Universitatea Politehnica Timisoara
Calculatoare si Tehnologia Informatiei, 2023
Baciu Bogdan

SunTracker System: Optimized Energy Harnessing Through Sun Movement Tracking

Candidat: Bogdan, Baciu

Coordonator științific: Prof.dr.habil.ing. Mihai V. Micea

Sesiunea: Iunie 2023

REZUMAT

Acest proiect are în vedere dezvoltarea unui sistem automat de urmărire solară care prezintă datele colectate la un interval de 1 secundă și generează statistici pe baza datelor înregistrate pe parcusul unei zile.

Cererea în creștere a energiei regenerabile are nevoie de tehnologii mai avansate pentru îmbunatatirea colectării energiei solare în vederea generării electricitații. Lucrarea va ilustra procesul de dezvoltare unui sistem de urmărire a soarelui construit pe platforma unui Raspberry Pi 2, modelul B de versiunea 1.1 care ruleaza o distribuție bazată pe Debian OS numită Raspbian.

Sistemul folosește o metodă de achiziție a datelor la un interval de 1 secundă pentru diferiti parametrii: intensitatea luminii, curentul generat, umiditate si temperatură. Monitorizarea continuă a intensității luminii și a curentului generat ajuta ca sistemul să se ajusteze automat la condițiile din jur prin poziționarea perpendiculară cu raza incidentă de lumină a celulei fotovoltaice pentru o productie mai de lungă durată.

Capitolele următoare acoperă in larg urmatoarele topice: contextual de dezvoltare, motivația, noțiunile teoretice folosite, arhitectura și implementarea sistemului. Conținutul va revizui soluția propusă și va menționa eventuale imbunătățiri ce pot fi aduse sistemului.

În continuare, proiectul va deservi ca un aparat de demonstrație dar este capabil de imbunătățiri pentru a putea implementa un sistem de urmărire solară real.

ABSTRACT

This project focuses on development of an automated solar tracking device that displays live data and enables reports on daily gathered data. The increasing demand on renewable energy calls for advanced technology that is able to enhance the solar power harvesting. This paper will illustrate the process of development of a sun tracking system built upon a Raspberry Pi model B v1.1 platform that runs a Debian-based operating system name Raspbian.

The system employs a data acquisition mechanism with a 1 second delay for data such as: light intensity, electrical power generated, humidity and temperature. By continuously watching over the light intensity and electrical power generated, the system automatically adjusts to environmental conditions by positioning to an orthogonal incident angle for keeping the power production higher.

The chapters enumerated broadly go through the following topics: context of development, motivation, theoretical notions used, architecture and implementation of the system. The content will also revise the proposed solutions and will also provide additional improvements that can be achieved.

The project will furtherly contribute as a demonstration device but is able to actively evolve into a better and better device for solar tracking development field.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF CODE SEQUENCES

1 INTRODUCTION

1.1 MOTIVATION

The software development was presents on two fundamental layers and thus, provided two different applications that run in parralel.

The motivation behind this project lies in my pursuit of creating a project that uses and requires a development on multiple layers such as mechanical, electrical and software engineering, that is able to illustrate and to react to the reality and environment around it.

1.2 DOMAIN OF ACTIVITY

The domains that are present within this project are the following:

- Electrical engineering – basic notions used for building the hardware
- Software engineering
    - Low level programming performed in order to interact with the integrated circuits and hardware modules needed
    - High level programming performed in order to build the website and creating specific classes designated to specific hardware modules

If there would be required to place the project in a more specific field of interest it would be the one of Internet of Things.

1.3 CONTEXT OF DEVELOPMENT

The project was a firstly presented on the proposed project list of the DSP Labs when I have got in touch with the coordinating professor Mihai Micea.

The scope of the project would be to to have a robust demonstration platform to be used in different events organized within the faculty.

The project was solely deveoped by myself under great guidance from the coordinator professor as well as having a small but great piece of advice and help from the following professors, Stângaciu Valentin and Răzvan Cioargă in regards to commuting the hardware from a breadboard to a semi-printed circuit board and facilitation of internet connection through a WLAN adapter, respectively.

1.4 GENERAL SPECIFICATIONS

The project consists of creating a means of creating a mechanism of following a source of light while monitoring are displaying in real time the state of the aquired information.

The overall functionality is splitted in two parts, the lower-leveled application assigned to interact with the hardware and also using the SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit) protocols. For the development of this part, C++ was the main language used alongside necesarry Python files reserved for interaction with the servomotors and the temperature and humidity sensor. This part is where the information is captured and processed.

The higher-leved application is the web application created using the Flask web development framework using Python and also HTML, CSS and plain Javascript. This part of the project has the responsability of displaying the live status of the system and presenting specific daily insights.

As a system environment tools, Makefile was used for building the lower layer application and for the higher layered application the *pip* package manager was used for tracking dependencies, installing necessary libraries and maintaing a virtual environment [1].

1.5 STRUCTURE OF THE DOCUMENT

The document is structured in six chapters. Firstly, the project is shortly presented followed by a theoretical background on what technologies were used and what kind of similar systems already exist. Also, they served as examples and means of inspirations and ideas for the deveopment of the system. Furthermore, the general architecture of the system is presented in depth alongside the decision made and reasons of them. This chapter will greatly help in understanding of the segment followed by it which deals with the implementation. The penultimate chapter will analyze the final state of the product through the lenses of the performance and the experimental results of it. In the end, my personal thoughts and conclusions are going to be shared on the entire development process of the project.

2 THEORETICAL BACKGROUND

2.1 SOLAR TRACKING DEVICES INSIGHTS

Having the global context of drastically increasing the green energy is production and to radically reduce the carbon emissions, the photovoltaic systems are increasingly searched for through various programs and companies that provide sollutions of such kind. As the demand for renewable energy increases, so does the technology advance to improve the efficiency of the existing mechanisms. The enhancements brought to reality in matter of solar panel construction and setup have been a thing from the moment they have been invented. As of today, there is a vast majority of programs that offer subsidies for implementing photovoltaic systems.

There are several sources of acquiring energy from nature, such as:

- Solar energy is converted into electricity through photovoltaic panels
- Wind energy is converted into electricity through the use of wind turbines
- Hydro energy of use in hydroelectric power plants and is harvested by drive turbines

It is common sense that some places are better suited for some kind of green energy producing system than others.

The photovoltaic systems are generally placed on the roofs of the houses or there is a dedicated field filled of solar panels but generally they are in a fixed state and have their movement restricted from their setup or by sourroundings. As action, there were developed different kinds of solar pannels have been developed according to the needs.

At this point in time there are three generations. The first generation solar cells are made out of single consistent silicon monolithic crystal and are rather expensive due to their efficiency being the highest. The second generation solar cells are manufactured from thin film, thus being more affordable with a lower efficiency while the third generation is implementing natural materials and can utilize regular printing press devices, plastics and organic dyes [2].

In the rapid development of the past years, there were developed large number of strategies of making photovoltaic cells more efficient by acting on the phisical event, which implies that only the orthogonal photons of the light wave interact with the atoms from the semiconductor materials. Generally, sun tracking system are using single or dual axis but there are other strategies such as: passive tracking, tilt-only tracking or chronological tracking. Chronological tracking is based on the fact that the sun moves 15° per hour and one example of passive tracking is implemented in [3] where chemical and physical knowledge are highly required.

For solar tracking systems there is generally used a PID (Proportional-Integral-Derivative) controller because it takes into consideration the error between the measured output and the desired output.

Solar panels are set up in the direction of the true south if the location is in the Northern Hemisphere or true north if the location is in the Southern Hemisphere. In the following image is presented the rotation of the sun in the Nothern Hemisphere.

