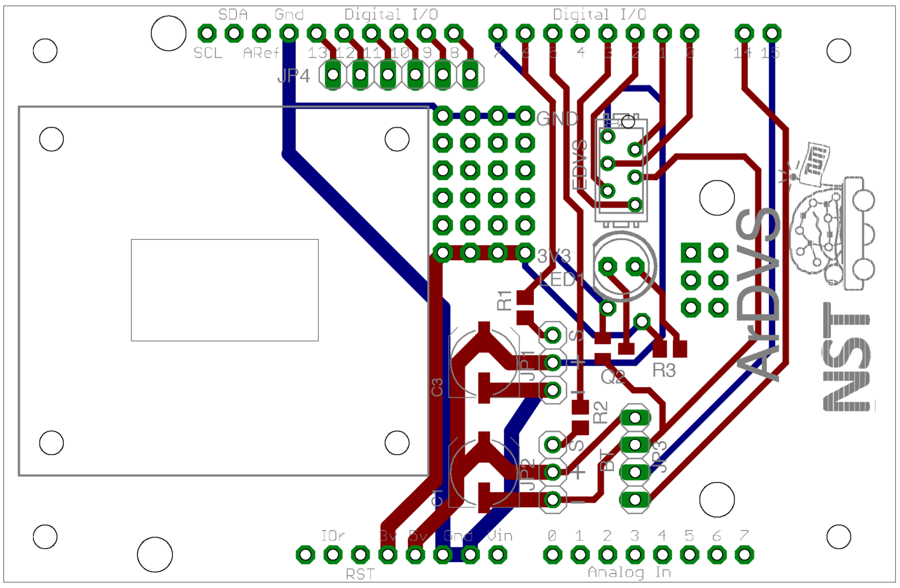


Figure 2.6: ArDVS board layout

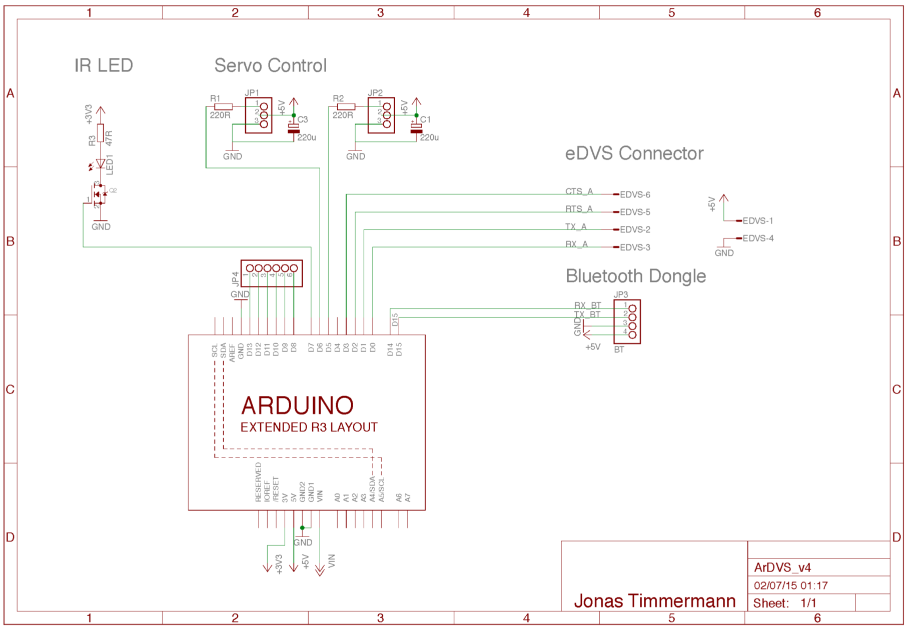


Figure 2.7: ArDVS board schematics

Chapter 3

Software

The software part of this practical course was initially limited to a basic library to use and control the DVS data stream with Arduino. The eDVS4337 board, offers a lot of more functionalities than the raw data stream. Via serial port it is possible to set and get parameters for motor driver, real time clock, baud rates and much more. This features can be very useful for ArDVS projects which may be realized by the Arduino community. Because of that, we decided to provide this additional functionalities to the ArDVS users. The whole ArDVS library is designed according to the Arduino Style Guide for writing libraries. This style guide is for Arduino library APIs developers. Some of these guidelines run counter to professional programming practice, but made it possible for so many beginners to get started with Arduino easily. The steps of developing the ArDVS library, discussed in the following sections, are definition, implementation, test and documentation. Finally, there are three example projects described, which are developed on top of the library. Offering example code to the Arduino user is state of the art for Arduino libraries. In order to do the development more efficiently, it is at some point absolutely necessary to visualize the data stream in some way. Of course the Arduino board has no possibility to visualize the stream directly. Currently, there are three ways mostly used to view the events coming from the eDVS. All of them require to connect a USB cable from the Arduino Board to the Computer. The first way is to show the raw output via an terminal emulator, described in the eDVS user guide of iniLabs. The second way is to use some Matlab scripts to visualize the event stream graphically. The third way is to use the officially supported software to view and process events coming via UART, called the jAER framework. All of them need multiple steps to set up and knowledge how to use terminal emulator, Matlab or the jAER framework. At this point, it might happen, that the Arduino user loses their motivation because of the big time investment in setting up the eDVS project. For this reason we decided to design and implement an Android application, which can connect to the ArDVS automatically and does not require any setting up to view the eDVS events graphically.
The ArDVS library, developed in this practical course, only works like a plug and

play product, if the eDVS comes with customized firmware. At default, the baud rate of the eDVS is set to 4 million baud. This speed is too fast for most of the Arduino boards currently in the market. Because of that, the last chapter describes modifications on the eDVS board operating system and how to apply the changes to the eDVS board.

