Vishnevskii Vladislav

CONCEPT OF A NEW GENERATION IoT COFFEE MACHINE

Bachelor thesis
Information Technology

2018

South-Eastern Finland
University of Applied Sciences

Author (authors)	Degree	Time
Vladislav Vishnevskii	Bachelor of Engineering	March 2018

Thesis title

Concept of a new generation IoT coffee machine

53 pages
5 pages of appendices

Commissioned by

XAMK

Supervisor

Matti Jutilainen

Abstract

In this thesis the prospect of creating a new IoT coffee machine was researched. The idea based on finding the jug content and implementing monitoring for the users. Features, made possible by the presence of this information were studied. Such new functionality included, was not limited to a new portion of coffee creation. With new data, the coffee machine would be safe from overflowing the jug.

To find a method for the coffee jug content detection many sensors and methods were studied and some tested. The trial and error was the main method of acquiring information followed by reading documentation on different components and libraries. This was caused by the very practical focus of the study.

The developed detection method based on ambient light sensors and on the fact that coffee does not pass the light. Thus, it is possible to identify when there is coffee in front of the sensors and when not. With an array of sensors, it is possible to get a reasonably accurate approximation and use it for many purposes.

Based on the study a proof of concept was built. It has been based on the ESP8266 12-E board, quite similar to Arduino UNO. This thesis had the full process of development described as well as building instructions. The proof of concept creation also required work with a web server, the creation of web interfaces, usage of EEPROM, modeling and printing the casing on a 3D printer.

As a result of all studies, a concept of a new generation IoT coffee machine was developed. This included recommendations on an operation principle, casing, web interface, method of resource supply, installation and usage. A speculation on the reasons why the device could be popular was also introduced as well as all the advantages it would have over traditional coffee machines.

Keywords

Information technology, Programming, C++, IoT, Web server, Arduino, ESP8266

CONTENTS

1 INTRODUCTION

Nowadays, Internet of Things (IoT) becomes more and more popular. A lot of work is done to bring a smart home idea to life. All IoT devices are connected to the local network. Everything could be monitored and controlled from a smartphone or PC. To make it possible many specialists in different IT fields are involved.

Some companies are already shipping smart devices to the customers. Unfortunately, their functionality is usually limited. I found an IoT coffee machine capable of brewing one portion of coffee on demand, on the market. The limitation is, that everything needs to be prepared beforehand for every brewing. There was also no way to check, whether it is safe to order a coffee, or there is simply no cup under the coffee machine.

In this thesis, I will research the possibility of improving the IoT coffee machines. I will use a Moccamaster KBG741 office coffee machine as a starting point. I believe that the current functionality is not sufficient, and it can be expanded greatly, if only a way to detect the coffee level in the jug will be found. Furthermore, I will work with creating a concept of a new generation of IoT coffee machines for office and home use.

To study the topic better I will create a proof of concept. It will extend the Moccamaster KBG741 functionality. Thus, my ideas can be put to the test. There are many questions to be answered for both creating a prototype and a final concept.

First, I want to find a way to have data about the coffee jug content at any given time. The method should be simple and inexpensive, but reliable. This will allow users to know how much coffee is left and to plan breaks accordingly. Some sort of sensors will be needed to get the data. The program of the device will process the readings from the sensors. It should provide the results to the users in a form of web page.

Then, the new possibilities in functionality need to be studied. Besides the basic ones, lots of features could be implemented even in the prototype. I will try to list, implement and test them.

As for the final concept, I believe that there are a lot of things that can be done with the IoT coffee machine, when the data about the current coffee level is available. For example, it is possible to start a new portion creation remotely without the risk of overflowing the jug. I will present my thoughts on perspective features and try to describe how they could work.

Also, a prototype of a new interface for the IoT coffee machine needs to be created. There are a lot of new data and possible functions, so they need to be properly presented. I want to define an appropriate way of doing that. I also hope to create a configuration tool, since the device will require some calibration and settings to be provided by user before it will start functioning.

As a result, a new generation of IoT coffee machines could improve the user experience for both families and office workers. I believe that it is beneficial for the companies to have happy employees who are well supplied with good coffee. Thus, there is a high chance of such devices becoming popular.

2 BACKGROUND

To achieve the goal, I need to study many topics. I will explore the possibilities to implement my idea. I need to develop the operation principle of the device. An appropriate sensor needs to be found. Also, I will study programming for Arduino like boards to create the logic of the device. Casing creation will require studying 3D modeling and printing.

My studies will include studying the topic and documentation and doing some experiments. I want to test the methods available for coffee detection on the jug of Moccamaster KBG741 before building the prototype. Network connection handling also needs to be explored.

2.1 Detecting the coffee

To get the data about the coffee level in the jug, I need to use some sensors. Based on their readings the device will be able to make an approximation of the coffee amount. I came up with some ideas related to this issue.

I believe that placing sensors inside the coffee jug is a bad idea. It will be corrupted, and it is simply unhygienic. Moreover, it will require users to remove it and put it back constantly. Also, I am against the idea of using a camera, since it must be positioned far enough to capture the whole jug and thus, it cannot be embedded into the device. It will lead to the camera installation related issues and difficulties with handling and securing it. Also, a neural network would be needed to process the data. Since it is not the most reliable method, it should be used only where any other methods are ineffective.

After filtering the bad ideas, I came up with two good ones. I could use an optical sensor, embedded into the device, which can provide data while staying outside of the jug. The other option is to use weighing-machine under the jug. Since there is usually a heating element underneath it and the water have more chances to ruin the device if spilled, I decided to proceed with the first option. Also, it would be much easier to add a device to the side of the coffee machine than to place it under the heating plate. Moreover, I believe that it is less likely for readings to be wrong if optical sensors are used. There are no moving parts in them and no sticky liquids or friction to spoil the results.

There are different optical sensors, but not all of them are suitable. Initially I wanted to use IR distance measurement sensor. The idea was to place it under the angle and measure the hypotenuse of the right triangle, as shown in a Figure 1. The brown box, and an image on the right represent the positioning of the sensor. The *h* is the known side, the distance from the bottom of the jug to the sensor, the *a* is the angle and it remains constant. Having the hypotenuse length, by checking the distance from the sensor to the coffee surface, we could use cosinus of angle *a* to get the left side of the triangle formed by the new hypotenuse.

This way the coffee level can be easily calculated by subtracting the new side from the initial *h*, the difference will be the the coffee level in the jug.

Figure 1. The distance sensor placed under the angle method

Unfortunately, it has not worked out, when I have done the testing. I guess that the problem was in the glass blocking and distorting the IR light waves. Maybe a laser distance measurement equipment could be used, but it is more expensive, and I am not sure that there is a portable enough component to be used in the IoT coffee machine.

During the same test, I have found that the ambient light sensor readings significantly vary based on the presence of the coffee in the jug in front of it. The coffee blocks the light and this gives a possibility to detect coffee on certain levels. If the sensor has been placed below the coffee level, it will show about 0 Lux. Meanwhile, the sensor placed above the surface of coffee showed around 60 Lux. Moreover, when the sensor was right on the level of coffee surface, it showed 2-5 Lux result. All these facts tell that it is possible to use ambient light sensors to detect the coffee.

It won’t give the precise amount of jug content, but by placing them as shown in Figure 2, it is possible to get enough data for a useful estimation. Also, the ambient light sensors are inexpensive and small. I assume that at least 4 can be easily placed inside of the IoT coffee machine casing. This number can be doubled by having two rows of sensors or using some other ways of positioning them.

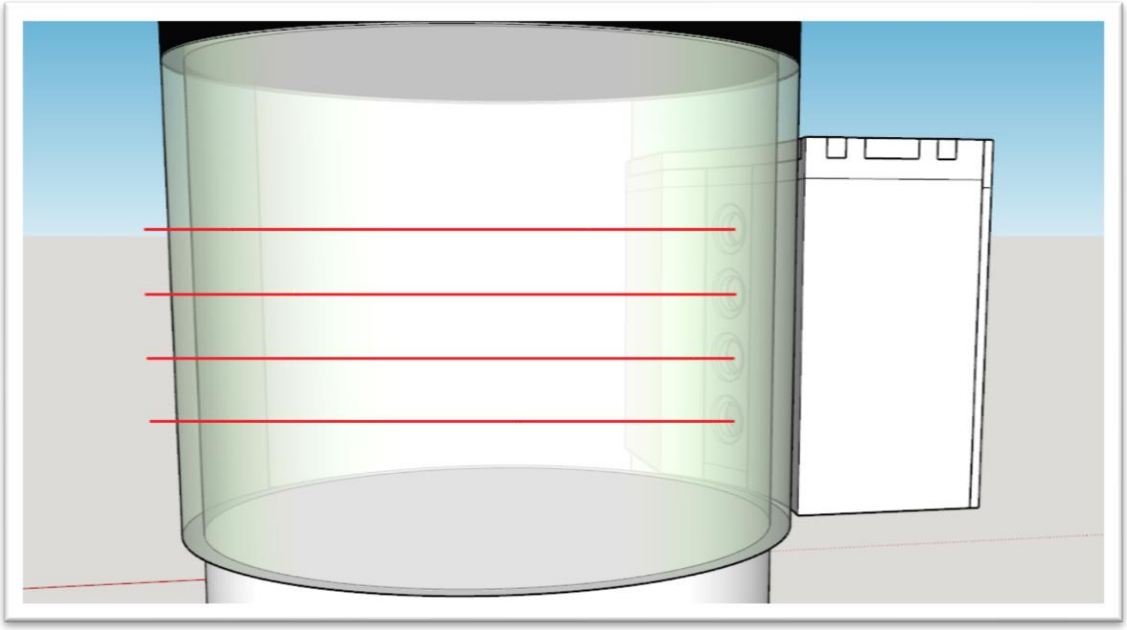


Figure 2. The ambient light sensors method

Ambient light sensors require a light source to be present near the device. It could be room lamps, or some dedicated source of light embedded into the device. Light levels are different for every room, so adjustments need to be made. This can be done by adding calibration feature.

Based on the tests, it is possible to detect 3 sensor’s states: the coffee level is above the sensor, the coffee level is below the sensor and the coffee level is on the sensor’s level. This allows to track 2n+1 coffee levels where n is the number of sensors installed.

Detection of the presence or absence of the jug is also necessary. The glass deflects enough light to make a significant difference in the light levels, which

makes it possible to detect it. This will help avoiding the situations when coffee would be ordered and poured onto the floor instead of into the jug.

2.2 Planning Logic

The purpose of the device's program is to make decisions based on the sensor readings. Thus, the threshold need to be defined. If the light level is above the threshold the readings will be interpreted as "coffee is below this level", near the threshold – "coffee is on this level" and below the threshold - "coffee is above this level". The value, determining how close to the threshold the readings should be, needed to detect when the coffee is exactly on the sensor's level. This value could be called "Range". For easier processing all the readings could be stored in ternary system number with an *n* digits, where *n* is the quantity of the sensors and with 0 representing absence of coffee, 1 - presence of coffee on the sensor's level and 2 - presence of coffee above the sensor.

Since the empty jug is also reflection some portion of the light, the maximum light level for the sensors can be also defined. If the readings are above this value, it means, that there is no jug in front of the sensor. Such result can be coded with the digit 9, for easier processing. Even if it makes the result not to be a ternary number anymore, it will be processed separately and will not influence the previously designed logic.

The total number will represent the state of the coffee level. The state will define which information should be sent to the user. I believe it is also possible to track the state changes and send additional data accordingly. This will enable to determine when the coffee is brewing, show its age, and detect that the lights in the room has been switched off.

