TECHNISCHE
UNIVERSITÄT
DARMSTADT

INSTITUT FÜR
SPORT
WISSENSCHAFT

FVF

FVF Documentation Documentation

Release 1

Thomas Gossmann

August 17, 2015

Contents

The Flicker Fusion Frequency (FVF) is used to measure central nervous activation. This is the technical documentation for the measurement system. Reach for the manual on how to run your own tests.

The FVF measurement system is created at the Institute for Sport Science at Technical University Darmstadt, see the Authors.

Table of contents:

CHAPTER 1

Authors

The FVF measurement system is built by:

Project lead:

- Prof. Dr. Josef Wiemeyer

Project members:

- Leonie Poetsch
- Gerrit Kollegger
- Thomas Gossmann

CHAPTER 2

License

CHAPTER 3

Contribute

There are a couple of ways to contribute:

1. Create an issue on github, if you realized something is wrong
2. You may clone the repo and send a pull-request which contains the fix
3. Get in contact with the team

CHAPTER 4

Structure

The FVF measurement system is split into various pieces and components.

4.1 Pieces

The FVF measurement system consists of multiple pieces:

- Tube
- Hardware LED controller (Arduino Uno)
- Client Software

4.2 Components

The software consists of multiple components written in multiple programming languages:

- Arduino Firmware (C++)
- LED Driver (Java)
- Client Software (Java)

4.3 Folders

The folders and what they contain in this repository:

- `docs/` - contains the source files for this documentation
- `driver/` - contains the source files for the Driver
- `firmware/` - contains the sources files for the Firmware
- `software/` - contains the sources files for the Software
- `manual/` - contains the sources files for the manual

CHAPTER 5

Firmware

The firmware runs on the **Arduino Uno** board. The board is connected via an Universal Serial Port (USB) to the host computer which runs the measurement software. It's main job is to handle incoming commands (via serial port) and send back feedback notifications.

5.1 Available Commands

- *on*
- *flicker*
- *off*
- *measurement*
- *ping*

Detailed information about the input and output about the firmware is described in the LED Protocol.

5.2 References

- Arduino Website
- Arudino IDE
- Arduino API

5.3 Verification

To ensure the flickering frequency a verification measurement has been done with VOLTCRAFT Universal SYSTEM MS-9150 Frequency Counter.

An important Note: The delay() method on the Arduino passes the delay in integer values, no floats are possible. For every milliseconds below 16383, delayMicroseconds() must be used. The gap happens between 30Hz and 31Hz.

5.3.1 Methodology

Voltage has been captured at the pins directly at the measured LED. Each frequency was measured two times and the latter value was used.

Note: Prior sample measurements showed, the value didn't changed after the second measurement for each frequency.

