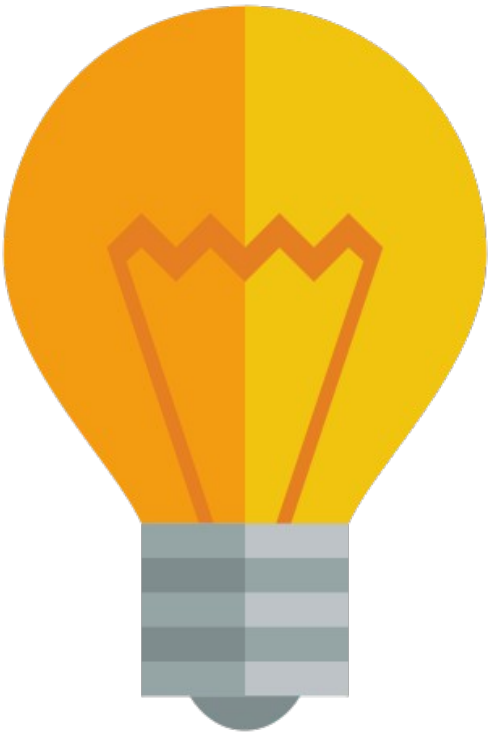


INF219
Project in Informatics

Light Switch

January - June

2015

http://storbukas.no/lightswitch

Declaration

The report on the project Light Switch handed in 10.06.2015, is written by Lars Erik Midtsundstad Storbukås and Ole Eirik Heggelund for the course INF219 Project in Informatics, at the Institute for Informatics, University of Bergen.

The following signatures prove that the information presented in this report as well as software associated or referenced in this report, is the result of the collaboration of the involved parties.

Lars Erik Midtsundstad Storbukås, Ole Eirik Heggelund

Table of Contents

1 Introduction

Imagine yourself living in a not too distant future. Where your daily chores no longer exist, and where you simply ask your house to do the dishes for you, or make you a sandwich, and you don't have to ask your housemaid, wife or even worse, having to do it yourself, urgh.

Isn't this a place you truly can call home? Yes, and maybe some day we get to experience this kind of futuristic home, but let's get back to reality. Today, the resources and technology we have available, doesn't meet the requirements for such a scenario, though, we're still able to get computers to do some amazing things. In this report we're going to show you how we can give you a little taste of the future, and how a computer can assist you in some non-trivial daily routines: like adjusting heat, turning on and off lights, etc.

We made a little "black box", which is able to control electrical power sources based on given environmental variables, like temperature. Even though this isn't quite like the futuristic scenario where the computer makes food for you, it's still pretty cool. It makes everyday life just a tiny bit easier, by not having to constantly worry if it's a couple degrees to warm, or having nightmares about having to turn on the lights by yourself in the morning. Not to worry, Light Switch is at your services, soothing your every electronical need. You just need to give it some basic guidelines on how to behave, and it'll cater to all your wishes. Do you want the house to be warm when you get home from work, without having to leave the heat on the whole day? No problem! Do you want to simulate people living in the house when you're away on holiday? Guess what, no problem at all!

See our point? Don't you just start to imagine the endless possibilities that such a device can do for you? When you suddenly realize that this has to cost a fortune, and that you probably can't afford it? Well, not to worry at all buddy, because we'll show you how you can build one yourself. What's that you say? You don't have any technical experience, nor are you a licensed electrician? Not to worry, we'll guide you through it. And at this point you might be jumping of joy, though we still haven't told you the best part yet, namely that the software is free and open source, and you'll be able to do absolutely what you want with it. Keep calm and carry on mate.

1.1 Goal

The goal of our project is to make a portable home-automation system that can control electronic devices such as lighting, heating and anything else that is connected to an electrical power source. You will be able to set up a time schedule that the devices will follow. Examples of use: «Call the cabin warm»-effect, remote controlling devices or devices controlled by a time schedule, where you can get up in the morning in a room that is already lit, the temperature is perfect and a hot cup of coffee is already made, just for you.

Our goal is to show that home-automation can be made simple and user-friendly. And we'll also try to give you a taste of what a fully automated home feels like. In the process of making this device we've always tried to make it small and compact and keep it's libraries easy to use, so that others can adapt it and modify it for their use.

The difference from standard remote controls for power outlets and our device is that we don't just want it to be used as a remote (i.e. everything has to go through an application at all times), we want it to be able to program behaviors (routines) and different modes so that all the lights turn on at a given time in the morning. You could also set the temperature to be lower in a mode call «away». We would also like to be able to communicate with our device from far away (GSM or Internet connection). So that we for example can give instructions saying that we are on our way home and then the device could preheat the house so that it's warm and cosy when you come home.

Optimally we would like to have additional devices for sensory input (that send data over a common medium like for example bluetooth or WiFi) that acts as master-slave communication. Which also would use a identification protocol to differentiate between the slave-units. This would compared to our current setup, be more user-friendly since all devices including sensors wouldn't need to be placed at one single location (as it is now).