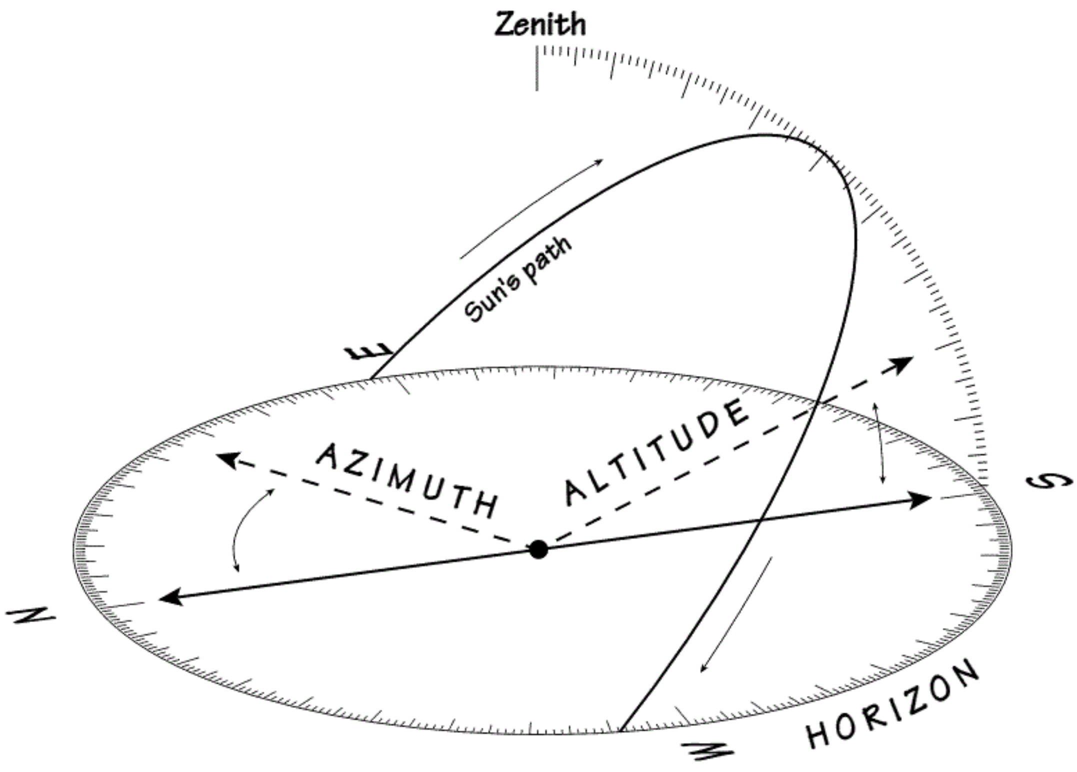


Figure 1. Sun Movement [4]

2.2 SIMILAR PROJECTS

Similar projects can be found here [5]. It is mandatory to mention the project found here because it was the first and and a great source of inspiration [6] from the YouTube video.

From an academical point of view, there a lot of papers that implement dual axis solar tracking systems in different ways and making use of different actuators or sensors. For example, this paper [2] used a Pyranometer which is a device used for measuring solar iradiance of a planar surface, others [ddd] use Light Dependant Resistors as well.

3 SYSTEM DESIGN AND ARCHITECTURE

In order to have a robust system design that is able to withstand a repetitive load and perform prolongued tasks, there was highly need of a carefully designed structure where the Raspberry Pi platform and the display screen could be attached to and built upon later. Although I have reserved some time for this specific task so that everything would fit perfectly, there are still some defects that can be seen. For example, the fitting of the screen did not match the calculations and I had to manually cut holes for the screws.

The architecture of the system can be generally splitted into two parts, hardware and software. The physical side of the project will be described in the hardware architecture sub-chapter whilst the more abstract layer of interacting with the physical world and the interfacing with the information received will be presented in the software sub-chapter as two sepparate applications.

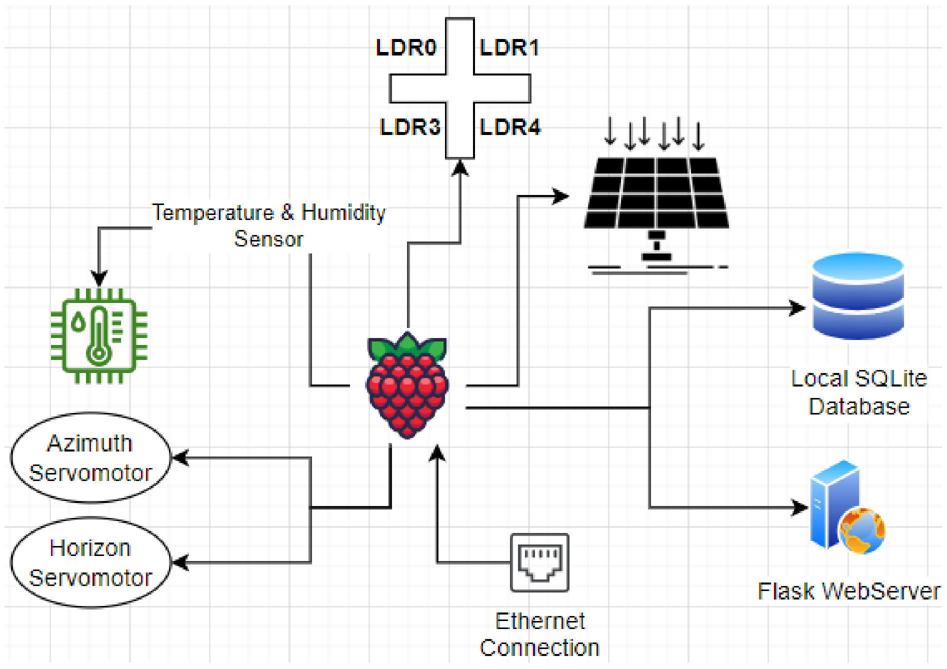


Figure 2. System Architecture

3.1 TOOLS USED

During the development process there were used several tools in order to make a prototype for the system in order to test the desired movement of the solar platform and some parts unfortunatelly broke down.

This is the list of the mechanicals tools used from development phase to get from prototyping to the actual final device:

- phillips screwdriver
- flathead screwdriver

- flathead screws
- phillips screws
- wood sticks
- 2 x lego cogwheels
- hot glue gun

3.2 PHYSICAL DESIGN

For desiging the structure of the system I searched for and ordered a 3D printed design onto which I could mount the Raspberry Pi along with the display. Inside the package is placed the main hardware circuit and on top of it is placed the turret [8] and a break-out cap connectors for separating the moving from non-moving parts.

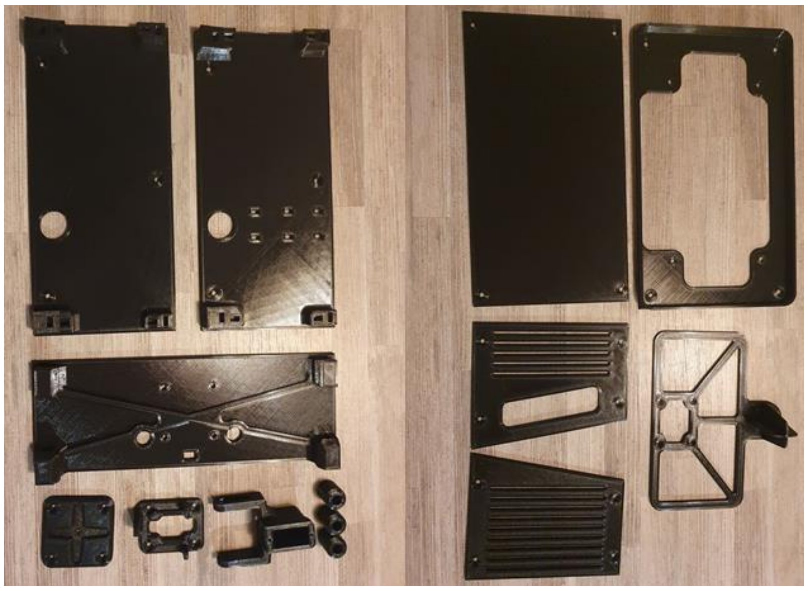


Figure 3. The 3D printed parts

The assembling was done with the help of the tools presented above.

