Swank-Rats documentation

true true true true true

January 10, 2015

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Introduction

TODO fancy description of project

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

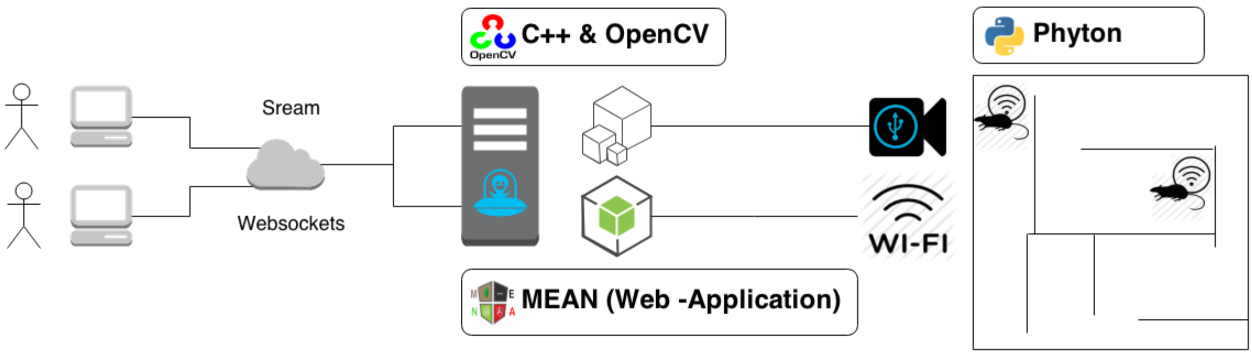

Abb. 1 – Architecture of Swank-Rats

1.2.1 Hardware

- 2 x “Rat-Robot” with WLAN Dongles to communicate with the server
- 1 x Webcamera (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 asyncronous 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 Game-Server

TODO fancy description of game-server

2.1 Requirements 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.

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

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 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

2.2 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.

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.

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

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

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

2.3 Game controls

A robot can be controlled with the arrow keys: - W will accelerate the robot and it will drive in the direction it is looking - S will accelerate the robot and it will drive backwards - A will rotate the robot to the left - D will rotate the robot to the right

With the 'L' the player can shoot. As long as the keys are pressed the robot will drive. When no key is pressed the robot will stop.

2.4 Game logic

1. Starting a game: To start a game a defined number of players (in our case two) is needed. This means the first player which gets to the games page will have to choose a color and start a new game. The second one can join the game after choosing a color. This means when no game is ready one has to be created. When a game in the ready status exists, players can join the game as long as the maximum number is reached. When the maximum is reached no player can join the game anymore. For the players which joined it the game will now start.
2. During a game: Each player can controll a robot with the arrow keys and he can shoot at the other player(s) by pressing the spacebar. You can find more information about the controlls in the game controlls section.

- Gameplay: The goal of the game is to reduce the lifepoints of the opponents by shooting him with the cheese bullets. This means when at least two players have lifepoints left the game will continue. The amount of lifepoints and the damage caused by cheesebullets should be configurated in the config-file. Also the multiplicator for fast wins should be configured there.
- Highscore: The final highscore will be calculated from the hits, the needed time and the remaining lifepoints. A highscore will be created for the winner.

3. Connection problems: When one player has problems with it's connection and the connection can not be restored the remaining player winns the game.
4. After the game: Is a game finished both players see a message and will then be redirected to the highscore page. The game itself will be set on finished and a new game can be started.

3 Image-Processing

TODO fancy description for image-processing

3.1 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.

3.2 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.

3.3 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.

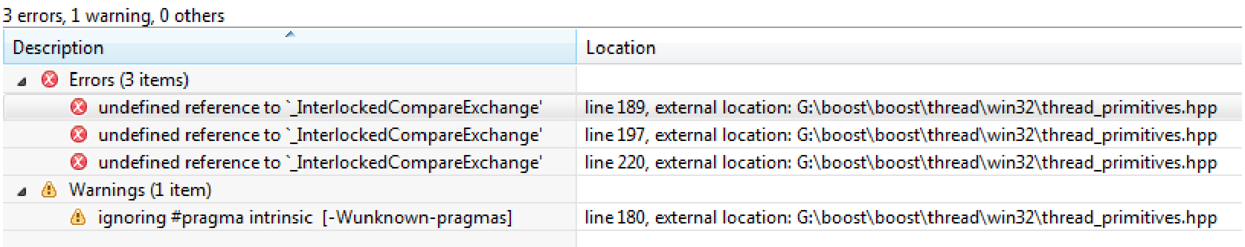

Abb. 2 – Eclipse errors when building project

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.".

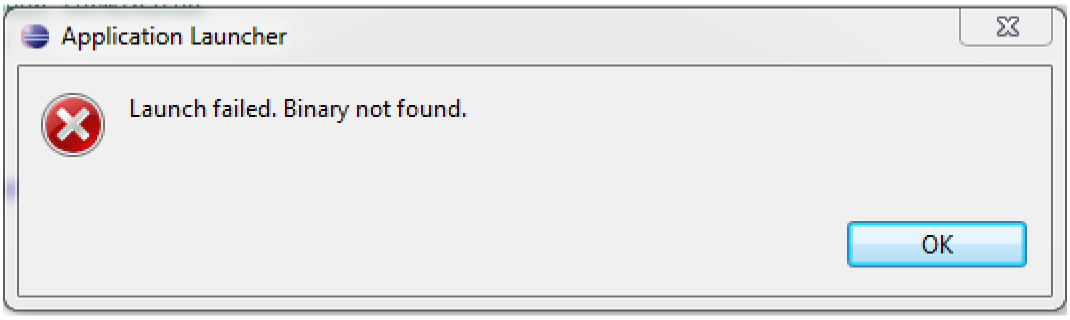

Abb. 3 – Eclipse errors when building project

The error log of the IDE did not mention anything helpful about this error. We got the same error with the previous Boost libary (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.