3.1 ArDVS library

This section is about the ArDVS library API of the ArDVS. In the subsection *Definition*, the requirements for the ArDVS library API are specified, mostly according to the Arduino style guide. The following chapters are about implementation, test and documentation of the library.

3.1.1 Definition

The ArDVS API consists of three different APIs, see illustration 3.1. Each API is responsible to control different components of the ArDVS. The components are the eDVS4337 API, Bluetooth API and the pan-tilt API. The ArDVS library can be easily used by an Arduino project by including the ArDVS.h file.

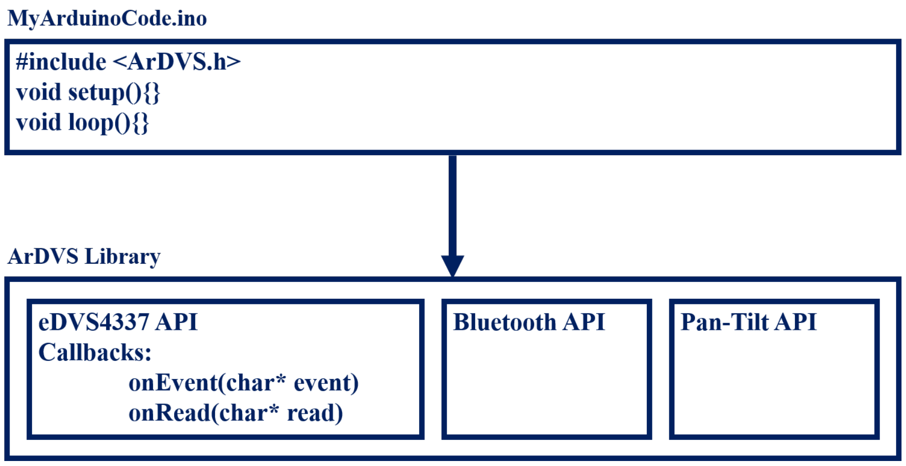


Figure 3.1: Contents of the ArDVS Library

Usability

The ArDVS library is designed to have easy control about the eDVS4337 module, the bluetooth module and the pan-tilt system. The library should fulfil the requirements from the Arduino Style Guide. It is necessary to assume that even a person can understand the library who has never programmed before. So a clear mental model of the concept and structure of the code is required. This is realized by avoiding nested structures and by using clear naming of the functions. The usage of the ArDVS library is planned to be simple as possible. The minimum code, which is required to get started with the ArDVS module should only be a few lines of code. All initialisation methods will be done in the constructor of the ArDVS library. So creating an object of the ArDVS library class is enough to set up the library.

eDVS4337 API

The eDVS4337 board, offers a lot of functions to the user. Via serial port it is possible to set and get parameters for motor driver, real time clock, baud rates and much more. The ArDVS library is designed to get simple access to the provided features of the eDVS module without entering any serial command. The ArDVS library offers a function called *writeToDvs()* in order to send commands to the eDVS module directly. But it is much more easier to use one of the provided functions of the ArDVS library which sends the required command over serial port in the background. And this is exactly what the Arduino Style Guide defines. In order to write an API according to the style guide, it is necessary to organize public functions around the data and functionality that the user wants. Very often, the command set for the eDVS module is overly complicated for the most common uses and can be hidden or re-organized with higher level functionality. So. for example, the ArDVS library API avoids setting register values, because this is a too unintuitive functionality of the eDVS module.

Bluetooth API

As mentioned in the hardware chapter, the Bluetooth module is a UART to Bluetooth converter. In order to set baud rates, name and other parameters of the Bluetooth module the ArDVS library offers some abstract functions, which do not require knowledge about the AT command set. That guarantees, that even an unexperienced user can change the Bluetooth name and serial settings.

Pan-tilt API

The pan tilt module is built with two micro servos. One servo controls the rotation around the vertical axis, the other servo controls the tilting. The ArDVS library offers one function to set the angle of rotation for both dimensions.

3.1.2 Implementation

The ArDVS library API is for controlling the ArDVS according to the Arduino Style Guide. Similar to most Arduino libraries, the ArDVS library consists of two files. One Cpp-file and one h-file. The h-file can be included by an Arduino project to get access to the ArDVS API. The most complex structures in the library are the callback functions, provided by the eDVS4337 API. In figure 3.2, there is a class

diagram, which illustrates the functionality of the callback function for events.
In order to read out the serial port of the eDVS module regularly, it is necessary to call the run() function of the ArDVS library in the loop() function of the Arduino project. Every time, when the run() function is called, the ArDVS library checks, if bytes are available at the DVS serial buffer, as depicted in 3.3.

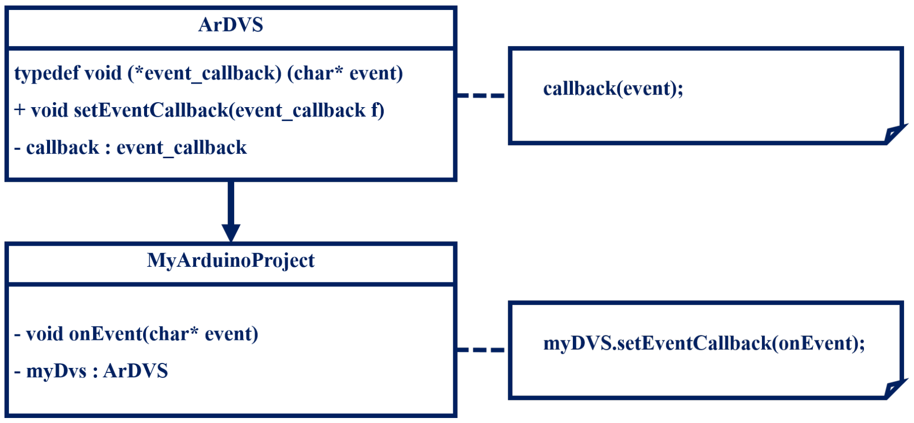

ArDVS
typedef void (*event_callback) (char* event)
+ void setEventCallback(event_callback f)
- callback : event_callback
callback(event);
MyArduinoProject
- void onEvent(char* event)
- myDvs : ArDVS
myDVS.setEventCallback(onEvent);