3.3 HARDWARE ARCHITECTURE

The total amount of hardware components are found in the following list:

- 2 x MG90S 180g Servomotors
- 4 x light dependant resistors 5528
- solar panel 5V, 1.1W, 220mAh
- temperature and humidity sensor DHT22

- analog-to-digital converter MC3008
- INA219 current sensor breakout board
- breadboard
- jumper wires
- insulation tape
- soldering station
- fusable metal
- wires
- different types of pliers
- cutter
- source meter

The most important hardware component used is the SD Card which is formatted with Raspbian OS. Another important hardware component is the WLAN adapter. The reason for it is that the wlan adapter provides the communication medium for external devices to connect to the system and interact with it, whether it means the system operates as an access point or as a client. The WLAN adapter scope is to act as an access point although during the development phase the main purpose of it was to enable a wireless ethernet access. The general hardware architecture is represented in the following figure.

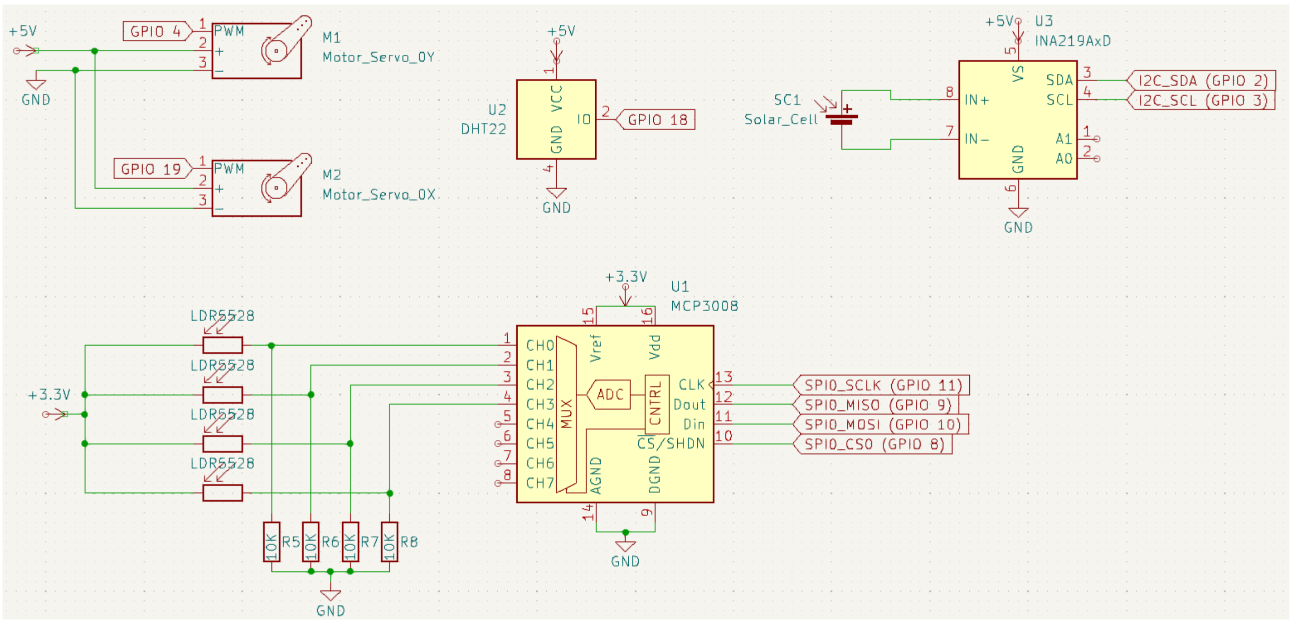


Figure 4. General hardware architecture

3.4 SOFTWARE ARCHITECTURE

The software development was presents on two fundamental layers and thus, provided two different applications that run in parralel.

The first application was developed with a clear scope in mind and that is to be as closer as it could be in relation with the hardware and run as smooth as possible. For such reason, C++ was chosen as the main language but due to several reasons in regards to the platform and the operating system, Python was used in particular cases with regards to hardware interaction. This particular case is presented as a pyton bridge. This application can easily run on reconfigured hardware because all of those properties are set and used from the Utils.hpp file.

The second layer is a web application that serves the purpose of sharing the captured, interpreted and then adapted to set of data by the first application. It creates a n easily readable interface that offers two main functionalities: to acknowledge the live state of the system and to have an insight about the past.

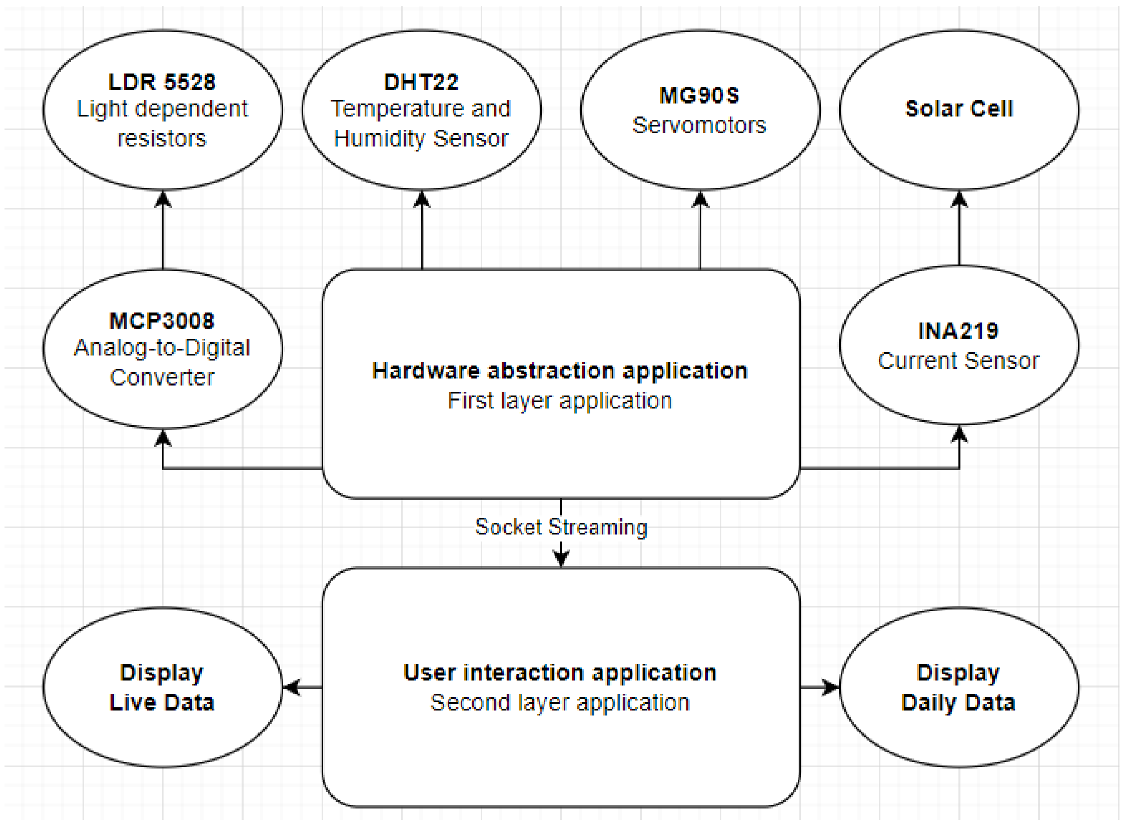


Figure 5. General Software Architecture

The Figure 5 is expressing the general software architecture of the application for creating an overall picture about the project. It gracefully depicts the main features of the entire system within a software perspective.

In order to have a basic understanding of how the project works, Figure 6 depicts an interaction between the user and the system which is represented by those two core applications. It illustrates the most specific interactions that are triggered whenever the system receives input from the user. It also represents a good metric of knowing how the user input is decomposed through the two layers of the project.