For better user experience, some degree of error resistance should be provided. If the digit representing the readings of the lower sensor is smaller than of the one of the higher, then one sensor is dirty, failed or its calibration is incorrect. In this case the error should be detected and, maybe, even fixed. The best option is to create a logic capable of determining which senor have failed and what its

readings should be. Also, a warning could be shown to the user and error should be logged for easier troubleshooting and repairing.

The current state is the last stable state detected in the past. The new state is the last result of reading the sensors. To provide better consistency these two needs to be tracked separately. The new state can become the current state if it has not been changed during several measurements. To check if the new state is stable, one more state, a contender state, and a counter could be used. It will help avoiding false data, in case someone passes by and prevents the sensors from capturing the light for a small amount of time. Only current state should be shown to the user. The state changes could be tracked right before the new state becomes the current state.

The process of a new portion of coffee creation can be detected by keeping track of sequential changes of current state from low to high. When the level has risen the creation time of coffee should be saved or updated. For better consistency, the coffee level should stably grow for few levels before the brewing process can be detected. Also, detection of the jug presence is crucial, since inserting a full jug should not be counted as a coffee creation process by the device.

Since the device will heavily relying on the room light or on the LED’s, the lights off event must also be detected. I believe that it is possible to do, by checking how the current state changes. The comparison with the previous state or checking if the coffee is brewing could help.

Since the device needs some data to be provided beforehand, a configuration tool is needed. The device logic should be able to work with it, by getting new configurations, storing and using them. I have found out that the EEPROM could be used to store some data between reloads. And since the device cannot connect to the access point before it is configured, it will have to start its own one. Thus, it should have a configuration and user modes and be able to switch between them.

2.3 Network connection

As any IoT device the new generation IoT coffee machine must use network to provide an information and an interface to the user. The device should use a wireless connection and have its own IP address to be accessible. It must be easy to connect the device to the wireless network and check the connection. The user should be able to choose from static and dynamic IP addresses and have a full access to settings. For better user experience a domain name for the IoT coffee machine IP address could be assigned on a local DNS server or a router.

The network connection is essential for IoT devices, so access point should not be far away. This should not be problematic, since the device is designed to be used in the common rooms. A stable Wi-Fi connection is supposed to be there anyway.

The device will use a web interface and act as a web server. It will return an html page as a response to the user requests. REST endpoints could be used for both sending data and controlling the device. This approach not only helps to keep things structured, but also makes it possible to create applications that will access the device using the same points.

2.4 Web interface

To access the IoT coffee machine users should insert its IP address or a domain name into the address field of any browser. The use of the web interface enables users to access and control the IoT coffee machine from any device. This is a simple and reliable way, but it is highly recommendable to create a mobile first web design for the device web page. Also, as I stated above, dedicated applications can be created for major operating systems.

The purpose of the interface is to provide users with information, including but not limited to the amount of coffee available at any moment, the presence of the jug

in the device, age of the coffee and any warnings and additional information. Information should be given to the user in a form of a simple graphical interface. For example, the picture of an abstract jug could be used. This image will be changed according to the current coffee level. Therefore, if the jug is full, the user should see an image of a full jug. There should be an easily understandable picture for every case such as coffee levels, empty jug, or absence of the jug. It is also possible to add a sign on top of it to give users an even better idea about the amount of coffee available.

All the graphics and other files should be stored somewhere, preferably on the device itself. For the proof of concept, I will use links to the external sources, to avoid some limitation. Since the device is supposed to have internet access, it is always possible to use it, in order to provide better user experience or to collect statistics. Automatic updates could also be implemented, so the device will have all the latest features at any given moment.

The device should also provide some controls, like ordering a new portion of coffee, scheduling the next brew and any other features that could be implemented by having data about the current coffee level, and which I will describe in the later sections of the thesis. The interface to these functions of the IoT coffee machine should also be simple. Good looking buttons and user-friendly controls could be used. The other concern is reliability and limitations that should be implemented to protect users from mistakes or misunderstanding. The button to brew a new portion must only be enabled, when this action won’t overflow the jug and there are no other problems with the device. The indicators and warnings must provide users with information about the reasons why certain actions can’t be performed.

Additional pages could be used to provide users with more features and information. A page with extended controls, configurations and debug information would be a good addition to the product. This page must have some sort of authentication. Also, a page with instructions could be useful. All supplementary pages should be accessible from the main device page, but should not attract too much attention to themselves.

For configuration purposes, a web or mobile application could be created. It can simplify the installation and management of the device. This program should be able to connect to the unconfigured device wirelessly. Good interface will help configuring the product. Since there will be many settings, they need to be structured and presented in groups or on separate pages.

2.5 Casing requirements

Since the device will use a lot of electronic components and it is supposed to be close to liquids all the time a proper casing is essential for it. First, the device must be water resistant, since users will spill some amount of liquid on it anyway. Second, the holes for the sensors and probably LEDs will be required. These holes must be tightly covered by transparent plastic.

A tight connection between the jug and the device is required, so the casing should be well designed to fit the form of the jug. For better connection, rubber band could be used to help users with the proper insertion of the jug and to prevent light from external light sources from affecting the measurements. It could be added to the corners of the casing to surround the jug. This is very important since all the new features will heavily rely on the accuracy of the device.

For the proof of concept, I will create my own casing for the IoT device for the Moccamaster KBG741 coffee machine. I will use 3D printed parts to show how things should be done and to test them. I will also try not to damage the Moccamaster, so I am planning to add as little permanent installations as possible and to design a proof of concept to be a removable external attachment.

To design the casing I will use a 3D modeling environment. With certain plugins, the model can be exported in an appropriate format for the 3D printer slicer. It will not tolerate even slight errors, so the model must be checked for hidden flaws. The 3D printer has its own limitations. For example, there should not be unsupported fragments. Otherwise the model will require a lot of post processing, since the slicer would add some material to support the model elements.

3 BUILDING A PROOF OF CONCEPT

To ensure that my idea is possible to implement and to test it, I will build a proof of concept. It will be an IoT module that will extend the Moccamaster KBG741 coffee machine capabilities. I will use one at the South-Eastern Finland University of Applied Science and test it with a help of the staff.

I will describe the requirements for components, choose the appropriate ones and test them. This will help me to improve the final concept by facing and solving problems during the tests of the proof of concept device. I will also create a logic, that could be used as a base for the final program of new generation IoT coffee machine.

All instructions, schemes and code will be attached to the thesis, so it will be possible to recreate the device for testing or for the use in offices. I will describe the components and include links to official documentation. Also, a guide for the device assembly is available in one of the sections below.

3.1 ESP-8266 12-E

The device needs a microcontroller to handle all the processing, network connection and logic. An appropriate board is the key to success. There are plenty of microcontrollers to choose from: Arduino, Raspberry pi, etc.

For the core of my module I decided to use the ESP8266 12-E Amica module from NodeMCU. It is a small Arduino like board with an embedded Wi-Fi module. According to the documentation, it has lots of features that will be useful for my device. The following list introduces some of them:

- 802.11 b/g/n;
- Integrated low power 32-bit MCU;
- Integrated TCP/IP protocol stack;
- Wi-Fi 2.4 GHz, support WPA/WPA2;
- Support Smart Link Function for both Android and iOS devices;
- Operating temperature range -40C ~ 125C;
- I2C capabilities.

- 512-bit EEPROM

The most important thing about the board is its pins and their function. Figure 3 shows them and their names. The most interesting ones for this project are 3 sets of 3.3V and grounds to power sensors and multiple GPIO pins that could be used for different purposes, depending on their configuration. For example, according to the documentation they could be used as I2C pins.

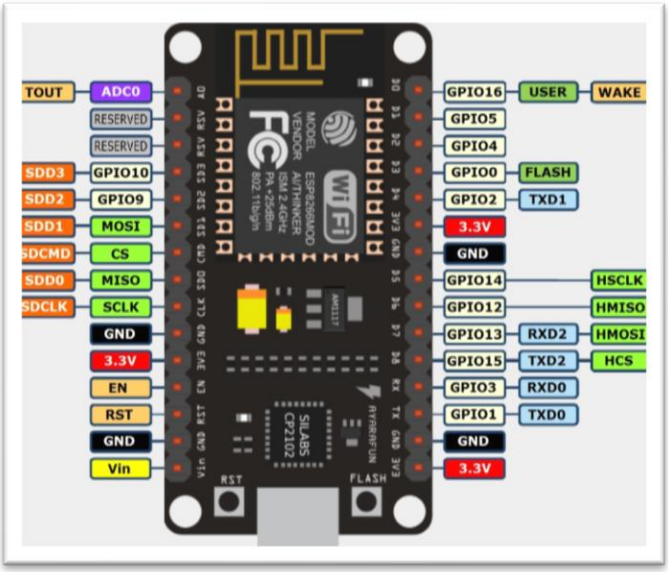


Figure 3. ESP8266 12-E pins

Not all pins are equally suitable for being used as SDA or SCL. I have used GPIO pins 5,12, 4, 14 and 13. I had trouble with 1,3, 9, 10, 15,16. Pins 0 and 2 should not be used, since they have other purposes, such as controlling flashing or re-seting processes.

There is also a MicroUSB port which could be used to power the board, program it and monitor the data coming to the port. To make a PC work with a board CP210x USB to UART Bridge VCP Drivers need to be installed.

To flash the board a NODEMCU tool named NodeMCU Flasher could be used. It needs to be downloaded and configured. To program the board, I used the Ar-duino IDE. It requires configuration to work with ESP8266 boards first. I will de-scribe tools and their usage in more detail in section 3.3 of this thesis.

Most of this information has been taken from the official documentation (NodeMCU board documentation). There is much more useful data about the board and standard libraries. Both the capabilities and the information about the syntax can be found in there.

3.2 GY-49

To detect the amount of light coming through the coffee jug, I decided to use GY-49 MAX44009 ambient light sensor, shown in Figure 4. It is an inexpensive ambient light sensor with an I2C interface. It could work using 1.7V - 3.6V power supply and require 0,65 μA. It is only 10,5mm*13,0mm in size. It is possible to easily fit them at any place.

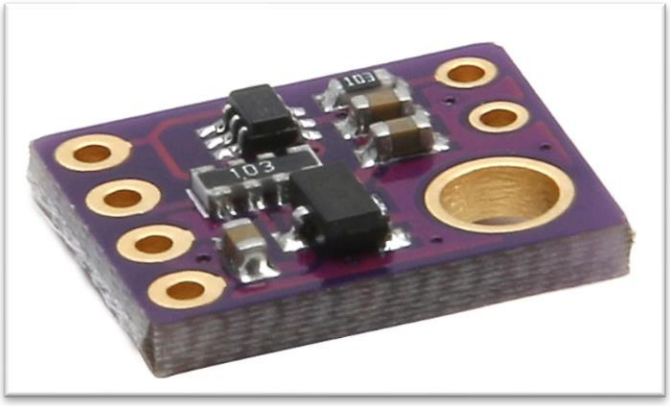


Figure 4. GY-49 MAX44009 ambient light sensor (Ebay)

This sensor uses an I2C interface, so it requires four wires to be connected to it. Power, Ground, SCL (Serial Clock Line) and SDA (Serial Data Line). All the sensors have the same address. It means that they can't be connected in parallel to the same I2C interface. This is a serious limitation for microcontrollers with a hardware implemented I2C controller, but the ESP8266 uses a software implemented one, so this problem can be solved by having multiple I2C buses.