5.3.2 Results

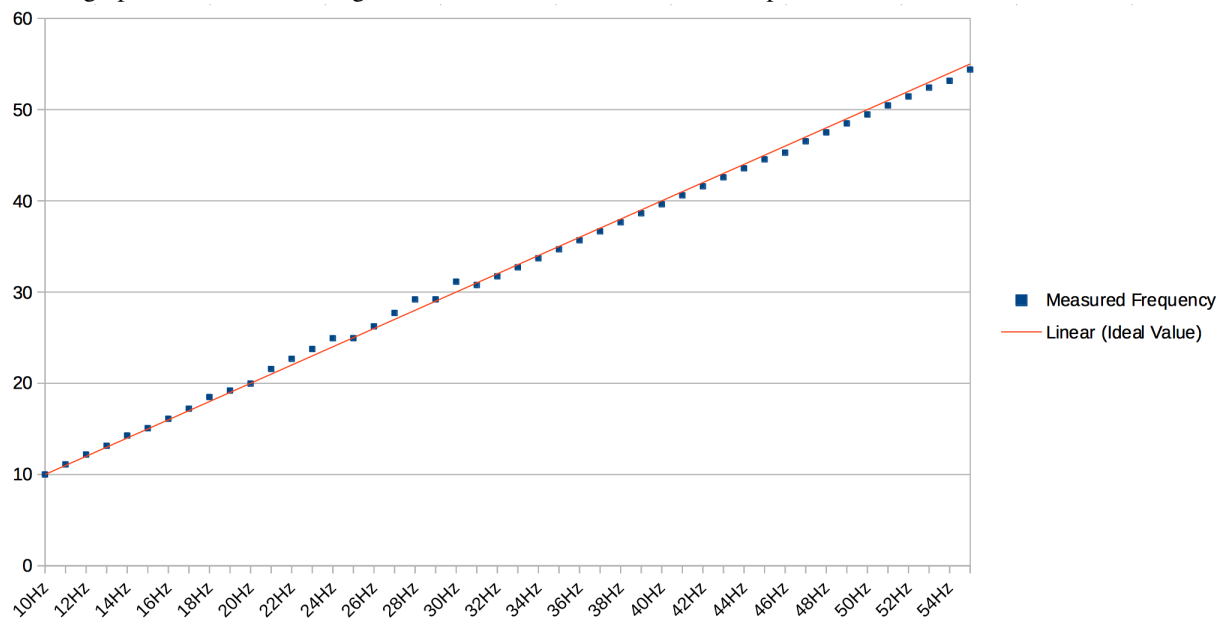
Frequency	Measured Frequency	Offset
10Hz	9,996	-0,004
11Hz	11,105	+0,105
12Hz	12,186	+0,185
13Hz	13,146	+0,146
14Hz	14,270	+0,270
15Hz	15,071	+0,071
16Hz	16,110	+0,110
17Hz	17,220	+0,220
18Hz	18,488	+0,488
19Hz	19,200	+0,200
20Hz	19,966	-0,034
21Hz	21,567	+0,567
22Hz	22,678	+0,678
23Hz	23,756	+0,756
24Hz	24,935	+0,935
25Hz	24,941	-0,059
26Hz	26,245	+0,245
27Hz	27,704	+0,704
28Hz	29,191	+1,191
29Hz	29,196	+0,196
30Hz	31,138	+1,138
31Hz	30,772	-0,328
32Hz	31,723	-0,277
33Hz	32,712	-0,288
34Hz	33,702	-0,298
35Hz	34,689	-0,311
36Hz	35,673	-0,327
37Hz	36,657	-0,343
38Hz	37,646	-0,354
39Hz	38,633	-0,367
40Hz	39,621	-0,379
41Hz	40,598	-0,402
42Hz	41,590	-0,41
43Hz	42,578	-0,422
44Hz	43,562	-0,438
45Hz	44,544	-0,456
46Hz	45,275	-0,725
47Hz	46,520	-0,48
48Hz	47,500	-0,5
49Hz	48,481	-0,519

Continued on next page

Table 5.1 – continued from previous page

Frequency	Measured Frequency	Offset
50Hz	49,465	-0,536
51Hz	50,454	-0,546
52Hz	51,436	-0,564
53Hz	52,414	-0,586
54Hz	53,152	-0,848
55Hz	54,389	-0,611
...		
500Hz	468,991	-32,009

The next graph shows the scattering of the measured values around the expected linear ideal values.



CHAPTER 6

Driver

The LED driver is a Java API to send commands to the Firmware and getting notified about feedback from the firmware.