3.4 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.

3.4.1 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 2013\vc
```

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\1.56.
```

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.

3.5 Requirements

3.5.1 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

3.5.2 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 make 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

3.5.3 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

3.5.4 Latency

TODO what we want

3.6 Architecture

TODO Architecture diagram

3.7 Object detection

3.7.1 Lessons learned

At the beginning of our object detection task we tried different detection solutions out.

- RGB detection
- HSV detection
- Contour detection with marker

3.7.1.1 RGB Color detection First of all we tried to solve the object detection by using a color detection. There we started with a simple RGB color adjustment. But this adjustment brought not the desired success. When using RGB we had too much influence by the light of the environment and when the object distance to the camera was to far we could not recognize the object any more.

3.7.1.2 HSV Color detection After the RGB detection we tried to detect the object via HSV colors. This worked alot better then the detection via RGB. But it also brought not the desired success. Also here we had too much influence by the light of the environment and when the object distance to the camera was to far we could not recognize the object any more. Compared to the RGB detection the distance was greater but for our solution to short.

HSV (hue-saturation-value) is the most common cylindrical-coordinate representations of points in an RGB color model. It rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the cartesian (cube) representation, by mapping the values into a cylinder loosely inspired by a traditional color wheel. The angle around the central vertical axis corresponds to "hue" and the distance from the axis corresponds to "saturation". Perceived luminance is a notoriously difficult aspect of color to represent in a digital format (see disadvantages section), and this has given rise to two systems attempting to solve this issue: Both of these representations are used widely in computer graphics, but both are also criticized for not adequately separating color-making attributes, and for their lack of perceptual uniformity.

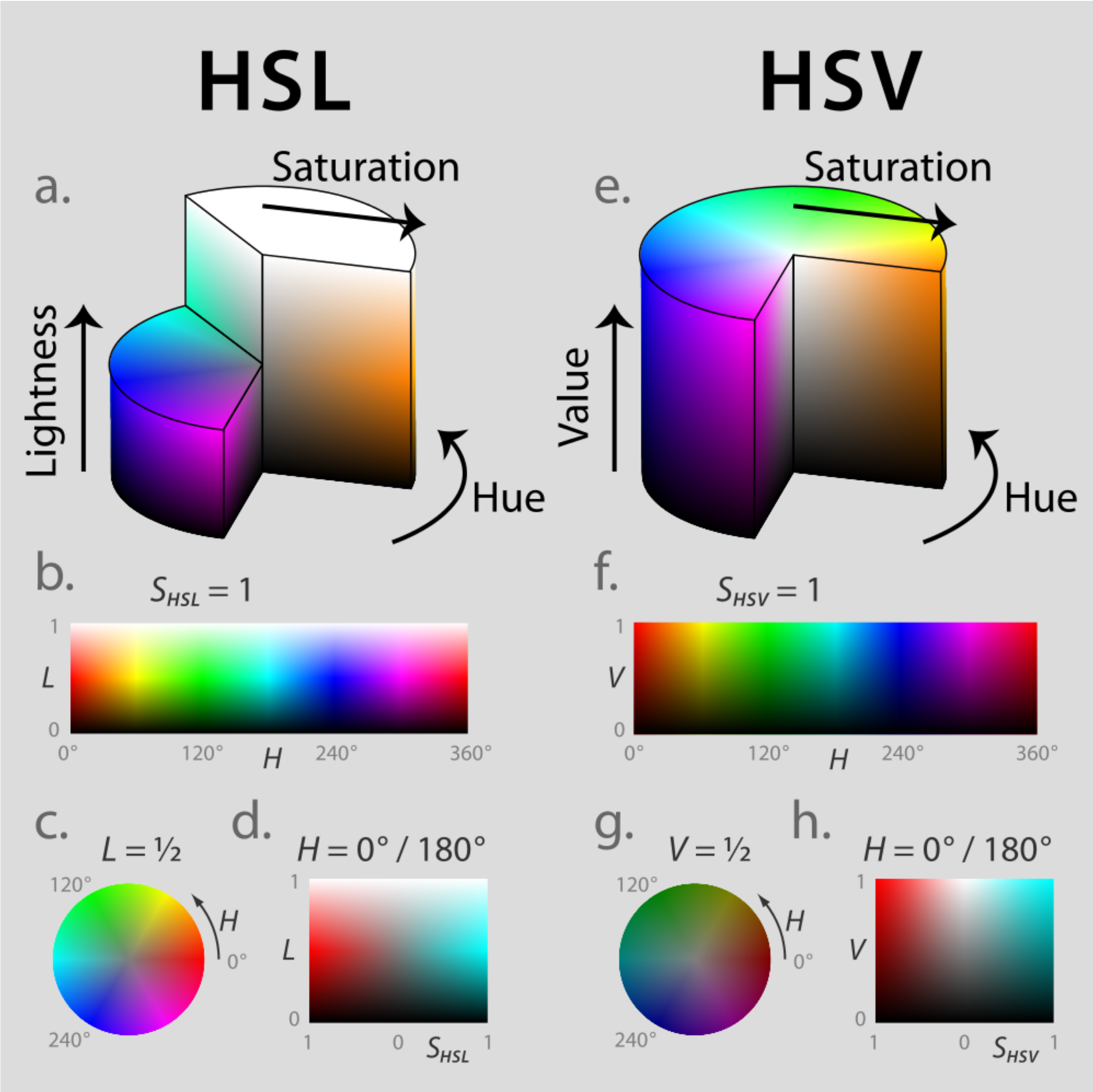

Abb. 4 – HSV model

Below you can see the detection result of our HSV detection. First you see the original image and then our detection result which detect blue forms.

