Swank-Rats documentation

Thomas Gaida Stefan Lässer Johannes Schwendinger
Johannes Wachter Michael Zangerle

March 4, 2015

Table of contents

Table of figures

1 Introduction

Swank-Rats is a mulitplayer, realtime, augmentmented reality, browser based, distributed and platform independent game. Which combines modern technologies and communication systems.

It was developed by a team of 5 people in the master course "S1: Coupling and Integration of Heterogeneous Systems" at University of Applied Sciences Vorarlberg in Dornbirn, Austria.

Figure 1 – University of applied science

The following movie shows the current development state which will be presented at the graduation date. The image processing has a few problems which results in:

- Some shots are not shown except of the end explosion.
- Sometimes the robot is not recognized which causes in faulty shot simulations.
- The live stream laggs sometimes

Figure 2 – Youtube presentation video

1.1 Game Idea

Swank-Rats 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. A camera is mounted above the game world, which offers the possibility to detect the rats position and to provide a live stream to the clients. It is also possible to detect obstacles like straight walls (e.g. wood slates). The walls serve as a limitation for the cheese-bullets.

The clients can see the robots via a live stream of the world displayed in a HTML UI in their browser. With buttons and keyboard shortcuts the player can control the real robot.

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

1.2 Architecture

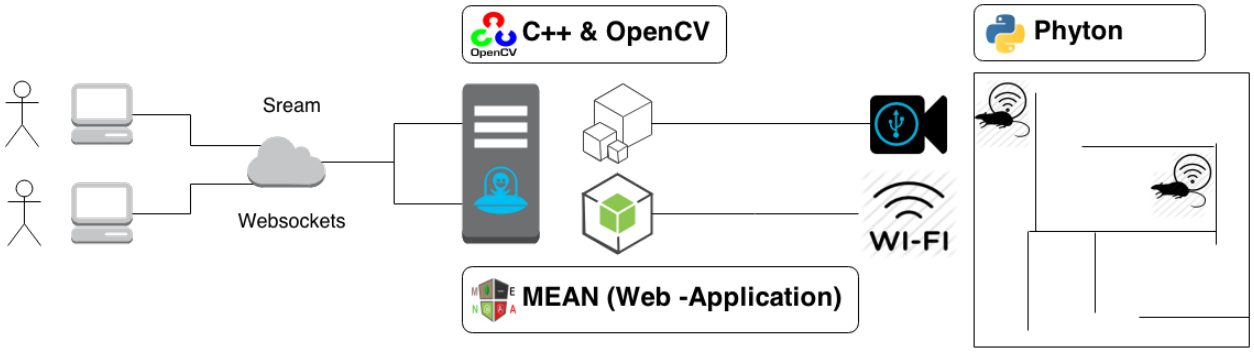

Figure 3 – Architecture of Swank-Rats

1.2.1 Hardware

- 2 x rat robot with WLAN dongles to communicate with the server
- 1 x Webcam for displaying the world and image processing like robot position detection,...
- 1 x Server (notebook or personal computer for image processing and game logic)
- 2 x Clients (notebooks or personal computer with modern browsers)

1.2.2 Server-Software

- Server Application (C++)
  - Image processing
  - Position detection
  - Overlay webcam stream with cheese-bullets
  - Offer webcam live stream to client
- Node.js 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 our used programming 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

The game server is the core component of the Swank-Rats game and controls and organizes all the other components and parties in the game.

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 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 web server 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

We used the MEAN stack provided by MEAN.IO because it provides already a lot of the basic needs like a user registration and some demo packages.

This stack consists of four different software components which work very smooth with each other. The components are MongoDB, which is a database, Express, the Angular.js JavaScript frameworks, and last 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 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 control a robot with the keys mentioned above and he can shoot cheese bullets at the other player by pressing the L key. You can find more information about the controls in the game controls 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 cheese-bullets should be configured in a config-file. Also the multiplicator for fast wins in the highscore calculation should be configured there.
- Highscore: The final highscore will be calculated from the needed time and the remaining lifepoints. A highscore will be created for the winner.

3. Connection problems: With the node module forever the app will be restarted in case of a major error. The connection of the players will be restored when they leave the game-screen and return later. When the connection is lost due to an error the client will also try to restore it.
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.

Below you can find an image of the general lifecycle of a game.

[Lifecycle of a game][game-server/images/lifecycle.png]

2.5 Security

To secure the game server all connections use SSL and the WebSockets have to use basic HTTP authentication. Otherwise the connection will be cancelled. You can find more information about the communication and its security in the communication section.

2.6 Connecting to the server

All parties connect to the Node.js server via WebSockets. To register themselves they send 'init' in the cmd-parameter and node server takes care of the rest.

2.6.1 Robot

When a robot registers himself at the Node.js server, its socket-connection gets stored along with some other information like its associated form (square or circle).

```
init: function(socket, params) {
   if (!!params.form) {
       RobotsSockets[params.form] = {
           socket: socket,
           params: params,
           send: function(msg) {
               ...
               this.socket.send(msg);
               ...
           }
       };
   }
}
```

When a game gets started all robots will receive a start message which tells the robots that they should execute all moving commands from now on. When a game is finished all robots will receive a stop message which indicates that they should not do anything unless they receive a start message again.

2.6.2 Image processing server

When the image processing server established the connection is stored in a variable along with some event listeners. The first listener gets called when the socket closes. The second listener gets called when a socket specific error occurs.

```
init: function(socket) {
   ImageServerSocket = socket;

   socket.on('close', function() {
      ...
   }.bind(this));

   socket.on('error', function() {
       ...
   }.bind(this));
}
```

Furthermore the image processing server tells the Node.js server when a player has been hit. Therefore a separate hit listener exists which gets the form of the hit and the precision passed as parameters. The precision is used to calculate the damage done by the shot.

Aside from those available listeners the image processing server accepts also a start, a stop and a shoot message. The start and stop message work in the same way as for the robots. The shoot message tells the image processing server that a specific player fired a cheese-bullet. This will trigger the needed logic and draw a bullet on the live stream.

2.6.3 Client

When a client registers himself on the server, the socket and its related user will be held in a list. Furthermore each client gets assigned to a robot. The robots are not defined by a name but by their form (e.g square, circle).

```
init: function(socket, params) {
   if (!!params.user) {
      ClientSockets[params.user] = socket;
      setRobotSocketForUser(params.user, params.form);
      ...
   }
}
```

Furthermore the client will accept following messages which will trigger an event on the client side: - **changedStatus** when a game state has changed - **hit** when a player has been hit - **connectionLost** when the connection to the image stream has been lost

2.6.3.1 Re-establishing the WebSocket connection to the server

When a client looses the WebSocket connection to the server (caused by an error), the client tries to reconnect to the server.

```
connection.onerror = function(error){
   console.error('Websocket error:',error);
   console.log('Trying to restart websocket...');
   this.init(username, form, wssUrl);
}.bind(this);
```

2.6.3.2 Re-establishing the stream connection

When the client looses its connection to the live stream the image processing server will inform the Node.js server and he will trigger an action on the client to reconnect (e.g. remove and add the stream-dom-element again).

2.7 Problems

2.7.1 MEAN.IO

MEAN.IO provides a lot of functionality which was useful, but it also complicated some things like e.g. the authentication via WebSockets or extending the user entity. It was not possible to get the session of a user when you do not use MEAN.IO's predefined structure. Furthermore it is also quite complicated to propagate new fields added to a user entity, because somewhere in the code of MEAN.IO some properties of the user object the get cut off.

These problems could have been solved easier but as MEAN.IO is quite a new framework it has also quite a small community and the resources in the internet are very few.

2.7.2 Certificate

Another problem was the certificate for the SSL connection. The different parties had different strict rules when to accept an SSL certificate. For example the Python implementation did not need a FQDN in the certificate but the image processing server (the used library was POCO) needed a FQDN or at least an IP address to accept the certificate. So it was not such a big problem but you have to know this. Otherwise you keep wondering why one party accepts the certificate and the other just declines it without telling you why.

2.8 Conclusion

Nonetheless it was interesting experience to work with such technologies!

3 Image processing server

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 game 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 the opportunity to develop with us. But the current version of the Boost library contains an already reported bug, which causes corrupt files when you build Boost with MinGW. This finally leads to an error in Eclipse, when you try to build the 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.".

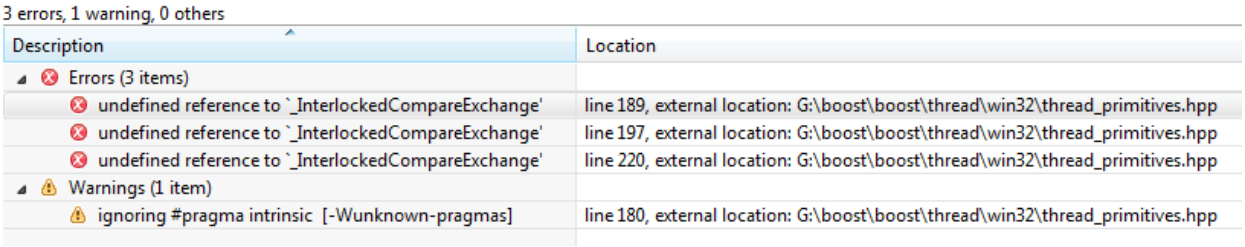

Figure 4 – Eclipse errors when building project

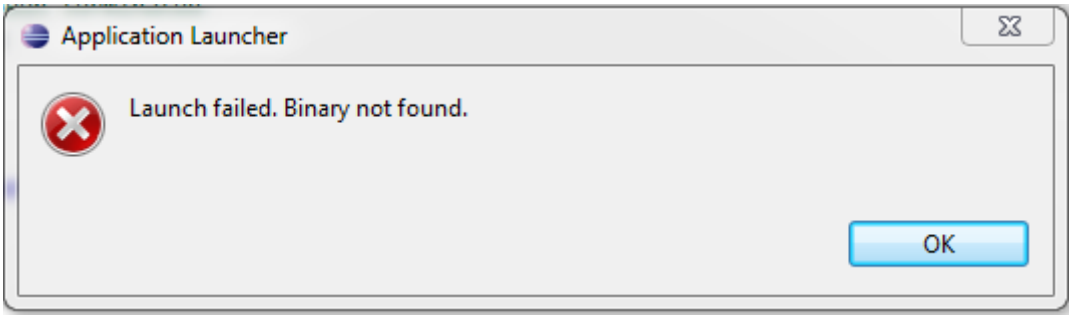

Figure 5 – Eclipse errors when building project

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 the reason why we changed from Eclipse to Visual Studio. But finally, when we were faced with the communication of the image processing server to the game 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\include\random line 2767
```

We could not figure out the source of this problem. 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\1.56.0\boost\function\function_template.hpp  153
```

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. 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
  - Shot/wall collision
- Position detection of walls and robots
- Communication with game server
- Simulation of shoots in the video stream

3.5.2 Communication with game server

It is necessary that the game server and the image processing server can talk with each other. The communication is needed, because the game 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 game server
  - Establishing the connection
  - If the video stream connection was lost to a client
  - If a shot hit a robot
- Messages by game server to image processing server
  - If a shot was made by a player
  - If game has stopped
  - If game has started

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 640x480 resolution with 15 frames per second 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 running the image processing server need the 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 1280x720 pixel for still image 1280x800. 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

3.6 Architecture

Figure 6 illustrates the component diagram of our program with the corresponding components and their package distribution and the relations between them.

3.6.1 Simulation Shot

Figure 7 illustrates how a the shot simulation process is done.

3.7 Object detection

3.7.1 Lessons learned

At the beginning we tried different detection approaches:

- RGB detection
- HSV detection
- Contour detection with marker
- Moving detection

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 anymore.

3.7.1.2 HSV color detection After the RGB detection we tried to detect the object via HSV colors. This worked a lot 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 anymore. 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