Figure 3.2: Class diagram section

3.1.3 Test

The maximum baud rate, which was achieved with a stable connection to the eDVS module via the standard Arduino API over the pins RX0 TX0, was 2,625,000 baud at the Arduino Due board. A higher baud rate results in an empty receiving buffer at the Arduino Due board. However, if the Arduino user applies filters and other algorithms on the event stream, this often results in an effectively slower baud rate, because the stream can not be processed that fast. Simple filters, similar to the DVS filter example, descrived in the next section has no influence on the effective baud rate.
In order to test the ArDVS library during development efficiently, it is necessary to have a fast implementation, compiling and testing cycle. In order to achieve this, we developed an Arduino project, which was developed for testing only. Using this project, most of the library functions can be called during runtime by serial commands, without recompiling and flashing the whole Arduino firmware. After implementing the callback functions for receiving the events, it was required to view the event stream of the DVS. With a bitmap representation, for example it is easy to see if an applied filter for the DVS events works correctly or not. Every user of the ArDVS library will need something like a video-view of the event stream from the DVS, in order to see if his project works correctly. Instead of learning the usage of jAER or Matlab, the Arduino user can use an Android application developed by us in order to view the eDVS output as a video. The Android application connects to the eDVS board automatically and can show the eDVS stream. The Android

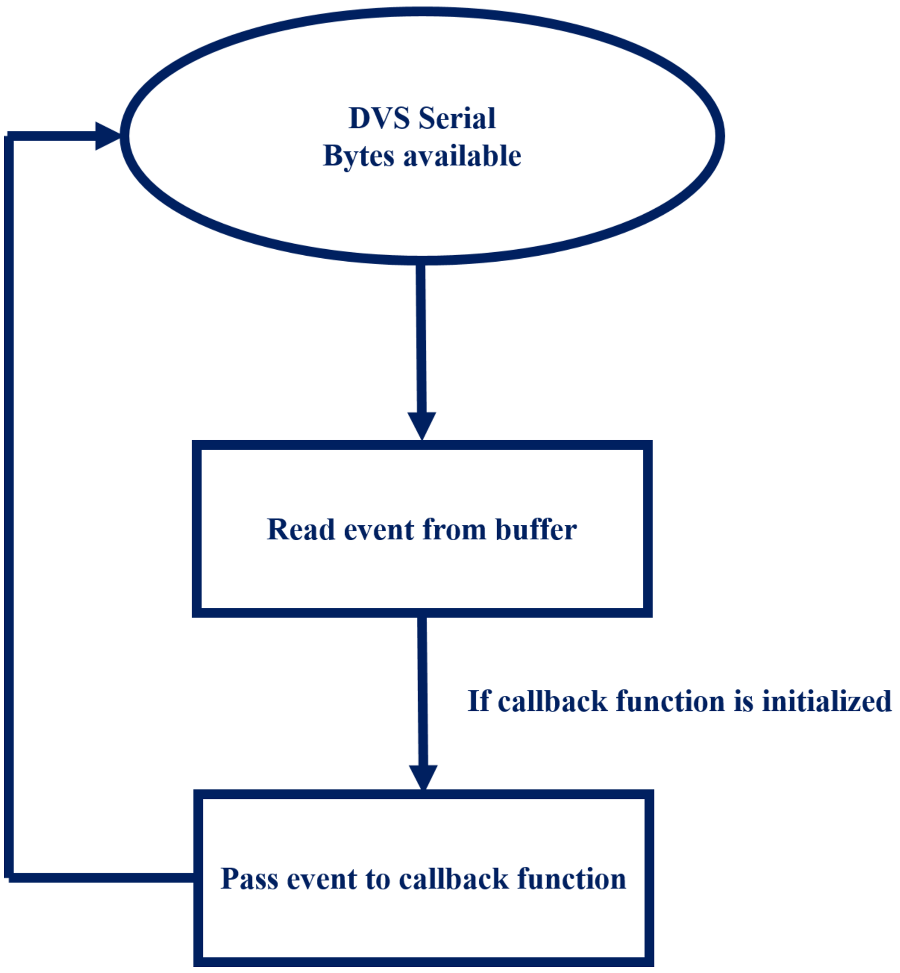

DVS Serial
Bytes available
Read event from buffer
If callback function is initialized
Pass event to callback function

Figure 3.3: Flowchart how the UART of the DVS module is read

user does not need any knowledge how to connect the ArDVS to the PC and how to view the events on Matlab or jAER. The maximum baud rate of the used Bluetooth module was only 230,400 baud because of financial reasons. Testing the Arduino project with higher baud rates up to 3 million baud is possible with the native USB port of the Arduino Due board.

3.1.4 Documentation

Similar to other Arduino libraries, the documentation of the library is available in HTML format. After a brief description, the user gets the download link and installation instructions. Installing the library is described in five steps:

1. Create a folder named ArDVS in you documents/Arduino/libraries folder.

2. Copy and paste the ArDVS.h and ArDVS.cpp under the ArDVS folder you just created.

3. Launch or restart the Arduino IDE to let it detect the new library.
4. Open one of the example codes (see below) to get started with the library.