3.7.1.3 Contour detection with marker

We also tried to detect the object via its contours. To realize this we went forth and first tried various geometry forms and tried to recognize them by there contours. Below you can see the detection result. First you see the original image and then our detection result.

In order to bring more security in the contour detection, we have decided to replace the simple contours by nested contours This enables us to detect the object more error-free and more stable then with simple contours. Below you can see the detection with nested contours. Only the rectangles with triangles in the rectangles boundaries are detected.

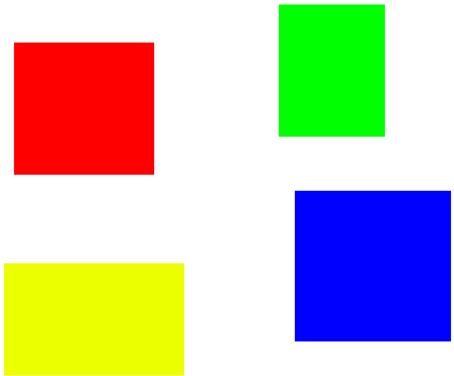

Abb. 5 – HSV detection original image

Abb. 6 – HSV detection after detect blue forms

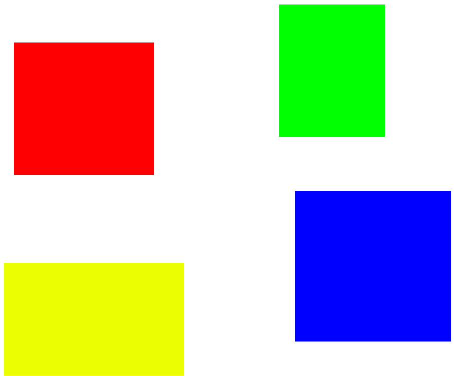

Abb. 7 – Rectangle model

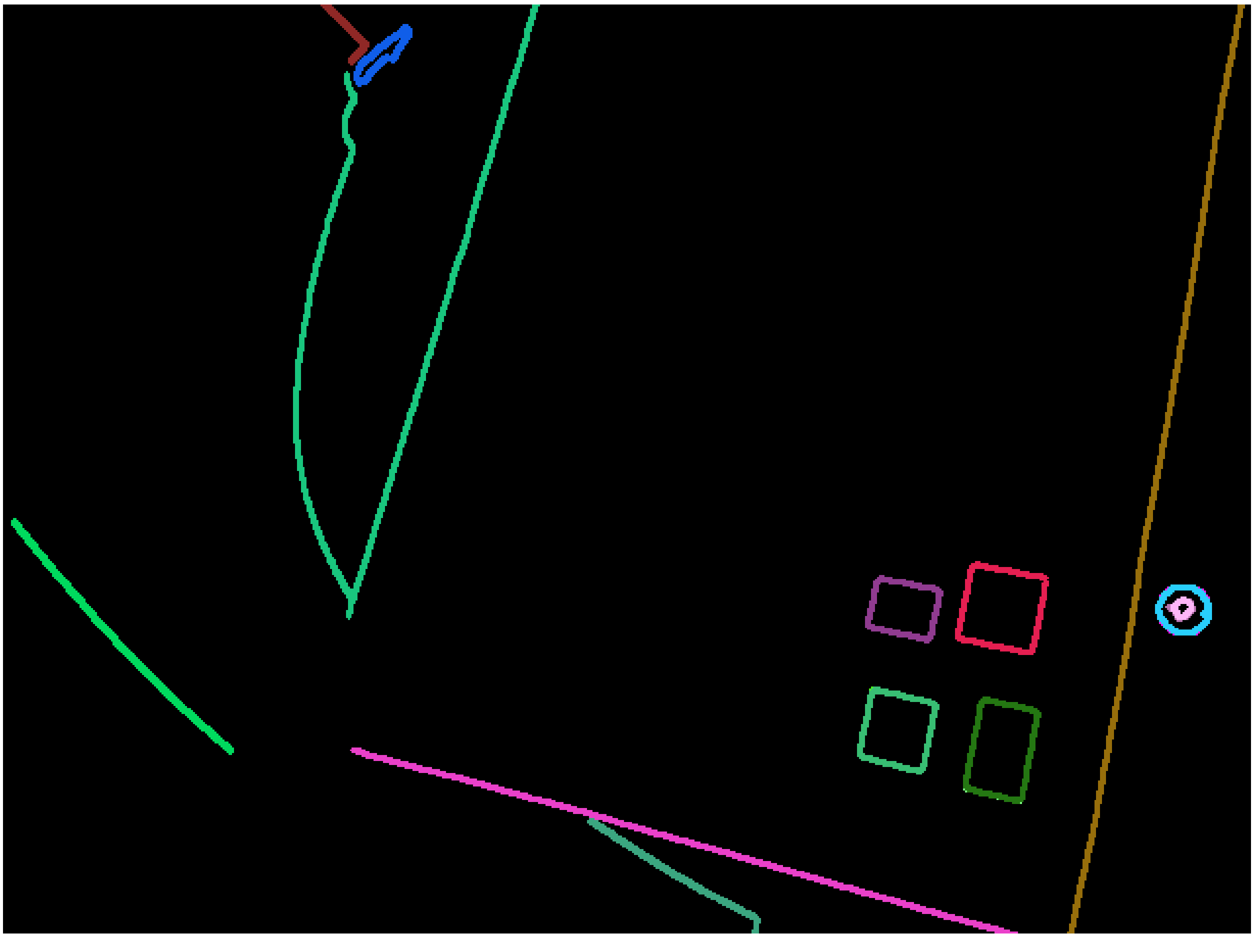

Abb. 8 – Rectangle model after detection

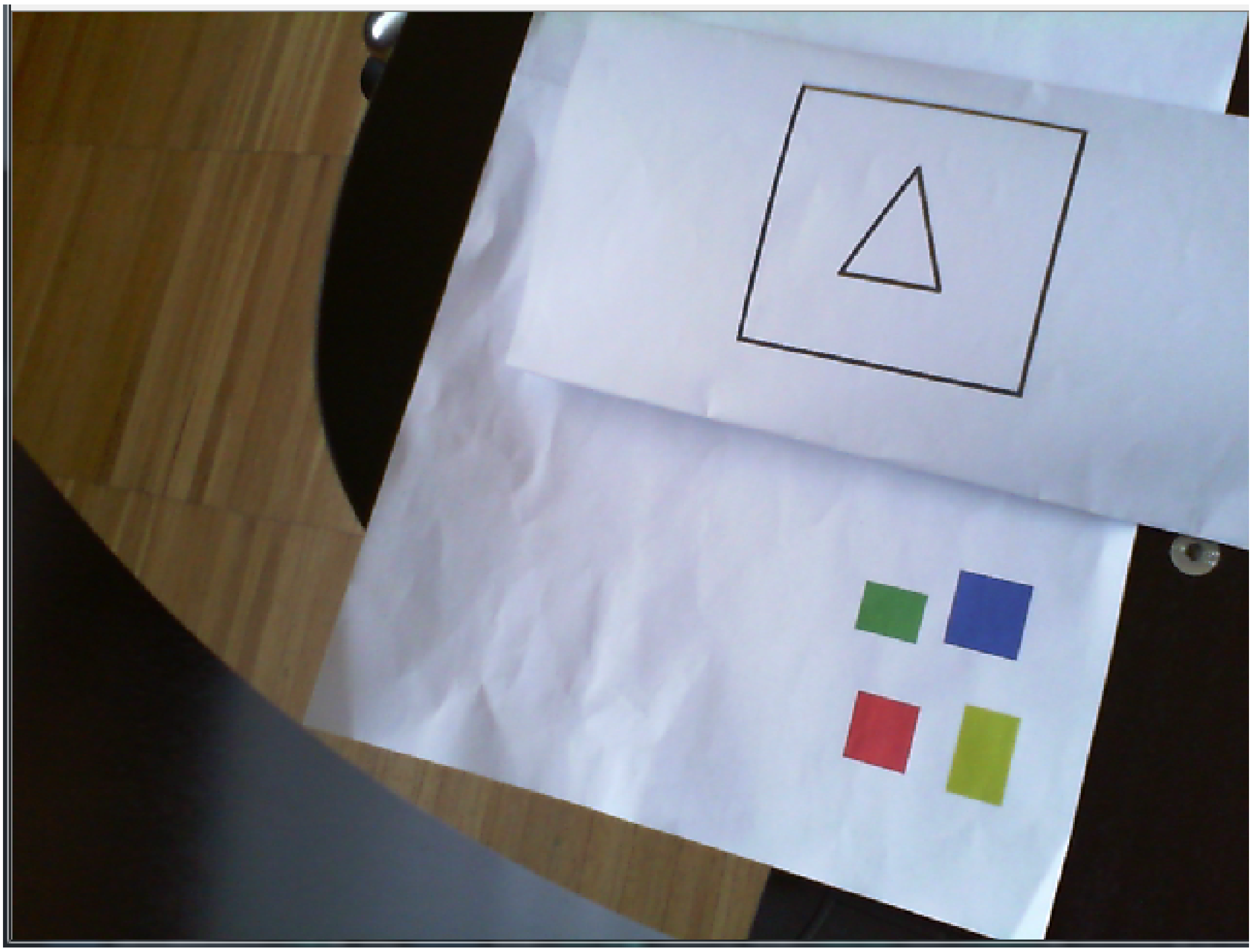

Abb. 9 – Rectangle model nested

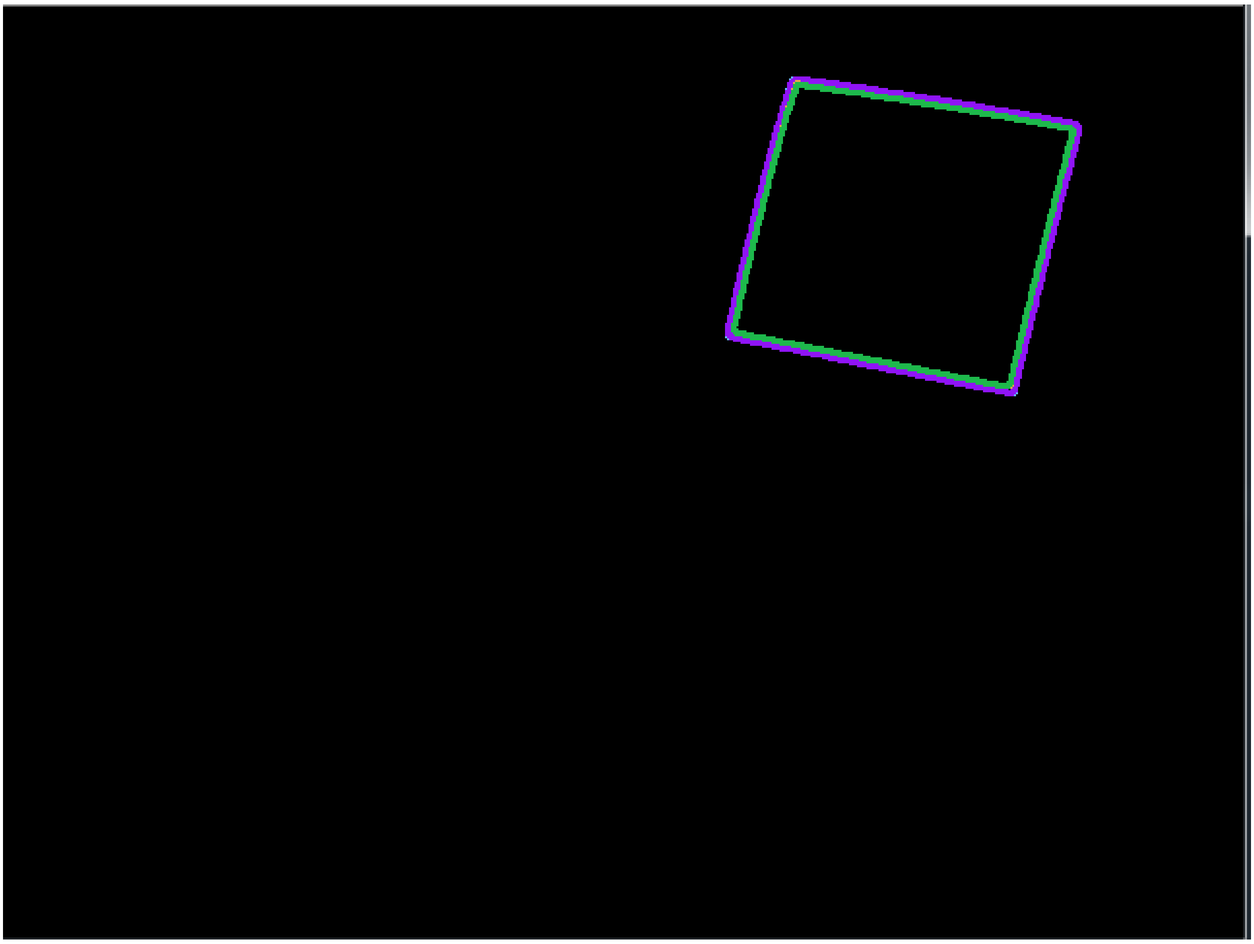

Abb. 10 – Rectangle model nested after detection

3.7.2 Conclusion Lessons learned

After our tests we decided to use contour detect for our object detect. The reason for this is that the detect via contours works faster, more stable and produces less errors during the detect produces than the RGB and HSV detection.

3.8 Websocket communication

TODO

3.8.1 Handling of connection loss

3.9 Video streaming to HTML client

We decided to use Motion JPEG (MJPEG) since it is very easy to implement, has only less restrictions and can be easily provided over HTTP.