6.1 Communication

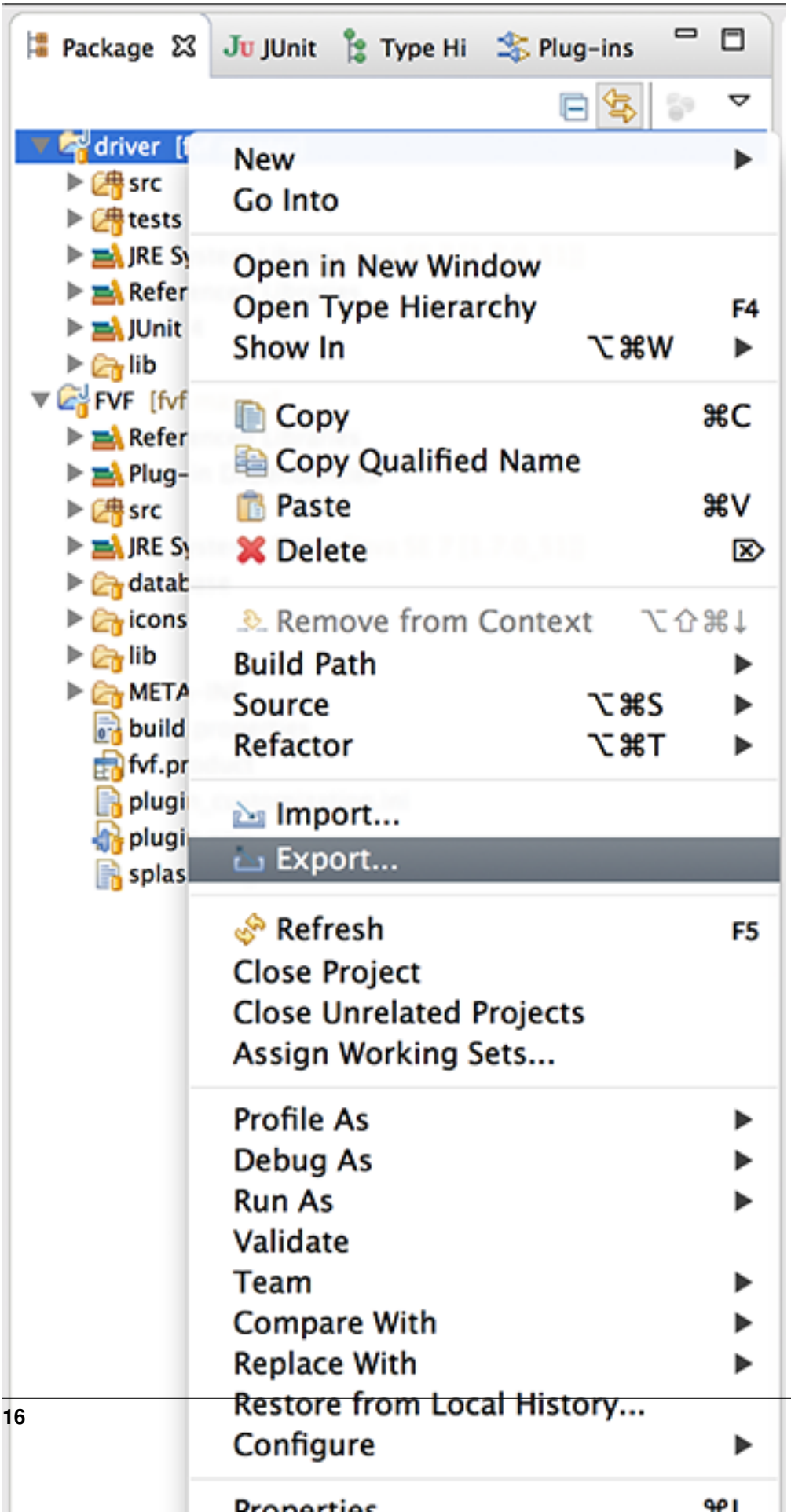
The client software talks to the Arduino board through a serial port connection. For this purpose the RXTX interface is used. In order to check whether the connection is still established, the driver periodically sends a ping to the board. Once this connection is interrupted for several reasons, it's assumed the connection is dead.