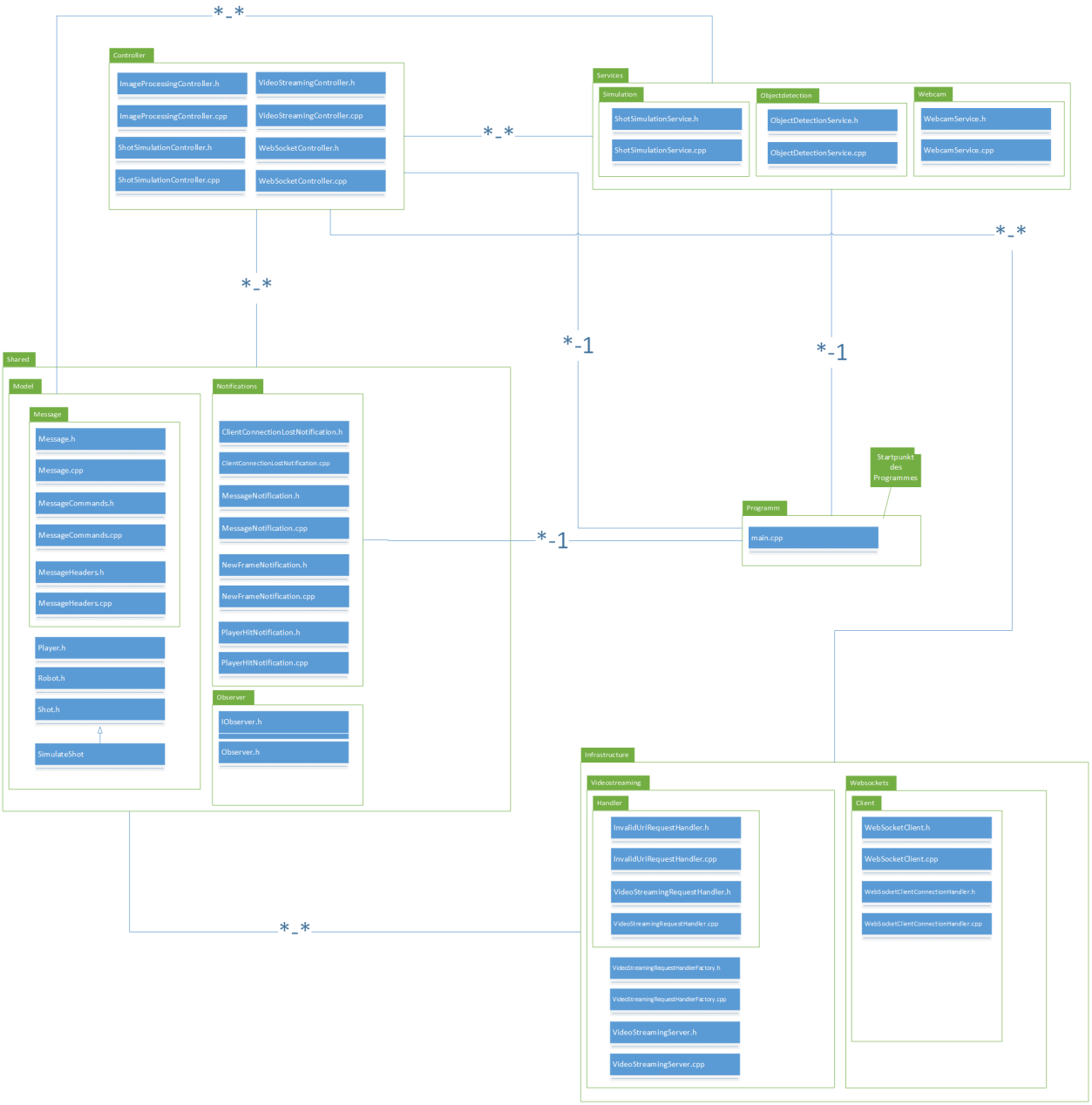

Figure 6 – component diagram

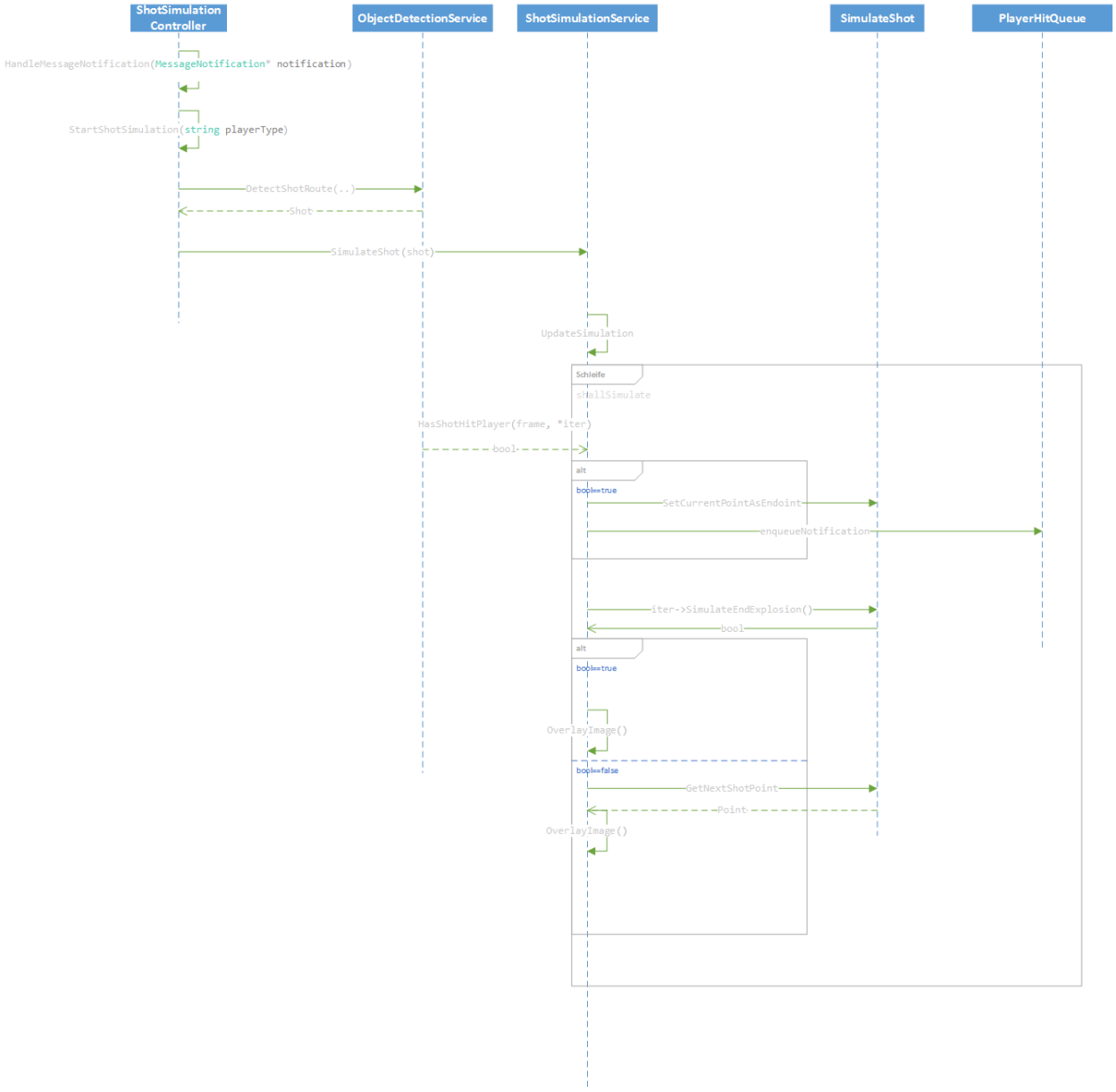

Figure 7 – sequence diagram shot

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. Figure 8 show this HSV model

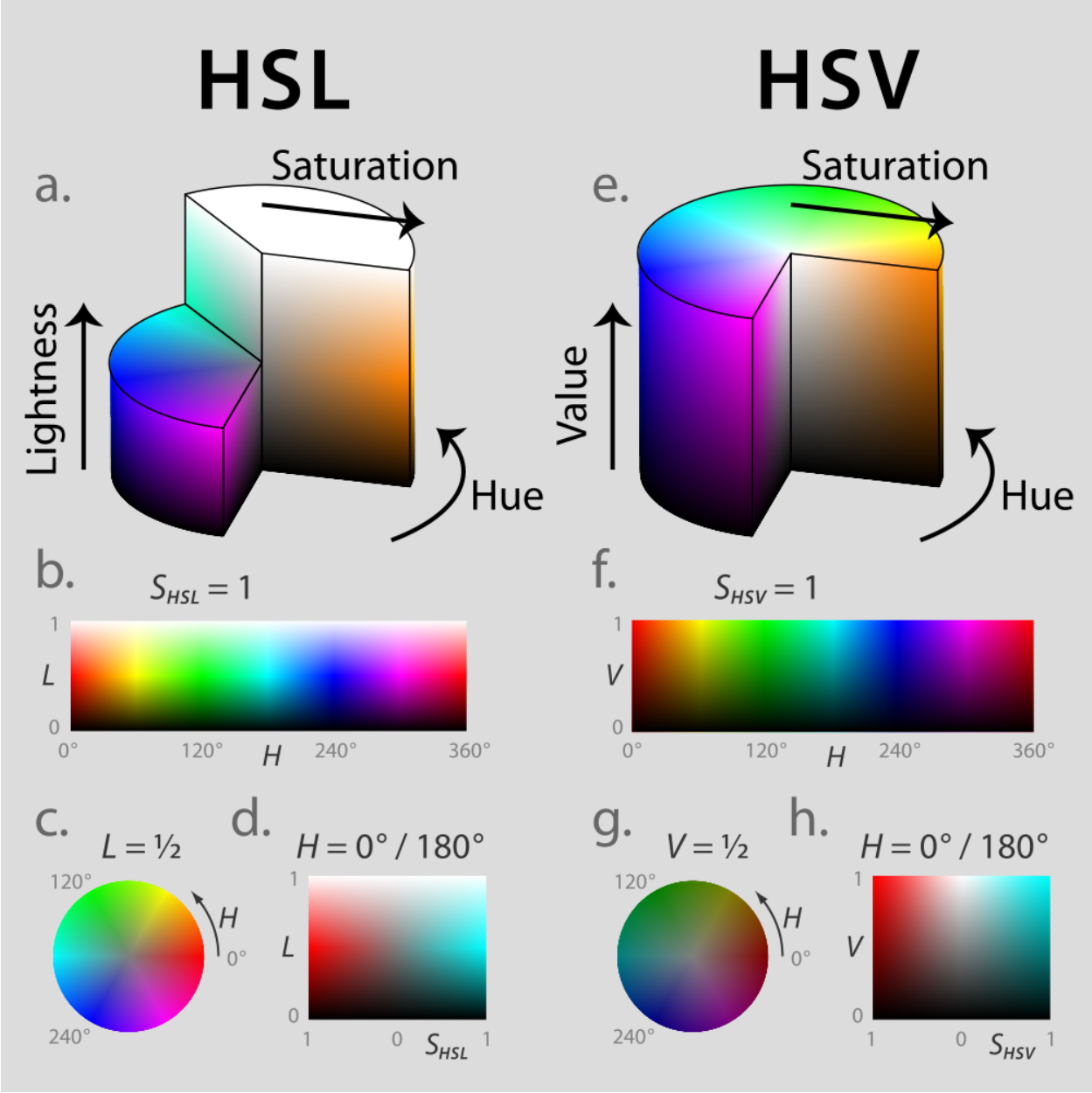

Figure 8 – HSV model

In figure 9 and figure 10 you can see the detection result of our HSV detection. In the first figure 9 you see the original image and the figure 10 then shows 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 first tried various geometry forms and tried to recognize them by there contours. On figure 11 you see the original image and on figure 12 the detection results.

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. Figure 13 and figure 14 show the detection with nested contours. Only the rectangles with triangles in the rectangles boundaries are detected.

