Connected lighting

Karl Gudmundson, Sofia Linevik & Theodor Westny

July 26, 2018

Version:	1.0
Revised:	

Contents

Abstract

Connected Lighting is an inhouse project at Cybercom Linköping made by three engineering students as part of a summer internship. The main goal was to offer an IoT-solution to control lighting in a room at the office and to show how it could be used in the connected industry.

1 Introduction

The IoT team at Cybercom Linköping is in the process of creating a show room at the office, filled to the brim with IoT related applications, primarly targeted at an industrial setting. New IoT applications can always be added, and it was decided that the summer interns this year should contribute to this with something they found interesting and fun. After discussion with the IoT team, the light panels in the ceiling were targeted.
The hardware in this setup will be described further in section 4, how the system is setup will be discussed in section 5 and how the software was created and how it works is explained in section 6 and 7.

2 Background

Lamps are one of the most revolutionary inventions of our time as it symbolized increased potential to us humans with the ability to replace the sun [1]. We no longer relied on the star in the sky to limit the amount of hours of the day, and the darkest corners of the earth could be lit up with the power of electricity. Because humans evolved through darker days when the sun was the only source of light except fire, it has a deep physiological effect to our bodies [2]. This means that we are very susceptible to the effects of light which this project aim to explore from an industrial perspective.

3 System overview

In this section a short overview will be given regarding all included parts and communication protocols. A flowchart showing the included components and protocols can be seen in figure 1.

3.1 General outline

The essential property of the product consist of remotely controllable lighting and the system offers two main features, the ability to control lamps from anywhere in the world with access to internet connection and the capability to autonomously trigger light programs using a range of smart sensors. The two features are developed modularly which means they can function separately: The lamps can be controlled by a person with a computer and the sensors can be integrated into a completely different case. If the lamps are used to signal an alarm the use of both features are favorable in an industrial setting. What if a toxic gas is released? Or a fire has broken out? If a sensor detects the danger and starts the alarm it could save valuable time before disaster is imminent. If the sensors malfunction however and doesn't signal an alarm when they should or vice versa, the ability to manually control the lights is essential.

Software development has solely been done using Python 3.6 together with

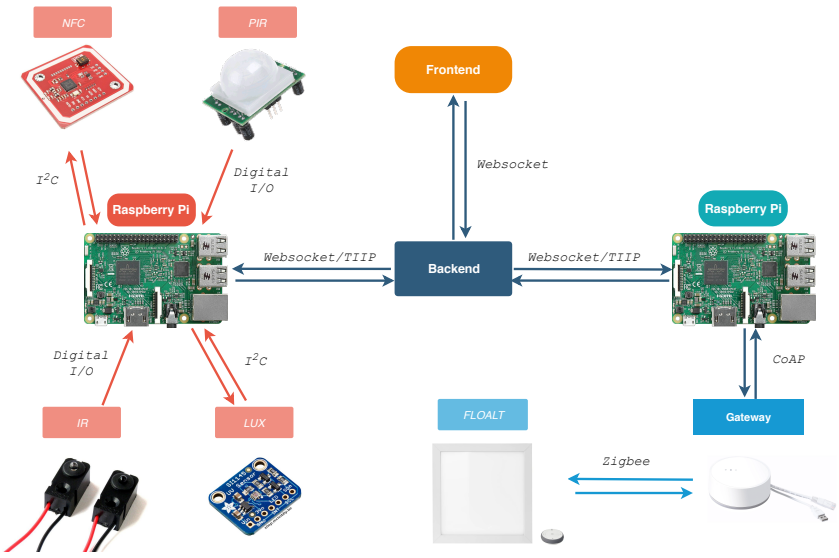


Figure 1: Flowchart of the system

different wrappers for C-based libraries. This is therefore the main language of the system, however the backend which the system communicates against talks in Javascript.

3.2 Communication

As mentioned briefly in the previous section the system communicates comprehensively against a backend server. This is a general aspect of all systems and products developed within the IoT team at Cybercom Linköping, everything should be connected to the backend and everything should follow the same communication standards. Communication against the backend is done using a protocol called *TIIP* (Thin industrial Internet Protocol), created by the team for IoT applications. Communication between all the subsystems can be seen once again in the flowchart (figure 1), and will be explained more thoroughly later in the document.

4 Hardware

In this section a brief introduction will be given regarding all included hardware. This contains introductions to the lights, control units as well as sensors.

4.1 IKEA Trådfri

Trådfri is a collection of IKEA home products that communicate wirelessly by a ZigBee protocol [3]. In this proof of concept, three Trådfri-products are included.