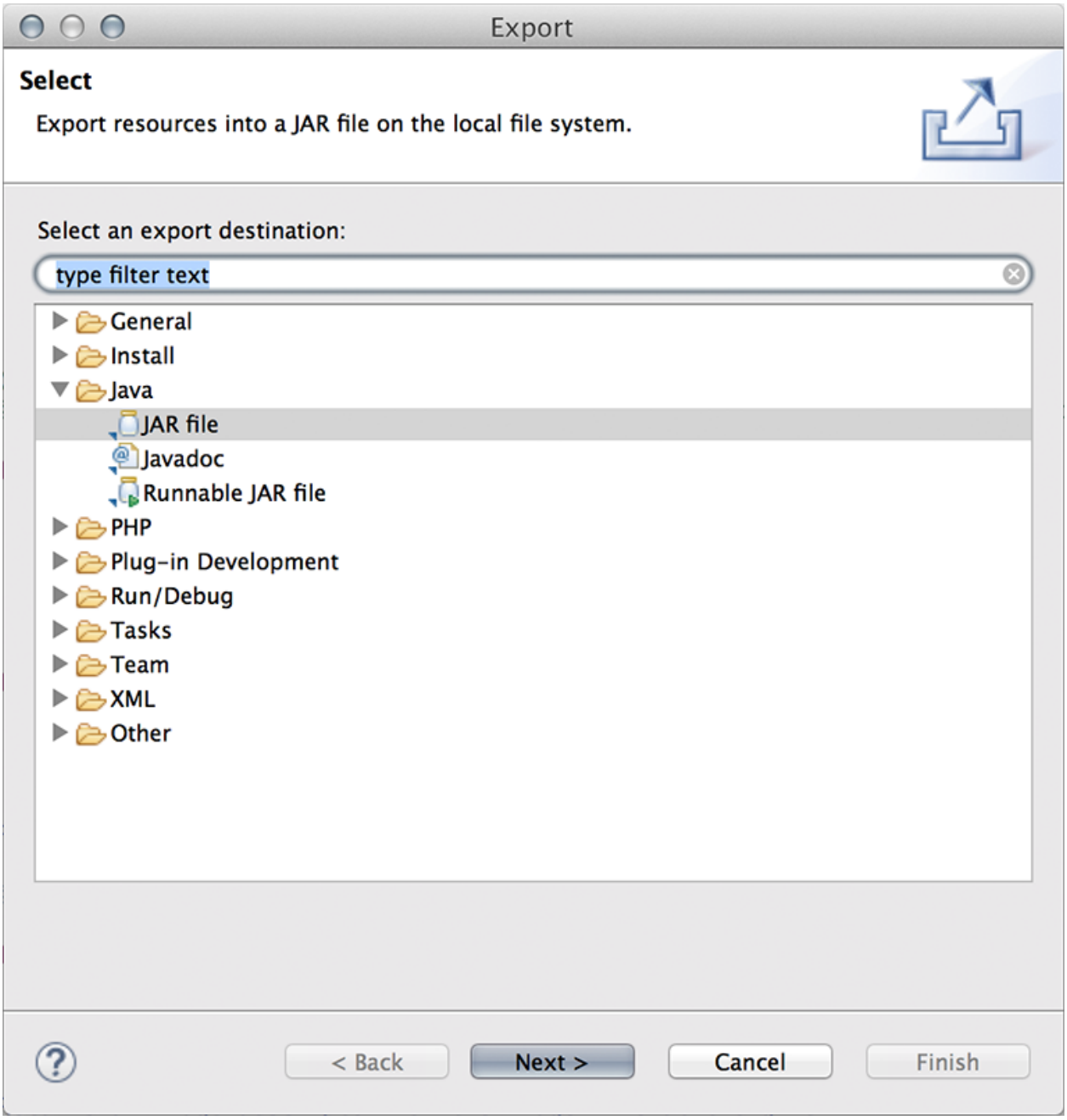
6.2 Protocol

The driver implements the LED Protocol.

6.3 Deployment

The driver is deployed as `*.jar` file into the softwares `lib/` folder.


Package
JUnit
Type Hi
Plug-ins
driver
src
tests
JRE Sy
Refer
JUnit
lib
FVF
Refer
Plug-
src
JRE Sy
datab
icons
lib
META
build
fvf.pr
plugi
plugi
splas
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In ⌥⌘W
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Remove from Context ⌥⇧⌘↓
Build Path
Source ⌥⌘S
Refactor ⌥⌘T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Profile As
Debug As
Run As
Validate
Team
Compare With
Replace With
Restore from Local History...
Configure



Note: This is not ideal and must be triggered manually. Better solutions are welcome.

6.4 Remarks

There is a rxtx wiki entry describing how to bundle the jni extension with an eclipse rcp application.

6.4.1 Mac OS X

The normal distributed `librxtxSerial.jnilib` is only in 32-bit mode which doesn't match a 64-bit processor architecture and can thus not be autoloaded. See here:

```
$ file librxtxSerial.jnilib
librxtxSerial.jnilib: Mach-O universal binary with 2 architectures
librxtxSerial.jnilib (for architecture ppc):  Mach-O dynamically linked shared library ppc
librxtxSerial.jnilib (for architecture i386): Mach-O dynamically linked shared library i386
```

Luckily there is a 64-bit version available, forged by Robert Harder. The eclipse plugin distributes the 64-bit version mentioned here:

```
$ file librxtxSerial.jnilib
librxtxSerial.jnilib: Mach-O universal binary with 4 architectures
librxtxSerial.jnilib (for architecture x86_64):       Mach-O 64-bit bundle x86_64
librxtxSerial.jnilib (for architecture i386): Mach-O bundle i386
librxtxSerial.jnilib (for architecture ppc7400):      Mach-O bundle ppc
librxtxSerial.jnilib (for architecture ppc64):        Mach-O 64-bit bundle ppc64
```

6.4.2 Disconnecting

There are some problems between RXTX and properly disconnecting connections. The problems are described in a forum thread. The mentioned hacks are implemented precautionally. Probably RXTX version 2.2 should have these issues resolved, yet wasn't available stable at the time of implementation.

6.5 References

- RXTX
- Arduino Java Interface

CHAPTER 7

Software

The heart of the measurement system is the software, which is responsible for managing your probands, running tests and browsing the results. It is an Eclipse RCP application. It uses the eclipse e3 API. Latest API docs are available at Eclipse Help.