3.9.1 How does it work

The protocol is quiet easy to understand. The browser of a client sends a normal HTTP GET request to our server. We need to answer the request with HTTP 200 OK and set the content type to “multipart/x-mixed-replace; boundary=–VIDEOSTREAM”. This signals the client to expect several parts delimited by the boundary name “–VIDEOSTREAM”. The TCP connection is not closed until the server or the client closes it.

The following image shows the communication between the client and the server. The TCP ACKs were not mentioned. If the packet size is bigger than 1500 bytes, it will be automatically split up into several parts.

Abb. 11 – MJPEG communication

Further information about MJPEG can be found here:

- Wikipedia Motion JPEG
- MJPEG protocol definition

3.9.2 Advantages and Disadvantages

MJPEG has the big advantages that it is easy to implement, no further libraries were needed and on the client side most of the modern browser like Google Chrome, Mozilla Firefox, Safari or Opera support MJPEG natively. Only Microsoft Internet Explorer does not support it.

The disadvantages were the inefficiency compared to more modern formats like H.264/MPEG-4 AVC as you have to always send the whole image. There is no interframe compression like in other, more modern standards. In our case we were also faced to some performance loss caused by the TCP connection, which we have to use since we talk to a browser.

3.9.3 Handling of connection loss

TODO

3.9.4 Handling of no available video stream

If no video stream is available, e.g. if no webcam is connected to the server, we cannot provide a video stream. In such a case all incoming video stream requests will be answered with HTTP/1.1 500 OK. The HTTP status code 500 means an internal server error occurred. Afterwards the connection is closed.

3.9.5 High delays

At the beginning we were faced with high delay rates of over 70 ms between each frame on the client. It felt like it was even more.

We figured out that there were several reasons for this. Two main problems were directly located in our implementation. We had some unneeded thread synchronization code and we also cloned each frame, which is not necessary since the used data structure (OpenCV Mat) provides reference counting. So a copy of a Mat object will not result in copying the whole image. Both instances will share the matrix, which represents the image.

Next we figured out that we send about 80 - 90 kb per frame. We solved the problem by decreasing the quality of the image we send. OpenCV provides the possibility to change the quality very easily during converting a Mat object into a vector of bytes. So we could decrease the size per frame to about 10 kb by setting the quality to 30 %.

TODO add image that compares original stream with stream received by client

With this few changes we could decrease the delay to about 20 ms, which is acceptable.

One big disadvantage, which costs a lot of performance, is the TCP connection overhead. Sadly it is not possible to provide an MJPEG stream via UDP to a browser.