Those two applications present a push-pull design as the C++ application pushes data at a known rate while the other one pulls it.

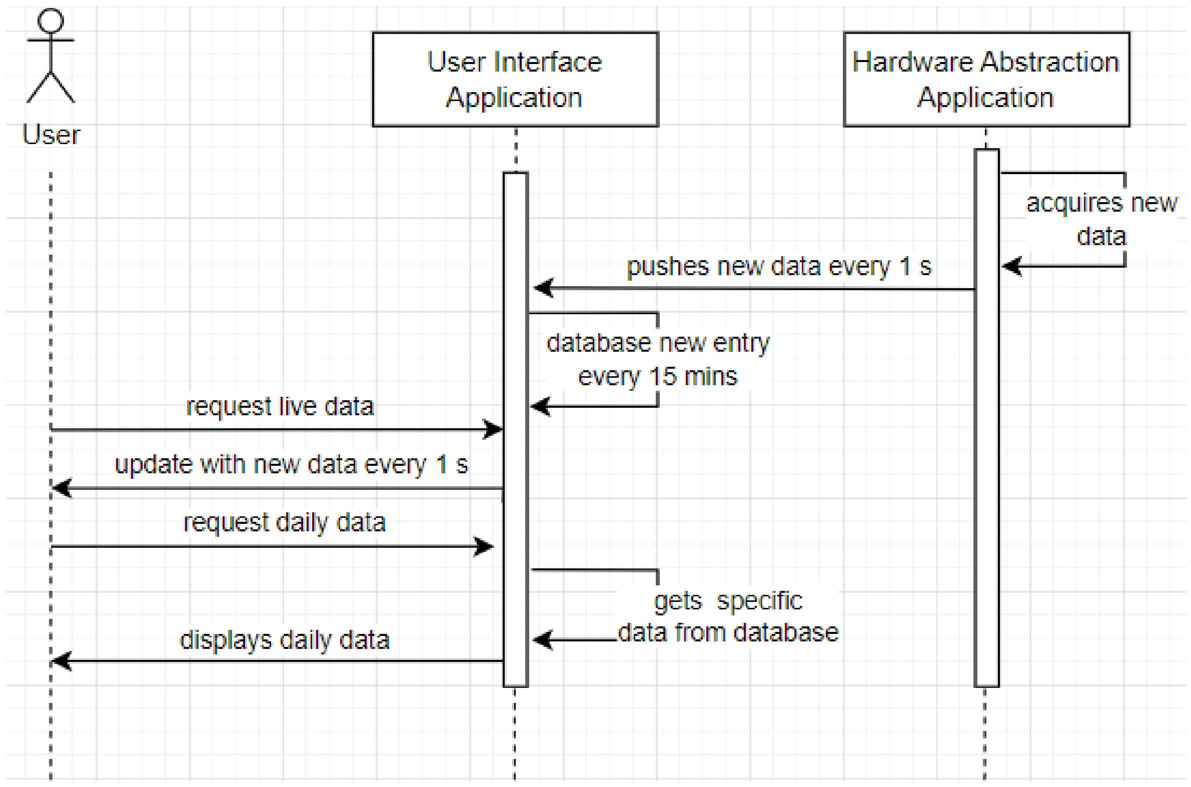

User
User Interface Application
Hardware Abstraction Application
acquires new data
pushes new data every 1 s
database new entry every 15 mins
request live data
update with new data every 1 s
request daily data
gets specific data from database
displays daily data

Figure 6. User System Interaction Sequence

Mention: first layer application can be referenced also as the base application or the main application whereas the the second layer application can be found as the web application or the webserver.

3.4.1 HARDWARE ABSTRACTION APPLICATION

The base application is splitted in the following modules:

- the turret control module
- the solar module
- the ambiental module
- the python bridge module
- the streaming module

Each of those modules are closely tied to the other ones yet independent. The first three modules are the basis of the system and are the ones that produce the later used data.

The application design is focused on the functionalities provided by the hardware system and was modelled using an OOP (Object Oriented Programming) approach. It consists of classes that concurently provide their specific data to an UDS (Unix Domain Socket) [9].

The turret control module is enacted through the TurretControl class. It is reponsible of handling the data received from the LDRs (Light Dependent Resistors) through the MCP3008 ADC (Analog-To-Digital Converter) and responds to this data with the help of the Servomotors. As such, the TurretControl class acts upon other two, Mcp3008 and Servomotor classes that independently run their job.

The ambiental module` solely purpose is to aquire the temperature and humidity from the environment. It is represented by the Dht22 class.

The particularity of those 2 modules stands in the python bridge they use in order to carry out their full set of responsabilities. After encountering several problems I took the decision to split the hardware interaction to a python script. Thus, the Servomotor class and the Dht22 class are delegating their hardware responsabilities to the python bridge through a UDS. In this case, the SocketCommunicator class launches into execution the python script that:

1. is a server by creating a UDS socket and handling the afferent connections
2. takes decisions based on the command code read from the UDS
3. delegates the job to the Servomotor and Dht22 python classes

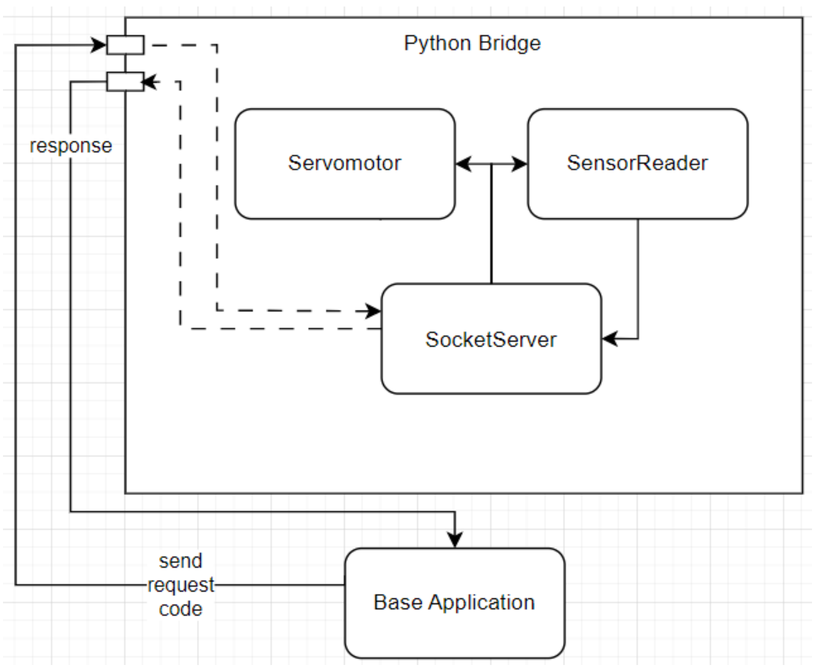


Figure 7. Main application Using Python Bridge

The solar module is represented through a single class SolarCell that bears the responsability of configuring the INA219 current sensor registers to read the most precise data.

At last, the streaming module is implemented into a single class that creates streams of data every second and are transmitted to the web application and such, the main application takes up the role of the server and the web application the one of the client.