4.1.1 FLOALT

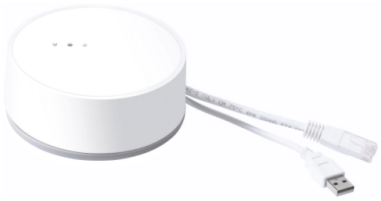
FLOALT is a dimmable led panel (Figure 2b) that allows a variation of programmable output levels as well as three different color schemes. IKEA offers two different sizes: 30x30 cm and 60x60 cm. For the project in question, 8 led panels of the larger size were used.

4.1.2 Gateway

The IKEA Trådfri Gateway (Figure 2a) is a standalone product that offers the ability to control the lights using a small mobile application. The Gateway is also needed to control the lights using exterior API:s or controlling them without using the official remote control.

4.1.3 Remote control

The Remote control is a Trådfri necessity as it is used to include and pair different products to each other and to the gateway. The control also serves to control all available light programs at the push of a button. A picture of the control can be seen in figure 2c.



(a) Trådfri gateway

(b) Trådfri FLOALT

(c) Trådfri remote

Figure 2: IKEA Trådfri products used for the IoT-application.

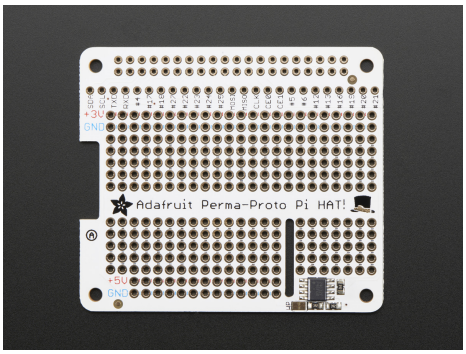


Figure 3: Adafruit Perma-Proto HAT

4.2 Microcontrollers

Two microcontrollers are included in this project. They're both a Raspberry Pi model B and are later referenced to as *Comm-Pi* and *Sens-Pi* which are both supplied with the latest Raspbian Stretch operating system. The Comm-Pi serve as communication link toward the Trådfri Gateway and the Sens-Pi is the microcontroller that all sensors are connected to as shown in the flowchart (figure 1).

4.2.1 Perma-Proto HAT

The Pi which has all the sensors connected to it also incorporates Adafruits Perma-Proto HAT [4] for easier connections and simplified soldering as shown in figure 3.

4.3 Sensors

The used sensors in the project to create some use cases for the connected lights are presented here.

4.3.1 Light level measurement

The light sensor in this project comes as a SI1145-chip from silicon labs distributed by Adafruit soldered onto their own PCB [5]. SI1145 can measure UV, Visable light as well as infrared light but are mainly used for its capability to detect visible light [6]. The PCB is easily connected to a Raspberry Pi using 3.3 V and communicate over I2C.

4.3.2 NFC-reader

For near-field communication applications a sensor with a PN532-chip is used which can read from both NFC and RFID sources [7]. The module communicates via either I2C, UART or SPI protocols and is powered by a 3.3 V pin of the Pi [8]. In this project communication over I2C is used.

4.3.3 Motion detection

The infrared break-beam sensor [9] offers a simple way to detect motion. They have an emitter side that sends out a beam of human-invisible IR light and a receiver across the way which is sensitive to that same light. When something passes between the two, and its not transparent to IR, then the 'beam is broken' and the receiver will output a digital high signal by emitting a small electrical current. The sensor draws current from the 3.3 V output of the Pi and outputs signals through a simple digital output to one GPIO-pin. The output however needs a pull-up resistor, where a 4,6 kΩ resistor is used.

Finally the project also utilize two passive infrared sensors for motion detection of model HC-SR501. The sensors draw current from the 5 V pin of the Pi. The PCB on which the chip is mounted offer a voltage regulator which enable the sensor to connect directly to one of the GPIO pins on the Pi by outputting signals at about 3 volts. The sensor can detect motion within a 6 meter radius with a span of 100 degrees [10]. When motion is detected a digital high signal is sent to the Pi.

4.4 3D-printing

A large amount of 3D-printed objects have been built over the course of the project mainly for the purpose of protecting the sensors and adding physical stability. All prints have been envisioned using *Onshape*, an online CAD-tool and materialized with a Flashforge Finder printer. A model can be seen in figure 4 and the 3D-printed version is shown in figure 5 where the NFC-sensor can be seen encapsulated inside a simple 3D-printed shell.

