

Swank-Rats documentation

Thomas Gaida (thomas.gaida@students.fhv.at)

Stefan Lässer (stefan.laesser@students.fhv.at)

Johannes Schwendinger (johannes.schwendinger@students.fhv.at)

Johannes Wachter (johannes.wachter@students.fhv.at)

Michael Zangerle (michael.zangerle@students.fhv.at)

Abstract

Semester Project for course “S1 - Kopplung und Integration von heterogenen Systemen”.

Inhaltsverzeichnis

Abstract	iii
1 Introduction	1
1.1 Game Idea	1
1.2 Architecture	1
1.3 Communication	2
2 Image-Processing	5
2.1 Components	5
2.2 Why OpenCV and C++	5
2.3 Why VC++ with VS 2013	6
2.4 Why using Poco instead of Boost	7
2.5 Requirements	8
2.6 Architecture	9
2.7 Specific requirements for the client:	9
2.8 Specific requirements for the server:	10
3 Used technologies	11
3.1 MEAN-Stack	11
4 Installation	13
4.1 Beaglebone	13
5 Appendix	17
5.1 Get the Bone in the internet	17

1 Introduction

1.1 Game Idea

Swank Rat is a rat fighter game. Two rats are trying to shoot each other with cheese. The rats are represented by robots which are controlled by two players. With a Camera over the Game-World can the software “see” where the rats are. In addition, the obstacles are detected over this camera. This obstacles are straight walls (e.g. wood slates with a red with a red mark). The Rats are able to throw pieces of cheese after the opponent. The walls serve as a limitation for the cheese-bullets.

To control the robots the live video of the world (overlaid with video of the cheese-bullets) will be displayed in a HTML UI in the browser. With buttons (and keyboard shortcuts) can the player control the real robot.

If a robot is hit (one or more) the game is over.

1.2 Architecture

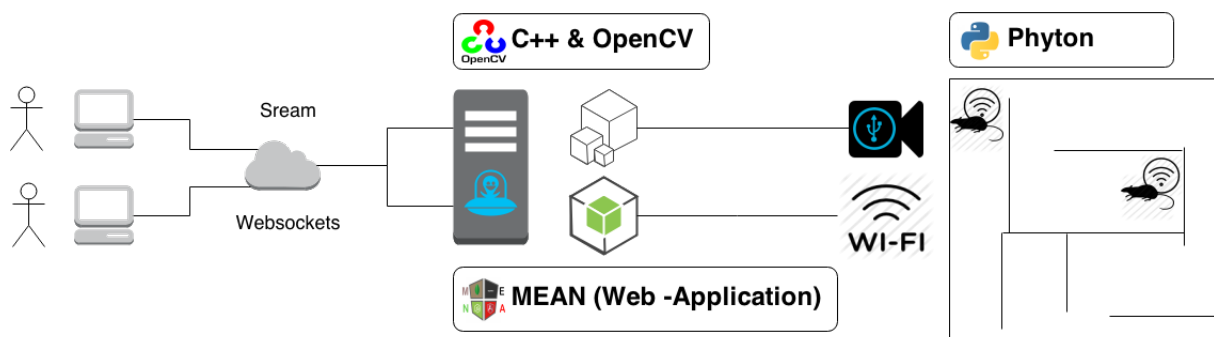


Abbildung 1.1: Architecture of Swank-Rats

1.2.1 Hardware

- 2 x “Rat-Robot” with WLAN Dongles to communicate with the server
- 1 x Webcam (for the detection of position and world)
- 1 x Server (Notebook or PC for image processing and game logic)
- 2 x Clients (Notebooks with modern Browsers)

1.2.2 Server-Software

- Server Application (Java)
 - Image processing
 - Position detection
 - Overlay webcam video with cheese-bullets
 - Stream video for client
- NodeJS Server
 - Robot control
 - Server UI (HTML)
 - User management

1.2.3 Client

- Browser Application
 - HTML5
 - Presentation of game stream
 - Javascript with Websockets
 - Buttons to control robot
 - Login
 - ...