To work with the sensor the Wire.h package needs to be included in the program. A function to send commands to the sensor and read its output is used. The output must also be converted to the Lux according to the special algorithm

The sensor should be close to the jug for more accurate results. The light should not be coming from any unexpected directions to ensure that the light level is solely defined by coffee level. Sensors should be encased properly and have a limiting sensor view hole. Figure 5 shows, how sensors could look like in the device.

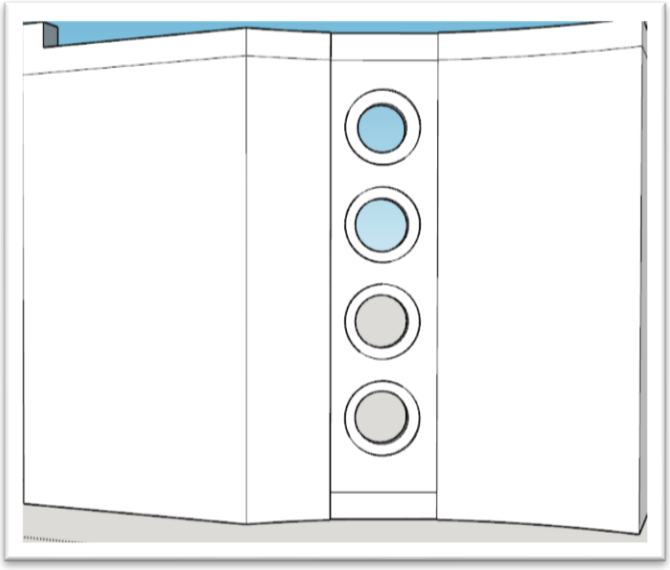


Figure 5. Sensor array example

The MAX44009 datasheet (Maxim Integrated 2011) was a primary date source about this component. It contains all the information needed to work with the sensor. For example, the control codes, required to make it work with I2C interface.

3.3 Tools and development environment

To start using and programming the ESP8266 boards the driver needs to be connected to the PC via USP port. The CP210x USB to UART Bridge VCP Driver for the port is also required. After the connection is established it is possible to flash and load the program to the device.

There are several tools required to work with the microcontroller and to create a program for it. One of them is NodeMCU Flasher, it is useful for updating the

board flash. It could be used to reset the microcontroller in case it would be impossible to load a new program on it due to overload. It can be found on the GitHub (NodeMCU Flasher). To flash the ESP8266 12-E Amica it is recommended to set the board baud rate to 115200, flash size to 4Mbyte, flash speed to 40MHz and SPI Mode to DIO. In configurations, it is possible to set the firmware needed to 0x00000. The firmware could be also found in the Internet (NodeMCU firmware).

Another useful tool is Arduino IDE. It could be downloaded from the official website (Arduino IDE). After the installation Arduino IDE needs to be configured to work with ESP8266 board. To do that the http://arduino.esp8266.com/stable/package_esp8266com_index.json URL needs to be added to an “Additional Boards Manager URLs”. A EP8266 package needs to be chosen and installed using a board manager. Then the proper board type should be selected. It can be done in the Tools -> Boards -> Board Manager. I have used NodeMCU 0.9 ESP-12 Module. Now the Arduino IDE is ready to create and upload firmware to the board. To monitor the port, it must be opened in Tools -> Port.

For creating the casing SketchUp and an STL Import & Export extension could be used. To become printable parts, they need to be properly designed and exported as .stl files. All models need to have all faces properly directed and all the volumes closed. Thus, it is recommended to check them with some plugin that will find hidden problems.

Most of the libraries could be downloaded using the Arduino library manager and included into the program. It is important, since they can gradually increase the capabilities of the board, without requiring programming everything from scratch. Libraries could also be loaded from .zip files.

3.4 Logic

The logic is the main part of the device creation. It must be able to handle the data properly and provide results to the users. There are many problems to solve. Some of them arise even before the device starts to work.

3.4.1 Configurations

One of such problems is the fact that some data required to connect to the network. Since reprogramming the device each time the SSID or password must be changed is not a good solution, the configuration mode must activate first. It will allow user to use tools, like applications or web configurator hosted on the device itself. This will allow to change many parameters and modify the device behavior.

The configuration tool will be described in the section 4 of the thesis. It will use the ESP8266 access point Wi-Fi mode and store the configuration data on the device memory. First is necessary to set the network credentials before connecting to any other network. The second solves the problem of storing the data between reloads, so the EEPROM will be used. The configuration mode will have to work together with the main logic of the device. In the user mode the device is supposed to read the data from the EEPROM if any is present or use default values.

3.4.2 Starting the device

After the necessary data is provided, it is possible for the device to start performing its main function. The first thing is to setup the Serial port, to enable debugging, and connect to the network. To connect ESP8266 to the Internet the ESP8266WiFi.h library could be used. The program must be provided with the SSID and password. It is also possible to set an IP address, a subnet and a default gateway, manually instead of relying on the DHCP server.

The web server on the ESP8266 board uses REST endpoints. Each has an address, a response code and calls a function. They need to be preconfigured in settings part of the code and server must be started. Usually the response is a web page. Current values of global variables will be used to build it. After the current state and events are determined, the variables should be changed accordingly. The user will get the new information once the page is reloaded.

The program of the controller mostly serves the clients. 999 out of 1 000 loops it will only handle a client if there is any and increment a counter. When the counter reaches 1 000, it is drops to 0 and the program will check for any changes in a coffee level.

3.4.3 Classes

To store the data, I have created some classes. The “State” class keeps all information about the state, like its code, image link and level of coffee it represents. “StateList” class is an array of “States”, with improved functionality. It holds all legal states and finds them by id. It also can check whether there is a state with certain id or not. These functions will be useful for getting new state from sensor readings. “Sensor” class stores the information like sensor wires numbers and a calibration. It has methods for getting sensor’s readings and processing them. Array of sensor objects will represent the physical sensors in the device.

3.4.4 Getting a new state

The first task is to read the outputs of the sensors. Sensor’s address used to transmit commands and receive data. Since all the sensors had the same address, I decided to take advantage of the ESP8266 software I2C interface and call for a new wire connection each time I need to use different sensor. The SCL is common for all sensors, but each of them has its own SDA.

The logic is built based on states and events triggered by the state changes. To determine the state, the sensor’s readings should be compared to the threshold. If the module of difference is smaller than range, the result is 1, if not and the readings are smaller than the threshold, then the result is 2, otherwise - 0.

The result of individual sensor should be multiplied by $10^{(n-i)}$ where n - is a number of sensors and *i* - is the index number of the current sensor, counting from the top. Processed readings form a *n* digit ternary number. The number represents the current sensor’s readings and the id of a new state.

Besides 1, 2 and 3 there is also digit 9 used in the state id. It shows that the light level of the sensor is above maximum level, if it's has been specified. This will help with detecting the absence of the jug. Everything related to 9's follows the special rules and the usual methods may be not applicable for processing 9's in the state id. The state with id equal to 9999 means that someone has removed the jug from the coffee machine. This will be changed to any other state, once a jug appear and state will be detected.

The current state could be changed if only there is a reliable new state. Thus, the new state, produced by processing new readings, should repeatedly appear for a certain amount of times before the current state will be changed. To store the data about the previous new state and count how much times it has appeared already a contender state and a counter are used.

If the new state is different from the contender state, the new state becomes a new contender state and the counter is nullified. Once the new state is equal to the contender state and the counter is equal to or bigger than the amount of stable readings required, the contender state becomes the current state and the counter nullifies. If the current state is about to be changed to any other one, the check for the events triggers.

3.4.5 Events

Events are the special occurrences that detected based on the state changes or sensor outputs and used for providing the users with more information. The following list introduces them.

- Coffee is brewing – This event triggers when the coffee level is steadily rising. If the current state has been changed to the higher one for certain amount of times in a row, it means that the coffee level is rising. The only logical option here is that the new portion is in the process of brewing. When this event is triggered it should update the coffee creation time and add an alert to the webpage, informing users about the event, for a certain amount time.

- Blackout – a state when the light has been switched off or the LEDs have failed. For the program, this could look as the presence of a full jug in front of the sensors, since all the sensors's readings are 0 Lux. It is necessary to distinguish blackout from the presence of the full jug. Thus, the event should trigger when the state has been changed to 2222 from any other then 1222 or 0222.
- Error detected – This event triggers whenever the state is invalid. The event should enable the warning on the web page and log an error message with useful information to the port. The warning is meant to notify the users that the information may be unreliable. The most probable coffee level should still appear on the web interface. The logs will help to find and solve the problem, by pointing at the sensor that most probably needs to be checked.

3.4.6 Timestamps

To use timestampt in the program a connection to NTP server needs to be established. I used a library NTPClient.h (Sandeep Mistry). I have defined an NTP server I wanted to use, and the program can request the current time from it. The main purpose of timestamp here is to reliably show when the coffee has been brewed. I keep this timestamp in a global variable and update it every time "Coffee is brewing" event triggered.

I also store the timestamp in a raw numerical form. This variable is used to determine wether the alert need to be shown or not. If less time passed from the moment coffee has been brewed than the reload interval, the user will be notified about the new portion has been brewed recently on next reload. To determine, if the alert needs to be shown, the second timestamp is received from the server. It contains the current time and used to calculate how much time has passed since the coffee has been created.

3.4.7 Error handling

To deal with errors, error correction could be implemented. It starts, if the id of the newly acquired state is not in the legal states list. In this case, while there is no legal state, the error correction code must produce a new id, which should represent the current coffee level as closely as possible. There are several steps done to detect and fix an error. First, the check for the absence of the jug should be performed. It counts 9s in the state id and if there is more than a half of these digits in the state id, it changes all other to 9s and produces a new id. Otherwise, it changes all nines to zeros and starts the main part of the error detection process.

After there are no more nines in the state id, all sensor readings are compared to each other to find conflicts between them. The conflict is an impossible pair of results, such as if the higher sensor has lower readings, if the lower sensor has higher readings, or two different sensors show 1 as a result. All the conflicts are calculated and used later for deciding which sensor's readings are most probably unreliable.

With an error count it is possible to find the sensor with the most conflicts. If there are sensors with equal number of conflicts, the one on the top is deemed failed and others are added to potentially failed list. Potentially failed sensors will be also logged. At the same time, the failed sensor's result will be changed to cause less conflicts with other ones. If the top sensor is found to cause problems, its state will be changed to 0, so it could no longer have conflicts. The bottom sensor should check the sensor right above it and change its readings accordingly. If the sensor above is greater than 0, the lowest sensor's readings must be 2. Otherwise it will become 0.

Things are a bit more complicated for the middle sensors. They need to check the one sensor below and the lowest one for any readings besides 2, and if there are any, they should change their value to 0. Otherwise, they need to check the sensor on top of them, and if it also shows 2, the failed sensor readings will be changed to 2. If the sensor above is showing 0, the readings of the sensor with

trouble will be set to 1. After that the new state id is produced and checked for validity. The process is repeated until a proper state id produced.

The principles of error handling are to trust the bigger number of sensors, to trust the lowest sensor and to produce the lowest possible coffee level. The first one is based on the assumption that in most cases there will be only one failed sensor. The second and third are meant to reduce the number of situations when the user will be disappointed. Later the maximum state number could be implemented to ensure that there is enough space for the new coffee portion.