2 Content overview

2.1 Software

Our setup consists of two parties, one client and one server. This means that the server will constantly be powered on and waiting for incoming requests/commands from a client. As well as interpreting requests, it also checks for changes in environment (i.e. clock, temperature, motion, etc.) and see if they have a relation to the predefined user settings.

When the software recognizes a command coming from the client, it analyzes the content of the message, and changes the output corresponding to the action given by the client. Thus, if we only look at the abstract concept of the communication between the server and client, we basically have a remote for controlling electrical equipment. By adding the ability to follow a set of rules and being able to adapt it's behavior based on these, then our device suddenly becomes much more then just a remote, it'll be able to perform actions by itself.

2.2 Hardware

The main parts of our device is the logical unit, circuit breakers, network connection and environmental variables (sensors). The logical unit decides if and when power should be provided to an outlet. The logical unit is connected to a local network via a WiFi-adapter, this means the unit is easy accessible either via your smart phone or a web-browser.

The device is equipped with a fuse to prevent short-circuiting, to not damage the device or the internal electrical networking of the house. You will also be able to track the power usage of the device and it's outlets, by using the built-in current measurement sensor. If the device should freeze or stop working, the device has a manual reset button, so that the electrical equipment connected to it is not affected if you should need to reset the computer.

Inside the box there's a power adapter, that supplies the Raspberry Pi (as well as relays and sensors) with power. The Raspberry Pi has six wires connected to the

relay, where each controls its assigned relay.

The device has a WiFi-adapter connected to it, so that it very easily connects to the local WiFi network once you supply it with the correct network name and passphrase (if there is one).

2.3 Status

All hardware is implemented and includes sockets, relays, power supply, fuse, computer board, temperature/humidity sensor, motion sensor and current sensor.

On the software side we've implemented network communication, status of GPIO-pins, modification of GPIO-pins, settings-parser, start-up script, android application for communicating with the server and a website that is hosted on the server that controls GPIO-pin statuses.

Yet to be done, is the communication between sensory inputs and the settings parser.

3 Architecture

The power comes from the power supply in the wall, and is connected to a socket in the device (three wires; two is the alternating current cables, and one is the ground wire). One of the power cables is connected in series across all the power outlets, whereas the same goes for the ground wire. The other power cable is connected in series to the relays after going through a fuse box. The relays are individually controlled by the Raspberry Pi. The individual relays, when powered on by the Raspberry Pi closes the circuit connection (making current be able to flow) to the corresponding power outlet. The relays acts as a 1-pole circuit-breaker.

See Appendix 9.7 for a comprehensive schematics of the architecture.

4 Implementation

4.1 Software (specific details)

The software implementation consists of several programs and scripts on the Raspberry Pi, and one Android application on the users smart phone. When the Raspberry Pi is turned on, a Bash script is run which in turn starts a server script for incoming connections.

After the server scripts starts, it runs the entire time the Raspberry Pi is turned on, until it is turn off, or manually stopped. The server is implemented using the TCP network communication protocol. It's constantly waiting for incoming connections, and when it gets a new connection, it forks and creates a new process for that connection (so every connection is separate from each-other, leaving less room for problems to occur). The Android application is a script that utilizes as TCP client to communicate with our server. The client sends messages either containing commands for specific power outlets to be turned on/off, or is requesting a update of the status for all relays (with a special delimiter separating the different relays and whether it is powered on or not), so that it in turn can update it's GUI-interface. All relay requests is passed on to the lightswitch.py script, which makes sure everything follows the correct syntax.

The software that handles the relation between sensory inputs and the controlled output is a script called the settings parser, it regularly checks the settings file and corresponding sensor pins for values, if the value and the relation to the settings file matches, a given output occurs on the server (Raspberry Pi), and is passed along to the lightswitch.py script that performs the GPIO-pin changes.

5 Further Development

There are several interesting ways the Light Switch project can continue it's development in the future. We didn't have the chance to implement every feature we wanted by the time we handed in the project, but we plan to keep working on it. The sensors we have connected to the pi, has not yet been implemented, but the software is almost implemented to be able to communicate across software. Sensors should be

able to be given a relation to an output or a specific event, so that our device can make decisions based on predefined settings values (i.e. lights off after bedtime).

This project was limited to use of digital sensor input, but with the addition of an Arduino, there would be more opportunities with analog sensors. Since Arduino can be made really small, we could also put sensors far away from the box connected with a bluetooth or radio frequency communication medium. The website for controlling the GPIO-pins is not yet password protected, and thus not ready to be globally available on the Internet, and should by now only be available on the local network. The Android application could be also improved in by adding some cool features, like for example voice control.