1.3 Communication

For the communication we use Websockets. This TCP-based protocol provides bidirectional connections between all stations of our infrastructure and is well supported by all languages.

1.3.1 Used Libraries

- Phyton uses the ws4py (Websocket for Phyton) library
- Node-JS uses the minimalistic implementation of Websocket Protocol ws (websocket)
- C++ uses POCO which provides the Websocket implementation
- JavaScript in Browser uses the nativ Websocket API (Tutorial) of the browser

1.3.2 Protocol

For communication between the stations we use asynchronous json-messages with a specific structure this structure is implemented for Node.JS in a Open Source module Websocket-Wrapper. The implementation of this module is part of this project.

2 Image-Processing

2.1 Components

For the implementation of our image processing functionality we decided to use C++ in connection with OpenCV 2.4.9 (<http://opencv.org/>). It will help us to get the video stream of a webcam, to detect the position of the robots and to detect collisions (e.g. collision between robot and wall, but also collisions between a shot and a wall or robot).

For networking, including HTTP and WebSockets, threading and logging we use the functionality provided by the Poco C++ Libraries 1.4.7.

For the communication between the NodeJS server and the image-processing server we decided to use WebSockets and JSON objects. To fulfil this purpose we can use the API of Poco.

For compiling our source code we use Microsoft Visual C++ Compiler 18.00.21005.1 for x86 platform. Therefore we also use Visual Studio 2013 as our IDE.

2.2 Why OpenCV and C++

We did some research and searched for possible free image processing libraries. We decided to use OpenCV, because it offers the biggest amount of functionality compared to the other libraries which were available for free like SimpleCV, OpenCV for Java, OpenCV for .NET (Emgu CV) or JAI. We do not want to take the risk to use a library which offers less functionality and finally we may be faced with the problem, that a functionality that we need is missing.

First we thought about using Java together with the ported OpenCV version, but then we were a little bit afraid about possible performance issues, instability and the fact, that you have to use native method calls in your Java code to get access to the OpenCV functionality since it is written for C/C++. Also offers the wrapper for OpenCV under java less functionality then the original OpenCV for C++.

Another possibility would have been to use C#.NET. There are several opportunities like EmguCV, Halcon or Aforge.Net. But since EmguCV is also just a wrapper for the

OpenCV library, the Halocn library is not available for free and our researched figured out that it is more recommended to use OpenCV before using Aforge.NET, C#.NET was no option anymore.

So we decided to use OpenCV in connection with C++. It is a little bit risky because we do not have much experience in using C++ and it is the first time we use it in a project. So we are looking forward to learn a lot of new things.

2.3 Why VC++ with VS 2013

First we wanted to implement our project with Eclipse CDT in connection with the Boost library. The reason for us to use Boost was that we were especially interested in the threading and networking functionality, because we have a team member with a Mac and we wanted to offer him opportunity to develop with us. But the current version of the Boost library contains an already reported bug, which causes the build of Boost with MinGW to create corrupt files. This finally leads to an error in Eclipse, when you try to build the project.

3 errors, 1 warning, 0 others

Description	Location
✖ Errors (3 items)	
✖ undefined reference to `_InterlockedCompareExchange'	line 189, external location: G:\boost\boost\thread\win32\thread_primitives.hpp
✖ undefined reference to `_InterlockedCompareExchange'	line 197, external location: G:\boost\boost\thread\win32\thread_primitives.hpp
✖ undefined reference to `_InterlockedCompareExchange'	line 220, external location: G:\boost\boost\thread\win32\thread_primitives.hpp
⚠ Warnings (1 item)	
⚠ ignoring #pragma intrinsic [-Wunknown-pragmas]	line 180, external location: G:\boost\boost\thread\win32\thread_primitives.hpp

Abbildung 2.1: Eclipse errors when building project

We adapted the fix, which is mentioned in the bug report, to our local boost source files and recompiled the library. The result was that Eclipse didn't run the application anymore. Instead it displayed the message "Launch failed. Binary not found."