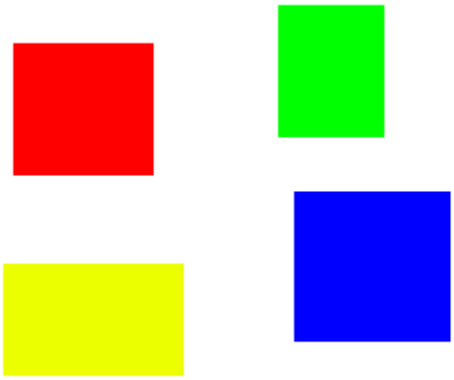

Figure 9 – HSV detection original image

Figure 10 – HSV detection after detect blue forms

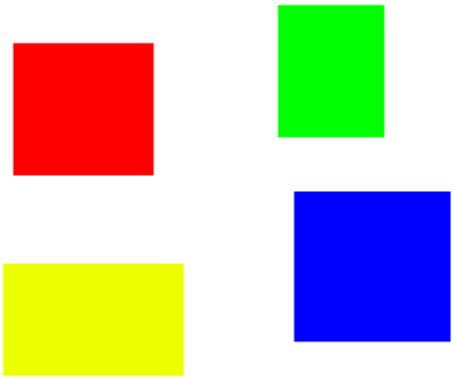

Figure 11 – Rectangle model

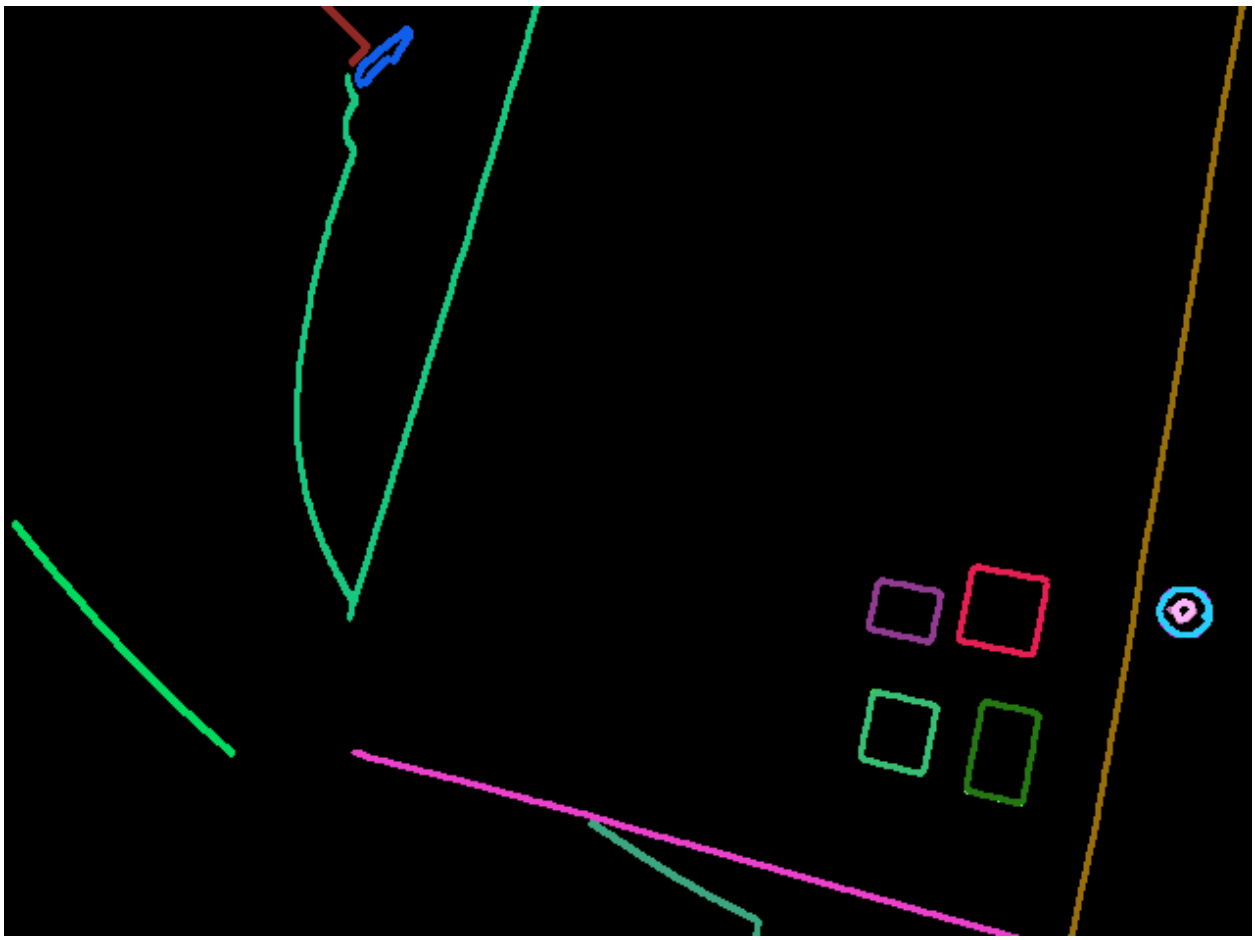

Figure 12 – Rectangle model after detection

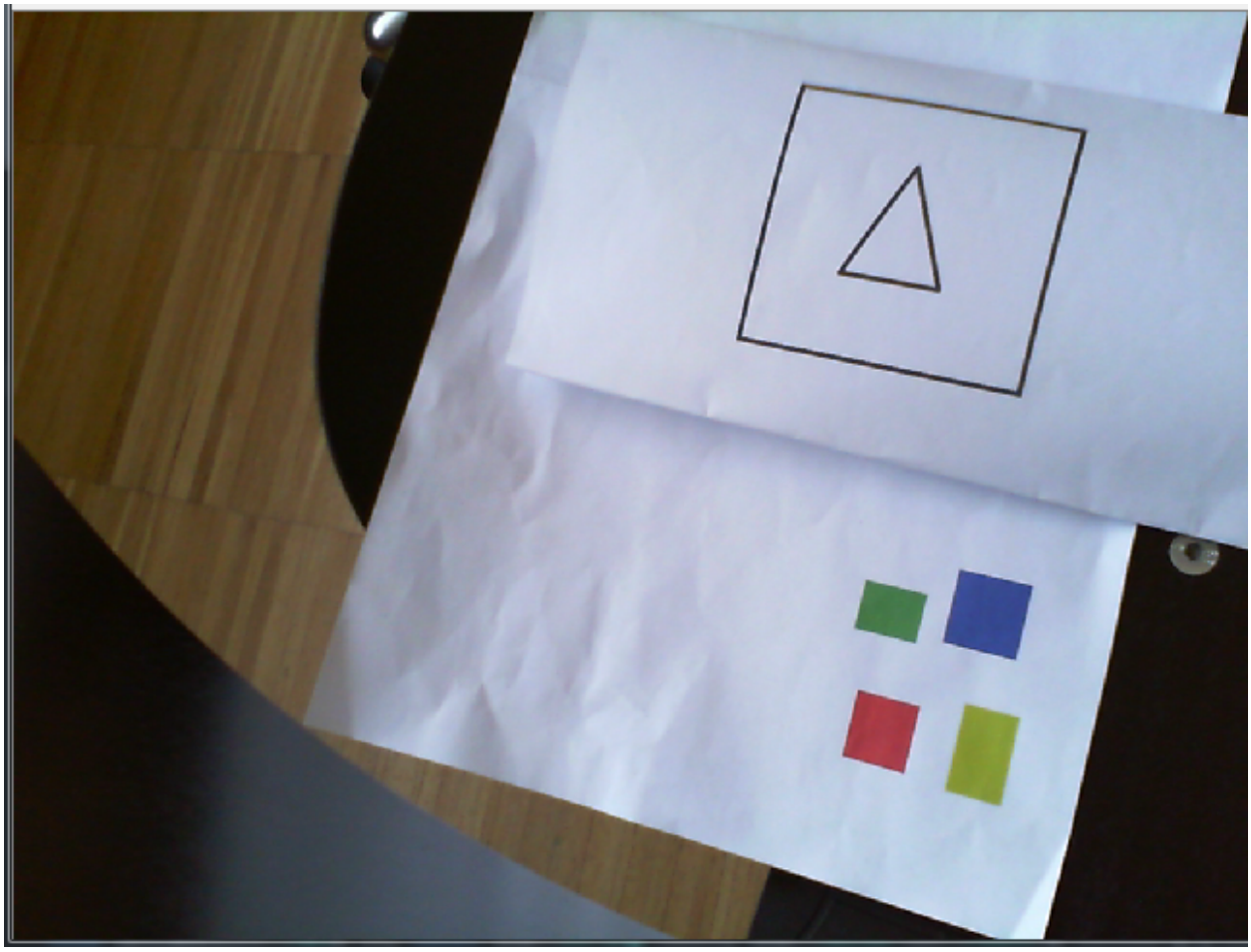

Figure 13 – Rectangle model nested

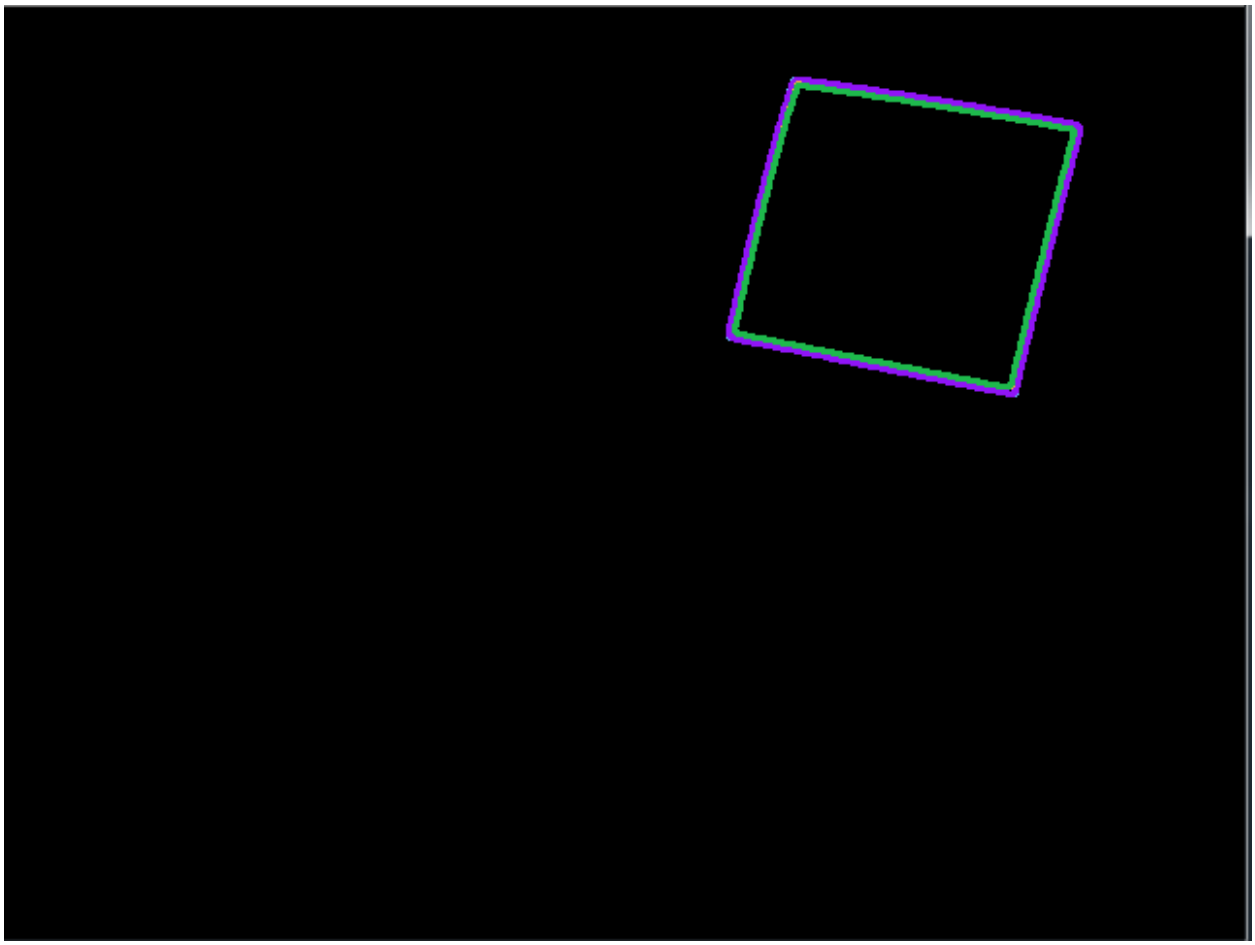

Figure 14 – Rectangle model nested after detection

3.7.1.4 Moving detection

We also tried to detect an object via his motion. This method has the advantage that the tracking distance is much higher as by the marker method. The disadvantage is that the robot must start in a certain position (cheese spin forward) to enable it using vector calculations to determine the new position of the cheese spin after a motion step. Furthermore, it is not possible to use this technology to turn the robot in the state, because that would have no effect on the motion detection. Because of the fact that the detection of the front of the robot is easier with the marker detection and also the detection of the front after a robot turn in the state we decided to not use the moving detection.

3.7.1.5 Conclusion Lessons learned

After our tests we decided to use contour detection in combination with HSV for our robot detection. The reason for this is that the detection via contours and HSV works faster, more stable and produces less errors during the detection process. We also decided not to use the moving detection because of the fact that the detection of the front of the robot is easier with the marker detection. Also the detection of the front after a robot turned in a static position.

3.7.2 Object detection realization

3.7.2.1 Robot detection

We decided to detect the robots via a contour detection method in combination with a HSV filtering

In order to bring more security in the contour detection, we have decided to use nested contours instead of simple contours. This enables us to detect the object more error-free and more stable then with simple contours.

We decided to used the following geometric forms: * White Rectangle with isosceles black triangle in it * Black Pentagon with isosceles white triangle in it

But at the end of our project we found out that the pentagon is not the best opposite form for the rectangle. Instead of the pentagon we now use a black circle with a isosceles white triangle in it.

3.7.2.1.1 Detection processing

In our detection process for a robot we first convert the original image of the video stream in to another color space via the following method

```
cvtColor(srcdetect2, imgHSV, COLOR_BGR2HSV);
```

We convert the original image BGR space into a HSV space and save this in a template image. After this converting process we use the following range function to find all white objects on the image.

```
int iLowH = 0;
int iHighH = 179;

int iLowS = 0;
int iHighS = 244;

int iLowV = 0;
int iHighV = 245;

inRange(imgHSV, Scalar(iLowH, iLowS, iLowV),
        Scalar(iHighH, iHighS, iHighV), imgThresholded);
```

After this function we only have a image which contains the white triangle of the circle marker and a white rectangle with a hole in shape of a triangle of the rectangle marker. In the next step we called the erode and dilate function on the image.

```
erode(imgThresholded, imgThresholded,
    getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
dilate(imgThresholded, imgThresholded,
    getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
```

The main objective of erode and dilate is to reduce noise. Such noise reduction is a typical image pre-processing method which will improve the final result.

The next step is that we make a canny edge detection to extract the edges

```
Canny(imgThresholded, canny_output, threshdetect2, threshdetect2 * 2, 3);
```

Canny algorithm aims to satisfy three main criteria: * Low error rate: Meaning a good detection of only existent edges. * Good localization: The distance between edge pixels detected and real edge pixels have to be minimized. * Minimal response: Only one detector response per edge.

The values of the thresholds must be set before the games started at the moment the thresholds are set between 30 to 60. The selected threshold value depends on the distance to the object and the environment of the detection area.

After the canny detection we run a threshold function over the image. Therefore we use a binary threshold method:

```
threshold(canny_output, canny_output, 128, 255, CV_THRESH_BINARY);
```

After all this preparation steps we call the find contours method which returns all founded contours.

```
findContours(canny_output, contours, hierarchy, CV_RETR_TREE,
                CV_CHAIN_APPROX_SIMPLE);
```

Contours: Detected contours. Each contour is stored as a vector of points. **Hierarchy:** Optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0], hiearchy[i][1], hiearchy[i][2] and hiearchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative. **CV_RETR_TREE:** Retrieves all of the contours and reconstructs a full hierarchy of nested contours. **CV_CHAIN_APPROX_SIMPLE:** Compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example an up-right rectangular contour is encoded with 4 points.

After this step we iterate over each founded contour and try to find our rectangle and triangle forms. In each iteration step we first call the approxPolyDP method, which approximates polygonal curves with the specified precision. It uses the Douglas-Peucker algorithm.

```
approxPolyDP(cv::Mat(contours[i]), approx,
    cv::arcLength(cv::Mat(contours[i]), true)*0.1, true);
```

Then we skip small or non-convex objects and only extract rectangles and triangles. The code below shows the rectangle marker detection:

```
if (std::fabs(cv::contourArea(contours[i])) < 100 || !cv::isContourConvex(approx))
    continue;

// Rectangles

if (approx.size() == 4)
{
    rectangles.push_back(contours[i]);
    rectanglesContourPositions.push_back(i);
}

if (approx.size() == 3)
{
    triangles.push_back(approx);
    trianglePositions.push_back(i);
}
```

After that we check via the pointPolygonTest method of OpenCV if one of the located triangles is in the located rectangle. This method returns the position of the triangles. This must be done to evaluate which triangle depends to which marker. The triangle which is in the rectangle contains to the rectangle marker.

With this information we can calculate the front and the throwing direction of the cheese spin. Below you can see the method which calculates this:

```
Point p1 = tri[0];
Point p2 = tri[1];
Point p3 = tri[2];

Point lP1P2 = p1 - p2;
Point lP2P3 = p2 - p3;
Point lP3P1 = p3 - p1;

double l1 = sqrt((lP1P2.x*lP1P2.x) + (lP1P2.y*lP1P2.y));
double l2 = sqrt((lP2P3.x*lP2P3.x) + (lP2P3.y*lP2P3.y));
double l3 = sqrt((lP3P1.x*lP3P1.x) + (lP3P1.y*lP3P1.y));

double shortest = l1;
Point toReturn = p3;
Point dir = Point(lP1P2.x / 2, lP1P2.y / 2) + p2;

if (l2 < shortest)
{
    shortest = l2;
    toReturn = p1;
    dir = Point(lP2P3.x / 2, lP2P3.y / 2) + p3;
}
if (l3 < shortest)
```

```
{
    shortest = l3;
    toReturn = p2;
    dir = Point(lP3P1.x / 2, lP3P1.y / 2) + p1;
}

Point direction = toReturn - dir;
vector<Point> points;
points.push_back(direction);
points.push_back(dir);
return points;
```

After we finished the robot detection we return a robot object with the position information:

```
if (pointsTriRect.size() == 2)
    return Robot(pointsTriRect[1], pointsTriRect[0], contoursRect);
```

3.7.2.2 Shoot route calculation

To detect the shoot route we perform following steps: First we get the actual position of the shooting robot. The actual position can be found in the representing robot objects. This robot objects are automatically updated by our robot position update process which tracks the robots every second frame.