3.5 Web Interface

The web interface I made is rather simple, but I have done my best to show some possible features. The actual interface can be seen in Figure 6. It has the header and the footer as well as a row of three divs: left, main and right. Overall, I have tried to represent the data in a more native way, so I have used graphics and always preferred to support the text with icons.

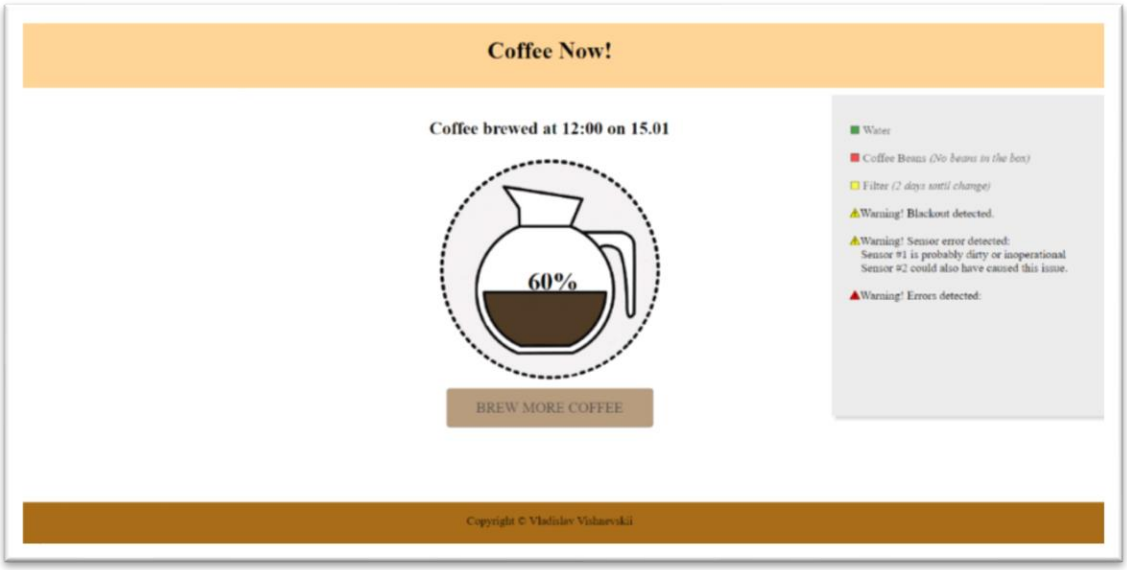


Figure 6. Web interface prototype (The bigger image is available in appendix 3/1)

For now, the left div does not serve any purpose except a decorative one. In future, all the links to additional pages could be placed there. In the mobile design,

these links must disappear to free space for the rest of the content. The burger menu with navigation will appear.

The main div contains the time of the last coffee brewing process. There is a jug image right below it. It has a sign on it, giving a more precise understanding of the current state. The jug image is loaded according to the current state. Each state has its own link. So, for different levels there are separate image file, with varying levels of coffee drawn. The jug will also be half transparent in case it should represent the absence of the real jug of the coffee machine. There is a big button below the image. It represents the future function to brew coffee. It is disabled now, for an obvious reason, but I believe it should look somehow like in Figure 6, since it is the main function of the IoT coffee machine.

In the right div there is a board with all the additional data. At the top, there are indicators which are supposed to give the user information about the device supplies, if there would be any. It also can help with keeping people informed about the need of maintenance, like filter changes. All this will help the device to work more reliably. After that there are warnings that appear if something is wrong. This includes sensor errors and blackout. A debug mode will also put the sensor’s outputs onto this board.

Also, there is a script that is added to the web page if the coffee has been recently brewed. It is a simple alert, noticing the users that fresh coffee is awaiting them. To get this alert, and to give users more recent data, the page is set to update itself after some period. I set it to 10 minutes in order to not hesitate the users.

The web page is built from parts. This way the code could be reused later. The main part is a “Layout” function which contains the header, footer and all three divs without content. To fill the data the special methods are used. The “Jug” function returns the html text, so it can be placed in the content div. The “Indicators” method is used to get the code for the right div. For every page built these functions can be different. To decide which data should be added parameter is

used. This is not the most efficient way, but I am restricted by the board I use and the time I have. In future page should be built using templating engine, for example Dust.js.

The web interface page is sent as a response for the request to the board endpoint. It is built using global variables like current state, the coffee brewed timestamp, logs and data about the events. This allows freely manipulating the data and showing it in a nice, understandable way.

3.6 Casing and connector

To connect the prototype to the Moccamaster KBG741 coffee machine, I had to create the casing. It should provide structural support for all the elements and protect the insides from liquid and damage. Also, since I did not want to ruin the coffee machine, I decided to use a connector to attach the device. This way it is easy to remove my upgrade and for example send the machine to the warranty service. The only permanent installation is a rail.

All the files of components and the whole model will be available in the appendix 2. It could be modified to adapt to any other coffee machine. Some improvements could be beneficial even for the Moccamaster KBG741 IoT module casing.

To create the casing, I firstly carefully measured Moccamaster KBG741. After that, I have created its 3D model. It is shown in Figure 7. This way, I could develop the casing by building it around the coffee machine.

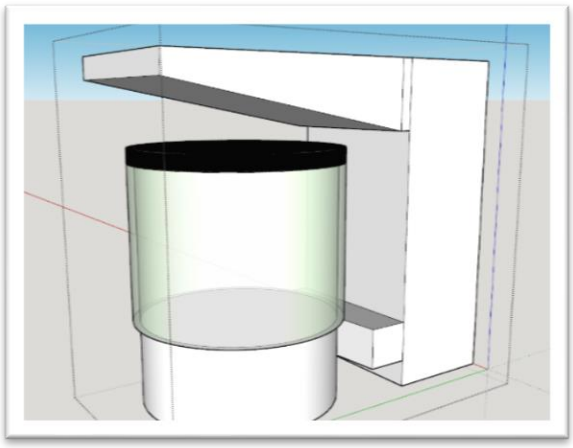
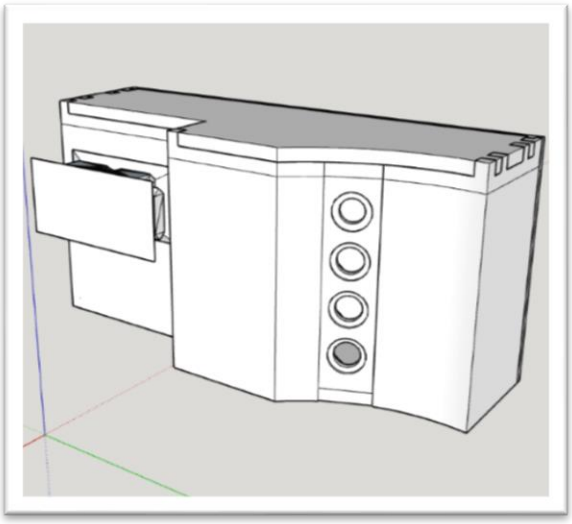
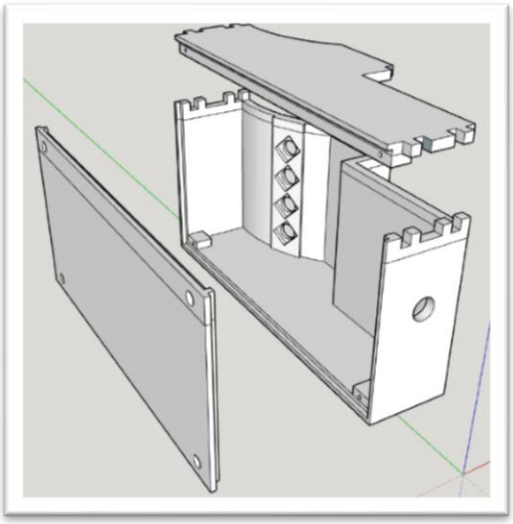


Figure 7. The Moccamaster KBG741 3D model

I have started with creating a curved part which is supposed to surround the jug. This can be seen in Figure 8-1. This was the hardest part, since work with curved lines lacks precision. Fortunately, minor flaws could be compensated with the rubber band on the curved part perimeter.





Figures 8-1 and 8-2. The casing

The casing was created to host the wires and the board. It was a simple box shown in Figure 8-2 and consisted of three parts. The first is the main one, with the bottom and most of the sides, including the curved section. The second one is the top part. It can simply be inserted into the locks or glued to the main part. And finally, there is a side part that can be removed for maintenance purposes.

The walls are 5 mm thick and all connections made as tight as possible to provide some degree of water resistance. The material I used was ACL plastic for printing.

3.6.1 Main part

The main part was printed in a single piece. It has some features that need to be described. First, it serves as the main structural support for both the board and other casing parts. The connector is glued to it and the whole weight relies on it. The extending pins on the top are used to hold the top part in place. On the back side, there is a hole of 12 mm in diameter for a microUSB cable, so the board can have the power and connection to the laptop for reprogramming without removing it.

The most interesting part is the sensor support shown in Figures 9-1 and 9-2. I made sensor holes with a support for the sensors using three forms removed from the wall. First, from the inside side there is a rectangular parallelepiped turned on 45 degrees, so it could be printed without much support. The sensor is supposed to fit inside it, so it would be easier to glue it and the connection will be stronger. Then there is a small hole in the center. It should provide a view to the sensor, but also keep it from falling off. The third part is on the outer side. It is a circle concavity where the transparent plastic should be inserted and glued to. Such multi-layer structure simplifies the assembly and helps to protect the sensors from being damaged or fall off.

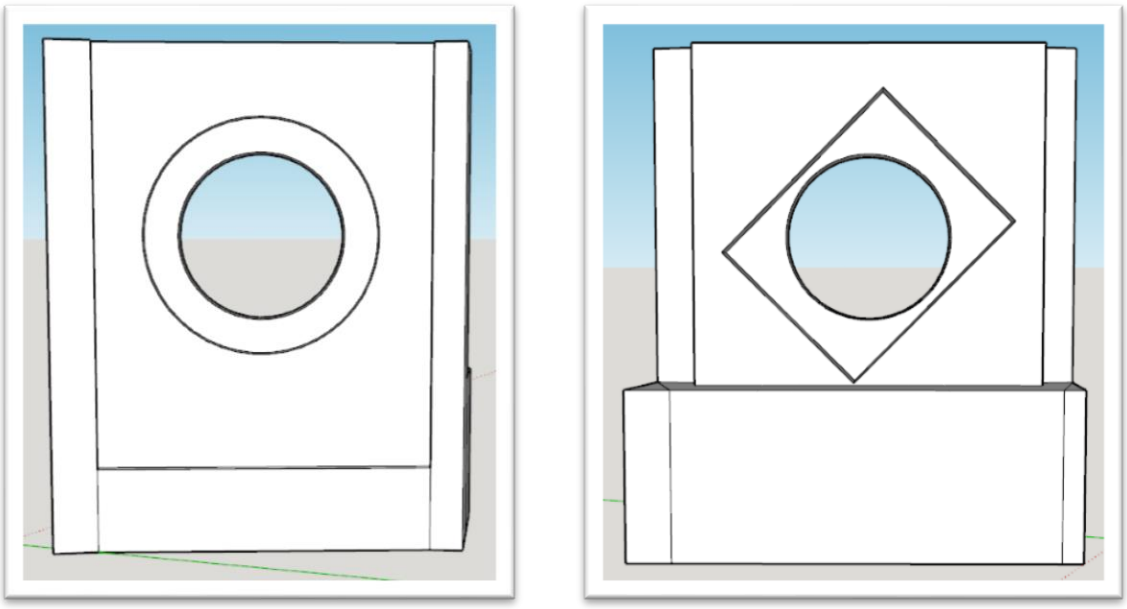


Figure 9-1 and 9-2. Sensor support

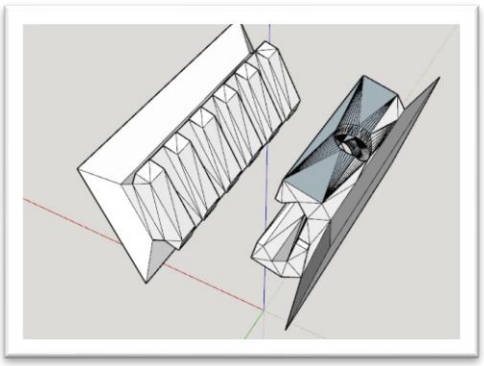
On the perimeter of the curved part the rubber band has been glued. It provides better placement and closes the sensors from the side light and makes measurements more accurate. I decided to use a P-profile window seal for 3-5 mm gaps. Therefore, I left 3mm between the closest position of the jug and the casing itself. I have glued the rubber on the perimeter of the curved part with a wider part pointing to the outside.