3.10 Cheese-throw simulation

The simulation of throwing a cheese is done by overlay the webcam stream with the images needed for the simulation. One Cheese-throw simulation consists of three parts:

1. Throw-animation: a small explosion in front of the robot shows the start of a throw
2. Cheese-throwing-animation: the explosion is followed by a cheese, which is animated to simulate a flying cheese beginning at the robot and ending at the calculated end position

3. Hit-animation: when the flying cheese reaches the end position a explosion is simulated

The following image illustrated all three states.

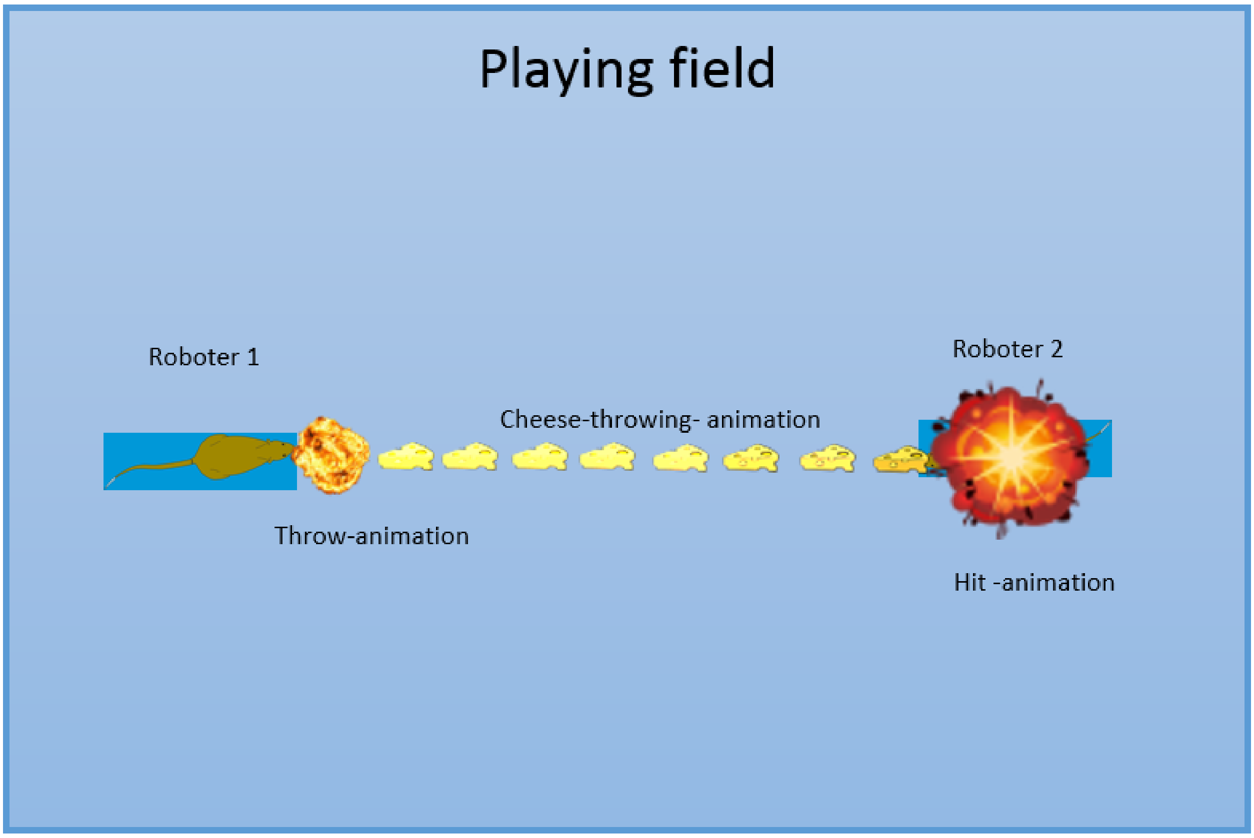

Abb. 12 – Cheese-throw simulation states

A simulation is started if the NodeJS server tells the image processing server that a cheese was thrown by a player. We can then determine the start and end point of a cheese-throw simulation, since we know the position and the viewing direction of the throwing player and by the fact that we are only simulating straightly throws.. The simulation is immediately started with the next occurring webcam frame and therefore also immediately visible for the clients. The decision, if a player or a wall was hit by the cheese is done when the simulation reached the end point. So we can ensure that the other player gets the chance to avoid a collision with the cheese.

The calculations for a simulation is not that complicated since the start and end point can be interpreted as a right-angled triangle, as the following image illustrates.

The simulation takes place along c (hypotenuse). We just have to calculate the length of a and b, shown in the image above by doing the following calculation:

- a = endPointX - startPointX
- b = endPointY - startPointY

By doing it this way we do not have to consider the direction in detail since the following cases were covered:

- start point x < end point x: next point x > start point x; a > 0
- start point y < end point y: next point y > start point y; b > 0
- start point x > end point x: next point x < start point x, a < 0
- start point y > end point y: next point y < start point y, b < 0