Second we calculate the normalized shooting direction vector. The shooting direction was calculated in the robot detection process. We added a multiplier to the normalized vector to reduced the time for the calculation.

```
double length = sqrt(pow(actuelRobot->shotDirection.x, 2) + pow(actuelRobot->shotDirection.y
int multiplier = 5;
Point normDirection = Point(actuelRobot->shotDirection.x / length * multiplier, actuelRobot-
```

After that we calculation the shoot route started from the cheese spin of the shooting robot to the end of the playing area (image borders). For that we start an iteration and in each of this iterations we add the normalized shooting direction vector of the shooting player to the shooting start point. We call this procedure as long as we run out of the playing area (reached the border of the image):

```
while (!found)
    {
        if (!rect.contains(currentPoint))
        {
            endPoint = currentPoint - normDirection;
            found = true;
        }

        currentPoint += normDirection;
    }
```

At least we return the shot route to the caller method

```
return Shot(player, hitPlayer, Point2i(actuelRobot->shotStartingPoint.x,
            actuelRobot->shotStartingPoint.y), Point2i(endPoint.x, endPoint.y));
```

3.7.2.3 Hit detection

For the hit detection we perform the following steps. First we fetch the actual position of the thrown cheese in the shot route.

```
Point2i tmp = shot.GetCurrentShotPoint();
```

After that we get the actual position of the shooting robot. The actual shooting position can be found in the representing robot objects. This robot objects are automatically updated by our robot position update process which tracks the robots every second frame.

After that we define the hit area but only if it is possible that the shot hit the robot. If the distance of the shoot and the hit robot is to far we automatically return false.

```
int diffx = abs(actuelPosition->x - currentShotingPoint.x);
int diffy = abs(actuelPosition->y - currentShotingPoint.y);

if (diffx > 100 || diffy > 100)
{
    return false;
}

vector<Point> hitArea;

Point x(actuelPosition->x - 50, actuelPosition->y - 50);
Point y(actuelPosition->x + 50, actuelPosition->y + 50);
Point z(actuelPosition->x - 50, actuelPosition->y + 50);
Point v(actuelPosition->x - 50, actuelPosition->y + 50);


hitArea.push_back(x);
hitArea.push_back(y);
hitArea.push_back(z);
hitArea.push_back(v);
```

With the shooting point and the hit area we determine if the point is in the hit area of the robot via the pointPolygonTest method. The result will be returned to the caller method.

```
if (pointPolygonTest(Mat(hitArea), currentShotingPoint, true) > 0)
    return true;
else
    return false;
```

3.7.3 Performance measurement object recognition

During our project we had the problem that our webcam streaming was very slow. After hours of searching we found out that one reason for the problem was that our object detection recognition was to slow. So we started a performance measurement were we measured each method call. By this time measurement, we found out that the most time of our object recognition is needed by OpenCV methods like cvtColor, blur, findContours and so on. On figure 15 you can see the results of this measurement.

According to this knowledge we have tried to improve the image processing. We enlarged the size of the objects for which we search. This brought us an improvement of about 20ms. Further we

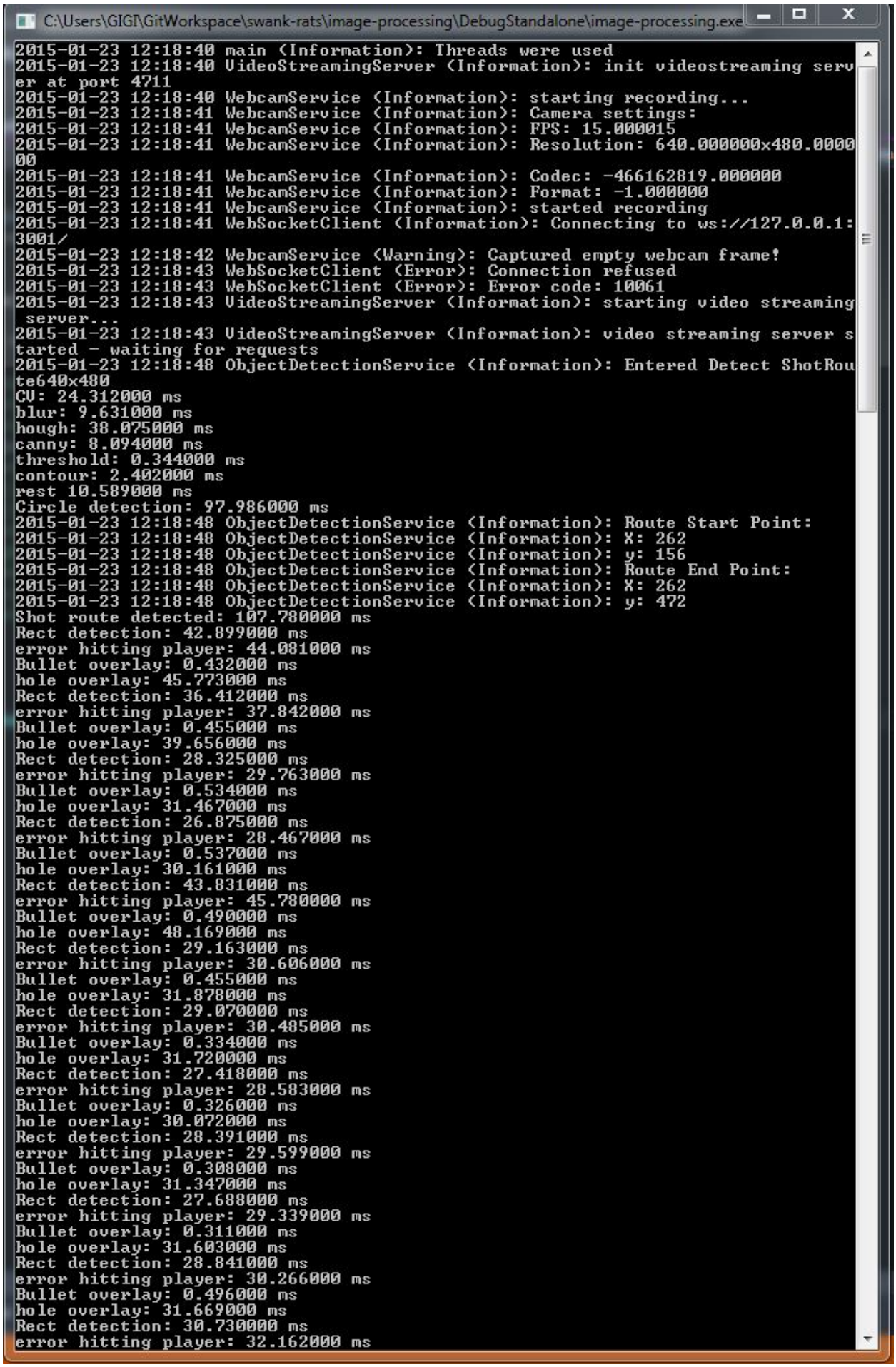

Figure 15 – image processing time

decided to track the robots continuously on every second frame. The reason why we skip one frame is again just a performance improvement, since the robots can not move that fast between 2 frames. So when the server starts an initial position detection of the robots on whole frame takes place. The needed time for this operation does not matter since it happens on start up. Once we have detected the position of the robots we store this information and use it to reduce the area where we have to search for the robots. We have introduced a 100x100px region of interest (ROI) around the last known position of a robot. This continuous tracking reduced the needed time for position detection to about 8ms per robot.

3.8 WebSocket communication

3.8.1 Connection establishment

The image processing server contacts the game server. The IP address of the game server is passed as command line argument. See figure 16.

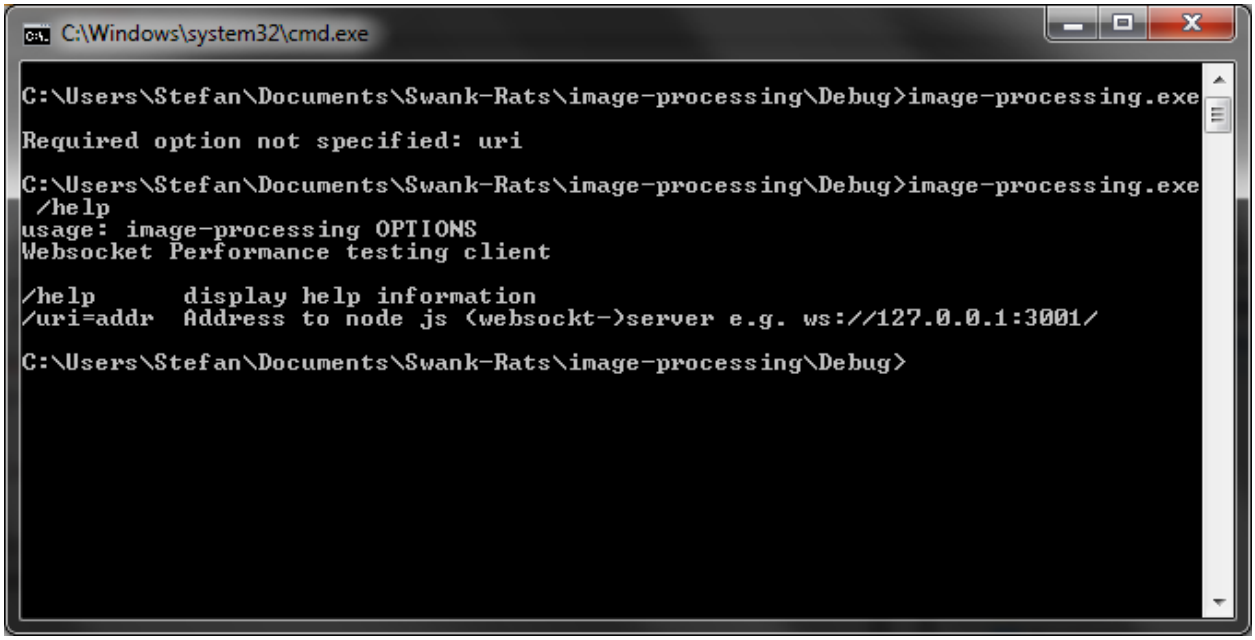

Figure 16 – Image processing server command line arguments

3.8.2 Handling of connection loss

If the connection to the game server gets lost we try to reconnect for one second. In this time all outgoing messages will be buffered and if a reconnect was successful, they will be sent. Otherwise they will be lost and not sent. After a reconnect we try again for one second to re-establish the connection.

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 figure 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.

Figure 17 – 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. MJPEG always requires 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

Handling of connection loss is not that easy since MJPEG functionality is embedded into the clients browser. But since the image processing server knows the IP address of the connected clients and the game server knows the IP address of the players, the image processing server notifies the game server about the connection loss. Afterwards the game server sends a message to the browser of the client which causes a reconnect by the browser.

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 to 18 kB by setting the quality to 30 % or the original image.

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.

We could also figured out two bottlenecks in our code. The encoding of a frame needs about 30 ms and this was done for each stream connection separately. This is now done only once per frame. Another mistake were the following lines of code:

```
....
std::ostream& out = response.send(); //output stream
....
vector<uchar>* buf = webcamService->GetModifiedImage(); //get frame
....
std::string content = std::string(buf.begin(), buf.end());
....
out << content;
....
```

The problem with those lines is the conversion to a string. This needs between 20 - 30 ms. We could improve this by reinterpret the vector containing the image like this:

```
....
std::ostream& out = response.send(); //output stream
....
vector<uchar>* buf = webcamService->GetModifiedImage(); //get frame
....
out.write(reinterpret_cast<const char*>(buf.data()), buf.size());
....
```

Figure 18 shows the performance improvement.

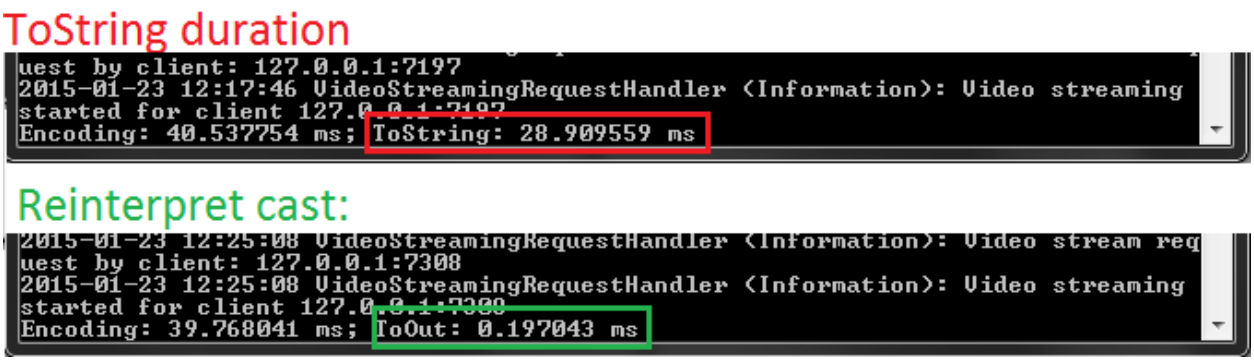

Figure 18 – Stream output improvement

3.9.6 Traffic

Out measurements showed that in 60 seconds play-time between 900 to 1000 frames with a total amount of 13 to 18 MB data were transferred. See figure 19 for one measurement result.

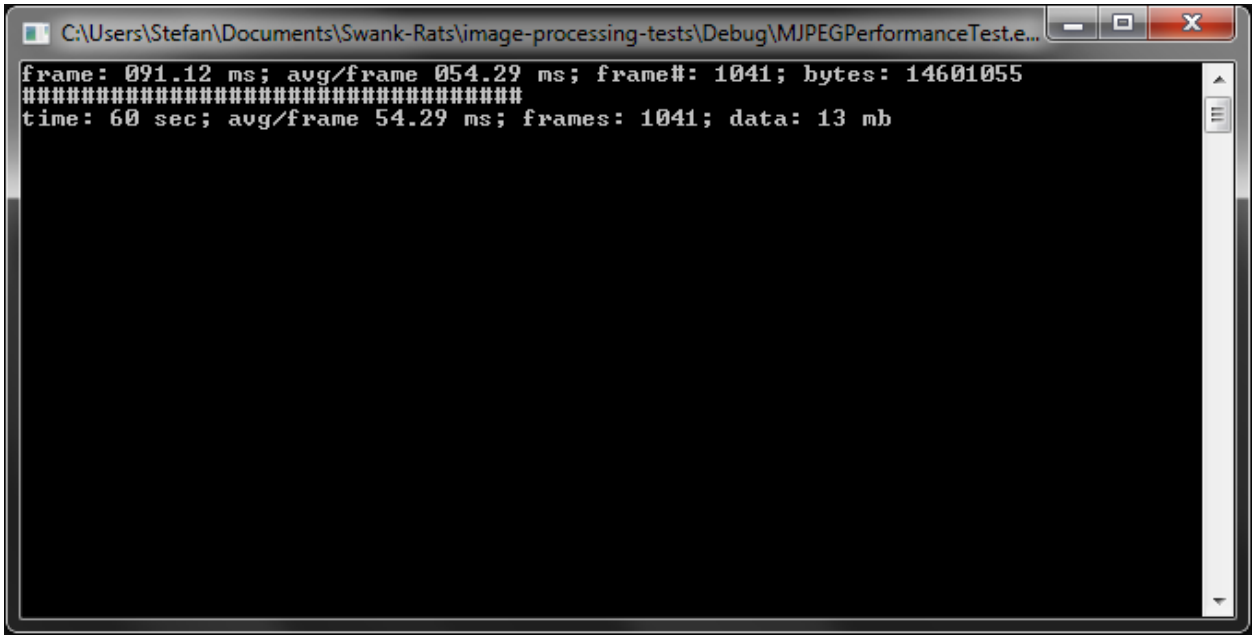

Figure 19 – Screenshot of one measurement result

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. Cheese-throwing-animation: to simulate a flying cheese beginning at the robot and ending at the calculated end position
2. Hit-animation: when the flying cheese reaches the end position a explosion is simulated

Figure 20 illustrated both states.