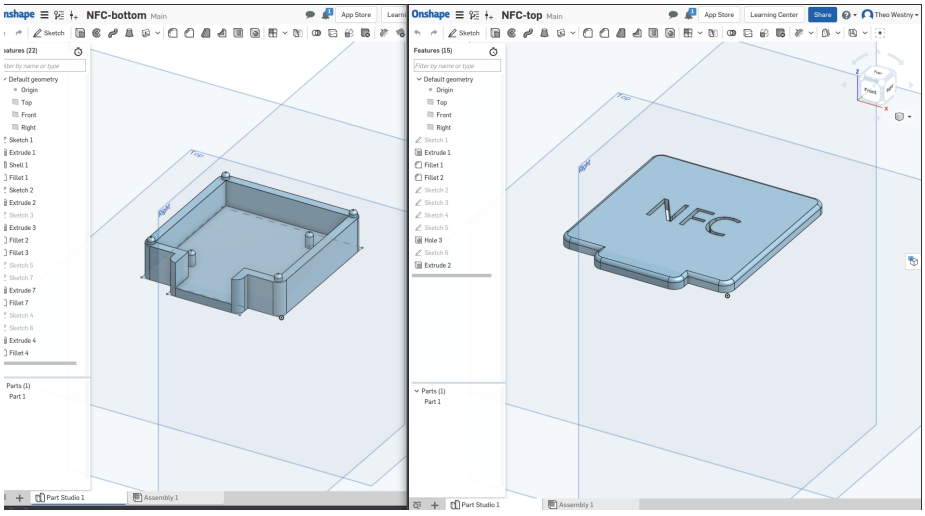


Figure 4: CAD-model of a shell for the NFC-sensor

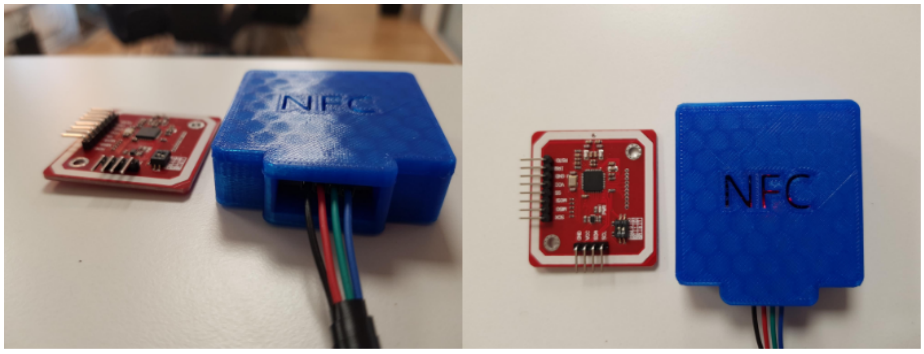


Figure 5: 3D-printed shell for the NFC-sensor

5 Setup

This section will explain mainly how the hardware is setup, how the microcontrollers, sensors and lights are installed.

5.1 Trådfri

All IKEA products include extensive setup instructions that also can be found online [11] [12]. Here only a quick overview will be given.

To start setting up the Trådfri network, the gateway needs power and connection to a router. Lights that want to be set up only need power. All FLOALT lights come with remote controls. The controls are essential to programming the products, however only one remote is needed to control up to 10 products. To be able to use the gateway, this also needs to be included into the same "network" (included with the same remote) as all the lights. Follow the IKEA-guide for the gateway to set it up with a remote and download the Trådfri-app from a general app-store. Once this has been done all lights can be included by following the application's instructions and repeating the same procedure for all the lamps. The FLOALT lamps can now be mounted onto the ceiling (seen in figure 6) at desired location where there is a power supply and in good range of the gateway.

For this implementation a group consisting of the eight light panels was created with the Trådfri application to control the lights synchronously. However, this does not imply that the lights can only be controlled in a synchronized manner. The light panels can still be controlled individually.

5.2 Raspberry Pi

This section will cover general and specific setup and installment of the two Pi:s used. Both Pi:s are installed with the latest release of Raspbian Stretch and equipped with tools for version control using GIT. All the code for both Pi:s are written remote and all necessary Python libraries come in configuration files that are installed in virtual environments on the Pi:s when programs are executed.

5.2.1 Comm-Pi

The Comm-Pi is what this project use as a communication channel toward the IKEA Gateway. The Pi is equipped with no external hardware except a protective case. To communicate with the backend server the Pi needs internet connection. Since the Pi is a Model 3 it offers options regarding the physical connection by either ethernet cabel or wirelessly. It's crucial however that the Pi is connected onto the same network as the IKEA Gateway as it sends commands

Figure 6: FLOALT led panels installed and mounted on the ceiling

to the gateway with reference to its IP address. The Comm-Pi only serves to run the software that will be explained in section 6.