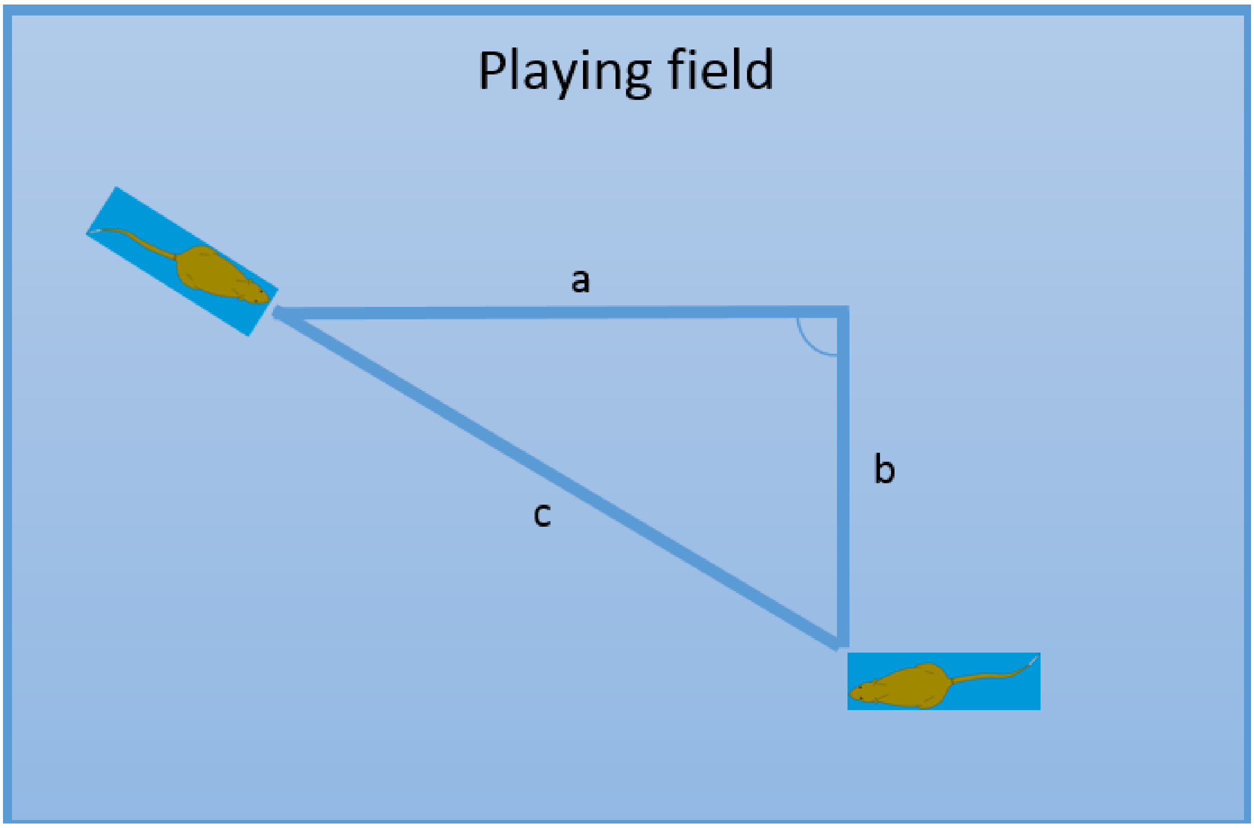

Abb. 13 – Cheese-throw simulation right-angled trigangle

- start point x = end point x: next point x = start point x; a = 0
- start point y = end point y: next point y = start point y; b = 0

This makes sure that all throw directions were possible without any additional magic in the code. The following image illustrates all directions.

During the simulation the next point, where we show a cheese image, is calculated the following:

- next point x = start point x + a * percentage
- next point y = start point y + b * percentage

Since we an image in the background we have to round the results to integers, because we cannot consider e.g. half pixels.

As you can see we always add a specific percentage of the length of a to the start point x value and of b to the start point y value. The initial percentage is 3% and is increased by +3% after each simulation step. We use 3% since our tests figured out that 3% offers us the best balance between throwing speed (the image does not move too fast and not too slow) and the gaps between each cheese image.