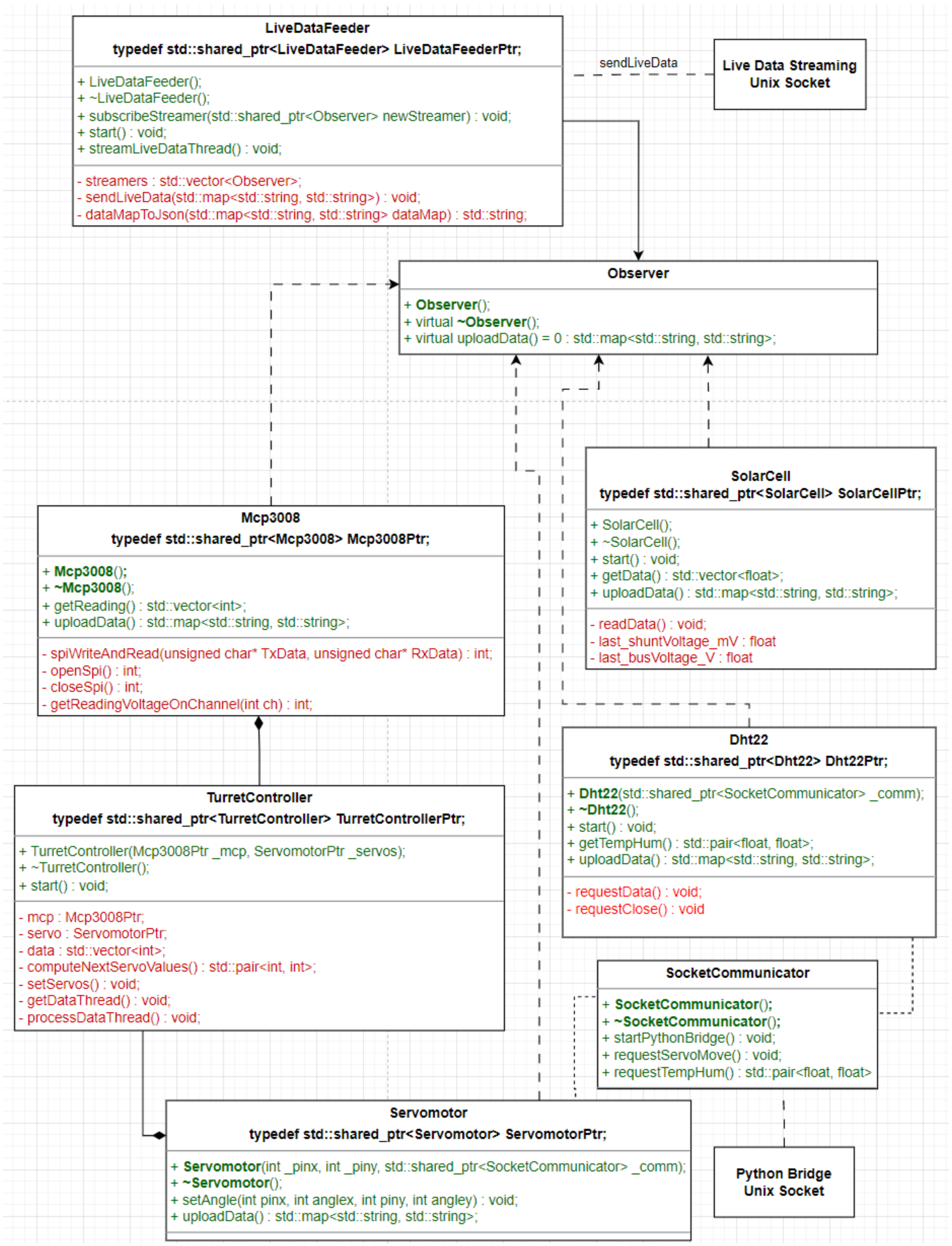

LiveDataFeeder
typedef std::shared_ptr<LiveDataFeeder> LiveDataFeederPtr;
+ LiveDataFeeder();
+ ~LiveDataFeeder();
+ subscribeStreamer(std::shared_ptr<Observer> newStreamer) : void;
+ start() : void;
+ streamLiveDataThread() : void;
- streamers : std::vector<Observer>;
- sendLiveData(std::map<std::string, std::string>) : void;
- dataMapToJson(std::map<std::string, std::string> dataMap) : std::string;
sendLiveData
Live Data Streaming Unix Socket
Observer
+ Observer();
+ virtual ~Observer();
+ virtual uploadData() = 0 : std::map<std::string, std::string>;
SolarCell
typedef std::shared_ptr<SolarCell> SolarCellPtr;
+ SolarCell();
+ ~SolarCell();
+ start() : void;
+ getData() : std::vector<float>;
+ uploadData() : std::map<std::string, std::string>;
- readData() : void;
- last_shuntVoltage_mV : float
- last_busVoltage_V : float
Mcp3008
typedef std::shared_ptr<Mcp3008> Mcp3008Ptr;
+ Mcp3008();
+ ~Mcp3008();
+ getReading() : std::vector<int>;
+ uploadData() : std::map<std::string, std::string>;
- spiWriteAndRead(unsigned char* TxData, unsigned char* RxData) : int;
- openSpi() : int;
- closeSpi() : int;
- getReadingVoltageOnChannel(int ch) : int;
Dht22
typedef std::shared_ptr<Dht22> Dht22Ptr;
+ Dht22(std::shared_ptr<SocketCommunicator> _comm);
+ ~Dht22();
+ start() : void;
+ getTempHum() : std::pair<float, float>;
+ uploadData() : std::map<std::string, std::string>;
- requestData() : void;
- requestClose() : void
TurretController
typedef std::shared_ptr<TurretController> TurretControllerPtr;
+ TurretController(Mcp3008Ptr _mcp, ServomotorPtr _servos);
+ ~TurretController();
+ start() : void;
- mcp : Mcp3008Ptr;
- servo : ServomotorPtr;
- data : std::vector<int>;
- computeNextServoValues() : std::pair<int, int>;
- setServos() : void;
- getDataThread() : void;
- processDataThread() : void;
SocketCommunicator
+ SocketCommunicator();
+ ~SocketCommunicator();
+ startPythonBridge() : void;
+ requestServoMove() : void;
+ requestTempHum() : std::pair<float, float>
Servomotor
typedef std::shared_ptr<Servomotor> ServomotorPtr;
+ Servomotor(int _pinx, int _piny, std::shared_ptr<SocketCommunicator> _comm);
+ ~Servomotor();
+ setAngle(int pinx, int anglex, int piny, int angley) : void;
+ uploadData() : std::map<std::string, std::string>;
Python Bridge Unix Socket

Figure 8. UML Class Diagram of Hardware Abstraction Application

As seen in Figure 8, the LiveDataFeeder makes of classes that implement the Observer interface and can be seen that all of them are implemented using a thread. It can also be seen clearly what classes interact with the python bridge through the SocketCommunicator class and which of them are part of the main component of the application which is the TurretController.

Regarless of the actual implementation of the Observer, it provides new individual data as a sample per second. The LiveDataFeeder is then requesting this type of adta from each one of those subscribed observers and pushes it into the UDS. This flow is illustrated in the figure below.