5.2.2 Sens-Pi

The Sens-Pi offers an interface to all the sensors in the project. Since two of the sensors used communicate over I2C this has to be setup during the Pi:s installment, or whenever via a terminal by writing `sudo raspi-config` and allowing I2C on the *Interfacing* option. The Pi is also equipped with a Adafruit Perma-Proto HAT described in section 4.2.1. The HAT comes unassembled out of the box and must therefore be soldered on a pin-strip before it can be attached to the Pi. The sensors are then connected to the Pi according to the small circuit diagram in figure 7.

The Sens-Pi also needs internet connection to communicate with the backend server and like the Comm-Pi both wireless and physical connection via ethernet cabel comes as option for the Pi 3. Unlike the Comm-Pi which network the Sens-Pi connects to does not matter. The Sens-Pi can be seen in figure 8.

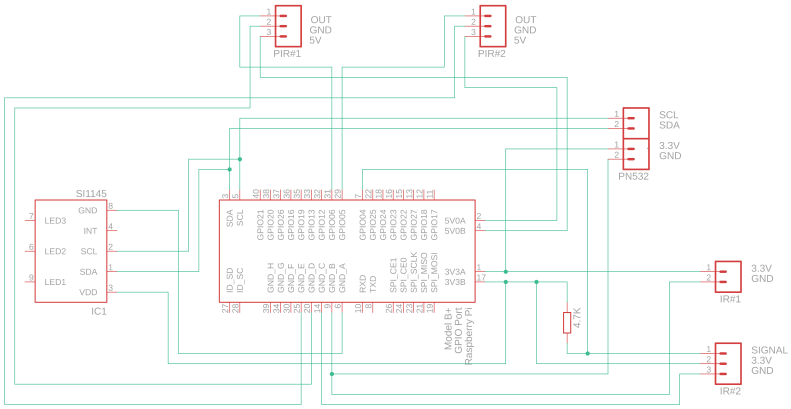


Figure 7: Circuit diagram for the sensor setup

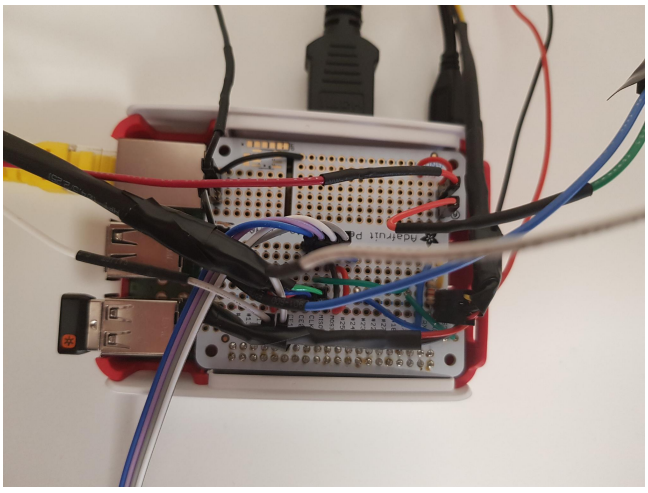


Figure 8: Sens-Pi in action

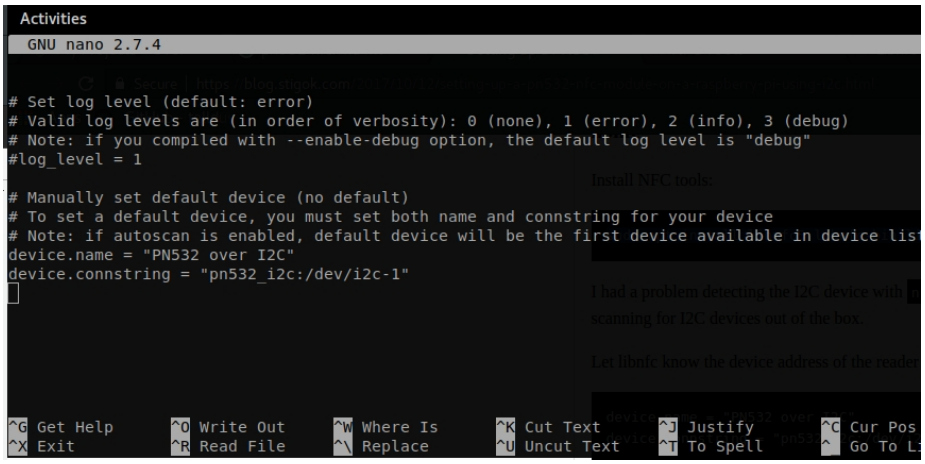


Figure 9: Libnfc configuration file

Libnfc

The Sens-Pi need external software to handle NFC-data generated from the PN532. *Libnfc* is a low level NFC SDK and programmers API which serves the purpose well in the project and the NFC tools was installed directly using the command `sudo apt install libnfc5 libnfc-bin libnfc-examples`. Libnfc does not scan for I2C devices without configuration so this has to be setup by letting libnfc know the device address of the reader in `/etc/nfc/libnfc.conf` as seen in figure 9. To list all NFC-devices, from a terminal simply run `nfc-list -v` or `nfc-scan-device -v` [13].