A simulation is started if the game 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

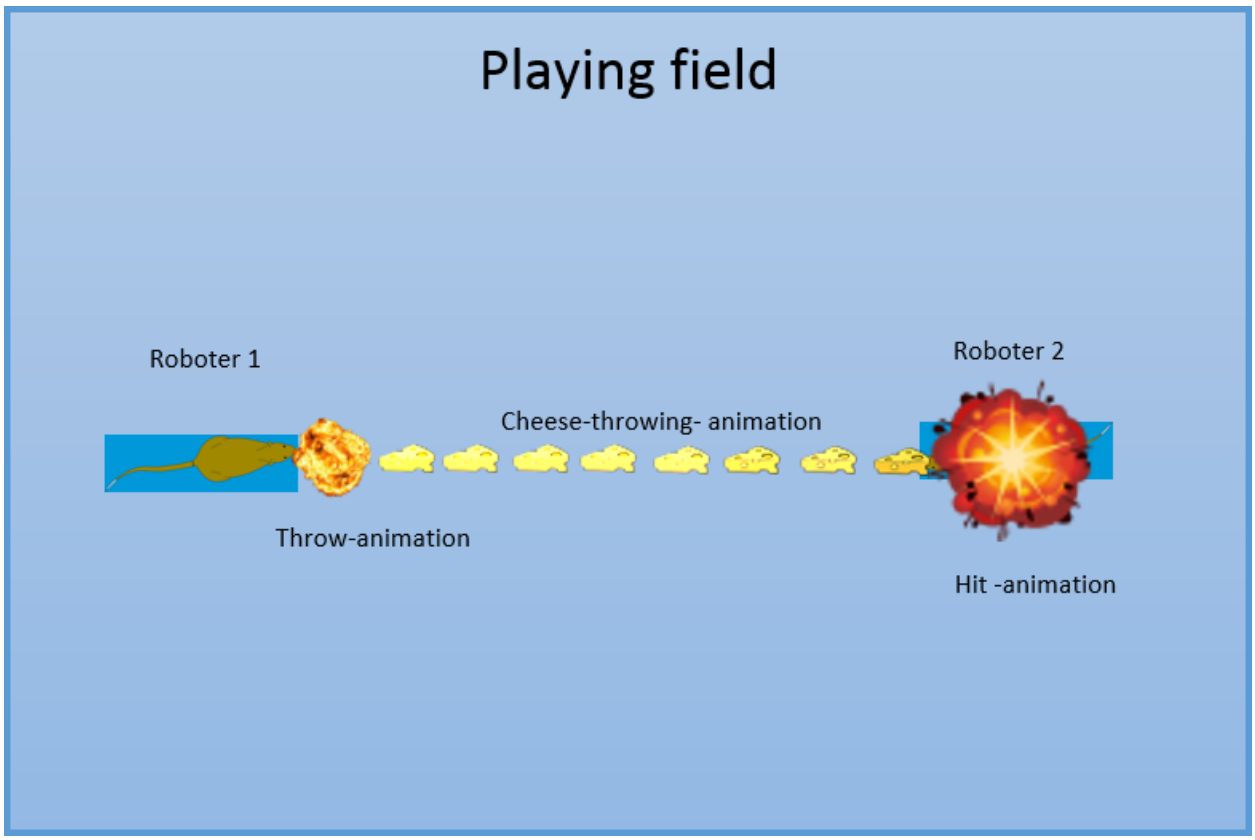

Figure 20 – Cheese-throw simulation states

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 figure 21 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
- 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. Figure 22 illustrates all directions.

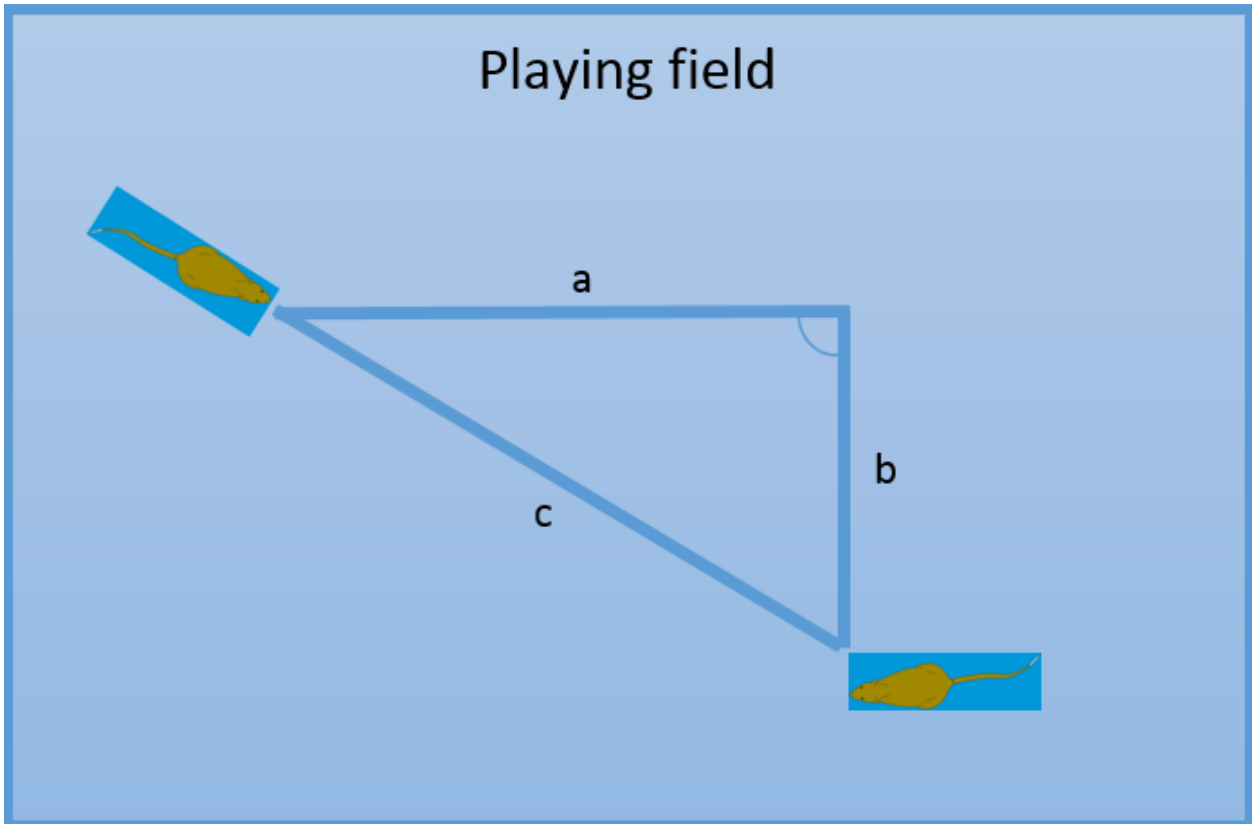

Figure 21 – Cheese-throw simulation right-angled trigangle

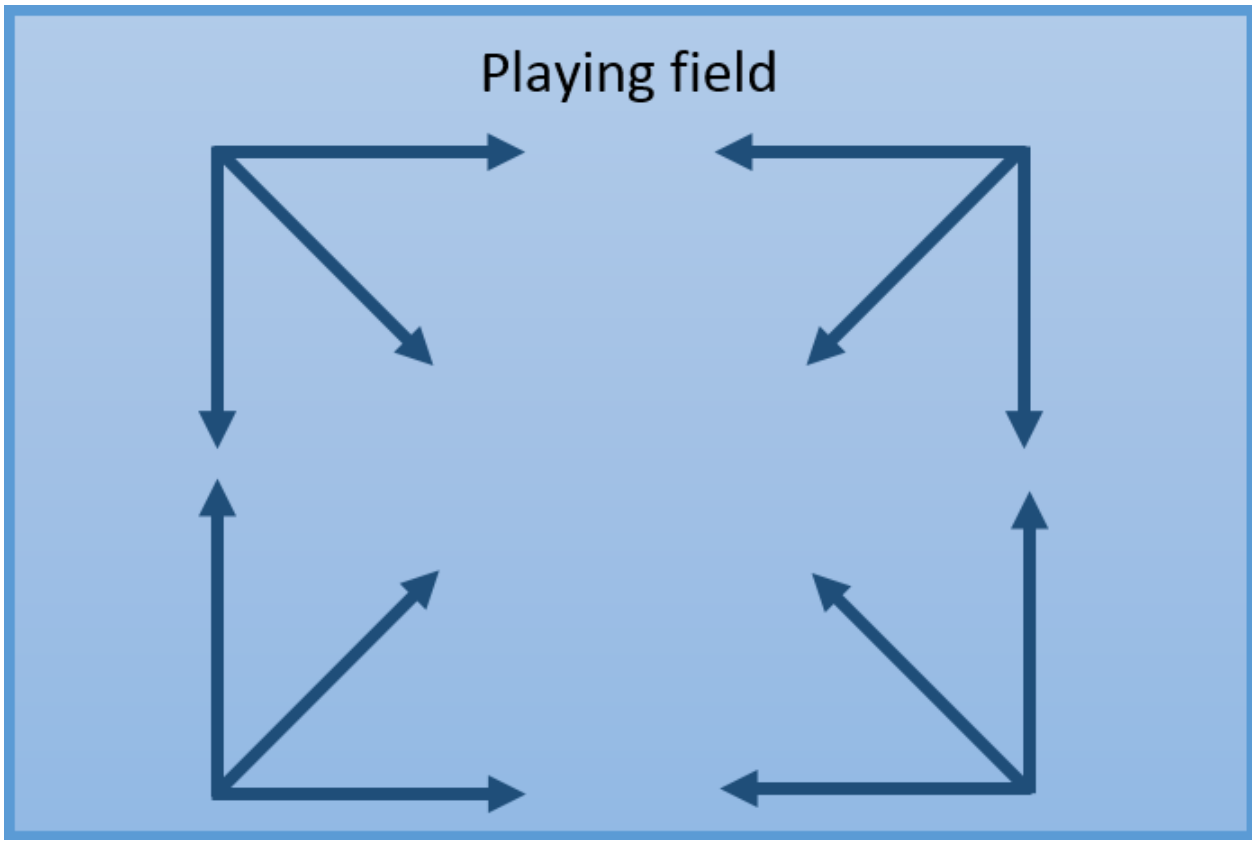

Figure 22 – Cheese-throw directions

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 have 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.

4 Communication

For the communication the system uses WebSocket on all stations. The protocol we use is self-defined and designed for our usage. But it is fully unit tested and it will provided by npm for third party usage.

4.1 WebSocket

WebSocket is a TCP based protocol which provides a bidirectional connection between a client and a server. Originally it is developed for real-time communication between web server and browser.

WebSocket uses the upgrade mechanism of HTTP 1.1.

4.1.1 Handshake

Before a WebSocket connection will be opened a HTTP request will be executed.

The request is nearly normal HTTP but there are some additional headers.

```
GET /socket HTTP/1.1
Host: thirdparty.com
Origin: http://example.com
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Extensions: x-webkit-deflate-message, x-custom-extension
```

Connection: Upgrade and Upgrade: websocket Request to perform an upgrade to WebSocket protocol

Sec-WebSocket-Version: 13 WebSocket protocol version used by the client

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== Auto-generated key to verify server protocol support

Sec-WebSocket-Protocol: chat Optional list of subprotocols specified by the application

Sec-WebSocket-Extensions: ... Optional list of extensions specified by the server