We would also like to change the communication from a typical client-server perspective, to a client-server-client perspective. In other words, to prevent the limitation of being on the same network to be able to communicate with the server (or alternatively make users having to set up port forwarding, though not everyone has the ability, nor the technical experience to do so). The setup we want to have is an external server with a public IP-address, where all communication has to go trough (for example the host http://storbukas.no), so that every client has a unique id and has corresponding software running on it. This scenario let's the communication take place no matter where you are, as long as you are connected to the Internet. This communication channel of course has to be password protected, and preferably also encrypted, so that a user only is able to control it's own devices. This feature also could support remote update of software (i.e. the server pushing patches to the software along the way), making a better user experience.

Another feature that should be added is exceptions, kind of as an manual override. If the settings file states that an output should be on based on a sensor value after a given time, but then the user later turns it off, an exception should be made so that it stays off for a given period of time (say 24 hours).

6 Insight

During our development of the finished product, we've embarked into the world of Raspberry Pi / Linux and learned many of it's awesome features. Python has been the main programming language we've been developing our applications in, and is a

pretty cool language. Python is straight forward, easy to understand and has many nifty tricks up it's sleeves. The reason why we have chosen to stick mainly with Python is due to it's easy syntax and automatic type conversion of variables, and also because Raspberry Pi seem to be heavily built in Python. Since we've chosen this path of development, we also see our software as easy to understand, which also fits nicely with our goal of being a introduction project for beginners to indulge into the world of Raspberry Pi, GPIO programming and home automation.

Of course, electronics have been a great part of our project, and thus we've had to really sit down and understand the principles of basic electronics, circuitry and of course Ohm's law. This has been a really interesting, and a rather different approach to learning compared to the typical classroom course in programming. We have had the opportunity to use our practical skills for creating something (hopefully) safe and usable, as well as pretty (or at least not ugly).

Android applications development has also been something we have had to learn, and this is maybe the most frustrating part of our project. Though it was rather easy getting started with creating an application that has buttons that corresponds to output-pins on the Raspberry Pi, it was a struggle to get the network communication implemented without leaving the application crashing all the time. After much back and forth on how we should do it, to prevent it from crashing/hanging, we found out that adding the network communication in threads running in the background was crucial for our application to work properly.

Although not a big part of our project, web development with PHP, HTML, CSS and JavaScript has been a part of it, making our Raspberry Pi host as a web server, and presenting a website with buttons to control our applications on the web server. Even though it's a fairly small part of our project, this has been one area where we've learned a lot. PHP is a very interesting language to be developing web applications in, and to now be able to understand the chemistry across these languages is a very useful knowledge to be able to use at a project in the future.

As mentioned, electronics has been a great part of our project, not only have we had to learn the basics of direct current applications (D/C low voltage), but we've also learned about alternating current (A/C high voltage), and how this differs from direct current. Of course since we're dealing with high voltages, it's many precautions we've had to take. We always work in a safe environment, use the right tools for connecting and testing the equipment for flaws and short circuits. During our project we also have consulted professionals for guidance along the way. As well as adding extra

precautions with fuses in case of a short circuit made by either the device or a connected device, to prevent the main fuse (in a house) from blowing. To comprehend the physics of alternating current we choose to simplify this by evaluating the two wires as positive and negative currents(although they in reality are alternating back and forth as the name suggests), to easier understand the connection between sockets. Our setup has what is called a 1-poled circuit-breaker, and we chose to do so, because we do not intend to use our device in areas with very high humidity or environments where a 2-poled circuit-breaker is a necessity.

Working with low voltage devices we can appreciate for example relays as electronically controlled light switches (which they also are), and the same current flow should be connected at both ends otherwise bad things can happen (short circuiting).

We've always tried to think ahead, for possible extra features being added to the device, and therefore being able expand with extra sensors.

7 Technical information

The device expects a grounded 230AC as power source. Inside the device there's a 5V DC power adapter supplying the Raspberry Pi and relays with about 1 amp of power, and should suffice in all circumstances. Maximum power consumption on the Raspberry Pi is about 700 milliamperes, leaving more than enough to the other components.

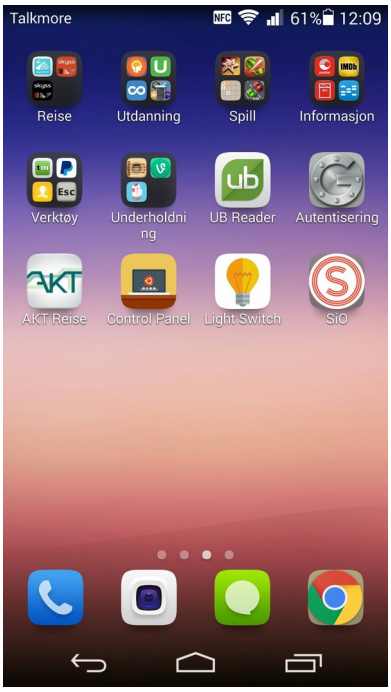
Considering all of the devices components, it supports up to 10 A total power consumption at a given time. Though equipped with a fuse, you should consider not using the device if you expect your appliances to draw more current than this. It should also be noted that the bottleneck also could be the main fuse connected to the power outlet you are using to supply your device with power (if the fuse is rated below 10A). If in doubt contact a certified electrician before use.