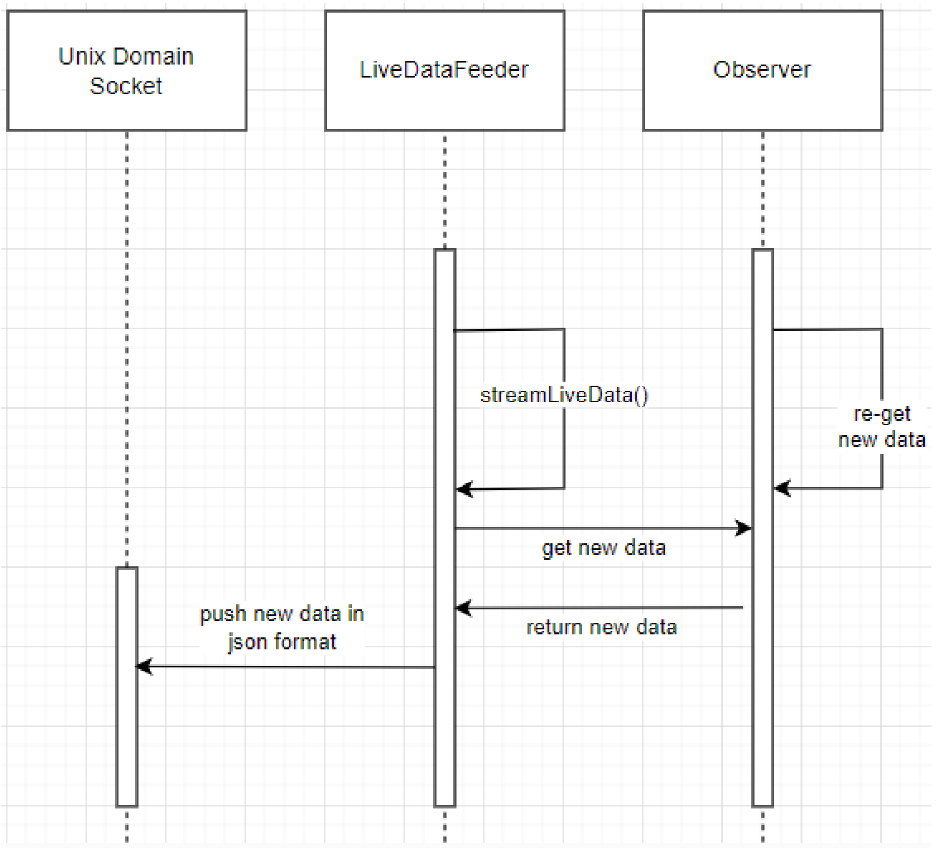


Figure 9. Flow of Data in Hardware Abstraction Application

3.4.2. USER INTERACTION APPLICATION

This second layer application is built upon a web framework called Flask [10] using Python, Javascript, HTML and CSS and acts like a client in regards to the socket streaming. This means it receives a data stream at a fix period of time of 1 second. The received data is then handled by a thread which ensures its validity, prepares it for publishing into the database and at last, gets the stream ready for the live display. The accessible endpoints are only the following ones:

- https://localhost/5000 (default)
- http://localhost/5000/statistics .

The default endpoint is where the live data can be seen and the /statistics endpoint is accesible for viewing selectable daily data.

The overall architecture and flow of data of this application is summarized in the figure below. Contrary to the the application presented before, the data flow of this one presents a pulling technique from the UDS through the arrow pointed to it.

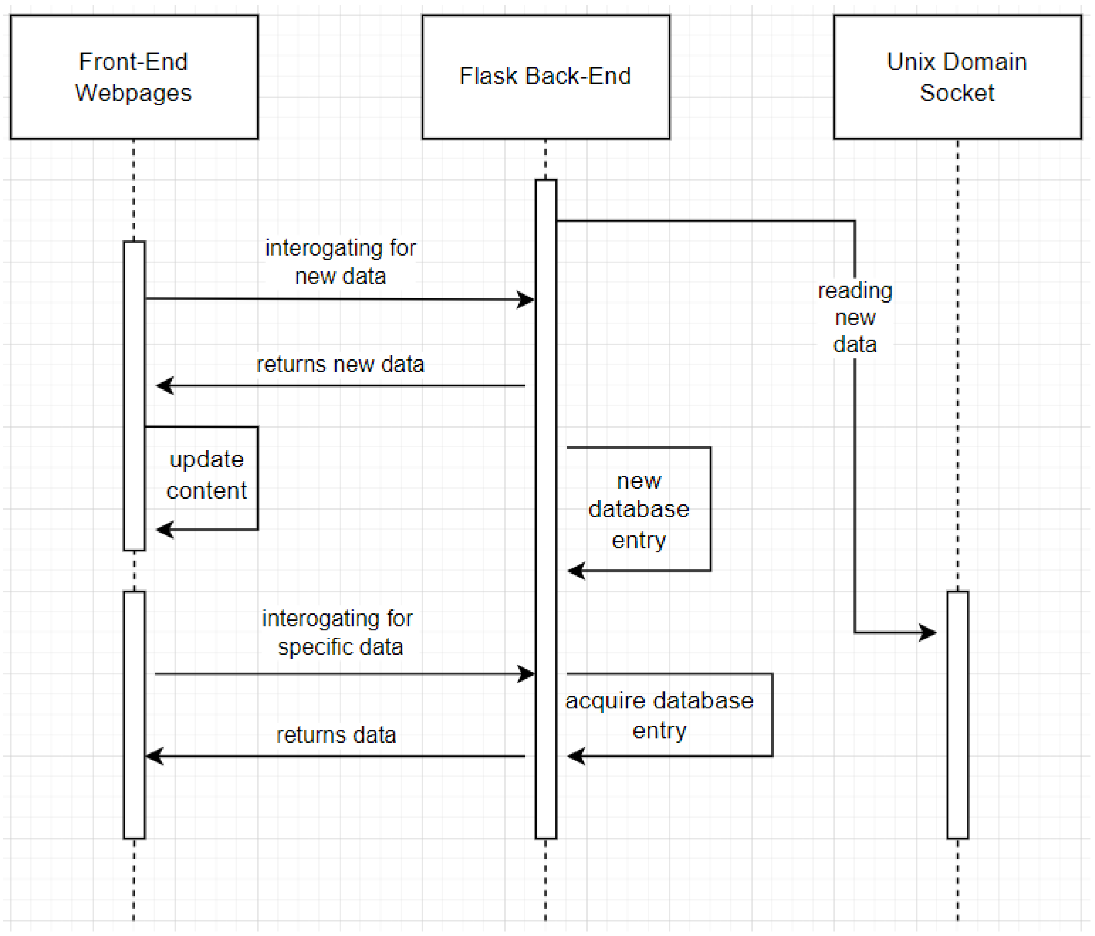


Figure 10. Flow of Data in User Interaction Application

4 HARDWARE ABSTRACTION APPLICATION IMPLEMENTATION

The implementation of the first layer application was designed to emulate the way of how the designated hardware modules work. Every module is implemented as a thread [11] that is either reactive or proactive, depending on its purpose while the data interaction is marked thread-safe through the use of mutex [12]. The implementation was focused on the Linux kernel and usage of standard C++ libraries. The only exception is centered on the python bridge that uses two external libraries: Rpi GPIO control library [13] and Adafruit DHT22 sensor library [14].

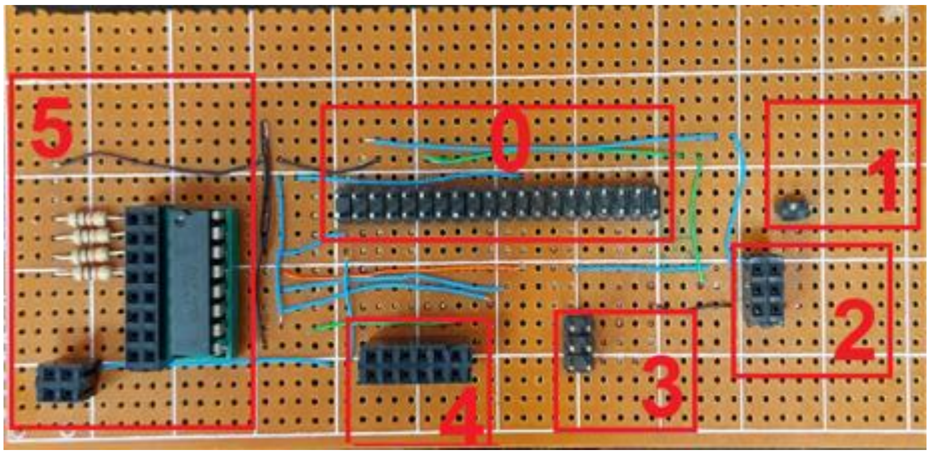


Figure 11. Custom Circuit Board

Every subchapter will include firstly describe its own hardware implementation but first, let`s introduce the whole of it through the Figure 11 above. First, there was a simple breadboard implementation but I managed to make a dedicated custom circuit board that is better suited for the needs of the project. The circuit board has six dedicated regions that are used for:

0. translating the Raspberry Pi pins onto the board
1. powering the display at 5V
2. powering the temperature and humidity sensor
3. powering the servomotors
4. powering the current sensor
5. powering the ADC and the LDRs

Out of these regions, some of them are dedicated to specific modules explained in the following subchapters as follows: **region 2** is implementing the ambient module, **regions 3 and 5** are the embodiment of the turret controll module and **region 4** is implementing the power module. The board uses pins and connectors to interface with the ICs and breakout boards such that it provides easier maintainability but also, soldering the ICs directly can cause damage through overheating.