6 Software

This section describes the software used in the project.

6.1 Overview

The scripts running on the Pi:s are communicating with channels which can be subscribed or published to. Each channel handles one type of sensor data. The apps running on the Sens-Pi are primarily publishing sensor data to separate channels, which the script running on the Comm-Pi subscribes to and use the incoming data to control the light panels. In the following sections, a brief description will be given on how the scripts work on the Comm-Pi and Sens-Pi.

6.2 Comm-Pi

The script that runs on the Comm-Pi to communicate with the IKEA gateway and the Backend server is called *lampsensor*. This script creates an object from the class in the python file called *tradfri*. The class is used to communicate with the gateway via a python package called *pytradfri* [15], which serves as an API. More information about this API is given in section 6.4.1

6.2.1 Lampsensor script file

The lampsensor file listens to remote procedure calls, RPC's, from the Backend and depending on which function that is called it performs the desired action. An example of such an action would be if the *set* function is called which can be used to set the color, intensity and state of individual or multiple lights. Class functions are called on the created class object in order to adjust the settings of the lights. Other functions that handles the remote procedure calls are *get*, *setMode* and *setTriggerMode*.

This script file subscribes to the channels that the sensors publish their data to. If it receives data from a channel, a callback function is fired that handles that data and performs an action on the class object with the class functions.

6.2.2 Tradfri script file

In the tradfri script file a class is implemented to control the IKEA light panels with the *pytradfri* API. The class functions are used to control the lights. The lampsensor script, 6.2.1, utilizes this class.

Main functions

The main functions of *tradfri.py* are described below. All functions apply to function for one or several lights at a time.

- **set_state**
  Function to turn on or off the light panels. The function takes two arguments, which light/group to set the state of and also if it should be turned on or off.

- **get_state**
  Function to get the current state of the lights. This includes both intensity, color and if it's turned on or not.

- **set_temp**
  This function enables the functionality to set the color "temperature" of the light panels. The function takes two arguments, which light/group to change and the value of the color temperature. The color is a value between 250 and 454, where 250 is white and 454 is orange. This function can also take a third, optional argument, for the time it takes to change the color from its current color to the desired color.

- **set_dim**
  Changing the intensity of the light panels are done with the *set_dim* function. This functions takes two arguments, which light/group to change and the value of the intensity. This value ranges between 0 and 254, where 0 is a turned off state and 254 is the maximum intensity. An argument that tells how long it will take for the lights to dim can also be passed with the function as a third argument.

- **alarm**
  Alarm includes a set of functions of self-made light programs to symbolize different alarms, including varying light intensity, color and different dim times.

- **get_device**
  This function returns all available hardware-specific information of the requested device: device ID, firmware version and so forth.

6.2.3 Modes

The script running on the Comm-Pi has different modes. These modes are set in order to tell the script running on the raspberry pi how the lights should be controlled. The mode can be changed with a remote procedure call, *setTrigger-Mode*, via the Lampsensor script. The standard mode, called *std*, is where the light panels are controlled from the user interface. This mode also allows for an alarm to set off if the IR beam, mentioned in 4.3.3, breaks. When the beam is solid again, the alarm can be turned off when the NFC sensor, mentioned in 4.3.2, reads the correct tag. The *pir* mode, which uses the signals from the PIR sensors, described in section 4.3.3, controls the lights depending on which of the motion sensors that detects motion. Lastly, the *light* mode is where the light panels are controlled by the light sensor, mentioned in section 4.3.1.

6.3 Sens-Pi

The Sens-Pi run scripts to control the lights with the sensors described in section 4.3. For each sensor, there is a sensor file that reads the incoming signals from the sensor and handles that data by publishing a message to their designated channel.

6.3.1 Light level measurement

The light sensor is used to measure the brightness in the room. By measuring the brightness, the lamps can be controlled to only shine as much as necessary. For example, if it's midday and the sun is shining bright through the windows, the lamps are essentially not needed at all, whilst at late in the evening the lamps need to shine with great intensity to provide enough light. By setting a desired light level and using control theory, in this case a PD-controller, this can be achieved.