The ArDVS library can be divided into different application sections. The most used section will be the API for setting up the ArDVS library.
The function *ArDVS::setupDvs(void)* must be called in the setup-body of the Arduino project. This is necessary to initialize the connection to the DVS module and the Bluetooth module.
In the Arduino project, two callback functions can be called from the ArDVS library. The first callback function is an event callback and the second one is an callback for incoming text messages from the DVS module. To set the event callback function, the method *ArDVS::setEventListener(event_callback f)* must be called. Setting up the callback function for incoming text messages from the DVS module is similar. The user must call the method *ArDVS::setRxListener(rx_callback f)*. More details about setting up and using the callback function can be found in the ArDVS library documentation.
The API for starting, stopping and parsing the event stream of the DVS is the core feature of the ArDVS library. The DVS data stream can be started by calling the method *ArDVS::startStream(void)*, and stopped by calling *ArDVS::stopStream(void)*. According to the Arduino Style Guide for libraries, it is recommended to offer an own function for each functionality instead of offering only one function with a few parameters. Parsing events is very easy. The Arduino user does not need any knowledge about the communication protocol of the DVS module. Instead of that he can call the following functions:

```
uint8_t ArDVS::extractXFromEvent(char* event)
uint8_t ArDVS::extractYFromEvent(char* event)
uint8_t ArDVS::extractPolarityFromEvent(char* event)
uint8_t ArDVS::extractTimestampFromEvent(char* event)
```

More details about the usage of the functions above can be found in the documentation of the library.
With these functions users can get direct access to the serial to the DVS module.

```
void ArDVS::print(String tx)
void ArDVS::print(int tx)
void ArDVS::print(char tx)
```

The user can write and flush bytes to the serial of the DVS module. In order to change UART settings, the user can call following functions:

```
void ArDVS::setUartEchoModeToNone(void)
void ArDVS::setUartEchoModeToCmdReply(void)
void ArDVS::setUartEchoModeToAll(void)
void ArDVS::setBaudRateOfDvsModule(int baudrate)
```

The naming of the functions is a trade-off between self-explaining and clarity, as required according to the Arduino API Style Guide.

The type of the timestamp of an DVS data event can be changed too. In order to do this, the Arduino user needs deep knowledge about data types and the DVS module itself. Because of this, the function are very well explained in the source code. In this report, the functions are listed, but more details about the timestamp format can be found on the support page of iniLabs. To set and get the current event type, following functions and methods can be called:

```
int ArDVS::getEventType(void)
void ArDVS::setEventTypeTimestamp0Bit(void)
void ArDVS:: setEventTypeTimestampVariBit(void)
void ArDVS:: setEventTypeTimestamp16Bit(void)
void ArDVS:: setEventTypeTimestamp24Bit(void)
void ArDVS:: setEventTypeTimestamp32Bit(void)
```

The ArDVS library additionally offers function to set the value of the current timestamp.
Really useful for ArDVS beginners and experts are the methods for setting the LED status. The library offers methods for turning the led off, on or keep it blinking.

The ArDVS library also offers methods for the built in Bluetooth module of the ArDVS. Calling this methods, the user can change the name of the Bluetooth module during discovery from a master device and set the baud rate of the UART-Bluetooth converter.

The DVS module comes with two motor drivers. The ArDVS library offers methods for enable and disable this motor drivers. Additionally, it is possible to set the PWM period and duty cycle of the motor driver in microseconds.

The built in real time clock of the DVS module can also be enabled, disabled, set and received by the functions offered by the ArDVS library.

The next chapter describes three Arduino example projects, which take advantage of the ArDVS library. After downloading the ArDVS library, this example projects will be included.

3.1.5 Example projects

The ArDVS library comes with three example projects: HelloDvs, AndroidDvs and FilterDvs. The complexity of every example project increases step by step. All example projects are described below.

HelloDvs

HelloDvs is the most simple example project, which is recommended to open with the Arduino IDE using the ArDVS module the first time. The next code snippet demonstrates how the example project looks like.

```
#include <ArDVS.h>


void onEvent(char* event);
ArDVS myDvs;

void setup() {
  //setup the ArDVS shield
  myDvs.setupDvs();

  //register Listener to receive events from the dvs
     module
  myDvs.setEventListener(onEvent);

  //start the event stream
  myDvs.startStream();
  //myDvs.setEventTypeTimestamp16Bit();
  //setup serial to communicate with the PC terminal
     emulator
  Serial3.begin(230400);
}

void loop() {
  myDvs.run();
}


//this function will be called, after receiving an event
   from DVS
void onEvent(char* event){

  //print the event to Serial in an intuitive format
  Serial3.print(myDvs.extractXFromEvent(event));
  Serial3.print(",");
  Serial3.print(myDvs.extractYFromEvent(event));
  Serial3.print(",");
  Serial3.print(myDvs.extractDensityFromEvent(event));
  Serial3.print(",");
  Serial3.print(myDvs.extractTimestampFromEvent(event));
  Serial3.println("");
}
```

All ArDVS projects need to include the ArDVS library in order to have access to the ArDVS class, defined in the library. In line 4 we declare our callback function, which will be called every time when a new event is received from the DVS module. In line 5 we create a ArDVS object, called myDvs. This object delcaration is always necessary in an ArDVS project. Declaring an ArDVS object causes a call to the constructor of the ArDVS class. In this constructor important default values will be set for the ArDVS library and we have created an object to get access to all public functions, available in the ArDVS library.
Every Arduino project has a setup() method. In this method we must call myDvs.setupDvs() in order to set up the serial port to the DVS module and to the bluetooth module.
In line 12 we set the callback function, which was previously declared at line 4. Now the ArDVS library is able to call our function. In line 15 we activate the event stream. That means, the eDVS will send the event data stream to the Arduino board. In line 18 we declare the baud rate of a free serial port, which is connected to a terminal emulator. For example Putty.
In line 22 we call the run() method of the ArDVS library. Calling this method causes a check for waiting events in the DVS serial buffer. Because of that we need to call the method periodically in the loop() method.
The last step is to initialize our callback function. This is the place where we can place code in order to manipulate the events, received from the DVS. In the following lines, we parse the received event with the built in functions from the DVS library and print them to the serial port of our choice in readable clear text format. The output could look like following:

```
1,12,0,24
1,15,1,26
4,127,1,28
```

Because of the simple complexity, it is recommended to open the ArduinoDvs example project after the HelloDvs project.