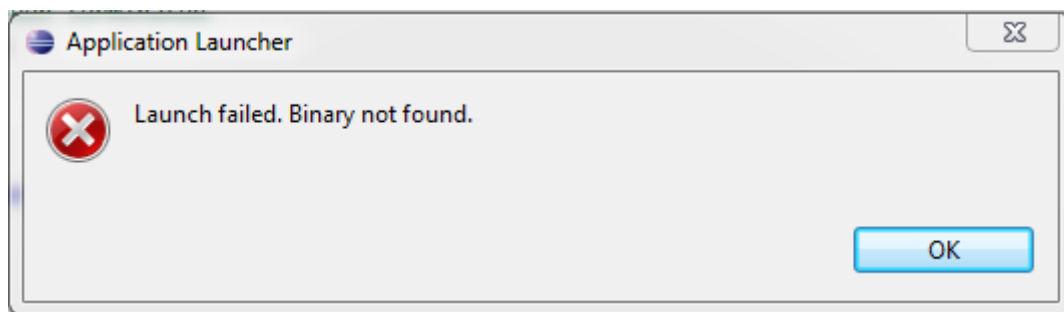


Abbildung 2.2: Eclipse errors when building project

The error log of the IDE did not mention anything helpful about this error. We got the same error with the previous Boost library (1.55). After some research about this error message we finally gave up at this point and decided to changed to VS 2013, VC++ and compiled Boost with the VC++ compiler.

2.4 Why using Poco instead of Boost

First we wanted to use Boost for the threading, networking and so on. Boost was also reason why we changed from Eclipse to Visual Studio. But finally, when we were faced with the communication of the image processing application to the NodeJS server, we investigated a lot of time to get WebSockets running with Boost and we failed.

2.4.1 Problems with WebSocket libraries

We tried to use Simple-WebSocket-Server. One of the big advantages of this library would have been that it uses Boost.Asio, but we got the following compiler error:

```
error C2338: invalid template argument for uniform_int_distribution g:\visual studio 2
```

We could not figure out where the source of this problem was exactly located. So we tried the next library.

Then we tried to use Websocketpp, which also uses Boost.Asio. Here we were faced to the following compiler error:

```
error C2064: term does not evaluate to a function taking 2 arguments c:\_libs\boost
```

We had contact with the developer of this library. First he recommended to use Boost 1.55.0 instead of 1.56.0, but the problem still occurred. Finally we could figure out that the source of the problem was in the file “websocketpp\common\functional.hpp” where some defines were wrong, which caused the error in VC 2013. The developer fixed the problem 2 weeks after we have decided to use Poco.

Finally we found Poco 1.4.7, which is a library like BOOST. The big difference is that Poco already contains an API for creating a WebSocket server/client and also a HTTP server/client can be easily implemented. Poco was very easy to compile and get things running with VS2013. So we changed (again) the library from Boost to Poco.

2.5 Requirements

2.5.1 Needed functionality

- Providing video stream for clients
- Collision detection
 - Robot/wall collision
 - Shot/robot collision
 - Robot/robot collision
 - Shot/wall collision
- Position detection of walls/robots
- Communication with NodeJS server
- Simulation of shoots in video stream

2.5.2 Communication with NodeJS server

It is necessary that the NodeJS server and the image-processing can talk with each other. The communication is needed, because the NodeJS server has not enough knowledge to make all the game logic decision by its own.