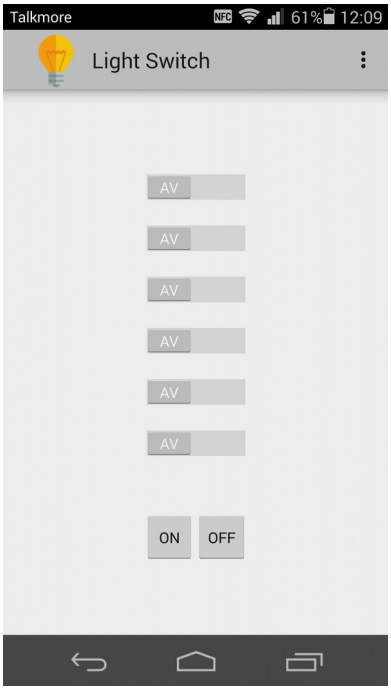
To connect to your device for the first time you need to find it's local IP on the network. This can be found by software such as: nmap (Linux) or Fing (Android, iPhone). If you don't have any of these available, you could connect to the Ethernet port on the Raspberry Pi for direct access. You also need access to the Ethernet cable

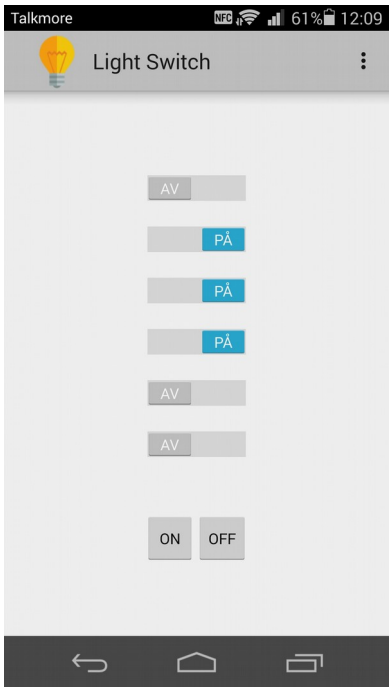
if you want to change connection settings (SSID, passphrase for network, etc.). The server operate on the port number `12345`, and this should be used for all clients connecting to it.

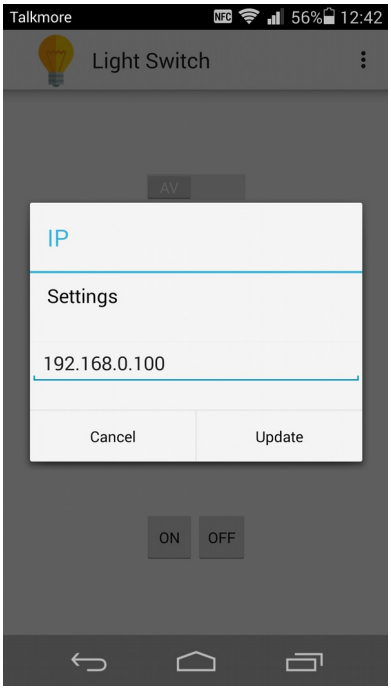
8 Demonstration

8.1 Android









Here is a demo of the Android application running (Light Switch). Buttons correspond to the relays, starting from 1 on the top, and ending with 6 on the bottom. You also have two master buttons, controlling the state of all relays (either ON or OFF). You're able to refresh/update the state from the server, disconnect if you want to finish, and also change IP of your device if it should do so.

8.2 Web browser

You're also able to control the device via the web browser if you want to do so. It looks very similar to the android application, with buttons corresponding to the relays, and also master buttons for controlling all of them. All you need to do to access the website is to type in the IP address of your device (i.e. http://192.168.0.100).

8.3 Command line

To use the command line for controlling the device, the easiest solution would be to use `telnet` to send/receive commands. You simply type (replace IP with your device's address):

```
telnet <ip-address> <port-nr>
```

```
telnet 192.168.0.100 12345
```