ArduinoDvs

The ArduinoDvs example consists of an Arduino project and a corresponding Android application. The Arduino part is very easy. Instead of parsing and printing the received events to a terminal emulator like the HelloDVS example does, the events are forwarded to the bluetooth module via the command *bluetoothSerial.print(event);*.
After that, the Android application is able to receive the events and to display them. More details about the Android application are given in the next section.

The ArduinoDvs project offers more features. If the event stream is disabled, it is possible to write and read to the serial port of the DVS module with an integrated terminal emulator in the Android application. A further feature is the possibility to call most of the ArDVS library functions and methods from the Android app too, during runtime. This is possible in the preference page of the Android application. The best feature of the Android App is that is always reconnects automatically to the ArDVS board and is able to show the DVS event data stream as live-stream as a 128 times 128 pixel pixmap.

FilterDvs

The FilterDvs example shows a simple filter, applied to the event data stream of the DVS. In this example project a rectangle of 40 times 40 pixels is filtered and forwarded to the serial port.

3.2 Android application

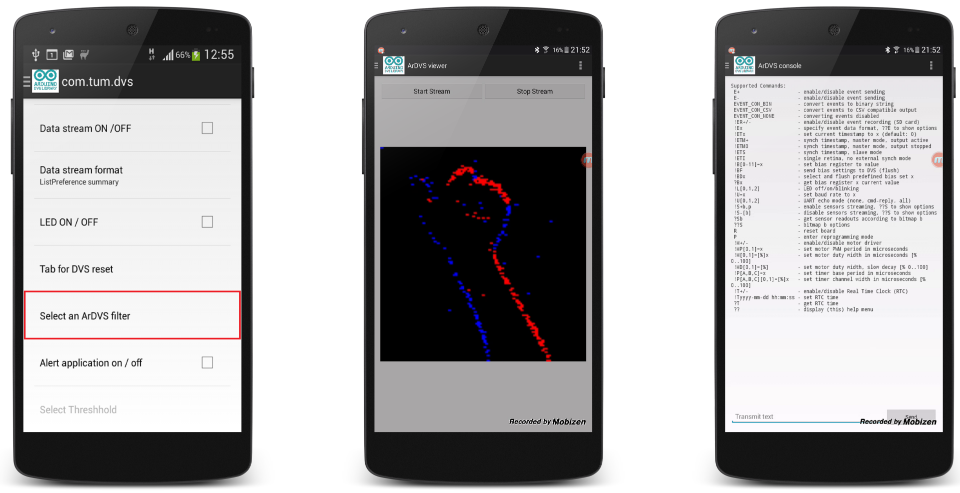
In this practical course, we developed an Android application, which is able to connect to the ArDVS via Bluetooth. This Android application can be used in combination with the example project, called AndroidDvs. The main purpose of this application is, that the android user can view the event data stream without installing software on his computer. The next two sections describe the definition and implementation phase of the Android application during the practical course in short.

3.2.1 Definition

The Android app is designed to be an easy-to-use alternative to the existing solutions to show the raw DVS and ArDVS data stream. The screenshots in figure 3.4 show the Android application HelloArDVSLibrary, available at the Play Store. The source code is also available in a bitbucket git-repository.

The left screenshot shows the preference section of the HelloArDVS application. The screenshot in the middle shows the live-view of an active DVS event data stream. The right screenshot shows the built in terminal emulator to the ArDVS.

The Android application offers a terminal emulator for debugging and offers a possibility to send and receive data directly to the DVS module. With the Android App, several functions of the library can be called during runtime. The most useful feature of the Android App is the automatically reconnection to the ArDVS board and the feature to show the ArDVS output as live-stream as a 128 times 128 pixel pixmap. The navigation through the menu is very simple. The block diagram in



```
Supported Commands:
 E+                   - enable/disable event sending
 E-                   - enable/disable event sending
 EVENT_CON_BIN        - convert events to binary string
 EVENT_CON_CSV        - convert events to CSV compatible output
 EVENT_CON_NONE       - converting events disabled
 !ER+/-               - enable/disable event recording (SD card)
 !Ex                  - specify event data format, ??E to show options
 !ETx                 - set current timestamp to x (default: 0)
 !ETM+                - synch timestamp, master mode, output active
 !ETM0                - synch timestamp, master mode, output stopped
 !ETS                 - synch timestamp, slave mode
 !ETI                 - single retina, no external synch mode
 !B[0-11]=x           - set bias register to value
 !BF                  - send bias settings to DVS (flush)
 !BDx                 - select and flush predefined bias set x
 ?Bx                  - get bias register x current value
 !L[0,1,2]            - LED off/on/blinking
 !U=x                 - set baud rate to x
 !U[0,1,2]            - UART echo mode (none, cmd-reply, all)
 !S+b,p               - enable sensors streaming, ??S to show options
 !S-[b]               - disable sensors streaming, ??S to show options
 ?Sb                  - get sensor readouts according to bitmap b
 ??S                  - bitmap b options
 R                    - reset board
 P                    - enter reprogramming mode
 !M+/-                - enable/disable motor driver
 !MP[0,1]=x           - set motor PWM period in microseconds
 !M[0,1]=[%]x         - set motor duty width in microseconds [%
0..100]
 !MD[0,1]=[%]         - set motor duty width, slow decay [% 0..100]
 !P[A,B,C]=x          - set timer base period in microseconds
 !P[A,B,C][0,1]=[%]x  - set timer channel width in microseconds [%
0..100]
 !T+/-                - enable/disable Real Time Clock (RTC)
 !Tyyyy-mm-dd hh:mm:ss - set RTC time
 ?T                   - get RTC time
 ??                   - display (this) help menu
```

Figure 3.4: Screenshots of the HelloArDVS Android application

figure 3.5 shows the navigation hierarchy of the Android app.