The following messages can be sent: - Messages by image-processing server to the NodeJS server - If a collision was detected (see above which cases exist) - If a shot was made to notify if a robot was hit or not - Messages by NodeJS server to image-processing server - If a shot was made by a player - If game has stopped - If game has started - If game has paused (e.g. connection problems) - Info about which player has which robot

2.5.3 Video quality and resolution

For our project we use the webcam LifeCam HD-3000 from the manufacturer Microsoft. We decided to use 720p resolution for the streaming. This allows us to provide the clients a gaming environment in a today acceptable resolution without too much traffic through the transmission. Therefore the system need following requirements - Intel Dual Core 3.0 GHz or higher - 2 GB of RAM - 1.5 GB - USB 2.0 required

The maximal resolution for motion video is 1280 X 720 pixel for still image 1280 X 800. The webcam has a maximal image rate up to 30 frames per second and a 68.5 degree diagonal field of view. The other image features of the webcam are Digital pan, digital tilt, vertical tilt, swivel pan, and 4x digital zoom - Fixed focus from 0.3m to 1.5m - True

Color - Automatic image adjustment with manual override - 16:9 widescreen - 24-bit color depth

2.5.4 Latency

TODO what we want

2.5.5 Connection loss

TODO what should happen

2.6 Architecture

TODO Architecure diagram #Requirement analysis for client and server

We defined a platform independent implementation as a general requirement for the client and the game-server. Furthermore we wanted to keep the possibility to play our game also via a tablet or even a smartphone and to keep it also extensible. Last but not least we wanted to use new, state of the art web-technologies for the sake of the web and for our continuing education.

2.7 Specific requirements for the client:

- clean and simple user interface
- the user interface should be responsive by default
- a login / ranking page should be visible for logged in users
- a welcome / introduction / registration page should be visible for all users
- a permanent connection to the server for fast and efficient transfer of game related commands should exist
- the client should be able to display the game universe with it's players and their interactions via a stream

2.8 Specific requirements for the server:

- the server should be able to communicate in an easy/efficient/fast way with the robots, the image processing unit and also the clients (bidirectional communication)
- the server should be able to communicate with a database to persist game results and also player specific data e.g. the user-/player-name, password
- the server should provide a fast webserver which also supports SSL for basic security
- the server has to provide interfaces for the client to process registrations, logins, page-ranking requests and normal page requests as well as game-specific commands
- the server should be able to communicate with the image processing component (more details here)
- FEATURE: game unrelated users with a smartphone should be able to watch the current game on their smartphones. This means that these users should film the game from defined positions with their smartphones and another image will be used as overlay to display player interactions and more

3 Used technologies

The technologies we used for this project for the client and the game-server is the mean stack.

3.1 MEAN-Stack

This stack consists of four different software components which work very smooth with each other. These components are the MongoDB which is a database, the and Express the Angular.js Javascript frameworks and last but not least the Node.js environment.

3.1.1 MongoDB

MongoDB (from “humongous”) is an open-source document database, and the leading NoSQL database. It’s main key features are document-oriented storage, full index support, replication and high availability, auto sharding, map/reduce support, in place updates, and a lot more. You can find more information about the database on their website.

3.1.2 Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications without obscuring Node.js features that you know and love. You can find more information about the framework on their website

3.1.3 Angular.js

Angular is an open source framework from Google and helps to make and structure single-page web applications according to the MVC pattern. Angular.js operates entirely on the client side. You can find more information about the framework on their website

3.1.4 Node.js

Node.js is an open source, cross-platform runtime environment for Javascript applications. Node.js provides an event-driven, non-blocking I/O model that makes perfect for data-intensive real-time applications. Internally Node.js uses Google V8 Javascript engine which is also used in the Chrome Browsers to execute the applications. A lot of the environment is also written in Javascript and it provides modules for file, socket and HTTP communication which allows it to act as a web server. Popular companies which use Node.js are for example SAP, LinkedIn, Microsoft, Yahoo, Walmart and PayPal. You can find more information about the framework on their website

4 Installation

In this chapter is described how to install the environment to run the software.