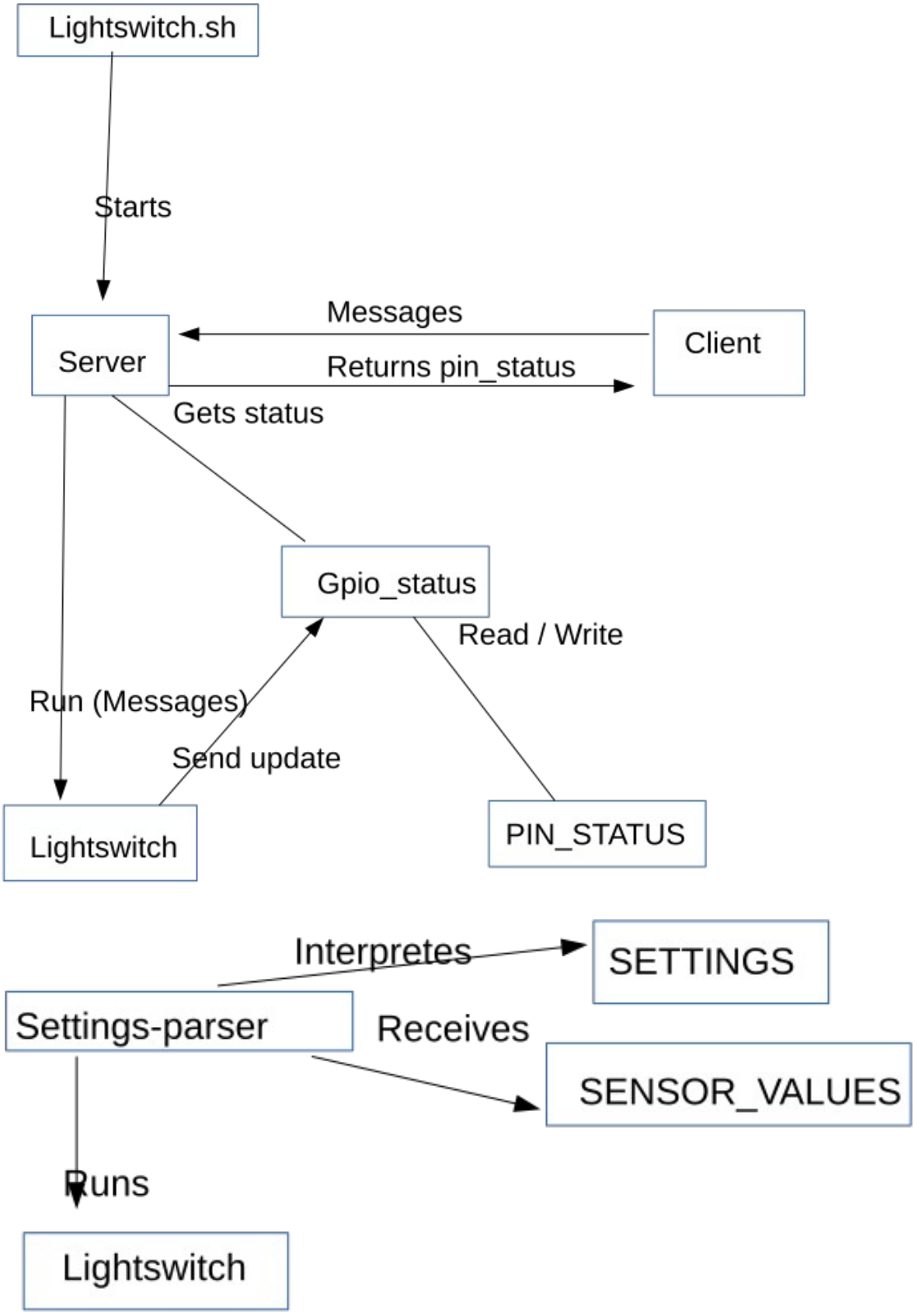
Where 12345 is the port number to send communication. Once you're connected via telnet, you can send commands directly from the command line.

Examples:

```
on all          - turns on all relays
on 1-4          - turns on relays 1,2,3,4
off 1,2,5       - turns off relays 1,2,5
off 3           - turns off relay 3
info            - gets information about which relays are on
```

9 Appendix

9.1 Software communication



9.2 Software description

lightswitch.sh

- Is run by the init.d script on startup
- Starts the server communication application

server.py

- TCP server, constantly checking for new connections
- Receives messages from client, creates new thread on connection
- Interprets messages, and choose which applications to run

lightswitch.py

- Switches status of GPIO-pins to on or off
- Informs GPIO-status of the updated pin values

gpio_status.py

- Receives new GPIO-pin values
- Stores updated pin values in PIN_STATUS
- Gives information about GPIO-pin' status

settings-parser.py

- Interpretes settings file
- Receives sensor values
- Updates GPIO-pins (via lighswitch.py) based on sensor values relation to settings

Client.java

- TCP network communication
- Receives message to send from MainActivity

- Notifies MainActivity about new incoming messages

MainActivity.java

- Receives user-input from the GUI-interface

- Converts GUI button-presses into network commands that are transfered via the Client-class to the server

- Updates GUI and button-status' based on message from server

9.3 Software snippets

gpio-status.py

```
def get_pins():
     """returns a list of ints with either 1 or 0 (active / unactive)"""
     # get first line from PIN_STATUS
     line = read_from_file(FILENAME)
     # put data into int-array
     pins = []
     for i in line.split(DELIMITER):
         pins.append(int(i))

     return pins

def get_string_representation(pin_array):
     """returns a string representation of an array of pins with delimiter"""
     return DELIMITER.join(map(str, pin_array))

def set_pins(pins, action):
     """modifies PIN_STATUS-file pins with action"""
     # get current pin status
     pin_values = get_pins()

     # modify pins with input
     for i in pins:
         pin_values[int(i)-1] = action

     # concatenate pin values to string
     string_representation = get_string_representation(pin_values)
     # write string representation to file
     write_to_file(string_representation, FILENAME)
```