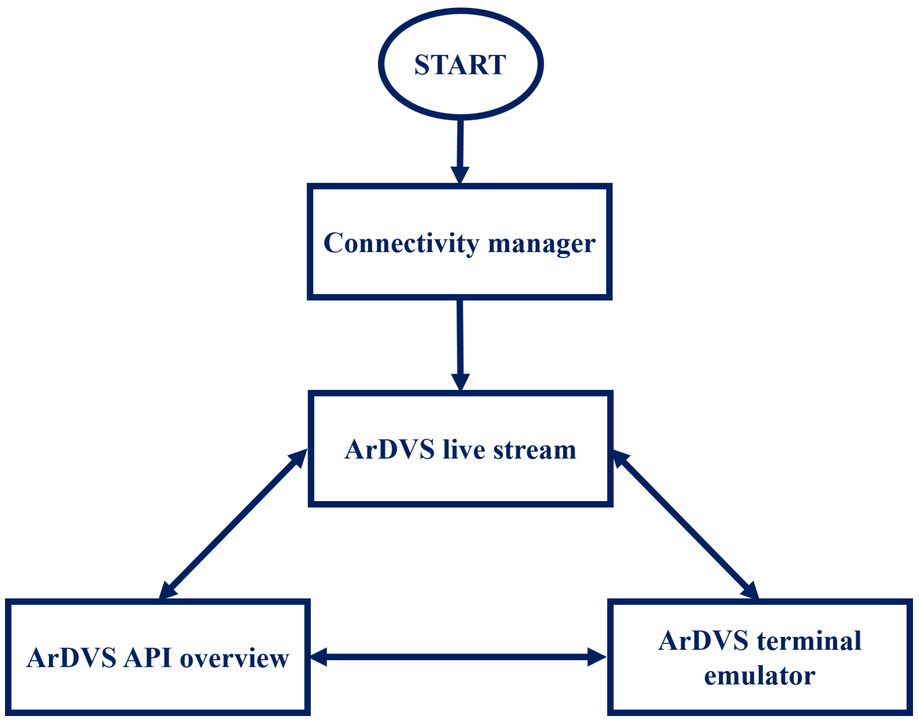


Figure 3.5: Navigation hierarchy of the HelloArDVS Android application

For good usability, the connectivity manager starts immediately after launching the

application. The application scans for bluetooth devices and the user can choose the correct one. Once connected, the application reconnects automatically to the ArDVS.
After that, the user can switch between three windows. The ArDVS live stream, where the user can activate, stop and watch the event data stream forwarded by the ArDVS. This is very useful, because in this way, the user can take a look, which effect their algorithms have to the event data stream without any knowledge about third party software.

3.2.2 Implementation

The Android application consists of two activities and one service. More detailed information about the Android API can be found in the API Guides of the official Android developers page. Block diagram 3.6 shows the relationship between the classes of the Android app.

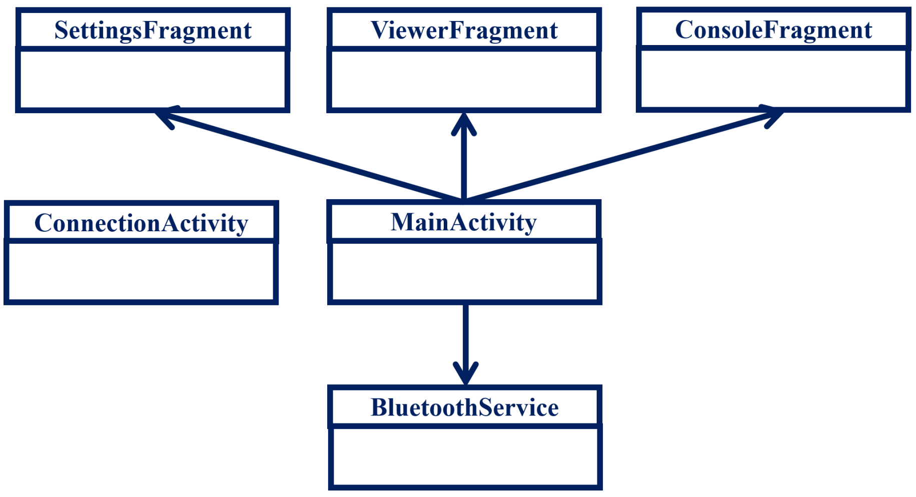


Figure 3.6: Including hierarchy of the classes of the Android application

SettingsFragment, ViewFragment and ConsoleFragment inherit from the class Fragment. These classes are included by the MainActivity. In this Android application, the MainActivity displays the main menu. The ConnectionActivity corresponds with the BluetoothManager in the hierarchy diagram. The BluetoothService is a background service and handles outgoing and incoming messages over Bluetooth. The Bluetooth Service uses the BluetoothSocket class from the Android API. This class provides functions for Bluetooth Serial Port Profile (SPP), which is an easy to use API in order to communicate with the ArDVS library. The Implementation part in this report is kept very short and simple. For more details please refer to the source code of the Android app and the well documented Android API.

Chapter 4

Application: Event based photography

Besides the examples, which are part of the ArDVS library, a full exemplary use-case of the ArDVS was implemented to comprehend the library's potential.

4.1 Motivation

A demo application serves as a tangible illustration of the ArDVS' capabilities. A great demonstrator of the DVS low latency event capturing in combination with the library's customizable filters is an event based camera trigger, pictured in figure 4.1. The focus was on creating a useful application that is not to difficult to grasp and to implement by the Arduino user as a first step to familiarize with the ArDVS library.