The server send a response with status-code `101: Switching Protocols`.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Access-Control-Allow-Origin: http://example.com
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Extensions: x-custom-extension
```

The connection stays open and the server or client can send data over the open TCP connections.

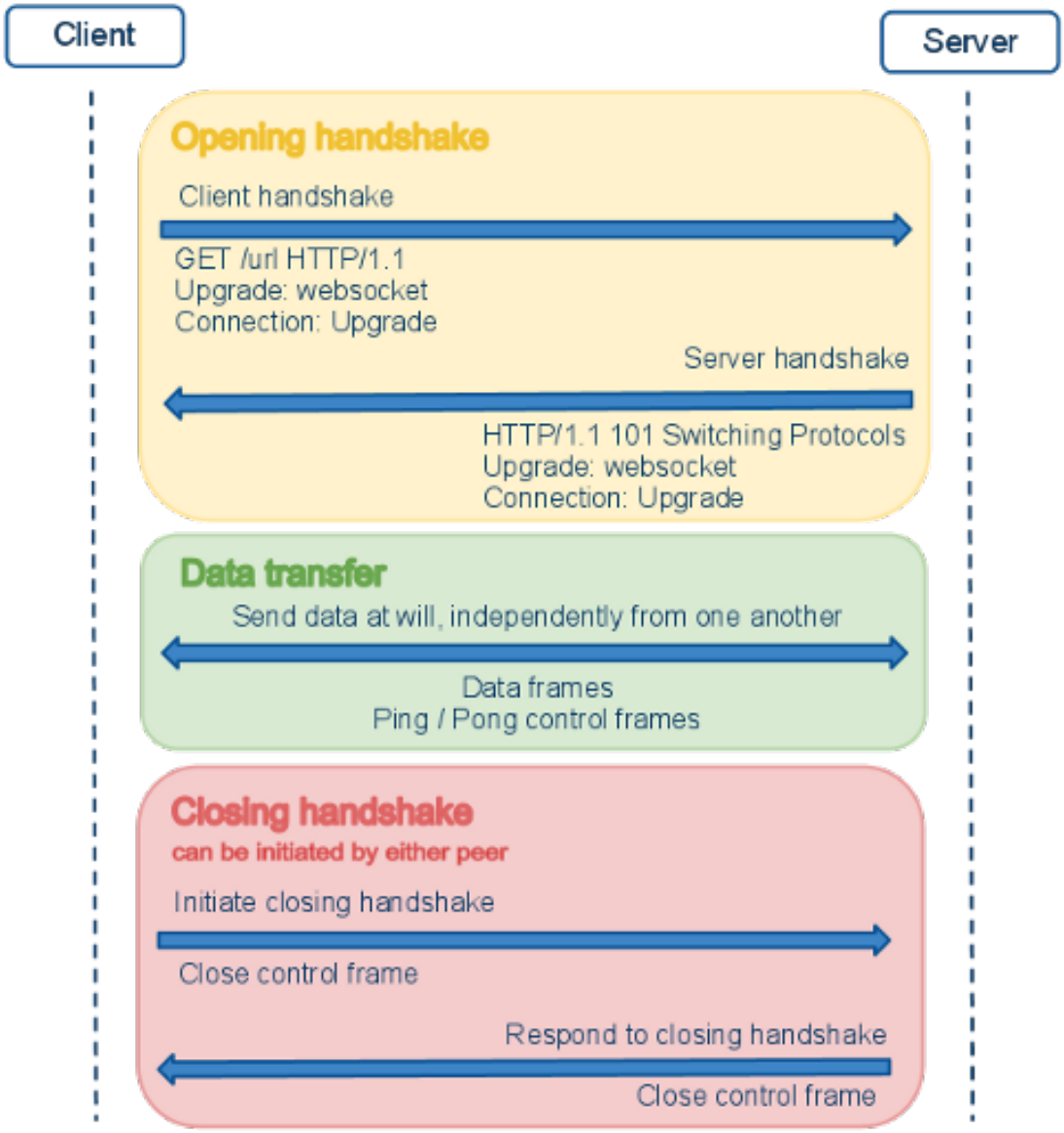

Figure 23 – WebSocket handshake

4.1.2 Data protocol

The data is minimally framed, with a small header followed by payload. WebSocket transmissions are described as "messages", where a single message can optionally be split across several data frames. http://en.wikipedia.org/wiki/WebSocket The receiver will be informed after receiving a full message.

Full description under http://presentation.asapo.at/websocket

4.2 Application protocol

The defined protocol uses JSON-encoded objects. These objects can be send in UTF-8 strings. The protocol and the library are able to delegate messages to a defined handler. The listener will be defined in the property `to`. The property `cmd` defines which function will be called for the given listener.

Example message:

```
{
    to: 'test',
    cmd: 'echo',
    params: {
        toUpper: true
    },
    data: 'testdata'
}
```

Example listener:

```
var echoListener = {
    echo: function(socket, params, data) {
        if (!!params.toUpper) {
            data = data.toUpperCase();
        }
        socket.send(data);
    }
};
websocketServer('test', echoListener);
```

CLIENT (wscat):

```
wscat --connection localhost:8080
> {"to":"test", "cmd":"echo", "params":{"toUpper":true}, "data":"testdata"}
  TESTDATA
```

The example describes how the messages and the listener should be structured.

Properties:

- to: used to find listener on the server
- cmd: command name to find function name in listener
- params & data: will be passed to the function

Remark: If no `cmd` is defined a `default` callback on the listener is called.

The communication library is only full implemented for the node.js server. The clients uses the raw JSON-string.

4.3 Traffic

Each packet has a size between 30 to 60 bytes. But 20 bytes of the packets are overhead caused by JSON and the protocol design (“to”, “cmd”,...).

5 Roboter

The Turtle 2WD Mobile Platform is a robot and will be controlled (in our case) by a person via a browser. In this chapter this robot will be described in more detail.

Figure 24 – 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.

- Chassis: The chassis is a round robot which has two electric motors and two wheels, which allows the robot to corner sharply.

- Energy supply: For the energy supply we use 5 batteries to power the motor and 4 Batteries to supply the BeagleBone (for example 5V).
- MotorControllerCape: Expands the board with the ability to control motors over a simple library.

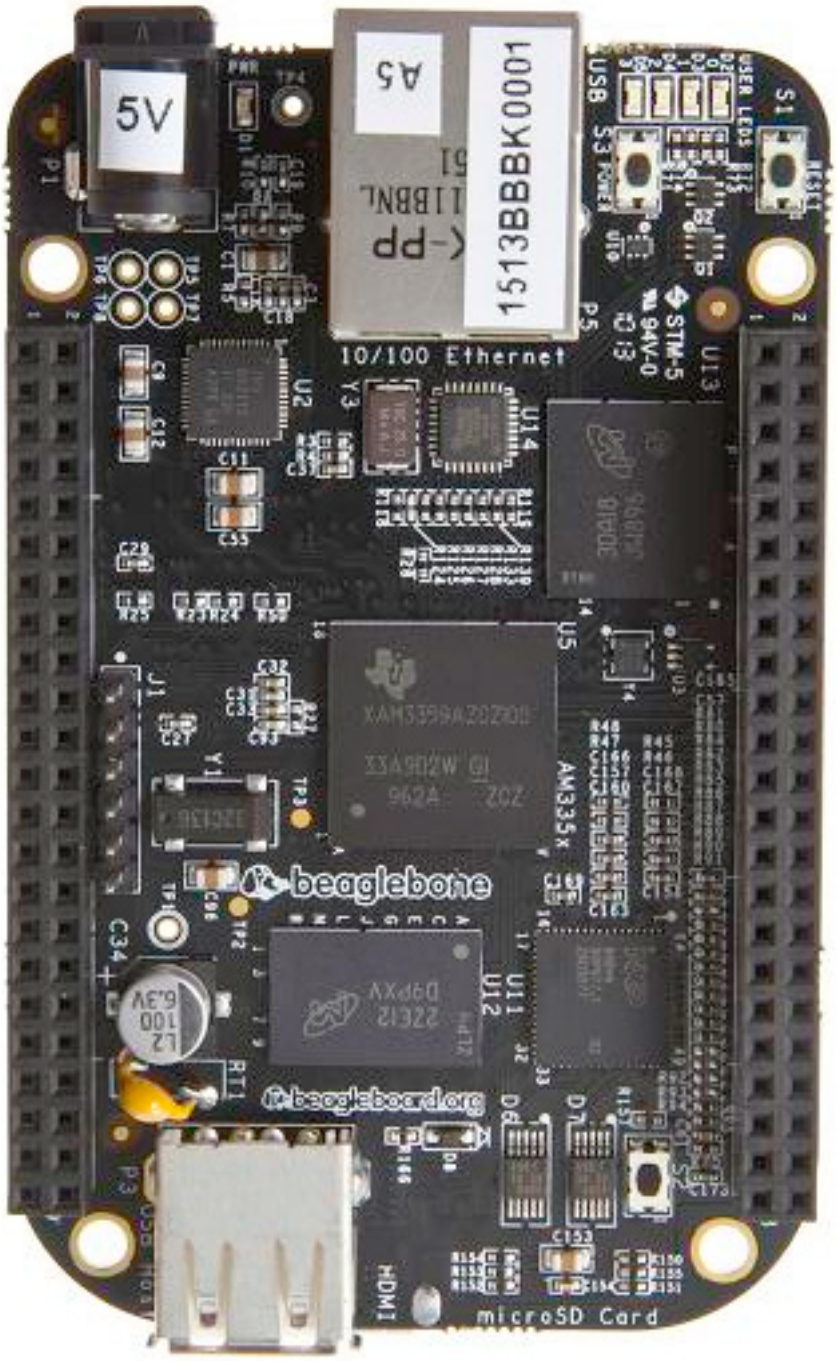

Figure 25 – BeagleBoneBlack from above

Figure 26 – Chassis example picture bottom

Figure 27 – Chassis example picture top

5.2 Specification

- BeagleBone Black is a low-cost, community-supported development platform for developers and hobbyists. Spezifications:
- Am335x 1GHz ARM Cortex-A8 Prozessor
- 12MB RAM
- 4GB Onboard eMMC Flash Speicher
- Motor
- Operating Voltage Range: 3~7.5V
- Rated Voltage: 6V
- Max. No-load Current(3V): 140 mA
- Max. No-load Current(6V): 170 mA
- No-load Speed(3V): 90 rpm
- No-load Speed(6V): 160 rpm
- Max. Output Torque: 0.8 kgf.cm
- Max. Stall Current: 2.8 A

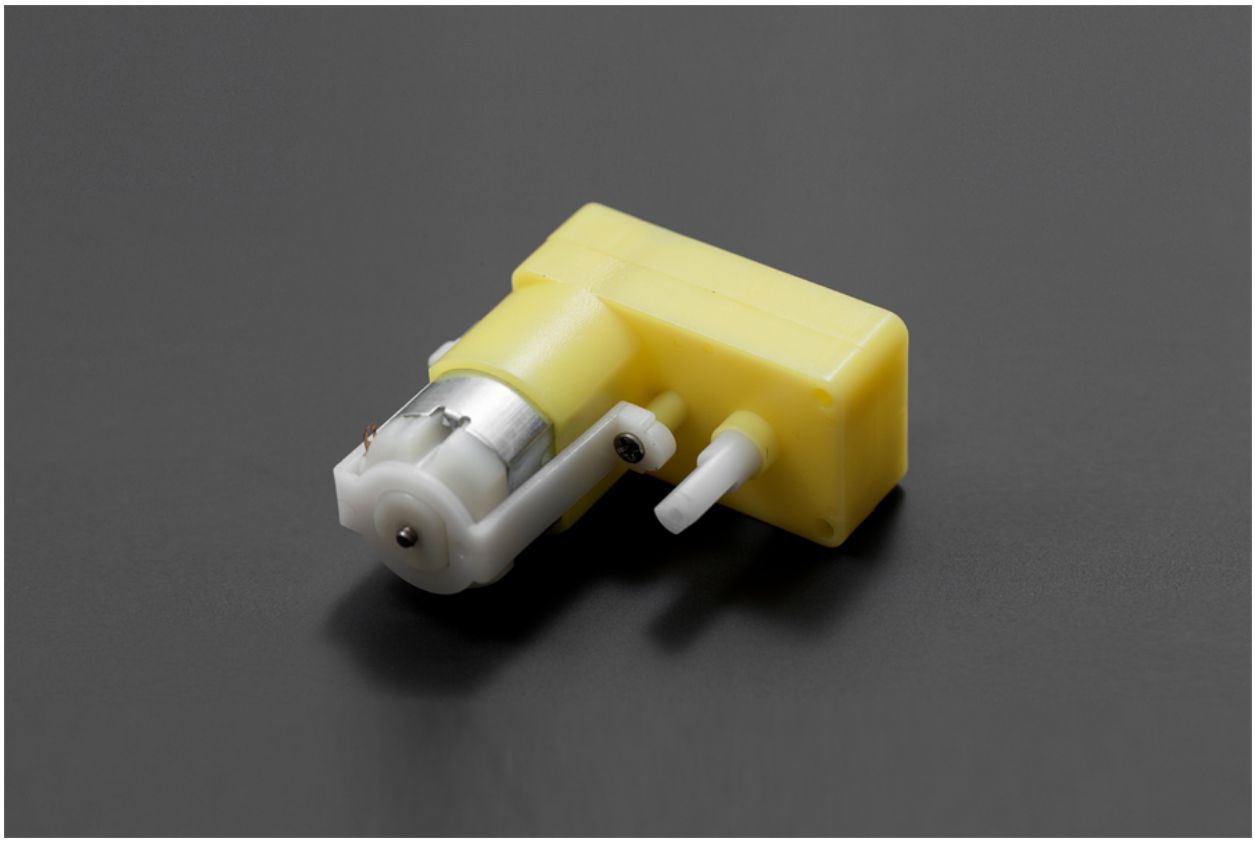

Figure 28 – Motor Picture

It can be ordered here.

5.3 Schema

See figure 26.

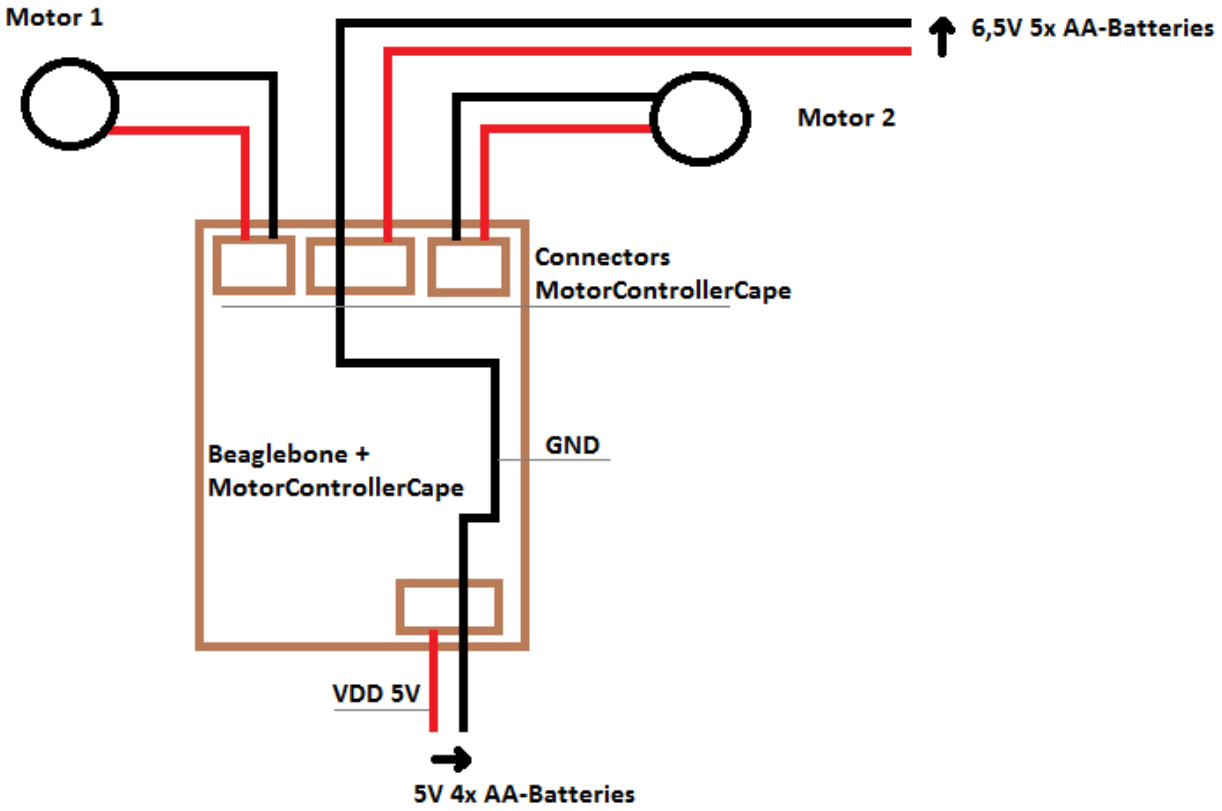

Figure 29 – Schematics

5.4 Energy consumption

5.4.1 BeagleBone Black

We measured 300mA current at 5V energy consumption. During booting it went up to 500mA peak.

5.4.2 Motors

The library DMCC provides a little tool to monitor the current consumption of the connected motors.

This output was produced when the robot drives with 40% which is the highest possible level without exceeding the current limiter of the motor control cape.