4.1 TURRET CONTROL MODULE

The turret control module is composed of three classes and most of the hardware thus it becomes the main feature of the project along with the power modules.

4.1.1 HARDWARE IMPLEMENTATION

The hardware implementation of the module is composed of the following parts:

- x LDRs (Light Dependent Resistors)
- MCP3008 Analog-To-Digital Converter
- 2 x MG90S Servomotors
- 4 x 10 Kohm resistors

The LDRs have an internal resistance of 10 – 20 Kohms at a maximum of 100 lumens or lux level of 10. In complete darkness the internal resistance is around 85 Kohms. It is important to mention that the LDR is focused around the 540nm wavelength value and such, the optimal wave color is green.

The MCP3008 ADC converter is an IC (integrated circuit) which has a 10-bit resolution and it has 8 input channels. It uses the SPI interface to communicate with a controller [15]. The IC was mandatory because the Raspberry Pi GPIOs do not have a built-in analog to digital converter.

The servomotors are powered by a 5V power source and has a physical range of 180°. The control PWM (Pulse Width Modulation) signal has a frequency of 50Hz and a period of 20ms with a 1 - 2ms pulse width corresponding for -90° and 90° respectively with 1.5ms corresponding to 0°. This servomotor was chosen for the metallic cogwheels and shafts it benefits of thus giving it more time of life.

The 4 later resistors were used as pull-down resistors. The reason behind this decision comes from the following benefits a pull-down resistor comes with:

- It defines a voltage level state whenever the LDR is exposed to little to no light. In this case, the value read by the ADC will be lower or equal to 0.
- It ensures a stability level and noise reduction because it provides a reference to the ground of the circuit.
- It protects the MCP3008 input channel from the potential unwanted current flow spikes.

4.1.2 SOFTWARE IMPLEMENTATION

As previously mentioned, the modules consist of three classes, Mcp3008 class, Servomotor class and the TurretControl class. The later one is composed of two parallel running threads that share resources.

```
void TurretController::getDataThread() {

    while(running) {
        if(initProcessThread) {
              processThread = thread(&TurretController::processDataThread,
   this);
             initProcessThread = false;
        }

        unique_lock<mutex> lock(m);
         data = mcp->getReading();
         dataReady = true;
         lock.unlock();
         this_thread::sleep_for(chrono::milliseconds(5));
         cv.notify_one();
         this_thread::sleep_for(chrono::seconds(1));
    }
}

void TurretController::processDataThread() {
    while(running) {
        unique_lock<mutex> lock(m);
        cv.wait(lock, [this]{ return dataReady;});
        setServos();
        dataReady = false;
    }
}
```

Code 1. TurretController Threads

At first, the getDataThread is spawned which firstly starts the second running thread that manages the newly received data from the light dependent resistors. The procedure of accessing the data from the ADC is similar to this one [16]. The system reacts almost immediately at changes but expects at least 1s of buffer time when nothing considerable should happen. Also, having the module react as fast as possible to any change means oversaturation of the servomotors which are not able of being controlled at a fast pace.

The servomotors are controlled by setting the incrementing or decrementing the current angle by 10° if difference between quadrants is larger than 100. This value was chosen as offset knowing that the values read from ADC channel that corresponds to a LDR is in the range of o to 1023.

4.2 POWER MODULE

4.2.1 HARDWARE IMPLEMENTATION

For this implementation there were chosen two components:

- Solar cell
- INA219 Current Sensor

The solar cell produces can produce a maximum voltage of 5V, 1.1W and 220mAh [17].

One important piece of information to remember is that those are the peak values and can only be sustained for as long as the optimal conditions are met:

- Light is focused and has perpendicular angle
- Temperature of the cell is lower. Excessive heat builds up lower performance on the physical layer.
- The cell has a clean surface.

When building the module, using a custom shunt resistor along with the ADC was considered when the implementation started and implied buying a shunt resistor or building one by making a parallel resistor connection. There were little to no benefits of this approach thus buying a dedicated sensor has been taken into consideration.

The INA219 current sensor offers high resolution measurements of 12-bit and uses an ADC converter. The sensor is interfaced with other programable devices using the I2C interface and enables easy integration. It is powered by 5V and has a low power consumption. It makes use of a low resistor of 0.1 ohms called shunt resistor and measures voltage drops across it [18].

At last, the INA219 current sensor was used for the following reasons:

- It creates a breakout circuit that can prevent damage to the GPIOs of the Raspberry Pi in case of any hardware failure
- It is a dedicated sensor that is very precise
- The IC provides a programmable configuration register to adjust for specific needs

The connection implies only a simple connection from the solar cell cathode and anode to the Vin- and Vin+ connectors of the integrated circuit breakout board.

4.2.2 SOFTWARE IMPLEMENTATION

The modules are abstracted into a single class that spawns a thread responsible for getting the readable data from the sensor: the shunt voltage and the bus voltage and then computing the power and the current based on the datasheet. This is the set of data that is of interest.

For using the I2C interface for communicating with the current sensor I had to use the Linux I2C libraries which are written in C and for that was used the “extern” keyword. The implementation started in the constructor of the SolarCell class that sets up the configuration register for the current sensor. The configuration of the register took in account the following [19] and was adjusted to specific needs:

- The bus voltage range (the INA219 sensor can be configured to have 16V, 32V or 40V)

- The gain and the shunt voltage (shows that the sensor can accurately measure a maximum shunt voltage of 320mV and it is amplified by a factor of 8)
- The voltage resolution across the buss and the shunt resistor (the resolution is set at 12 bits)
- The conversion time for measuring the current
- The operating mode of the sensor (mode is set to be continuous meaning both the shunt voltage and bus voltage are measured continuously)

The spawned thread is computing a set of values using 5 samples at a rate of 150ms and averaged. After the set of values are computed they are stored into adequate private fields that can be later accessed in a thread-safe manner using a getter.

4.3 AMBIENT MODULE

4.3.1 HARDWARE IMPLEMENTATION

The temperature and the humidity values are acquired using the python bridge and makes use of a sensor that transmits data on a single bus through a digital protocol. It is a 2-in-1 sensor powered by 5V. The sensor was chosen for the following reasons [20]:

- It is accurate (has a temperature error of ± 0.5°C and a humidity error of ± 2%
- Measures temperature within a range of -40°C to 80°C
- It has a lower consumption
- It is robust and reliable (has a protective shield which makes it more resistant to external factors)

4.3.2 SOFTWARE IMPLEMENTATION

The class is based on a thread that periodically request data through the python bridge. The later sends a specific code to the python script through a UDS and gets a response. The reading can often run into errors and the hardware cannot support high data acquire rates and thus, the data is received at a period of 10s if is a correct reading, else, it retries every 2 seconds until the sensor receives a correct reading. As found in the datasheet of the module, the collection period should be at least 2 seconds [20]. The actual hardware interaction is performed in the python script using a designated class along with the help of Adafruit Dht22 library. Whenever the values are incorrect or the sensor fails the process of computing the values, the Python script returns a sample of data that does not have a root in reality. This approach was used to simply manage eventual errors in consideration to the communication environment which is the UDS.

4.4 PYTHON BRIDGE MODULE

This module is a connection layer below the first layer application. It enables the application to interact with the servomotors and the temperature and humidity sensor, mentioned previously at the ambient module. In fact, both the turret control module and the ambient module make use of this one.