4.1 Beaglebone

To run the control-software for the robot you have to install:

- WIFI (TP-Link TL-WN725N)
- Phyton

4.1.1 WIFI (TP-Link TL-WN725N)

This tutorial is inspired by: <http://brilliantlyeasy.com/ubuntu-linux-tl-wn725n-tp-link-version-2-wifi-driver-install/>

Important: Run in **root**

1. Install Kernel-Sources

```
apt-get update
apt-get install linux-headers-$(uname -r)
```

If the linux-header version does not exists search for deb file in <http://rcn-ee.net/deb/precise-armhf>.

Example:

```
wget http://rcn-ee.net/deb/trusty-armhf/v$(uname -r)/linux-headers-$(uname -r)_1.0trusty-armhf.deb
dpkg -i linux-headers-$(uname -r)_1.0trusty-armhf.deb
```

2. Install dependencies

```
apt-get update
apt-get install build-essential git
```

3. Build driver

```
git clone https://github.com/lwfinger/rtl8188eu
cd rtl8188eu
make all
make install
insmod 8188eu.ko
```

4. Check installation

```
iwconfig
```

5. Reboot

```
reboot
```

6. Install and Configure WPA-Supplicant

```
apt-get install wpasupplicant
wpa_passphrase <ssid> <password> > /etc/wpa.config
```

7. Add config to start script: App following to config file /etc/network/interfaces

```
auto wlan0
iface wlan0 inet dhcp
    wpa-conf /etc/wpa.config
```

4.1.2 Phyton

```
apt-get install python
apt-get install python3
cat > hello.py << EOF
#!/usr/bin/env python3
# Mein Hallo-Welt-Programm fuer Python 3
print('Hallo Welt!')
EOF
python hello.py
chmod u+x hello.py
./hello.py
```

4.1.3 Troubleshooting

1. ERROR: mach/timex.h: No such file or directory

```
cd usr/src/linux-headers-$(uname -r)/arch/arm/include
mkdir mach
touch mach/timex.h
```

source: <https://groups.google.com/forum/#!msg/beagleboard/1IkTdkdUCLg/8th83TmgdPkJ>

2. WARNING: perl: warning: Setting locale failed.

```
sudo locale-gen de_AT.UTF-8
```

source: <http://stackoverflow.com/questions/2499794/how-can-i-fix-a-locale-warning-from-perl>

5 Appendix

5.1 Get the Bone in the internet


This will tell you how to connect the beaglebone to the internet. There are three possibilities how to achieve that goal

5.1.1 Easy

1. Connect an ethernet cable with your bone and the other end with your router.
2. You're done.

5.1.2 There is only WIFI or It's not that easy

Requirements: Computer which can connect to the WIFI and with a free Ethernet Port
Ethernet Cable

1. On the host computer (the computer whose Internet connection you plan to share) open Network Connections by clicking the Start button  Picture of the Start button, and then clicking Control Panel. In the search box, type adapter, and then, under Network and Sharing Center, click View network connections.
2. Select your WIFI and your LAN adapter by holding CTRL and clicking on both.
3. Right click on the second and choose "Bridge Connection"
4. Connect an Ethernet cable with your bone and the other end with your router.
5. You're done

5.1.3 Why should I use another cable?

1. Connect to your bone via SSH
2. Enter the following:

```
ifconfig usb0 192.168.7.2
route add default gw 192.168.7.1
echo "nameserver 8.8.8.8" >> /etc/resolv.conf
```

3. On your computer: running Linux:

```
sudo su
#eth0 is my internet facing interface, eth3 is the BeagleBone USB connection
ifconfig eth3 192.168.7.1
iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
iptables --append FORWARD --in-interface eth3 -j ACCEPT
echo 1 > /proc/sys/net/ipv4/ip_forward
```

running Windows: follow this Tutorial: <http://windows.microsoft.com/en-us/windows/set-internet-connection-sharing#1TC=windows-7>

change the IP-Adress of your USB-Networkadapter to the old "192.168.7.1"

4. You're done