lightswitch-startup

```
case "$1" in
     start)
      echo "Starting light switch"
      /opt/lightswitch/lightswitch.sh
      ;;
     stop)
      echo "Stopping light switch"
      killall lightswitch.sh
      ;;
     *)
      echo "Usage: /etc/init.d/light-switch-startup start|stop"
      exit 1
      ;;
esac

exit 0
```

lightswitch.py

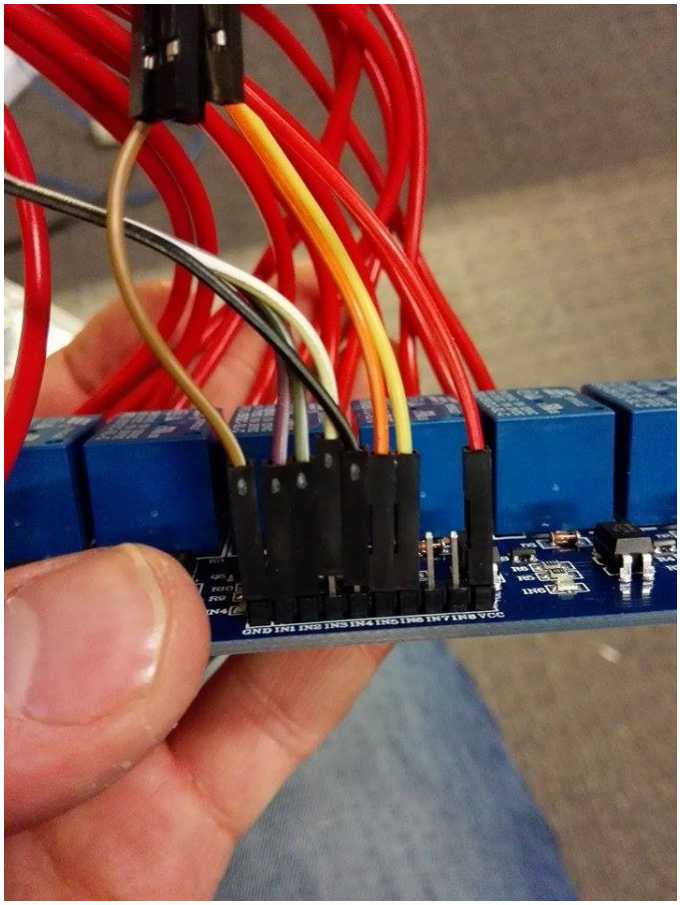
```
# handles out the action on pins in the list
def perform(pins, action):
    # perform action on hardware
    wiringpi.wiringPiSetup()
    for i in pins:
        wiringpi.pinMode(i-1,action)

    # update gpio-status
    gpio-status.set_pins(pins, action)
```