7.1 Database

Database development is realized via sormula ORM. The models are placed in the `de.tu_darmstadt.sport.fvf.model` package. The respective ERM is available as appendix. It is a SQLite database which is realized with SQLJet and connected with SQLite JDBC.

7.1.1 Migrations

Further version might require a migration of the underlying database. The mechanics for this are already implemented and ready to use. SQLJet allows to stores a user version number along with the database file. The purpose is to read this number at start and run a migration if necessary. The class `de.tu_darmstadt.sport.fvf.database.DatabaseLoader` handles this logic. The required code to read the version number is already available in the `initialize()` method, yet commented out but provides good start.

7.2 Icons

Icons are Silk by famfamfam and Fugue by Yusuke Kamiyamane.

CHAPTER 8

Setup

Learn how to setup your development environment for FVF.

8.1 Installing Arduino

Installing the Arduino IDE is straight forward. From the Arduino Website download the Arduino IDE for your platform and open `firmware/fvf/fvf.ino` to start your firmware development.

8.1.1 Arduino Drivers

Some systems require a manual driver installation for the Arduino board. Please refer to the Getting Started Guide from the Arduino website if this is required for you and how to get this done.

8.1.2 CoolTerm

CoolTerm is a simple serial port terminal application. CoolTerm can be used to send commands to the Arduino Board and test your firmware.

8.2 Install Git

Git is used as VCS and GitHub as repository master.

8.2.1 Windows

Luckily GitHub provides an application with GUI to access git repositories. Download GitHub for Windows and install it; Clone the repo from GitHub and you are ready to go.

8.2.2 Mac

Also Mac got a GitHub app with GUI to access git repositories. Download GitHub for Mac and install it; Clone the repo from GitHub and you are ready to go.

8.2.3 Linux

You are on Linux, you know how to use your personal package manager to install yourself a git package and of course you can handle it from your favorite shell.

8.3 Installing Eclipse

Eclipse is the main development environment. A good start is to download the Eclipse for RCP and RAP Developers package.

8.3.1 Install PDE Tools

To help and assist you with programming (Javadoc + proper code completion), install the following plugins from "The Eclipse Project and Updates" update site (Help > Install New Software ... Update Site: http://download.eclipse.org/eclipse/updates/4.4 - replace "4.4" with the current version number):

- Eclipse Plug-In Development Environment
- Eclipse Platform SDK
- Eclipse Java Development Tools

Note: Some of them might already be installed.

8.3.2 Install Deployment Tools

To deploy the FVF application bundle to multiple platforms the eclipse "DeltaPack" is required for this. Read here for installation: https://stackoverflow.com/a/12737382/483492

8.3.3 Install Optional Tools

There are more useful plug-ins to support your development. They are available via the current releases update site (Help > Install New Software ... Update Site: http://download.eclipse.org/releases/luna - replace "luna" with the current release):

- SWT Designer
- Eclipse GIT Team provider

CHAPTER 9

Deployment

This page describes, how to deploy your own version of the measurement software.

9.1 Required Plugins

Since this is an e3 Plug-In deployed in an e4 environment all required compatibility layer plugins must be added.

9.2 Export the RCP application

Open the `fvf.product` file in eclipse. On the *Overview* page there is the export section with a link to open the "Eclipse Product export wizard" (which is also available from the toolbar of this editor). Make sure to check "Export for multiple platforms" (which is only available if you followed the *Install Deployment Tools* instructions) and "Synchronize before exporting". Click "Next" which shows the available platforms to deploy to.

CHAPTER 10

Documentation

The process of writing documentation is also known as *continuous documentation*. The raw source files are written in reStructuredText and Sphinx is used to generate the documentation in various formats. Read the Docs hosts this documentation.

CHAPTER 11

Follow-Up Projects

Some ideas for follow-up projects.

11.1 Automated Builds

Currently deploying the software is a manual job. It would be more pleasant to have automated builds. This especially means two tasks:

1. Driver and FVF are two independent projects right now, which means deploying the driver first to use it from FVF, it would be way easier to just refer to the driver project instead.
2. Deploy the software (with all it's required libs). Either automatically, by committing, tagging a release or trigger the build manually.

For option #2 (continuous deployment) there are some online services available, which must be checked individually if they are eligible for the task on hand:

- semaphore
- codeship
- dploy
- drone.io
- circleci

Additionally, there must be found a good place to distribute binaries to.