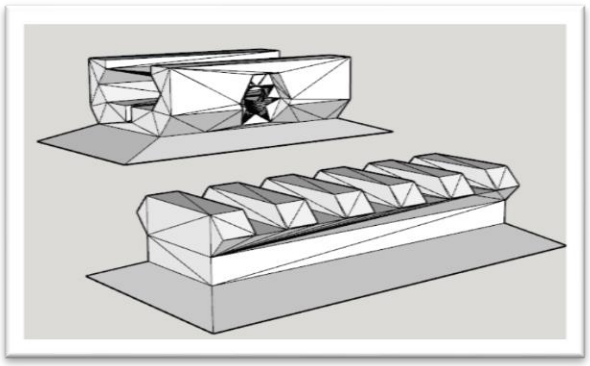
The main part required a lot of post processing. The 3D printer added a lot of supporting material on every hole. This material was quite hard to remove. I have used sharp knives and sandpaper to remove excess plastic and to smooth the shapes. This must be done cautiously, or in a completely different way, because there is a risk destroying the casing, or injuring oneself.

3.6.2 Connector and rail

I used existing models of the rail and mount and modified them. For the rail, I used Picatinny Rail (Mykail 2014). For the mount, I have printed Mobius Picatinny Rail Mount (JimmyU 2015).

Since I have used the rail similar to the Picatinny rail, the connector must be wrapped around the edges and have a hole to insert the screw as shown in Figures 10-1 and 10-2. There is a special place for a nut on the other side of the hole. The connection is not perfect, since I could not tighten the connector and did not want to change the rail, which would cause it to be non-standard. Nevertheless, the connection is reliable enough and the device can be pressed against the wall for a little bit better positioning.





Figures 10-1 and 10-2. Connector and rail

The rail must be glued to the coffee machine, while the connector to the main part. I have used super glue, and it was very effective for gluing plastic to plastic or plastic to metal. Once both parts are glued properly, it is possible to slide the device onto the rail and fix it with the screw.

For the testing purposes I have printed the Picatinny rail and mounted and glued them to a metal plate using Ethyl cyanoacrylate glue. I managed to lift a computer holding the glued part. It will definitely be able to support the weight of the device, although I have no idea for how long the glue will hold plastic and metal parts together.

3.6.3 Side part

The side panel’s purpose is to protect the device, while providing access for the maintenance and upgrades. This why it is made to be removable. The form is

quite interesting on the side that faces the insides of the device. It is made to fit the main part better.

On the open side of the main part there are four wider sections with screw holes. I have created a 6 mm in diameter cone shaped recess for a head and a hollow 2 mm in diameter and 2 cm long to make it easier to apply a screw. This hollow is separated across two pieces to hold them together.

Unfortunately, only the bottom screw holes are usable, since the top ones are printed badly. Moreover, it is hard to insert or remove the screws, so it must be done with caution, not to destroy it. It will be troublesome to remove it otherwise.

There is also a thinner wall on the perimeter. Its purpose is to ensure a good connection between the main part and side part. It can be fit into the main part only in a single way and then screwed tightly.

3.7 Assembly

After everything is ready it is finally possible to assemble the device. Some instruments such as a screwdriver and scissors will be needed. Also, here is a list of components required:

- Casing main top and side parts;
- Connector and rail;
- A nut and a screw for the connector;
- P-profile window seal for 3-5 mm gaps;
- Breadboard;
- ESP8266 12-E board;
- MicroUSB cable and adapter;
- 4 GY-49 ambient light sensors;
- Simple and male-to-female wires;
- A piece of transparent plastic;
- Superglue (Ethyl cyanoacrylate), plastic glue, 4 screws.

First, the casing needs to be assembled. The top part should fit into the pins of the main one. The connection can be strengthened with glue. The side part should also fit the open side of the main part. I recommend checking it and screwing the parts together beforehand to spot all the imperfections and complete the screw holes by using them. The M6 nut should be placed on the special spot of the connector. Due to some inaccuracies in the model, the spot need to be made smaller or widened with a soldering iron and a M7 nut placed instead. The connector itself should be glued to the main part around 20cm below the top edge, in parallel to it, so that the nut would be on the bottom. The screw should be added, so that it would be possible to fix the connector on the rail.

Once the casing is assembled, an ESP8266 12-E should be placed into the breadboard, so that the legs will fit into the different rows of holes. Every leg is supposed to have a free spot for a wire close to it. The breadboard itself should be glued to the main part from the inside, so that the microcontrollers MicroUSB port will be accessible through the hole made for the microUSB cable.

To prepare the breadboard to work with sensors, the wires need to be placed to give power and I2C wires to the sensors. The 3.3V pin of the ESP8266 should be connected to the power row of the breadboard and the microcontrollers ground to the ground row. Since SDAs are individual for the sensors, the wires can go straight into the holes near to GPIO pins. But since the SCL is common, it needs to be paralleled, so that all the sensors can have access to it. The schema is shown in Figure 11.

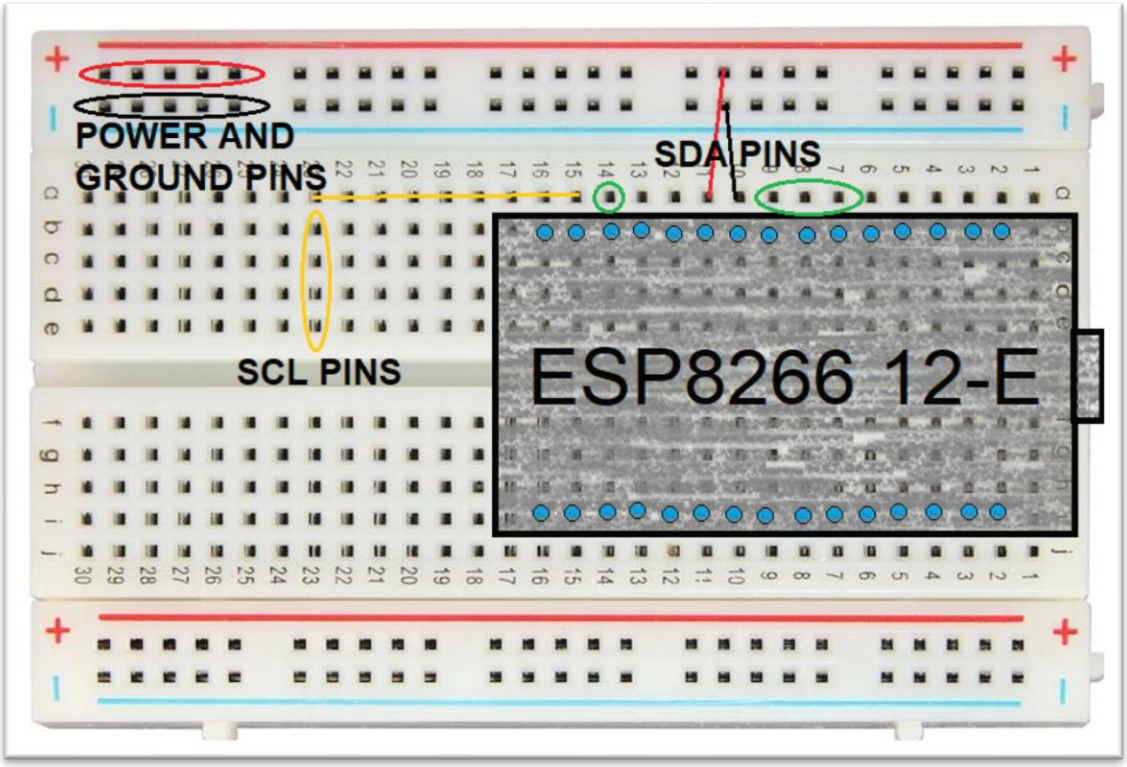


Figure 11. Wiring

With the breadboard with the controller glued, it is possible to add the sensors. Each sensor comes with legs, so it is possible to solder them to it. In this case, the male-to-female wires will be needed, to connect it to the board. The other option is to solder a wire directly to the sensor. The end connected to the breadboard should be male.

The sensors should be fitted into the rectangular parallelepiped spot on the inside of sensor support. If it sits well, it can be glued with a plastic glue. On the other side the round piece of plastic needs to be glued in the same way. It needs to be cut first, its diameter is 14mm. Some adjustments could be required to fit it onto its spot. The rubber band should be placed on the perimeter of the curved part. Wide half of the profile should face outward.

With the sensors on their places, they need to have four wires each: 3.3v power, ground, SDA and SCL. I recommend using wires of different colors or mark them with duct tape to avoid mistakes. The power and ground wires should go into the holes of + and – rows of the breadboard. I have wrapped all such wires together

to form an array. This way they will support each other and the probability of one to fall off is minimized. The SDA wires should go into the holes near to the respective GPIO pin. All the SCLs must go to the line where the SCL of the ESP8266 has been paralleled.

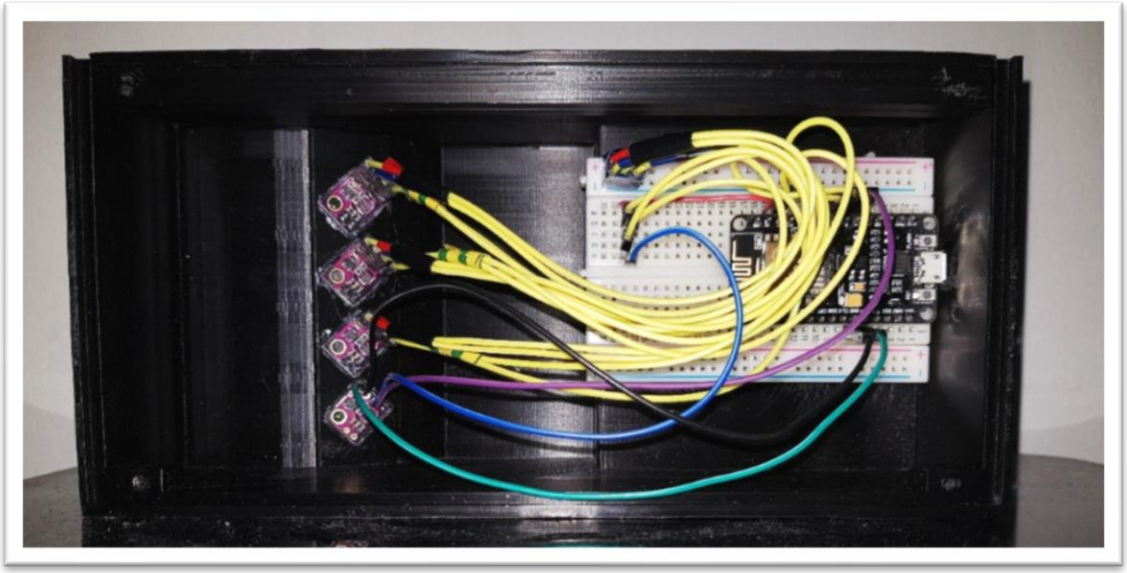


Figure 12. Insides of the assembled device

After all the manipulations the device's insides should look like in Figure 12. The board also needs to be flashed and programmed. This has already been described in previous parts of the thesis. With everything in place and working the side part can be placed on its spot and screwed tightly. Then the device can be installed and put to work.