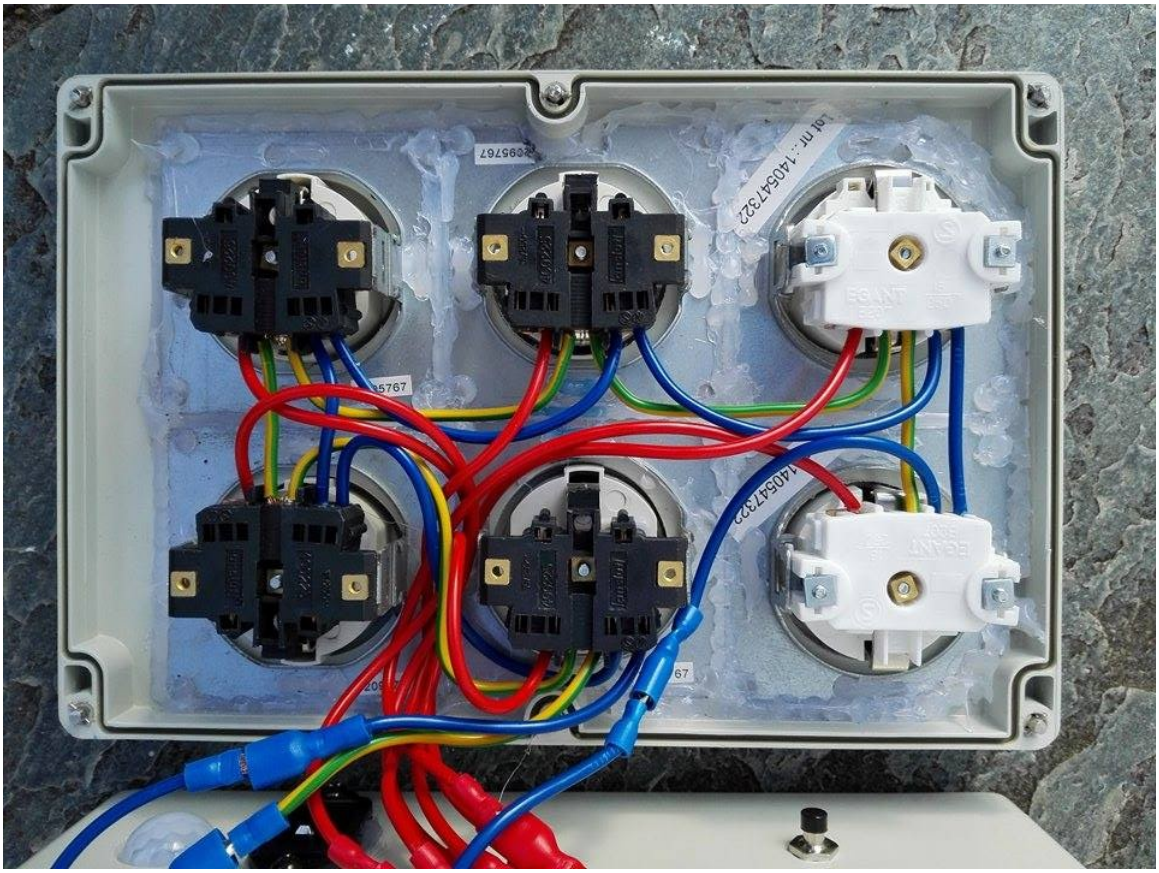
server.py

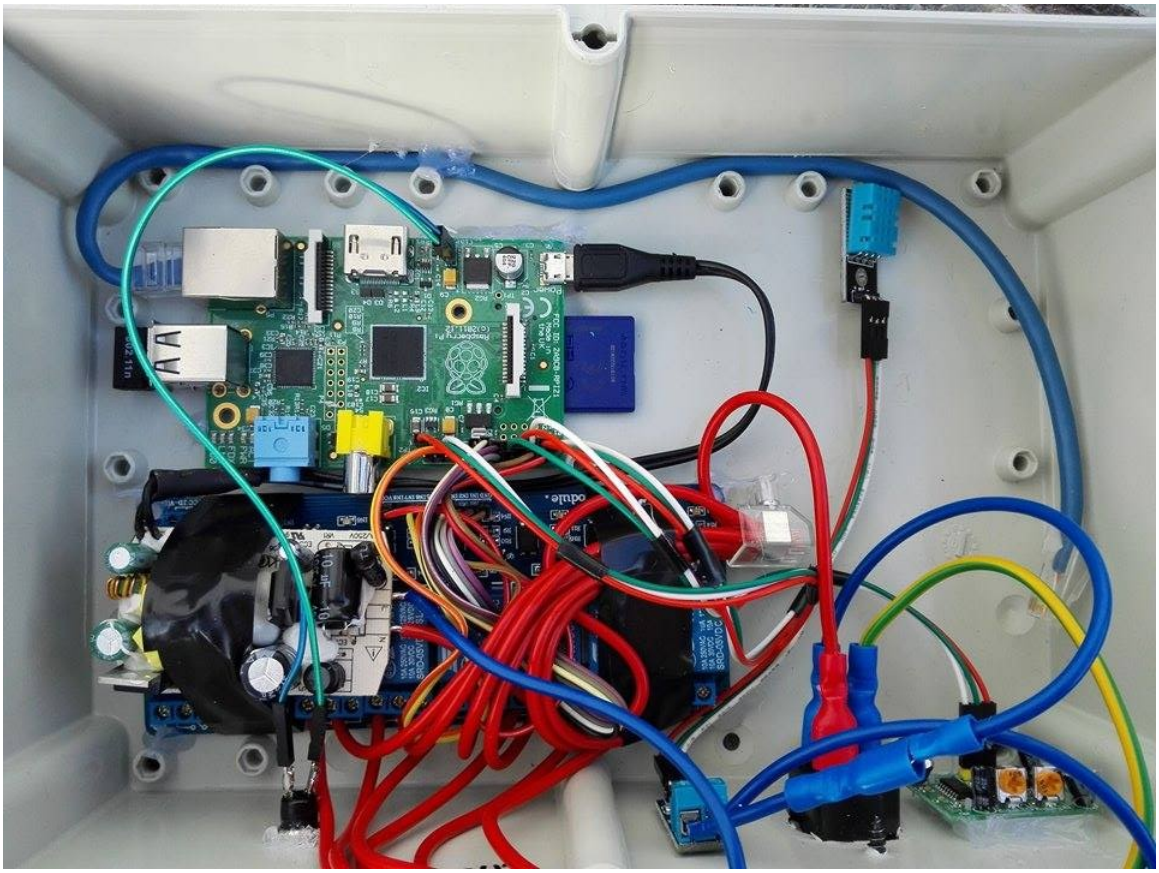
```
def clientthread(conn):
# infinite loop so that function do not terminate and thread do not end.
    while True:
      # wait for input from client
        data = conn.recv(buffer_size)
```

```
        # make sure it's not empty
        if not data: continue

        # print input from client to terminal (for debugging purposes)
        print data

        # client wants status of pins
        if("info" in data):
            # get pin status as string representation
            pin_values = gpio_status.get_pins()
            list = gpio_status.get_string_representation(pin_values)

            # send list to client
            conn.send(list + "\n")

      # clients ends connection
      elif("exit" in data):
            break

        # clients sends a command
        else:
            # compose and evaluate command
            command = LIGTHSWITCH_COMMAND + str(data)
            command = command.replace("\0", "").replace("\n", "")

            # perform system command
            os.system(command)

      # close connection to conn in each thread
      conn.close()

# always listen for connections
while True:
    conn, addr = sock.accept()
    start_new_thread(clientthread,(conn,))
```