11.2 Streamline Web Presence

The current web presence is cluttered among this docs and the manual. A formal webpage introducing FVF, what it is, who is responsible for that, where to download, how to contribute and contains links to the docs and manual is missing. Probably GitHub pages are a solution to island this or possibly put this up on the IFS website.

11.3 Internationalization (i18n)

Right now, the docs are in english, software and manual are in german. All tools within the toolchain support internationalization. This can be used to better translate all occurring strings. Transifex is an online service to keep track of

all translations, which can be used as a managing instance. There is even an integration between Transifex and Sphinx, to translate docs.

11.4 Self-Validation

A self-validation routine built into the Firmware that averages the deviation for each frequency and applies it during the measurement routine.

11.5 Post-Processing of Results

The results can receive some post-processing by either showing statistics and displaying graphs or providing exports to various formats for further processing, e.g. exporting to SPS.

11.6 Instructions to setup your own FVF measurement system

Instructions to setup one's own FVF measurement system. With technical specifications of the tube and the oculus adapter to connecting the software. Likewise a step-by-step manual for a self-construction-kit.

11.7 Port to eclipse e4

For historical reasons, the software is built on eclipse e3 API. At the time of writing, e4 is the current API and contains modern programming approaches to simply development. It can be worth to port the codebase to the new e4 API.

CHAPTER 12

LED Protocol

The firmware communicates with the protocol described here. The driver sends a command with arguments (*Input*) and the hardware send feedback on what is happening (*Output*).

12.1 Schema

The following schema is used for every command, in every direction:

```
cmd [arg1] [arg2] ... [argN]
```

12.2 Input

Commands send to the hardware

12.2.1 on

Turns on a led:

```
on [led]
```

Arguments

- `led` (int) - the led number

12.2.2 flicker

Flickers a led:

```
flicker [led] [frequency] [duration] [[light]] [[dark]]
```

Arguments

- `led` (int) - the led number
- `frequency` (int) - the flicker frequency [hz]
- `duration` (int) - the duration the led flickers [ms]
- `light` (int) - the light part of the light/dark ratio (optional)

- `dark` (int) - the dark part of the light/dark ratio (optional)

12.2.3 off

Turns off a led:

```
off [led]
```

Arguments

- `led` (int) - the led number

12.2.4 measurement

Runs a measurement sequence:

```
measurement [mode] [flickerLed] [frequency] [onDuration] [offDuration]
```

Arguments

- `mode` (int) - *2* for two leds and *4* for four leds
- `flickerLed` (int) - Which led will flicker
- `frequency` (float) - The frequency for the flickering led [hz]
- `onDuration` (int) - The duration, the leds will be on [ms]
- `offDuration` (int) - The duration, the leds will be off [ms]

12.2.5 ping

Sends a ping:

```
ping [[seq]]
```

Arguments

- `seq` (misc) - An optional sequence identifier (optional)

12.3 Output

Feedback received from the hardware.

12.3.1 on

Send when a led is turned on (or flickering, when measurement command was used):

```
on [led]
```

Arguments

- `led` (int) - the led number

12.3.2 flicker

Send when a led is flickering:

```
flicker [led]
```

Arguments

- `led` (int) - the led number

12.3.3 off

Send when a led is turned off:

```
off [led]
```

Arguments

- `led` (int) - the led number

12.3.4 measurement

Send when a measurement is started or finished:

```
measurement [state]
```

Arguments

- `state` (string) - `on` when the measurement started and `off` when it's finished

12.3.5 ons

Send when more than one led is turned on (during a measurement):

```
ons [mode]
```

Arguments

- `mode` (int) - is either 2 or 4, depending the first value of the `measurement` input command

12.3.6 offs

Send when more than one led is turned off (during a measurement):

```
offs [mode]
```

Arguments

- `mode` (int) - is either 2 or 4, depending the first value of the `measurement` input command

12.3.7 pong

Answers a ping with a pong:

```
pong [seq]
```

Arguments

- `seq` (misc) - The returned sequence identifier (optional)

12.3.8 error

Send when an error occured:

```
error [number]
```

Arguments

- `number` (int) - the error number (see below)

12.4 Error Codes

Error code explanation:

- *0* - Unknown Error
- *1* - Malformed command
- *2* - Unknown command
- *3* - Too few arguments for flicker command

12.5 Troubleshooting

1. Getting a "Port in Use" exception on OSX, when connecting to the Arduino Board -> See here: https://marcosc.com/2011/10/arduino-java-error-serial-port-already-in-use/

CHAPTER 13

ERM

The softwares database entity-relation-model (ERM).