3.8 Installation and usage

To install the device, it is necessary to attach a rail and try it on to find a good spot. The rail must be parallel to the ground and the top and bottom of the device must be as close to jug's ones as possible. Once the proper placement is found, it should be marked, and a rail glued to this place. With the rail on its place, the device can be installed as described in the casing connector part. After that it must be connected to the power grid using MicroUSB and a 2.1 A power adapter.

With the device in place the only left is to configure and calibrate it properly. This can be done with the configuration tools described later in the thesis. Otherwise, it is possible to provide the device with all the data by reprogramming it. It needs at least a SSID and a password to connect to the closest access point.

Once it is connected and powered it will start measuring the coffee level. The device could be accessed from the same network, or from any other connected network using its IP address. The web interface allows using any device with a screen of a reasonable size.

3.9 The result

During this part of the thesis I have successfully developed a basic concept of an IoT coffee machineband built a prototype. While the list of features implemented is limited, due to the resources I had, it is still a working proof of concept. It can show the basic functionality and interact with the users.

This device has been installed onto the University of Applied Sciences coffee machine (Figures 13-1 and 13-2) and used by teachers for two months. The feedback was good overall and new functionality has been used with enthusiasm. Information gathered helped to develop the device further.

Figures 13-1 and 13-2. Installed device

The interface was simple and looked good on both PCs and mobile devices. It was easy to use, since the device could be accessed via its IP address. Also, presence of an advanced device with the IoT functionality seems to improve people's mood.

Nevertheless, there are plenty of things to be improved or redesigned. The casing could be smaller and lighter and the program more optimized. I have not performed crush tests, so the minimum wall thickness or the density of the filling are unknown to me. The reduction in these parameters can decrease weight, cost and time required to print the casing. Unfortunately, it can decrease structural strength and water resistance. I would also recommend increasing the diameter of the hole for the MicroUSB cable from 12 mm to 15 mm.

4 CONFIGURATION TOOL

To be useful the IoT device need to be connected to the local area network, so people can access it. Thus, the device must have at least an SSID and a password configured. The IP address should also be known to the user, so it must be shown once acquired, or configured by the user manually. In both cases the configuration tool is needed. It could be a configuration web interface hosted on a device, or any sort of application capable of connecting to the device wirelessly, using the network or Bluetooth.

4.1 Concept of configuration tool

In this part I will create a web interface, which will allow to use any device with a browser to configure the device. I want it to be able to change not only the network settings, but also all the other parameters, like calibrations, intervals and other global variables. This will allow not to reprogram the device unless major changes are needed.

There are few issues need to be solved, for the configuration tool of the ESP8266 board-based device. The first one is that the connection needs to be established, while it is impossible to connect the IoT module to the existing network. So, my

solution is to create a new network by making the device to create an access point upon its start. The second question is where to store the data. Since it need to be preserved between reloads, I chose to use the EEPROM. It is a small but reliable way to store configurations.

The hardest part, is to properly embed configuration tool into the current program. I want the device to load in the configuration mode and then, after some time, start executing the main program. This way, it will be easier to set up the device and hard for someone unauthorized to mess with the IoT coffee machine.

4.2 Starting an access point

The first thing the device needs to do is to create its own network by starting an access point. The name should remain constant, while the password could be re-configured in the web interface of the configuration mode. The ESP8266 can act as a DHCP server, so it will be easy to connect any device to its access point. The network settings should be configured, so it would not cause any problems. I recommend using the simple class C network. For the security purposes, it could be changed to the network with 255.255.255.252 mask to allow only one user at a time. For the demonstration I will go with the simple network, so anyone could connect to the device.

In the ESP8266 the Wi-Fi module has its own configurations stored on a permanent memory, so they are preserved between reloads. Therefore, to start the network, it need to be configured to work in WIFI_AP mode. I have used the same library that has been used in the section 3 of the thesis. In the ESP8266WiFi.h library the “WiFi.mode(WIFI_AP)” command will do the job.

The AP should only exist until the device goes to the user mode. The Wi-Fi module will be repurposed to be used in the client mode. This can be achieved with “WiFi.mode(WIFI_STA)” command. Then the board should act normally and serve clients in the network.

4.3 Web interface for the configurator

The web interface I have created is based on the layout, used before. This way, I tried to keep things standard and make it easier for the user to adapt to both interfaces. I believe that the familiarity of the user with the interface is a benefit that should not be ignored. The changes are in the content of the page. As shown in Figure 14 it consists of the input fields, divided into the groups. There is a network configuration forms, a sensor calibration form, a state settings form and a form to set up other things as the global variables and a configuration mode access point password. Also, there are two separate buttons. First is responsible for clearing the EEPROM and the second is for making device change to user mode.

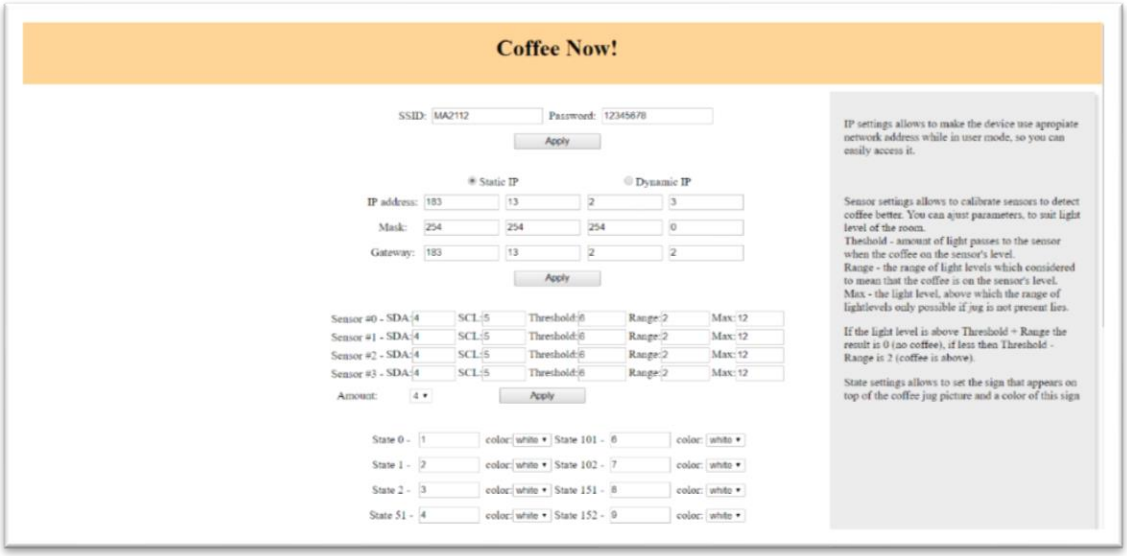


Figure 14. Configurator web interface (The bigger image is available in appendix 3/2).

Inputs on the page has a validation function. It is vital for both providing a better user experience and keeping the device functional and its behavior predictable. The validation is done by using appropriate controls for different data and minimum, maximum and length checks of the values. The inputs will display an auto generated hint if the user will try to submit the form with bad data.

The forms send the html requests to the device with the data from the fields. The endpoints will call corresponding functions and reply with a redirect to the main page if the changes been successful or with an error code if not. The functions

are responsible for checking the data by verifying their presence in the request, casting it to the required data types and validating values. No actions will be performed if the user have failed to provide the proper data.

On the right there is an information about the forms. The interface is not intuitive, so additional textual information could be used. The user will not use the page very often, so it should not worsen the user experience. The guidance and recommendations are done in a similar way as the indicators and the additional information in the main interface. Thus, I believe, the user will know where to look in case of troubles.

The main page itself is also served as a response to the html request and it builds with all the data in the EEPROM available. The simple data could be read at the beginning. Contrary, the information about the sensors and states is similar and processed dynamically in a cycle. Some other parts of the page are also built this way. This method saves space in the code and helps avoiding mistakes. If some of the values missing or wrong, they could be set to default one. Such checks are necessary to provide data integrity and improve user experience. Also, it is highly recommended to add a script that will block the form submitting by pressing enter. This can be achieved by adding this event listener: window.addEventListener('keydown',function(e){if(e.keyIdentifier=='U+000A'||e.keyIdentifier=='Enter'||e.keyCode==13){if(e.target.nodeName=='INPUT'||e.target.nodeName=='SELECT') {e.preventDefault();return false;}}},true).

The clean memory button could be used to set the device to its defaults. I will use it to clean the memory after tests and experiments, to avoid unexpected results. This button should not be activated by pressing enter and must have a confirmation. This way it would be harder for users to accidentally restore the device to defaults.

The last one is “To user mode” button. It is responsible for nullifying the variable that holds the number of seconds the device will be in configuration mode. Pressing it will cause the device to go into the user mode immediately. This way the user will not have to wait any longer after the configurations are done.

4.4 Usage of the EEPROM

The EEPROM is a small permanent memory on the ESP8266 board. It is only 512 bytes of size, so it must be used wisely. Only the most important values should be stored, preferably in the most efficient way. This way, there will be some limitations, as such for example the inability to change the image link, since there are 10 of them and they all could occupy a lot of bytes. With some limitations, such as the restriction to only Drop Box links, the usage of the same links for different states, the required space could be lowered to a reasonable number. The problem is that I don’t believe that this feature could be useful with all these issues.

To use the EEPROM, I added the EEPROM.h library. The EEPROM needs to be started with the “EEPROM.begin(512)” command first. To write a byte to the memory “EEPROM.write(<byte>, <memory cell address>)” could be used. After the data has been written the changes need to be saved, so the “EEPROM.commit()” should be called. To read the byte, there is an “EEPROM.read(<memory cell address>)” method which returns a byte.

While it is enough to set and get the simple data in bytes, methods for working with strings and numbers are needed. For the strings I decided to create a “WriteString” function which uses a source string and the beginning and ending of an address space and returns Boolean. The response is used to determine the success or failure of the function. The string is checked to be less or equal to the allocated space. If it fits the given address range, it is converted to the array of bytes, using ASCII encoding. The bytes are then written one by one and if the string is shorter than the allocated space, all free cells get a null value.

The second function is “ReadString”. It takes the beginning and ending of an address space and returns string. All bytes are read one by one, decoded from ASCII to character and added to the result string. If there is a null value, it is ignored. By having functions functioning this way, the strings could be of any size and the only limitation for them is to fit them into the allocated space. It is extremely useful, since it is impossible to accurately predict the length of the string passed as some parameter value, without creating unnecessarily strict limitations.

For storing numbers, I decided to use the unsigned short data type. It occupies 2 bytes only and can have a value from 0 to 65535, which is more than enough. The need for using them comes from the fact that the byte can only store the numbers from 0 to 255. It is enough for most of the variables. But there are some designed to have higher values. For example, the interval between the processes of checking the sensors is 1 000 by default.