9.4 Photos







9.5 Development stages

Here is a rough overview of our development stages:

1. Be able to control an LED from the computer.

2. Read sensor values on the computer, and print these values to the command line.

3. Get the machine to perform action based on environmental values (like sensors, clock, etc.), and control electronical devices based on theese.

4. Develop software for server-client communication. So that we're able to remotely control the device (via network communication).

5. Figure out how the electronics should be hooked up (relays, sensors, etc.), and hook it up for testing on a breadboard.

6. Mount all electronics as the "finished product".

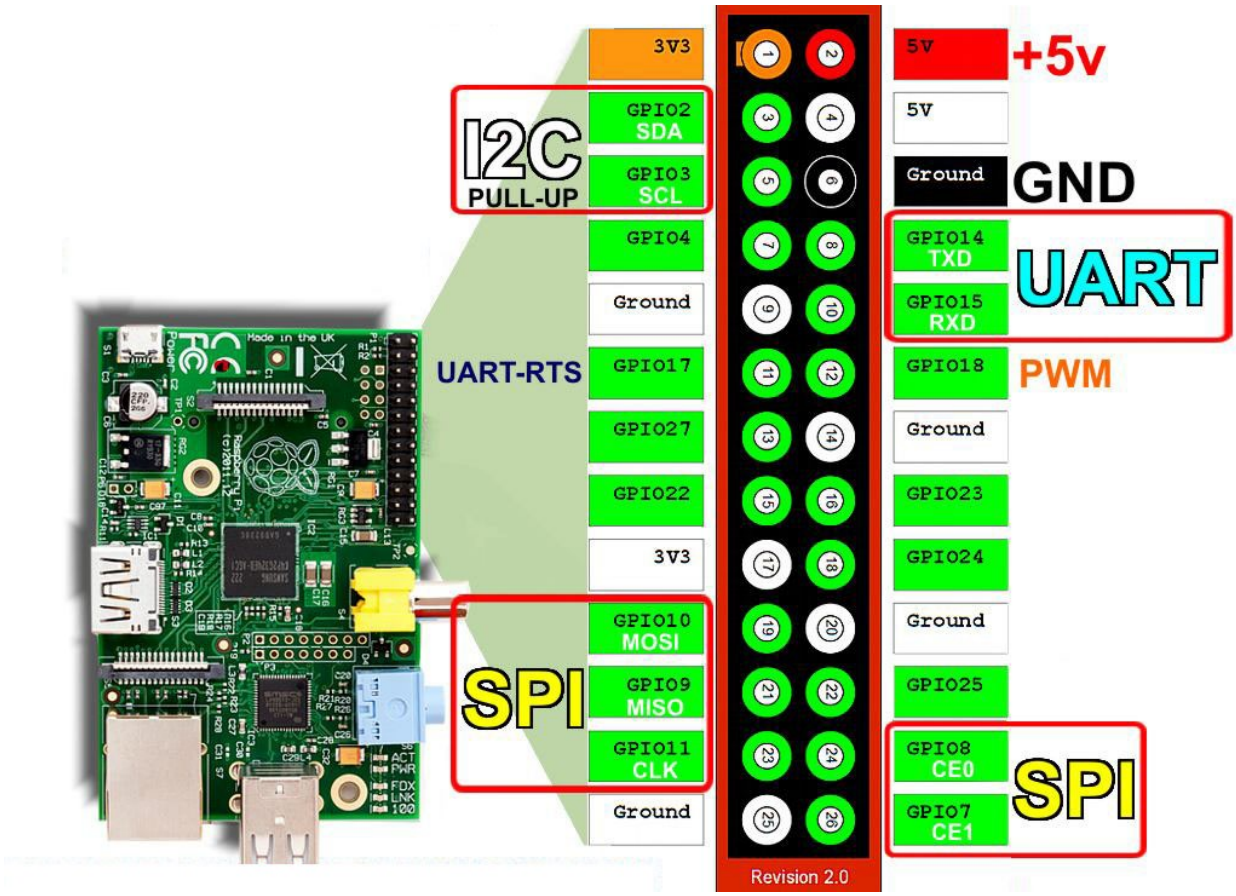
7. Modify the software to be suited for the electronical setup.

8. Modify the software to ble able to set relations between sensor values and output.

9. Develop a simple user interface available on the web browser.

10. Create a simple Android application for controlling the system.

9.6 Miscellaneous



9.7 Architecture