As described in section 4.3.1 the light sensor communicates over I2C and can provide measurements of three different types of light: UV, IR and visible light. In this case, only the visible light is measured. To get the measurements from the sensor, the Python package *smbus2* is used to read from and write to addresses. The package is more thoroughly described in section 6.4.5.

PD-controller

Control loop feedback mechanisms are widely used in different industrial control systems and there are a broad range of different controllers. The PD variant used in this case, is a proportional-derivative controller defined by equation 1.

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} \tag{1}$$

where $u(t)$ is the control signal and $e(t)$ is the continuously calculated error value. Since the controller works in discrete time, the equation is slightly adjusted to:

$$u(k) = K_p e(k) + K_d \frac{e(k) - e(k-1)}{T_s} \tag{2}$$

where k is a discrete sample and T_s the sample time. K_p as well as K_d are coefficients for the proportional and derivative terms. In a broad sense, large K_p generates a fast response but reduced stability and a large K_d gives better stability but increases measurement error influence. The coefficients have to be produced through testing and adjusted until a desirable result is obtained. How the controller works in practical terms is done as follows: The visible light level acquired from the lights sensor is compared to the desired light level (the reference signal), giving an error signal which is used by the PD-controller to send light levels (control signals) to the lamps. The measured light level sent to the controller is an average over 100 samples, measured at 100 Hz, which means that the light levels of the lamps are updated at 1 Hz. This, as well as

the desired light level, are tuning parameters. An update frequency of 1 Hz was deemed effective for a visual demonstration.

6.3.2 NFC-reader

The script to handle NFC-readings is built around *Libnfc* briefly explained in the end of section 5.2.2. The code works to wrap around libnfc with the help of the *os* library and the function *popen* to open pipes, saving the response and converting it to a string. The outputs from running different command line functions from libnfc when a NFC-source is read returns comprehensive data, so the main goal is only to extract the unique ID. This is done with different string manipulations. The script can be used to simply read the UID of different sources, or utilize already known UID:s to verify different security restricted commands.

6.3.3 Motion detection

The break-beam IR-sensor and PIR-sensor function in roughly in the same way. Once the 'beam is broken' for the IR or the PIR has 'detected movement' a digital high signal is sent to the GPIO-pin of choice. This is handled with python:s built in package *RPi.GPIO*. Once a high signal is detected by the Pi it sends a message to the backend.

6.4 Python packages and modules

The python packages used for the implementation of this project in addition to the python standard library [14] are *pytradfri*, *RPi.GPIO* and *smbus2*. *Asyncio* and *time* was also used for the implementation, which both are a part of the python stdlib. In this section an overview will be given of how they were used in this project.

6.4.1 pytradfri

Pytradfri is the IKEA trådfri API which communicates with the IKEA gateway [15]. It enables the raspberry pi to control the IKEA lights with the python standard library. The library was used in the *tradfri* script, 6.2.2, to change the settings of the lights.

6.4.2 asyncio

The *asyncio* [16] package is a part of the python stdlib and was used in the *tradfri* script so that the application would be asynchronous, meaning several threads could be used to run separate processes without the need of a queue. It enables the functionality to send several commands asynchronously to the IKEA gateway.

6.4.3 time

Arranging so that the lights would flash in an alarm like way was done with the help of the *time* [17] package. The alarm function was implemented as a class function in the *tradfri* script. Using *time.sleep(sec)* in between calls to the IKEA gateway to set the light settings would allow for the change in the lights to occur before another call to the gateway would be made.

6.4.4 RPi.GPIO

The GPIO pins on the Sens-Pi are controlled with the *RPi.GPIO* [18] package. The script files that use the *RPi.GPIO* package are the ones that reads signals from the IR beam and PIR sensors.

6.4.5 smbus2

SMBus is used to communicate over I2C. *smbus2* [19] is a Python implementation of this, making it possible to communicate with I2C modules with Python. Communication here includes write/read of different sizes. The package is used to communicate with the light sensor.

7 User Interface

In this section a description will be given of how to control the light panels using the UI which is available at a webpage for illustrating IOT-solutions. ReactJS was used to implement the controlling component, which is a javascript library to build user interfaces.

7.1 The controlling component

The component to control the light panels is presented in Figure 10. The figure displays two subfigures that illustrates how the user interface can be used in order to adjust the lighting.