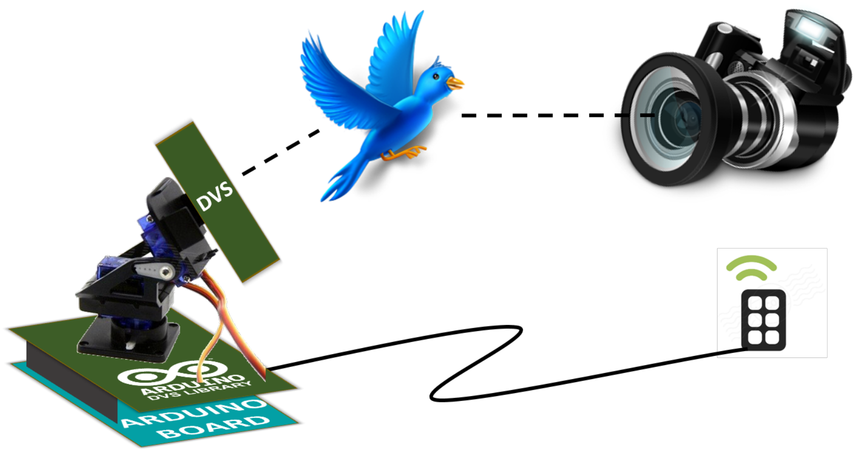


Figure 4.1: Visualization of demo application

4.2 Components

The main components used for this use-case is the Arduino Due with the ArDVS shield, an infrared LED connected to the shield, an Android smartphone with the ArDVS app for configuration as well as a Canon 6D single-lens reflex camera on a tripod.

4.3 Filters

The filter for selecting the region of interest is similar to the filterDvs example. The following code shows how the filter is implemented.
If the event is in the region of interest, the eventCounter increases. If an event is older than 100 milliseconds, the eventCounter is decreased again. If the eventCounter reaches a selectable threshold, then the function for shooting a photo will be called.

```
void onEvent(char* event){

  //filter all events, which between xStart, xEnd, yStart
     , yEnd
  if(myDvs.extractXFromEvent(event)>xStart  &&
                  myDvs.extractXFromEvent(event)<xEnd &&
                  myDvs.extractYFromEvent(event)>yStart
                     &&
                  myDvs.extractYFromEvent(event)<yEnd ){

          onEventCount(myDvs.extractTimestampFromEvent(
             event));

  }
}
```

4.4 Camera remote control

The camera is remotely triggered by infrared light. Most single-lens reflex cameras possess a remote control photo sensor with a simple burst based protocol. The application makes use of an open-source library that supports the different timing requirements for the manufacturers Canon, Nikon, Sony, Olympus, Minolta und Pentax.
For activating the shutter on a Canon camera there are 16 pulses with a duty cycle of 50% and a period of 22 microseconds necessary, followed by the same sequence of pulses after a delay of 7330 microseconds, as shown in figure 4.2.

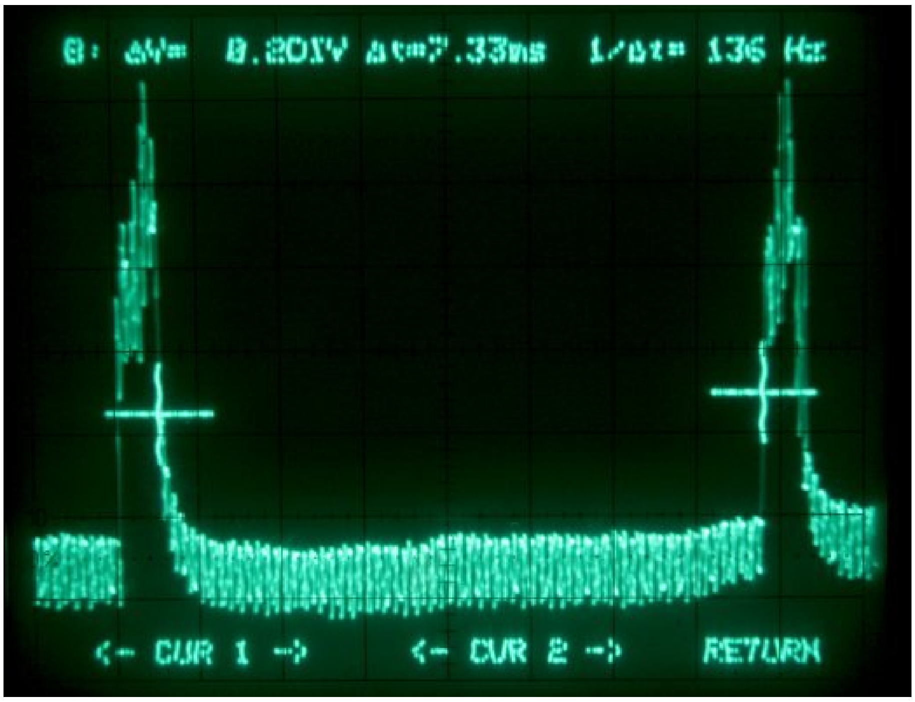


Figure 4.2: Graph of the two bursts required for taking a photo[Luk15]

4.5 Implementation

In this demo setup we adjust the direction of the DVS using the pan-tilt-system and point the LED towards the front of a Canon 6D where the remote sensor is located. The DVS points in the direction of interest and in the Android app the regions of interest are manually selected. Further, the Canon 6D is adjusted on the tripod for the desired framing of the image as well as the settings like shutter speed, aperture and exposure. When an object enters the region of interest, e.g. a bird leaving its hiding place, a selectable output action is triggered. In this use-case the output action is a burst by the infrared LED that serves as a remote control for the camera. The result is a crisp picture of the bird leaving its hideout in the desired location of the picture like in the sample picture in figure 4.3.