```
$ root@beaglebone:~/DMCC_Library# ./getCurrent 0
Current Motor 1 = 81 (0x51), Motor 2 = 71 (0x47), Voltage = 6322 (0x18b2)
Current Motor 1 = 64 (0x40), Motor 2 = 70 (0x46), Voltage = 6259 (0x1873)
Current Motor 1 = 73 (0x49), Motor 2 = 73 (0x49), Voltage = 6277 (0x1885)
Current Motor 1 = 65 (0x41), Motor 2 = 69 (0x45), Voltage = 6341 (0x18c5)
Current Motor 1 = 87 (0x57), Motor 2 = 75 (0x4b), Voltage = 6341 (0x18c5)
Current Motor 1 = 90 (0x5a), Motor 2 = 72 (0x48), Voltage = 6268 (0x187c)
```

```
Current Motor 1 = 85 (0x55), Motor 2 = 72 (0x48), Voltage = 6341 (0x18c5)
Current Motor 1 = 73 (0x49), Motor 2 = 70 (0x46), Voltage = 6350 (0x18ce)
Current Motor 1 = 68 (0x44), Motor 2 = 76 (0x4c), Voltage = 6268 (0x187c)
Current Motor 1 = 85 (0x55), Motor 2 = 70 (0x46), Voltage = 6341 (0x18c5)
Current Motor 1 = 83 (0x53), Motor 2 = 70 (0x46), Voltage = 6322 (0x18b2)
Current Motor 1 = 76 (0x4c), Motor 2 = 78 (0x4e), Voltage = 6259 (0x1873)
Current Motor 1 = 85 (0x55), Motor 2 = 67 (0x43), Voltage = 6295 (0x1897)
Current Motor 1 = 69 (0x45), Motor 2 = 71 (0x47), Voltage = 6322 (0x18b2)
Current Motor 1 = 70 (0x46), Motor 2 = 69 (0x45), Voltage = 6240 (0x1860)
Current Motor 1 = 79 (0x4f), Motor 2 = 67 (0x43), Voltage = 6304 (0x18a0)
Current Motor 1 = 78 (0x4e), Motor 2 = 70 (0x46), Voltage = 6350 (0x18ce)
```

This data in a boxplot diagram displays information about the the power consumption of the robot.

Statistics:

```
Population size: 36
Median: 72
Minimum: 64
Maximum: 87
First quartile: 69.25
Third quartile: 78
Interquartile Range: 8.75
Outlier: 87
```

The motor is powered with four batteries (each 1900mAH), the entire energy of the batteries are 7600mAH. The calculation with the maximum current of both motors (172mA) results in a maximum working time of about 40h.

5.5 Software

The software is a small python script that uses ws4py as websocket library and DMCC to interact with the motor controller cape.

5.5.1 WS4PY

Excerpt from the documentation of ws4py:

ws4py provides a high-level, yet simple, interface to provide your application with WebSocket support. It is simple as:

```
from ws4py.websocket import WebSocket
```

The `WebSocket <ws4py.websocket.WebSocket>` class should be sub-classed by your application. At the very least we suggest you override the `received_message(message) <ws4py.websocket.WebSocket.received_message>` method so that you can process incoming messages.

For instance a straight forward echo application would look like this:

```
class EchoWebSocket(WebSocket):
    def received_message(self, message):
        self.send(message.data, message.is_binary)
```

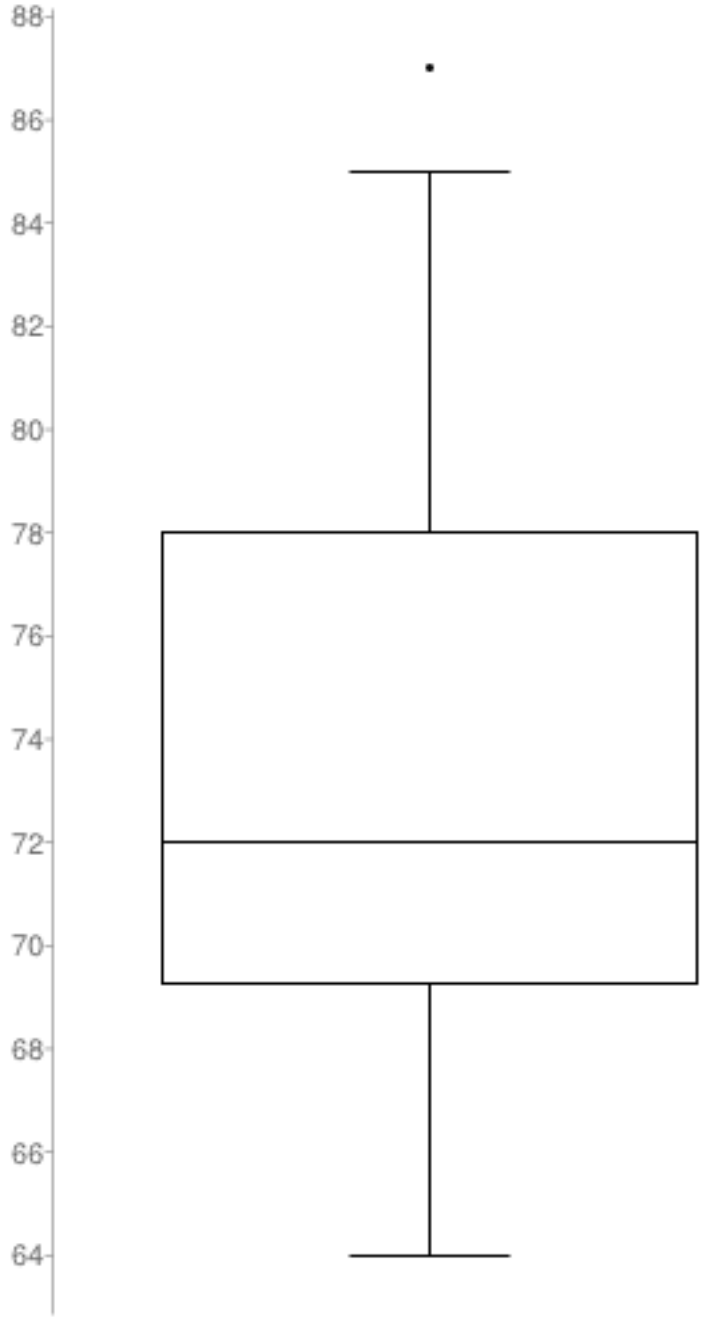

Figure 30 – box plot current

$$7600mAH/(86mA * 2) = 44,19H \simeq 40H$$

Figure 31 – calculation of working time

Other useful methods to implement are:

- `opened() <ws4py.websocket.WebSocket.opened>` which is called whenever the WebSocket handshake is done.
- `closed(code, reason=None) <ws4py.websocket.WebSocket.closed>` which is called whenever the WebSocket connection is terminated.

You may want to know if the connection is currently usable or

`terminated <ws4py.websocket.WebSocket.terminated>`.

At that stage, the subclass is still not connected to any data source. The way ws4py is designed, you don't necessarily need a connected socket, in fact, you don't even need a socket at all.

```
>>> from ws4py.messaging import TextMessage
>>> def data_source():
>>>     yield TextMessage(u'hello world')

>>> from mock import MagicMock
>>> source = MagicMock(side_effect=data_source)
>>> ws = EchoWebSocket(sock=source)
>>> ws.send(u'hello there')
```

5.5.2 DMCC

The DMCC library enables python to interact with the motor controller cape. The cape generates a PWM proportional to a value between -10000 and 10000 which can be configured over a Python interface. The cape can be stacked 4 times.

For the Swank-Rats robot we use one cape to control two motors, one for left and one for right.

It provides an easy to use python interface.

```
import DMCC

# turns on motor 1 on board 0 with 50% power
DMCC.setMotor(0, 1, 5000)

# reverse direction on motor two with 70% power
DMCC.setMotor(0, 2, -7000)

# turn off the motor
DMCC.setMotor(0,1,0)
```

5.5.3 State machine

The state machine calculates the current speed of the motor left and right. Therefore the websocket library forwards the keys pressed and released to the current state, which is initialized with the state stop.

Example:

- The current state is `Stop`
- Press `left` the state `Stop` returns new state `L`
- Press `straight` the state `L` returns new state `FL`

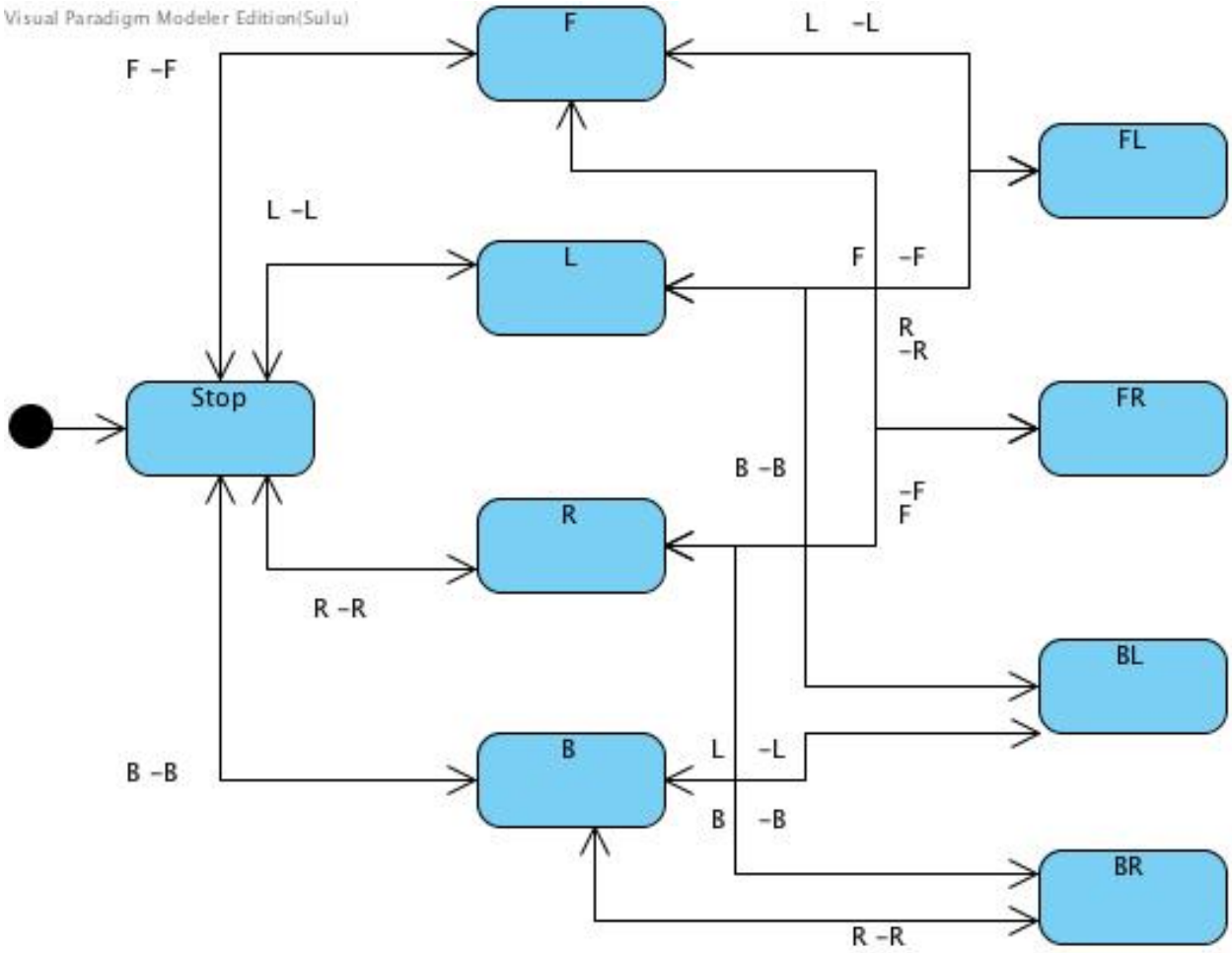

Figure 32 – robot state machine

- Release `left` the state `FL` return new state `F`

```
class State:
    def __init__(self):
        pass

    def press(self, key):
        return Stop()

    def release(self, key):
        return Stop()

    def getLeft(self):
        return 0

    def getRight(self):
        return 0


class Stop(State):
    def press(self, key):
        if key == "left":
            return L()
        if key == "right":
            return R()
        if key == "straight":
            return F()
        if key == "backwards":
            return B()
        return Stop()

    def release(self, key):
        return Stop()


class F(State):
    def press(self, key):
        if key == "left":
            return FL()
        if key == "right":
            return FR()
        return Stop()

    def release(self, key):
        return Stop()

    def getLeft(self):
        return 100

    def getRight(self):
        return 100
```

```
class MessageParser:
    """Parses JSON messages an performs the according SwankRatsRobot action"""

    def __init__(self):
        self.robot = Robot()
        self.currentState = StateClasses.Stop()

    def execute(self, key, pressed):
        if pressed:
            self.currentState = self.currentState.press(key)
        else:
            self.currentState = self.currentState.release(key)

        self.robot.set(self.currentState.getLeft(), self.currentState.getRight())
```

5.6 Supervisor

Supervisor is a client/server system that allows its users to control a number of processes on UNIX-like operating systems. It was inspired by the following:

- Convenience
- Accuracy
- Delegation
- Process Groups

Swank-Rats use it to automatic start and restart the robot script if it crashes. For this it is configured to try restart 100 times after crash or restart the system.

This file is a example config file which is used on the robots.

```
[program:swank-rats]
command=python /root/roboter-software/SwankRatsRoboterSoftware/Main.py
directory=/root/roboter-software/SwankRatsRoboterSoftware
autostart=true
autorestart=true
startretries=100
stderr_logfile=/var/log/swank-rats.err.log
stdout_logfile=/var/log/swank-rats.out.log
```

To enable web we added this to the config file of supervisor (more informations under the install section).