Each light panel is represented by a square in the user interface. The panels can be controlled both individually and in group, by checking the checkboxes. A click on a square will turn on or off a light panel. The intensity and color of the lights are adjusted with the gradient label. The intensity of the light is adjusted on the y-axis, and the color on the x-axis. The lowest value will return an intense white light and the highest value will return an orange light. The time it takes for the light to change intensity can be set with the slider.

In Figure 10a all the lights are turned off, hence the dark gray color of the squares. If turned on, the gray color will change to the color of the previous state when the light was turned on. How to adjust multiple light panels at the same time is illustrated in Figure 10b. By clicking the option *check all*, all of the checkboxes are checked, which means that all light panels are chosen. Unchecking checkboxes is done in the same manner, but instead clicking the *uncheck* option.

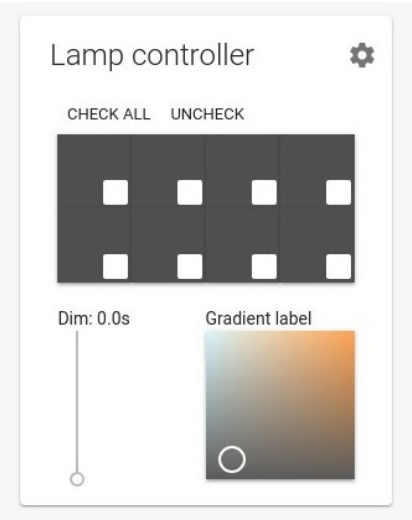
7.1.1 Other features

Pressing the settings button, illustrated in the top right corner in the figures in Figure 10, will open a menu for other functionalities. This menu is displayed in Figure 11. From here, the user can set different trigger modes. These modes are explained in section 6.2.3.

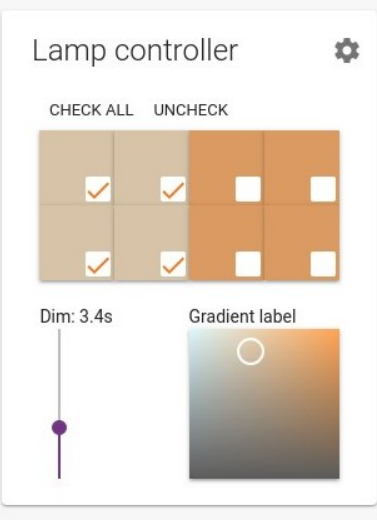
From this menu the user can also set off a light sequence called *cool_lights*. This light sequence illustrates how the light settings can be modified by varying the color and intensity. The second option in the menu is *alarm_pulse* where a pulsating light in orange sets off. The third option is *temp_change* where the color of the lights varies between a white and orange light. The last option is *alarm_snake* where there is only one light panel that is turned on, and then turned off at the same time the light panel next to it is turned on. This creates an effect that resembles a snake moving around. The modes can be stopped by clicking the *End mode* menu option.

7.2 Use case of the controlling component

In order to control a single or multiple light panels, firstly choose which light panels to control by checking the checkboxes. Thereafter, if desired, set the dimmer time with the slider and lastly pick the intensity and color of the light in the gradient label. This will set off an action to change the lightning to your desired settings.



(a) The gray color of the squares indicates that the panels are turned off. Clicking a square will turn the light panel on or off, depending on the current state.



(b) The checkboxes in the squares makes it possible to control a single or multiple light panels at the same time. The slider for the dim time controls the time for which it takes to dim the lights, in seconds. Color and intensity of the lights can be adjusted with the gradient label.

Figure 10: The component that controls the eight light panels, showing different ways to adjust the lightning. Each colored square represents a light panel.