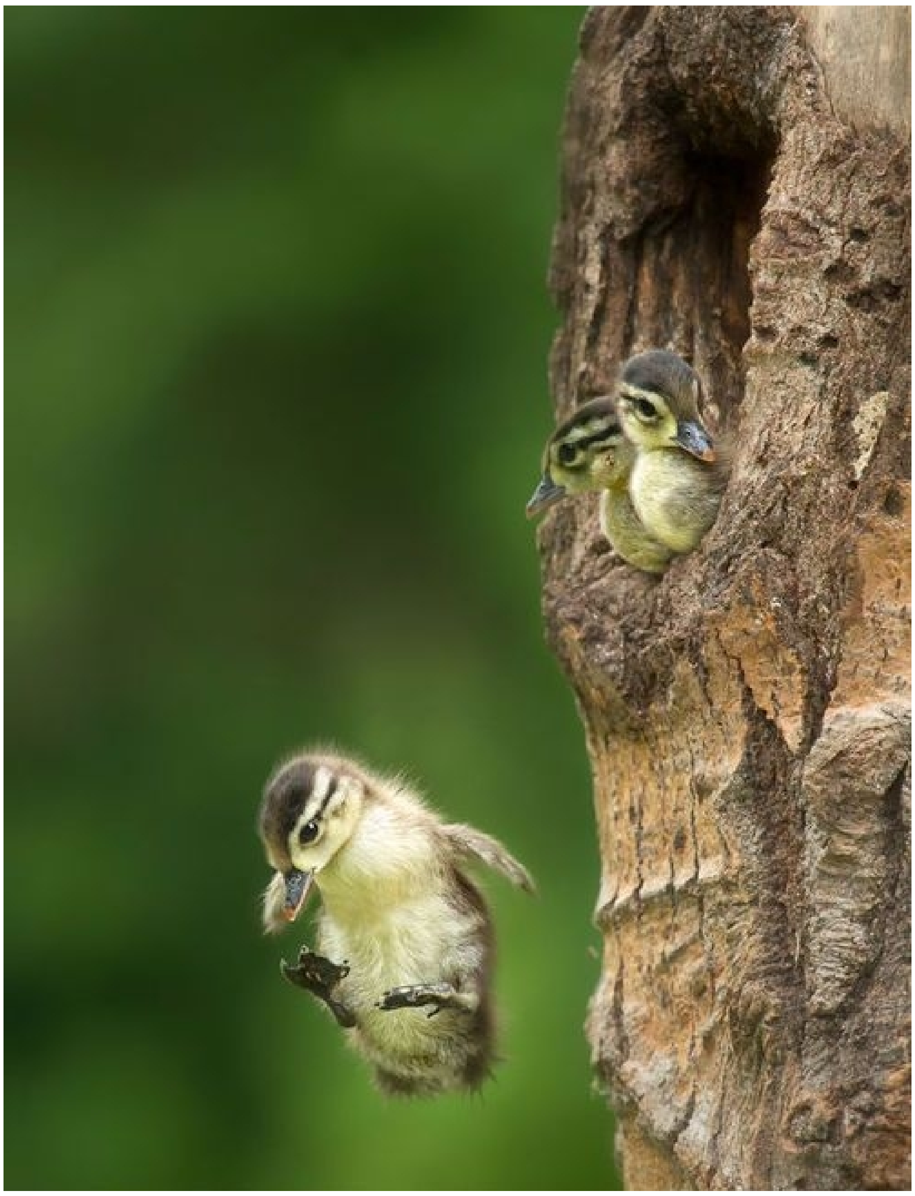


Figure 4.3: Sample image for event based photography[Ama15]

Chapter 5

Conclusion

Processing live image data with an Arduino is very rare until now. The experience from this practical course shows, that processing the event data stream from a DVS is possible with low computational power. Additionally, we developed the ArDVS development kit, which can be used by the Arduino community.
Equipped with an ArDVS and an Android phone the typical Arduino users with very limited programming skills have everything they need to get started with the DVS. The implemented example projects and demo application demonstrate, how simple it is to realize an own project using a DVS. The Arduino IDE can be used to implement simple algorithms for the DVS after including the ArDVS library. The Android phone is used to show the ArDVS output only by downloading an Android application.
There are still possibilities to extend the existing ArDVS library together with the Android application. The most important functions for starting, stopping and parsing the DVS data stream are already implemented. Now it is up to the creativity of the Arduino community to come up with new features and exciting applications.

List of Figures

Bibliography

[4tr15] 4tronix. 2 DOF Pan and Tilt, 2015. URL: http://4tronix.co.uk/store/index.php?rt=product/product&product_id=232.

[Ama15] AmarnaZenter. Photobucket, 2015. URL: http://s1028.photobucket.com/user/AmarnaZenter/media/1601500_662617957138046_1104535053_n_zps890a1052.jpg.html.

[ard15] Fritzing Boards, 2015. URL: http://fritzing.googlecode.com/svn/trunk/fritzing/parts/svg/core/breadboard/Arduino_DUE_V02b_breadboard.svg.

[ini15] iniLabs. eDVS user guide, 2015. URL: http://www.inilabs.com/support/edvs.

[Luk15] Luk. Canon RC-1 IR remote control reverse-engineered, 2015. URL: http://www.doc-diy.net/photo/rc-1_hacked/.

[Pas15] Pascal Romeyer. Bluetooth CAT System ou Data, 2015. URL: http://f4cvm.free.fr/realisation/connexion/bluetooth/data/cat.htm.

License

This work is licensed under the Creative Commons Attribution 3.0 Germany License. To view a copy of this license, visit http://creativecommons.org or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.