```
[inet_http_server]
port = 9001
username = admin
password = admin
```

After a restart the web ui can be located for example at `http://192.168.43.242:9001/`:

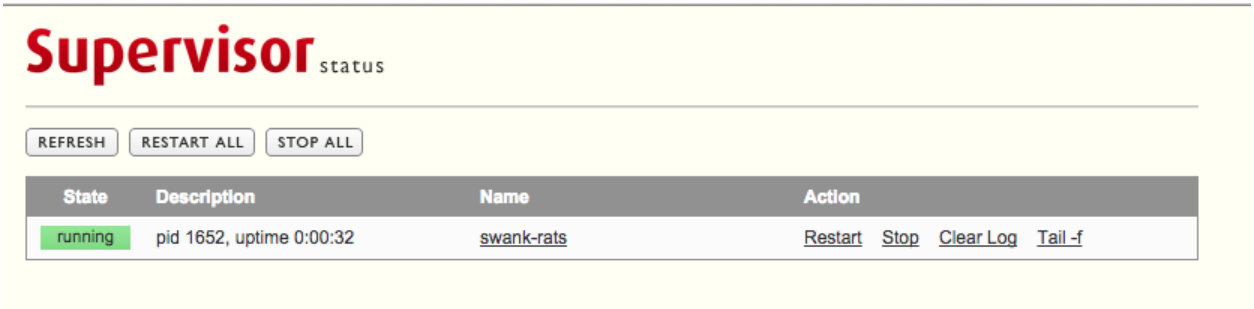

Figure 33 – Supervisor web ui

6 Installation

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

6.1 Installation of mean.io and game-server

6.1.1 Prerequisites

To get started with the mean stack we need to install Node.js and MongoDB.

6.1.1.1 Node.js Get the appropriate installer for Node.js for your OS on their website or just use your favourite package manager. After the installation you should get something like this when you type `node --version` and `npm --version` on your commandline:

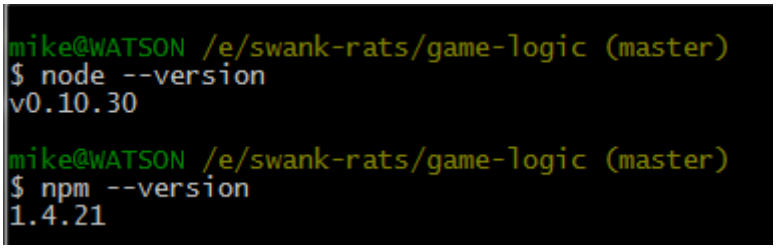

Figure 34 – commandline node and npm

6.1.1.2 MongoDB To install MongoDB follow this link and get a installer or use your package manager again. After the installation create the following directory structure `data\db` in the MongoDB installation directory.

Windows

To start MongoDB on **Windows** just execute the following exe-file (from the commandline to see possible error messages):

```
~MongoDBDirectory\bin\mongod.exe
```

add the dbpath-parameter to the command when you did not install MongoDB in the default location:

```
~MongoDBDirectory\bin\mongod.exe --dbpath "d:\path\to\data\db"
```

When the mongod.exe launch was successfull you should be able execute the mongo.exe to start working with MongoDB.

Ubuntu

To start MongoDB on **Ubuntu** type following on your cli:

```
sudo service mongod start
```

Mac

To start MongoDB on **Mac** type following on your cli:

```
mongod
```

6.1.1.2.1 More details on the installation process

This guide is just a short summery of the installation process - when you need more details just follow this link and you will find a lot of information for Windows, Mac and Linux.

6.1.1.2.2 Security

To enforce security please follow the the steps described here because ... > **Warning:** > MongoDB is designed to be run in trusted environments, and the database does not enable "Secure Mode" by default.

6.1.2 Installation

When Node.js and MongoDB are installed we install bower and grunt with following command:

```
npm install -g bower grunt-cli
```

Bower is a package manager for Javascript libraries like e.g. jQuery and will help us to get all dependencies with just one command and Grunt is a taskrunner and will be used to build the files for the application (Javascript / CSS / etc.) and also to run the server.

These are the basic requirements for this repository to work - if you need more information about the mean stack take a look at https://github.com/linnovate/mean.

6.1.3 Start the app

After these steps the basic requirements for this application are installed and you can clone the repository and execute

```
clone git@github.com:swank-rats/game-logic.git
cd game-logic
npm install
grunt
```

in the repository directory. You should see the project at `http://localhost:3000`

6.1.4 IDE integration

For a very cool integration into Webstorm or Intellij IDEA from Jetbrains watch this tutorial on youtube.

In short: - add mongoose, angular and express Settings > Javascript > Libraries > Download from the "TypeScript community stubs"-list in the dropdown - add mongo plugin to explore MongoDB in the IDE - add a configuration for remote debugging of node.js and enter the host with the port - 5858 in our case

6.2 Beaglebone

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

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

6.2.1 WIFI (TP-Link TL-WN725N)

This tutorial is inspired by this tutorial

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: Add following to config file `/etc/network/interfaces`

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

6.2.2 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
```

6.2.3 DMCC Library

Library to control the motor-cape.

```
git clone git://github.com/Exadler/DMCC_Library
cd DMCC_Librahttp://rcn-ee.net/deb/precise-armhfry/
make
python setupDMCC.py install
```

6.2.4 WS4PY

Library to communicate over Websocket.

```
php install ws4py
```

6.2.5 Supervisor

```
apt-get install supervisor
cp roboter-software/SwankRatsRoboterSoftware/swank-rats.conf /etc/supervisor/conf.d
service supervisor restart

cat > /etc/supervisor/supervisord.conf <<EOF
[inet_http_server]
port = 9001
username = admin
password = admin
EOF
```

6.2.6 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

6.3 Open CV and POCO for image server

To get this project running you need OpenCV 2.4.9 and Poco C++ Libraries 1.5.4. We developed on Windows 7 by using Visual Studio 2013 as IDE and the Microsoft Visual C++ Compiler 18.00.21005.1 for x86 platform. The installation instructions are for Visual Studio 2013 and Windows.

It is necessary to add new system environment variables. So do not close the window, if you have opened it during the installation process.

6.3.1 Install OpenSSL

1. Download OpenSSL Installer for Windows
2. Run installer
3. Add a new system environment variable. To do so open Control Panel -> System -> Advanced system settings -> Environment variables.
4. At system variables press the "new" button and add a variable with name "OPENSSL" and path to e.g. "C:/OpenSSL/" (or to your new location) (with "/" at the end!)
5. OpenSSL installation is finished

6.3.2 Install Poco C++ Libraries

1. Download Poco C++ Libraries 1.5.4 (development version) all - direct link - download poco-1.5.4-all.zip
2. Unpack the archive file to e.g. C:/Poco
3. Navigate to the folder
4. Open the file "components" and remove "CppUnit", "Data", "Data/SQLite", "Data/ODBC", "Data/MySQL", "Zip"
5. You have to edit the path to your OpenSSL installation in the file "buildwin.cmd", if it is not "C:/OpenSSL"
6. Double click build_vs120.cmd -> this command will build the needed files
7. Make sure that the folder e.g."C:/Poco" contains a folder "bin" and "lib".
8. Again add a new system environment variable.
9. At system variables press the "new" button and add a variable with name "POCO" and path to "C:/Poco/" (or to your new location) (with "/" at the end!)
10. Edit the variable PATH
11. Add "C:/Poco/bin" at the end (between the last and the new entry must be a ";"!)
12. Poco installation is finished.

6.3.3 Install OpenCV

1. Download OpenCV 2.4.9
2. Unpack the archive file
3. Copy all the files to the location where you want it to have on your compute
4. OpenCV is already delivered with prebuild VS120 libs. So we have nothing to build.
5. Again add a new system environment variable. To do so open Control Panel -> System -> Advanced system settings -> Environment variables.
6. At system variables press the "new" button and add a variable with name "OPENCV" and path to e.g. "C:/opencv/build/" (with "/" at the end!). This is the path to the OpenCV installation including the folder "build". The folder "build" must contain the folder "include" and "x86/vc12/lib".

7. Modify the PATH variable. Add “%OPENCV%/x86/vc12/bin;” (without “) at the end of the value of your PATH variable.
8. OpenCV installation is finished.

6.3.4 Project setup

1. Clone repository
2. Open the solution with VS 2013
3. Build the project
4. Click right on the solution and go to -> properties -> debugging -> additional command line parameters
5. add /uri=ws://127.0.0.1:3001/ where the IP and port should be the address of the NodeJS server
6. Finish - now you can run the application!

6.3.5 Troubleshooting

- OpenCV: install instructions windows
- OpenCV: instruction for setting up the environment variables
- OpenCV: general install instructions
- OpenCV: Installing & Configuring OpenCV with Visual Studio
- Poco: Building On Windows

6.3.6 Build & Start application

1. Open the solution with VS 2013
2. Right click on the solution image-processing -> Properties
3. Now navigate to Configuration properties -> Debugging -> Command line arguments
4. Enter /uri=wss://IP:3001/ (replace the IP with the IP of your game server)
5. Build it
6. Run it

When you run the .exe manually do not forget to pass the /uri parameter.

```
//to get help:
image-processing.exe /help

//to start the image processing server (important use wss and add a "/" at the end!)
image-processing.exe /uri=wss://127.0.0.1:3001/
```

7 Appendix

7.1 Get the Beagle Bone into the Internet

In this section we will tell you how to connect a beaglebone to the Internet. There are three possibilities how to achieve this goal.

7.1.1 Via lan (easy)

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

7.1.2 Via a bridged wireless lan (not that easy)

Requirements: * a computer which is connected via WIFI to the Internet * a free Ethernet Port and a ethernet cable

1. On the host computer (the computer whose Internet connection you plan to share) open the network connections by navigating to the controll 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 beagle bone and the other end with your computer.
5. You're done

7.1.3 Via Eduroam wireless lan (not easy at all)

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:

```
wpa_supplicant -i $WLAN -D wext -c /etc/wpa_supplicant/wpa_supplicant.conf&
```

Note:

- Instead of $WLAN type your interface name.
- To view the name type iwconfig
- Wait until the authentication has been finished.
- 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 Via usb 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

7.2 Reflash BBB

- Download Image (that works with WLAN-Dongle).
- Follow this instructions to copy image to an SD-Card.
- Insert SD-Card when BBB is unplugged
- Press following button and plugin BBB with a smartphone-charger (more power than PC)

- Keep holding down the button until you see the bank of 4 LED's light up for a few seconds. You can now release the button.
- Wait about 25-45min while the BBB will be flashed
- Once it's done, the bank of 4 LED's to the right of the Ethernet will all stay lit up at the same time. You can then power down your BeagleBone Black and remove SD-Card.

The appropriate linux-headers can be found here.

Detailed information can be found here.

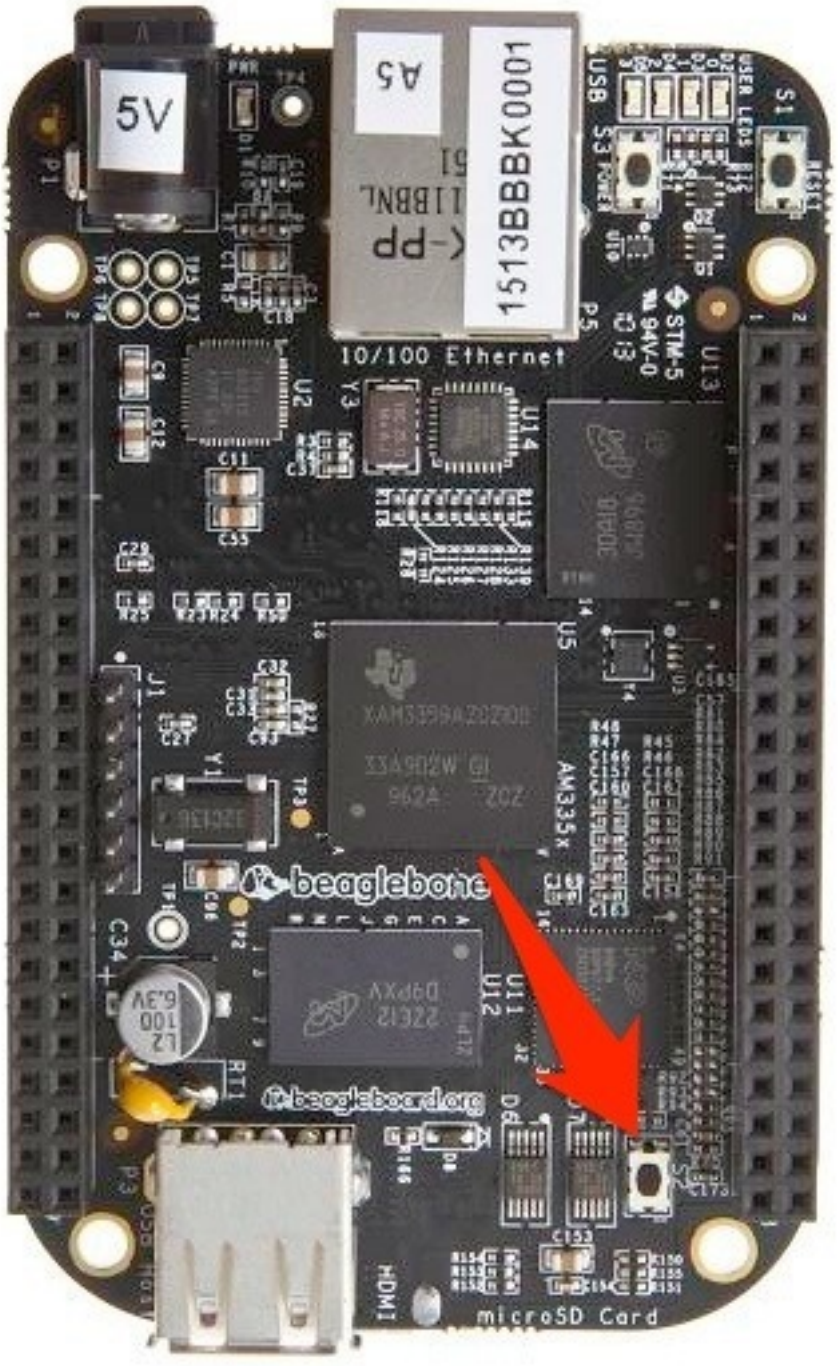

Figure 35 – Boot-Switch button