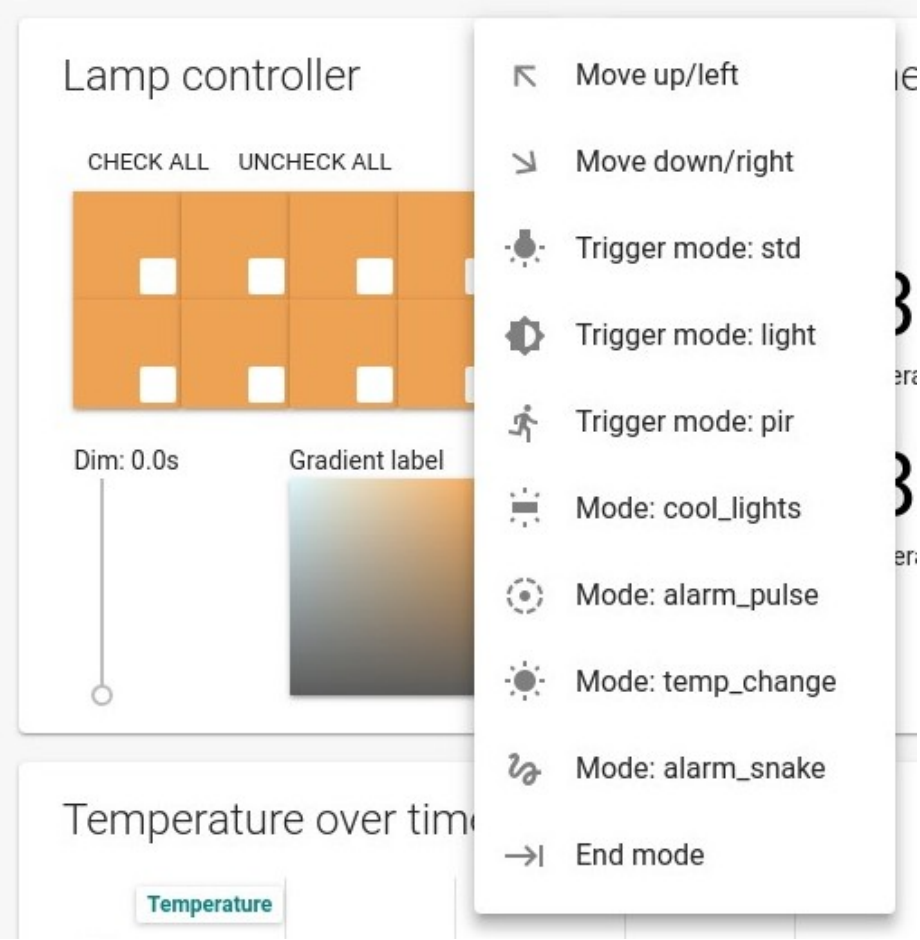


Figure 11: The drop down menu that appears after pressing the settings button. From this menu the *trigger modes* can be set to either *std*, *light* or *pir*. There are also four different options that are called *modes*, that starts a light sequence where the intensities and colors varies depending on the chosen mode.

References

[1] Sarah Reisert. *Let There Be Light*, The arrival of electrical light.
https://www.sciencehistory.org/distillations/magazine/let-there-be-light

[2] Melanie Rüger. Time-of-day-dependent effects of bright light exposure on human psychophysiology: comparison of daytime and nighttime exposure
https://www.physiology.org/doi/pdf/10.1152/ajpregu.00121.2005

[3] Compability & protocols, IKEA smart lightning, 2018
https://www.ikea.com/gb/en/customer-service/smart-lighting-support/faq-smart-lighting/compatibility-protocols/

[4] Adafruit Perma-Proto HAT for Pi Mini Kit - With EEPROM
https://www.adafruit.com/product/2314

[5] SI1145 Digital UV Index / IR / Visible Light Sensor
https://www.adafruit.com/product/1777

[6] Silicon labs. PROXIMITY/UV/AMBIENT LIGHT SENSOR IC WITH I2C INTERFACE.
https://cdn-shop.adafruit.com/datasheets/Si1145-46-47.pdf

[7] PN532 NFC RFID Module User Guide
https://dangerousthings.com/wp-content/uploads/PN532_Manual_V3-1.pdf

[8] NXP. Near Field Communication (NFC) controller.
https://www.nxp.com/docs/en/nxp/data-sheets/PN532_C1.pdf

[9] IR Break Beam Sensor - 3mm LEDs
https://www.adafruit.com/product/2167

[10] HC-SR501 PIR MOTION DETECTOR
https://www.mpja.com/download/31227sc.pdf

[11] IKEA. Trådfri FLOALT led-planel assembly instructions.
https://www.ikea.com/se/sv/assembly_instructions/floalt-led-ljuspanel-med-tradlos-styrnin

[12] IKEA. Trådfri Gateway assembly instructions.
https://www.ikea.com/se/sv/assembly_instructions/tradfri-gateway__AA-1908212-3_pub.pdf

[13] Libnfc. NFC SDK and Programmers API.
http://nfc-tools.org/index.php/Libnfc

[14] Python standard library. Version 3.6.6.
https://docs.python.org/3.6/library/

[15] pytradfri, python package. Version 5.5.1.
https://pypi.org/project/pytradfri/

[16] asyncio, part of the python stdlib.
https://docs.python.org/3.6/library/asyncio.html

[17] time, part of the python stdlib.
https://docs.python.org/3.6/library/time.html

[18] RPi.GPIO, python package. Version 0.6.3.
https://pypi.org/project/RPi.GPIO/

[19] smbus2, python package. Version 0.2.1.
https://github.com/kplindegaard/smbus2