To put the numbers into the EEPROM I have created a “WriteShort” method. It takes and unsigned short and 2 memory cell address. It divides the short by 256 to get the number that should be stored in the first byte and then takes the rest of the remainder of the division and stores it in the second byte. The 256 here is a mask, dividing the first octate from the second in the number represented in the binary system. To read the data, the “ReadShort” function reverses the operations. It reads 2 bytes from memory cells, multiplies the first by 256 and returns a sum of numbers. The addresses of the cells are provided to function as parameters.

To clean the memory, I have created the function “Clean”. It takes the beginning and ending of an address space and writes 0s to every cell within the range. This particular function is responsible for clearing the EEPROM to restore the device to its defaults.

With all types of data ready to be written and read it is possible to use all 512 bits. But before doing so, the memory must be mapped. The map will define where to write and where to read the data from. The data need to be stored should be

listed. Then the data type and the length of every value stored must be defined. This will show which methods and how much space should be used. Then the memory cells must be distributed. Similar values should be stored in one chunk. This way their addresses could be determined dynamically. The map that I have created is attached in the appendixes.

4.5 Logic

The device starts in the configuration mode. The AP is configured and started, the server is created, and the REST endpoint has been added. Each loop the device will serve clients, but once in 1 000 loops it will check, if it needs to stay in configuration mode or move to user mode. To do this, it compares the time from the device start, acquired by dividing “millis” method response by 1 000, with the “configTimeLimit” variable that holds the time limit in seconds for the device to be configurable. Every time any endpoint is accessed it is set to a value of the current board time plus 180 seconds. This should be enough for the user to send a new request to the web server.

When the time will come, the device will reconfigure itself into the user mode. The AP will be shut down, the Wi-Fi module settings changed. The device in user mode will try to get values from the EEPROM. If there will be nulls, it will use the default values. The device will try to connect to the configured network and start serving clients as usual, but with new configurations.

4.6 Embedding configuration code into the main program

The configuration is done in the “ConfigMode” function. It takes the name of the mode as a variable and acts accordingly. I have not found a method to remove the REST endpoints, so they all will be disabled by making the functions to check the current mode and allow access to the endpoints related to the current mode only. So, it is possible to add a function to the user mode, which will make device return to the configuration mode.

The global variables will be created and set to the default values on the device start up. To get the new values the “SetConfigs“ function called by the “Config-Mode” function will get the data from the EEPROM and try to use it to set the configurations for the user mode. All the data validated, for any unset values the default one will remain.

The functions to build the page remain accessible all the time, and the same “Layout” function is used to build the main part of the page. However the function accept a parameter with a mode name. Depending on its value the “Layout” will add certain data into the page by calling other functions.

The behavior of the device will also change. In the function which fired once 1 000 loops, there is a check for the current mode, stored in a global variable. This way the behavior could be easily changed. The variable should only be changed in the “ConfigMode” function, since all the parts of the device behavior are inter-connected.

5 NEW GENERATION IOT COFFEE MACHINE

After creating a prototype and verifying that my assumptions are correct, it is time to describe how all this can be applied. I believe that it is possible to create a de-vice, using the principles developed during my thesis. Such a product could not only have all the above-mentioned functionality, but also a lot of new features. With more resources available and the ability to design a new coffee machine to fit the purpose, there are a lot of improvements available.

The device could become popular, since there is a constant need for making things simpler and allowing people to do less everyday routines. Therefore, the concept will be created so that it would require as little effort to use as possible. I believe that someday there will be no need to brew coffee by ourselves and a fresh cup will be always available, without much preparation. Besides cleaning the utensils, people should not pay attention to such a device more than once a month. With a new generation IoT coffee machine, it should only take one click to get the coffee for the whole office or family.

5.1 Concept of the device

The device, which will represent the new word in coffee brewing automatization, will be bigger than the simple coffee brewing machines. It is related to many new functions that should help with making the device require less maintenance and improving the coffee quality. It will also be more expensive, but since it could be the only coffee brewing machine in the office, some money could even be saved in comparison to buying three to five ordinary coffee machines.

The IoT coffee machine should have a coffee bean tank, grinder, changeable filters and connection to the water supply. This way, better coffee could be brewed, while less operations are required. It should be also easy to replenish the beans and change the filters. The coffee beans bunker should be on the top of the device, so that people can see that the coffee is brewed from freshly grinded beans. To make this easier the indicators in the user interface should be used. The filter has a life time, and people tend to forget to change them, so the device should keep track of the filter state instead.

Some sensors need to be used. They need to detect the presence of water in the pipe and the amount of the coffee beans. The filters could have some magnetic marks or chips to identify when the new one has been placed and how much coffee has been brewed since. This will help to provide better user experience and avoid bad coffee from exiting the device.

The main part of the device should be a half cylinder node with a jug. The 3D models I have created could be used as an example to design this part. I believe that the jug should be held in a proper position with the form of the device. There should be something to wrap around its edges in the places where the jug and node connect to prevent external light from interfering. For example, a rubber band could be used. It this case, it should be dirt resistant. The tests performed in the University of Applied Sciences show that the coffee will inevitably end up spilled on it. The sensors could be placed in one, or two rows. They are cheap and small after all. The sensors should be protected from the liquids. It should not

be hard to do with professional machinery. I also recommend placing LEDs on the other side of the sensors, so that there will be no need to rely on room lights.

The device should be available in the network, preferably via its domain name. I believe that it could be assigned to the IoT coffee machine on the local DNS server. It is also possible to make the device identify itself in the network in the same way printers do. Finally, smartphone applications should be able to find the device in the same network and connect to it.

5.2 Installation

The installation process will require the device to be placed on a spot. It will not be possible to easily move it after it has been installed. The water supply must be provided through the water pipe. Then, the filter for the water and coffee should be installed. Finally, the coffee beans could be loaded into the bunker.

With all the physical preparations done, it is possible to configure the device. A display on the device could be used for some basic configurations, but some sort of application should be able to provide a better interface and greater options. There are a lot of configurations available for the prototype and even more could be added to the actual product’s configurator.

5.3 Maintenance

The maintenance should be simple. To make sure that the device is running and produces good quality coffee it needs beans to be loaded from time to time. The filters should be changed according to the schedule. With water and coffee supply and filter indicators this should be simple. One more thing is to clear the jug at the beginning or at the end of the work day. All this could be accomplished by a janitor or by someone from the staff. In some distant future there may be a cleaning system in place. Two small pipes, one with water and/or soap and second to suck the liquid from the jug could do the job. They can extend into the jug from above and do the cleaning automatically. This will make the device even more automated.

The filters should have hatches to be accessed and replaced. Any person should be able to do this. The filters should have a unique form, so that they cannot be mistaken with inappropriate ones. The chips will help to ensure that there is a new valid filter and to keep track of its usage. This will also help to avoid health problems due to bad or dirty filters used.

To make the maintenance even easier, it is possible to add a link to the online store to the web interface of the device. Filters, coffee beans and maintenance could be bought from the official vendor this way. This will make the user experience better and ensure the long life of the device.

5.4 New functionality

Besides of all the prototype's functions, the factory produced device could have much more of them. The main one is the ability to brew coffee ordered by the user with a help of the web interface. With the sensors it will be possible to ensure that it is safe to brew a new portion. The requirements are: The presence of the jug, the water supply, the beans in the bunker, clean enough filter, and enough of available space in the jug.

Even if the jug is half full it is still possible to add some more coffee by brewing half of the usual portion. It should not be a problem with a full control over the water and coffee dust used in every brew. This also makes possible to provide the user with functionality to decide the amount of coffee and, moreover, its strength. However, due to approximation and error fixing process it might be needed to store both the most probable state and the maximum one, in case of an error. The decision to allow or forbid coffee brewing will be made more reliably, using the second value.

The filter's lifetime should depend on the estimated amount of liquid passed through it. Two half portions should only use the same resources of filtering medium and one full one. Thus, it should be properly counted and the information

about the way how the filter's resource calculated must be present to the customers. Otherwise, they could be trying to save on filters by forbidding themselves from using half portion functionality.

The coffee brewing process could be automated even further by adding a schedule. The device will try to brew coffee according to it. If it is not possible, then this brew will be skipped. But in case the possibility to fulfill the schedule appeared soon after the time set by the user, the IoT coffee machine should still create a new portion. This way the everyday morning coffee brewing process could start without anyone have to order it every time. It is important to provide a tool to create and modify schedule. The rules should include the time, the dates, the days of the week and the holidays, so there will be no coffee wasted during these days.

Besides of the schedule that regulate the repeating coffee brewing processes, it could be useful to add a single time order possibility. If the person knows that he wants to have the cup of a coffee in a few hours, he could order it beforehand. The device will try to ensure that by the time the user will come, there will be coffee for him.

If the device is in the public network, it might be useful to add security and authentication. The user account with individual password could be used. This way, there will be no unauthorized access. Moreover, the permissions could be set, so the users will have different functions available. For example, the schedule should only be set by the responsible person and not changed by someone unauthorized.

The LEDs installed near the jug should give stable light to the sensors. This will not only remove the need to calibrate the device every time the lights in the room has changed but will make the coffee machine look better. The colors could be configurable. The changes in light intensity should be tracked and applied to the current calibration.

The configuration tool should also have more functions. It would be useful to make the device search for the available networks. It could also be connected to it by choosing it and entering the network, the same way as a smartphone. This is the most native way.

5.5 New Interface

The prototype of the web interface could be used as a starting point for the actual product's web interface design. The indicators should start doing their work. They could show the state of the device and explain all the problems with a text near to them. The button which activates the coffee brewing process, should be in the most visible place. It should only be enabled when the requirement for brewing coffee met. If they are not, it should be disabled and the message with the reason should be shown.

The strength of the coffee and the required amount should be shown as analog indicators with a draggable pointer. If some part of the range is unavailable, it should be shown as disabled. This can happen in case the jug is half full, so only half of the portion could be possibly brewed.

The scheduling tool could be on an additional page. It should look as a scrollable calendar. When the user presses the day, the box with an interface for creating a rule pops up. It should give the user choose the time and how he wants to set the schedule. He could add the rule, so the device would brew coffee every such day of the month or every such day of the week. It should be also possible to add an everyday rule and the exceptions should be added in in the same constructor, to forbid the device from the automatic coffee brewing. The schedule should show the rules with colored dates. This way the user could instantly get the idea about the current schedule.

If the user wants to add only one deferred coffee brewing order, he should be able to do that form the main page. The controls to choose the time should be near the "brew coffee" button. If nothing has been chosen, the process should start immediately, and everything works as with usual instant order. If the time

has been set, the device will brew the coffee according to it. It should be impossible to set the time of the order prior to the current moment.

The control over LEDs could also be on a separate page. Different patterns, colors and intensity levels should be available. The only thing that most probably is not possible is blinking because it will disrupt the sensor's work. This could be solved with good self-calibration algorithm.

An android and IOS application could be developed to provide more options to access and use the device. An individual device should allow or block access from applications based on the configuration. The applications should have the same functionality but present it in a different way. This will be especially useful for the devices used at homes. The application should search for the devices in the local network or connect to the coffee machines nearby using Bluetooth.

6 CONCLUSION

During this thesis, I have faced a problem of finding the current coffee level in the jug and come up with an ultimate solution. I have studied many topics new to me and learned a lot in order to implement my idea. The proof of concept has been built and successfully used in the live environment. The program of the device works stably and produces good results. The feedback was good and the demand for such devices became obvious to me.