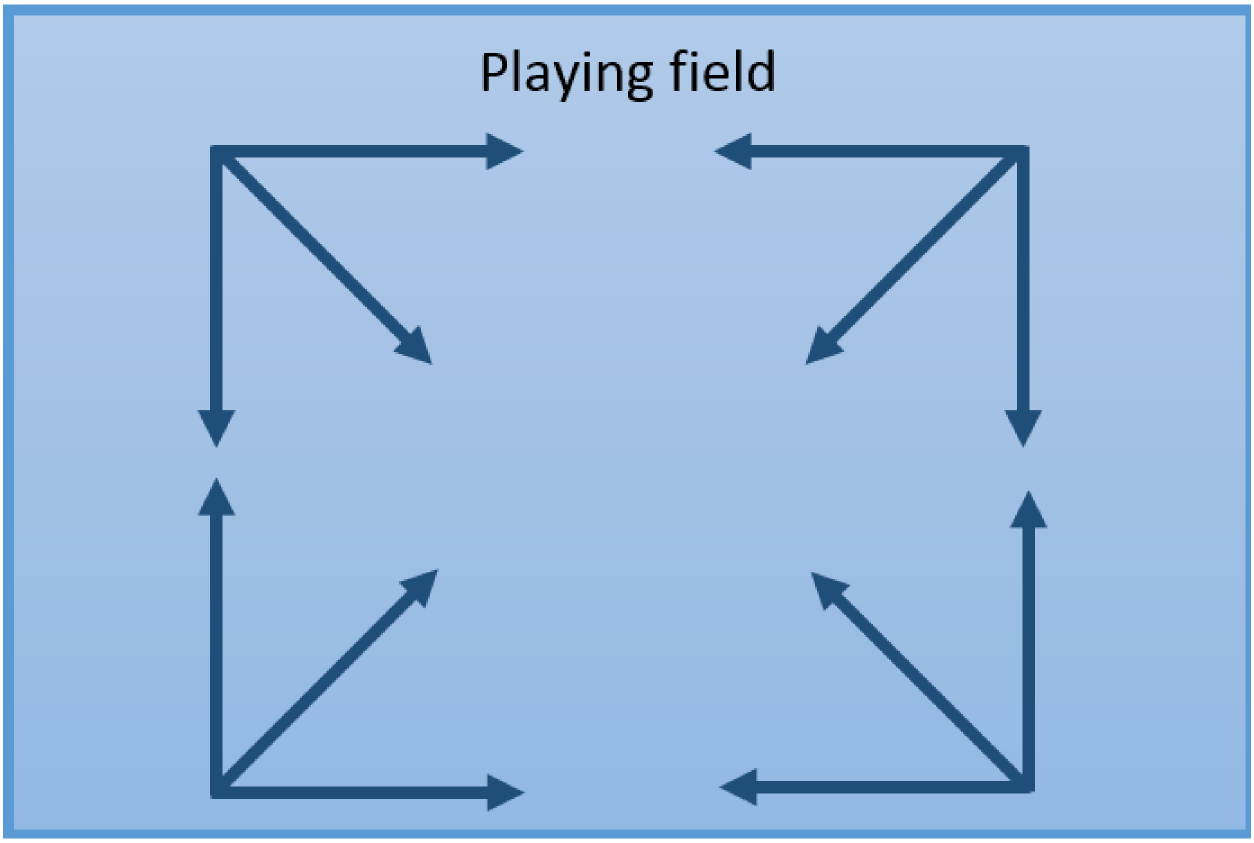

Abb. 14 – Cheese-throw directions

4 Communication

TODO copy and adapt following doc to this doc.

- https://github.com/swank-rats/websocket-wrapper/blob/master/README.md

5 Roboter

The meeple is a real robot which moves in the real world pitch. It is controlled by a person on the browser. In this chapter this robot will be described in more detail.

TODO picture of real robot

5.1 Hardware

The robot hardware is a composition of several parts:

- BeagleBoneBlack: is the control unit of the robot. It communicate with the server and controls the wheels and motors. The software for this board is written in Phyton which provides a library to communicate with the Common IO of the main board. To connect with the LAN the board uses a WLAN-Dongle.

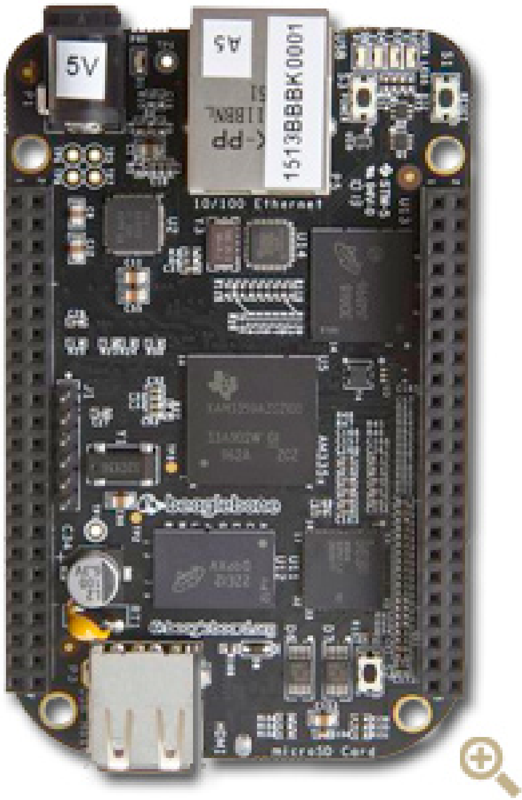

Abb. 15 – BeagleBoneBlack from above

- Chasis: The chassis is a round robot which is powered by two electric motor and two wheels, which provides to corner sharply.

- Engergy supply: For the energy supply we use 8 (TODO ???) batteries which provides directly the power for the motor and supply the BeagleBone with 5V, over a POWER SUPPLY CAPE.

5.2 Specification

- TODO bbb
- TODO motor

Abb. 16 – Chassis example picture

5.3 Energy consumption

- TODO energy consumption

6 Installation

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

6.1 Beaglebone

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

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

6.1.1 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_arm
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
```

6.1.2 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
```

6.1.3 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

7 Appendix

7.1 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

7.1.1 Easy

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

7.1.2 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

7.1.3 Eduroam Wlan-Network

Step 0

```
apt-get install wpasupplicant
```

Step 1

- Login as root
- Create a file `/etc/wpa_supplicant/wpa_supplicant.conf`

*Type the following parameters: `ctrl_interface=/var/run/wpa_supplicant network={ scan_ssid=1 ssid="eduroam" key_mgmt=WPA-EAP eap=PEAP identity="xyz1234@students.fhv.at" password="XXXXXX" ca_cert="/etc/ssl/certs/AddTrust_Extern_Root.pem" phase1="peaplabel=0" phase2="auth=MSCHAPV2" }`

Step 2

Run the command: '`bash wpa_supplicant -i $WLAN -D wext -c /etc/wpa_supplicant/wpa_supplicant.`

Note: * Insted of $WLAN type your interface name. * To view the name type iwconfig * Wait until the authentication is completed. * To receive an IP address type: dhclient

To view if you are connected type:

```
ifconfig -a
```

Or use script from FHV

Download it from: https://inside.fhv.at/pages/viewpage.action?pageId=54198344

```
chmod +x eduroam-linux-Fachhochschule_Vorarlberg.sh
./eduroam-linux-Fachhochschule_Vorarlberg.sh
wpa_supplicant -i wlan0 -D wext -c /root/.eduroam/eduroam.conf&
```

7.1.4 Why should I use another cable?

This method creates a default route on the beaglebone which can cause errors with other networks! Try an other solution first!

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 change the IP-Adress of your USB-Networkadapter to the old “172.168.7.1”

4. You’re done