During my studies I have found a valid way to detect coffee in the jug using ambient light sensors. This could be used in a smart IoT coffee machine to get information about the amount of coffee available. The method has been tested and used in the proof of concept device.

The device I have developed could be built and used with any Moccamaster KBG741 coffee machine. Links to the program and files to print the casing are provided in the appendixes. The construction cost is low, with the highest expense being the 3D printing material. I believe that all the problems could be solved, and the device successfully used as a nice self-made extension to the

coffee machine. The casing could be changed to fit other models of devices, but the principle remains unchanged.

The web interface for the device has been created. It includes both the one for the user mode and the one for the configuration tool. They are developed in order to provide users with information in the most native way possible. Icons and images used to represent data along with text. The design is simple and mobile-friendly.

The configuration tool allows users to change the behavior of the device. This way it could act in a way, most suitable to the actual circumstances. The calibrations of the sensors, the signs of the coffee levels, and the limits of the counters could be changed. This is useful since there are always differences in ambient light levels of different rooms, in people’s view on the definition of jug fullness and user demands in terms of accuracy and responsiveness.

The date about the jug content is important to enable groups of people to work with a smart device simultaneously. Only by having this information, they could have a good idea about the current state of the device. This will allow them to make decisions based on the situation. It would be impossible if there were no data about the jug content, since chaotic commands could ruin the everyone’s experience. Since the data is available across the network, it is possible to access it from any device at home or in the office. Even if the smart coffee machine is far away.

It is certain that it is possible to build the device I describe in the section 5 of the thesis. I have built a proof of concept specifically to show it. The approximation of the coffee level could be enough to the user and to implement a lot of new functionality. Having the ability to control the coffee brewing process, this device can reliably supply an office or a house with fresh coffee.

The proof of concept device has some limitations. The main one is inability to brew coffee since it doesn’t have any connection to the coffee machine and cannot control it. This also limits the information provided to the user to the coffee level and few related things only. These problems can be solved by creating a new device from scratch or based on the existing coffee machines. Sensors and microcontroller should be designed to be inside the casing and work with all the coffee machine systems.

The existing proof of concept can be improved. The device connection to the coffee machine is not perfect and this makes it hard to position the jug properly. The casing is also bigger and heavier than it should be. Thus, it would be great to change the connection a bit and reduce the amount of material used, without losing in liquid resistance or durability. The program could also be improved. It is impossible now to change the image for the states, since there is not enough space in the EEPROM. The files are stored in the cloud, which is not preferable. I believe some problems could be solved by studying the ESP8266 board capabilities further. This would allow to make great improvements for the next version.

New functions of the new generation IoT coffee machine are not limited to just brewing coffee remotely. The coffee brewing can be scheduled or ordered in advance without any risk of overflowing the jug. Additional portions could be cooked based on the amount of coffee already present. In other words, the users and the device could make decisions based on the new data available and adapt their behavior accordingly.

The device won’t have much different casing from the usual ones. The controller is already installed in smart coffee machines and could be reprogrammed. The sensors cost is low. So, the device can be easily produced by any vendor.

The only remaining question is, if there are better ways of implementing this idea. Maybe the other type of sensors, or interfaces will do the work better. I believe that at least some parts of this thesis could be used in the creation of an IoT coffee machine.

REFERENCES

Arduino IDE. 2018. WWW document. https://www.arduino.cc/en/Main/Software [Accessed 24 February 2018].

eBay. 2018. WWW document. GY-49. https://www.ebay.com/itm/MAX44009-GY-49-Ambient-Light-Sensor-Module-for-Arduino-with-4P-Pin-Header-Module-/141849177830 [Accessed 24 February 2018].

I2C interface. 2018. WWW document. https://nodemcu.readthedocs.io/en/master/en/modules/i2c/ [Accessed 20 February 2018].

Flashing the Firmware. 2018. WWW document. Available at: https://nodemcu.readthedocs.io/en/master/en/flash/ [Accessed 20 February 2018].

JimmyU. 2015. Mobius Picatinny Rail Mount. WWW document. Available at: http://www.thingiverse.com/thing:1137790 [Accessed 20 February 2018].

Maxim Integrated. 2011. MAX44009 datasheet. WWW document. Available at https://datasheets.maximintegrated.com/en/ds/MAX44009.pdf [Accessed 20 February 2018].

Mykail. 2014. Picatinny Rail. WWW document. Available at: http://www.thingiverse.com/thing:449015 [Accessed 20 February 2018].

NodeMCU board documentation. 2018. WWW document. Available at: https://nodemcu.readthedocs.io/en/master/ [Accessed 20 February 2018].

NodeMCU firmware. 2018. WWW document. Available at https://github.com/nodemcu/nodemcu-firmware/releases [Accessed 25 February 2018].

NodeMCU Flasher. 2018. WWW document. Available at https://github.com/nodemcu/nodemcu-flasher [Accessed 25 February 2018].

Sandeep Mistry. 2018. GitHub repository. Available at: https://github.com/arduino-libraries/NTPClient [Accessed 20 February 2018].

Wi-Fi Module. 2018. WWW document. Available at: https://nodemcu.readthedocs.io/en/master/en/modules/wifi/ [Accessed 20 February 2018].

EEPROM MEMORY MAP

{} – Memory cell range;

[] – possible values;

() – Validation;

- SSID – String max 16 bytes {0-15};
- Password – String max 16 bytes {16-31};
- IP – 4 oct 1 byte each {32,33,34,35};
- Mask – 4 oct 1 byte each {36,37,38,39};
- Default gateway – 4 oct 1 byte each {40,41,42,43};
- Sensnum – 1 byte [0,4] {44};
- (Threshold + Range < Max, Threshold - Range > 0);
- Sensor #0:
    - SDA – byte [1,5] || [9,16] {45};
    - SCL – byte [1,5] || [9,16] {46};
    - Threshold – byte [1,255] {47};
    - Range – byte {48};
    - Max – byte {49};
- Sensor #1:
    - SDA – byte [0,5] || [9,16] {50};
    - SCL – byte [0,5] || [9,16] {51};
    - Threshold – byte {52};
    - Range – byte {53};
    - Max – byte {54};
- Sensor #2:
    - SDA – byte [0,5] || [9,16] {55};
    - SCL – byte [0,5] || [9,16] {56};
    - Threshold – byte {57};
    - Range – byte {58};
    - Max – byte {59};
- Sensor #3:
    - SDA – byte [0,5] || [9,16] {60};
    - SCL – byte [0,5] || [9,16] {61};
    - Threshold – byte {62};
    - Range – byte {63};
    - Max – byte {64};
- States
    - 0000
        - Sign – String max 24 bytes {65-88};
        - Color – String max 8 bytes {89-96};
    - 0001
        - Sign – String max 24 bytes {97-120};

        - Color – String max 8 bytes {121-128};

    - 0002
        - Sign – String max 24 bytes {129-152};
        - Color – String max 8 bytes {153-160};
    - 0012
        - Sign – String max 24 bytes {161-184};
        - Color – String max 8 bytes {185-192};
    - 0022
        - Sign – String max 24 bytes {193-216};
        - Color – String max 8 bytes {217-224};
    - 0122
        - Sign – String max 24 bytes {225-248};
        - Color – String max 8 bytes {249-256};
    - 0222
        - Sign – String max 24 bytes {257-280};
        - Color – String max 8 bytes {281-288};
    - 1222
        - Sign – String max 24 bytes {289-312};
        - Color – String max 8 bytes {313-320};
    - 2222
        - Sign – String max 24 bytes {321-344};
        - Color – String max 8 bytes {345-352};
    - 9999
        - Sign – String max 24 bytes {351-376};
        - Color – String max 8 bytes {377-384};
- AP password – String 16 bytes max {385-401};
- Static address – byte [0,1] {402};
- Change state interval – byte {403};
- Loop interval – short {404-405};
- Coffee rises interval – byte [0-3] {406};
- Reload time – short {407-408};
- Debug mode – byte [0,1] {409};
- Time zone raw – byte [0,14||101,112] {410};
- Daylight saving – byte [0,1] {411};

FILES

Final program

IoTCoffeeMakerv3 -https://github.com/RazgrizX/NewGenerationIoTCoffeeMachine/blob/master/IoTCoffeeMakerv3.ino

Casing Parts

Device model –https://github.com/RazgrizX/NewGenerationIoTCoffeeMachine/blob/master/DeviceModelCasingFinal.skp

Printables:

Main part –https://github.com/RazgrizX/NewGenerationIoTCoffeeMachine/blob/master/MainPart.stl

Top part –https://github.com/RazgrizX/NewGenerationIoTCoffeeMachine/blob/master/TopPart.stl

Side part –https://github.com/RazgrizX/NewGenerationIoTCoffeeMachine/blob/master/SidePanel.stl

Rail –https://github.com/RazgrizX/NewGenerationIoTCoffeeMachine/blob/master/PikatinnyRail.stl

Mount –https://github.com/RazgrizX/NewGenerationIoTCoffeeMachine/blob/master/PikatinnyConnector2.stl

Coffee Now!

Coffee brewed at unknown time

BREW MORE COFFEE

- Water
- Coffee Beans *(No beans in the box)*
- Filter *(2 days until change)*

⚠Warning debug enabled! Sensor outputs:
Sensor #5 - Ambient Light Level (Lux) = 21.42
Sensor #5 - Ambient Light Level (Lux) = 24.48
Sensor #5 - Ambient Light Level (Lux) = 30.60
Sensor #5 - Ambient Light Level (Lux) = 30.60

Coffee Now!

SSID: Aleksandr Password: 02061951

Apply

Static IP Dynamic IP

IP address:	192	168	1	199
Mask:	255	255	255	0
Gateway:	192	168	1	1

Apply

Sensor #0 - SDA:	12	SCL:	5	Threshold:	2	Range:	1	Max:	20
Sensor #1 - SDA:	4	SCL:	5	Threshold:	3	Range:	1	Max:	25
Sensor #2 - SDA:	14	SCL:	5	Threshold:	4	Range:	1	Max:	35
Sensor #3 - SDA:	13	SCL:	5	Threshold:	4	Range:	2	Max:	40

Amount: 4 ▼ Apply

State 0000 -	 0%-20%	color:	black ▼	State 0122 -	 60%	color:	black ▼
State 0001 -	 20%	color:	black ▼	State 0222 -	 60%-80	color:	black ▼
State 0002 -	 20%-40	color:	black ▼	State 1222 -	 80%	color:	black ▼
State 0012 -	 40%	color:	black ▼	State 2222 -	80%-100%	color:	white ▼
State 0022 -	 40%-60	color:	black ▼	State 9999 -	Jug missing	color:	white ▼

Apply

Configuration AP password	12345678		
Change state interval	5	Check interval	1100
Coffee rises interval	2	Reload Time	600

IP settings allows to make the device use apropiate network address while in user mode, so you can easily access it.

Sensor settings allows to calibrate sensors to detect coffee better. You can ajust parameters, to suit light level of the room.
Theshold - amount of light passes to the sensor when the coffee on the sensor's level.
Range - the range of light levels which considered to mean that the coffee is on the sensor's level.
Max - the light level, above which the range of lightlevels only possible if jug is not present lies.

If the light level is above Threshold + Range the result is 0 (no coffee), if less then Threshold - Range is 2 (coffee is above).

State settings allows to set the sign that appears on top of the coffee jug picture and a color of this sign