As mentioned in the architecture, the module is launched into execution from a specific class called SocketCommunicator. This is how the main functionality is wrapped up:

```
    if __name__ == "__main__":
        parser = argparse.ArgumentParser()
         parser.add_argument("pinTh", type=int, help="GPIO pin number of DHT22 sensor")
        parser.add_argument("pinx", type=int, help="GPIO pin number of horizontal servomotor")
         parser.add_argument("piny", type=int, help="GPIO pin number of vertical servomotor")
         parser.add_argument("socket_name", help="Name of Unix socket to create")
         args = parser.parse_args()
         GPIO.setmode(GPIO.BCM)

        servox = Servomotor(args.pinx)
         servox.setup()
         servox.move(0)
        servoy = Servomotor(args.piny)
         servoy.setup()
         servoy.move(0)
         sensor_reader = SensorReader(args.pinTh)

        server = SocketServer(args.socket_name, sensor_reader)
        server.start()
```

Code 2. Python Bridge Main Function

4.5 STREAMING MODULE

The streaming module is software based only and makes use of the overall architecture of this specific application. The classes that hold information that needs to be known by the user inherit an interface or a pure abstract class that enables this module to make use of the run-time polymorphism concept to later send the data to the web application. This abstract class is demonstrated in the following code sequence:

```
#pragma once

#include <string>
#include <map>
```

```
class Observer {
    public:
        Observer();
        virtual ~Observer();
        virtual std::map<std::string,std::string> uploadData()= 0;
};
```

Code 3. Abstract Class Declaration

Such use of this abstract class also implements the Dependency Inversion principle. This principle comes from the SOLID principles in object-oriented programming and promotes lose coupled classes or modules and strongly encourages the use of abstraction to make the low-level and high-level modules as decoupled as possible. It also states that the abstractions shall not depend on details. This was taken into consideration considering that the Observer class only exposes a single method to be overridden instead of supplying a more specific abstraction for such classes because every class job is strictly dependent on the hardware.

The module is concentrated in a single class named LiveDataFeeder that firstly creates a UDS that is used for sending data to the user interaction application. After the Unix Domain Socket is created the application shall register all the observers as streamers. Only after all the observers are subscribed as streamers, the main execution of the this class cam begin.

When the main execution body of LiveDataFeeder class begins, the class remains in a blocked state until the Flask app connects. When a request for connection is encountered, it is accepted and the streaming of the live data starts by starting the streaming of the live data thread. When the transmission is started, data is gathered from the vector of classes that implement the abstract class Observer making efficient use of polymorphism. The gathered data is then mapped to a JSON (JavaScript Object Notation) with the help of a custom function. After the data is succesfully mapped it is returned to a local variable that stores the entire JSON string. The JSON string is then sent to the Flask client and it is verrified everytime that the transmission is succesful. This workflow can be seen in the following code snippet at Code 4.

```
void LiveDataFeeder::streamLiveDataThread() {

    map<string, string> liveData;

    while(streamingLiveData) {
        liveData.clear();
        for(auto streamer : streamers) {
             auto streamerLiveData = streamer->uploadData();
             liveData.insert(streamerLiveData.begin(), streamerLiveData.end());
            }
        sendLiveData(liveData);
        this_thread::sleep_for(chrono::seconds(1));
    }
}

string LiveDataFeeder::dataMapToJson(map<string, string> dataMap) {
    string json = "{";
    bool first = true;
    for (auto& pair : dataMap) {
        if (!first) {
            json += ",";
        }
        first = false;
        json += "\"" + pair.first + "\":" + pair.second;
    }
    json += "}";
    return json;
}

void LiveDataFeeder::sendLiveData(map<string, string> liveData) {
    string serializedData = dataMapToJson(liveData);
    int bytesSent = send(flaskClient, serializedData.c_str(),
serializedData.size(), 0);
    if(bytesSent == -1) {
        cerr << "Failed to send data" << endl;
    }
}
```

Code 4. Streaming Module Threads

5 USER INTERACTION APPLICATION IMPLEMENTATION

5.1 BACK-END IMPLEMENTATION

The back-end of the web application is implemented using the Flask framework along some particularities found in Flask SQLAlchemy [21] to interact with an instance of the Linux embedded sqlite database.

The SQLAlchemy extension was used for the following reasons:

- Simplified database operations
- Very easy integration

Those two important factors enabled full focus on application rather than writing and testing the database queries. The whole data is captured into a single table:

```
class Readings (db.Model):
  id = db.Column(db.Integer, primary_key=True, autoincrement=True)
  read_time = db.Column(db.DateTime)
  lul = db.Column(db.Integer)
  lur = db.Column(db.Integer)
  ldl = db.Column(db.Integer)
  ldr = db.Column(db.Integer)
  temp = db.Column(db.Float(2))
  hum = db.Column(db.Float(2))
  angle0X = db.Column(db.Integer)
  angle0Y = db.Column(db.Integer)
  mW = db.Column(db.Float(2))
  mA = db.Column(db.Float(2))
  v = db.Column(db.Float(2))
```

Code 5. Using SQLAlchemy To Define Tables

What happens basically is that the data is averaged at the period of 15 minutes and then uploaded into this table where it can be later retrieved from.

This Flask application gets the live data stream from the hardware abstraction application through a UDS and for that, there are two threads designated to do the job of retrieving the data from the socket and handle it. The reading from the socket is done in a non-blocking manner using the select module [22] since it provides a way to monitor input or output sources for reading, writing and error events. The new set of data is then disposed of in a queue where it is later used. This approach was intended to try to limit the data loss and improve handling by using a buffer zone. This implementation is observed in the first part from Code 6.

The handle_new_data_thread() is then responsible for getting data from this queue and handling it according to the needs. The default gateway for this is refreshing the local stored set of data that is used for displaying the live data and if 15 minutes elapsed the data is prepared for a new entry in the Readings table by averaging it. The method was used based on the fact that the system will not have such fast changes and 15 minutes is a considerate time of having an update on environment.

In order to create some fake values for testing the application I tried to use the Faker library in correlation with SQLAlchemy extension but to no avail, thus I reverted back to basics and used the random package for creating mock values and kept the Faker library to mock the date time field only. For better accuracy, I should have added that the timestamps shall be at a 15 minutes period.

The endpoints provided are basically only for, the two presented above that are for main use and other two that come in help for those, which are:

- /get_data
- /get_date_data

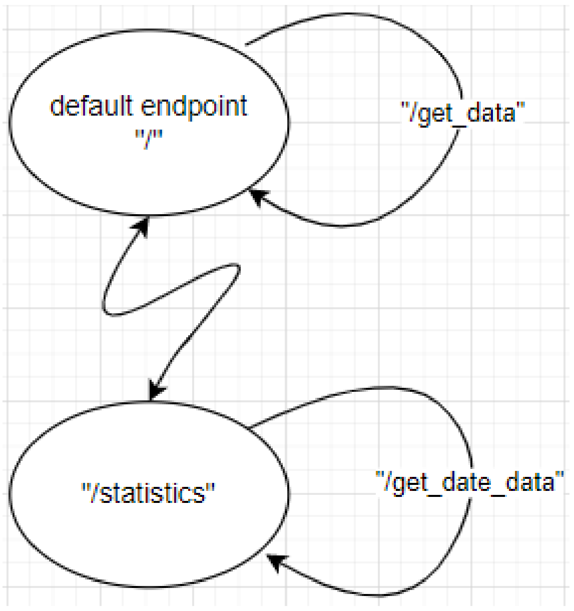


Figure 12. Endpoint Collaboration

The “/get_data” endpoint is used for updating the live data screen with the latest data that is stored in the latest_data variable and updated with every reading from the UDS. It is formatted as a json string and packed as following : jsonify({'data': latest_data})

The “/get_date_data” enpoint is used for getting all the entried of a specific day. It makes use of the SQLAlchemy extension using the filter [23] it provides as it can be seen in the ending part of the snippet from Code 6. The data was ordered at the end to ensure consistency and integrity of the data. After filtering the required information from the database, another filter is added for getting only the needed data important for a daily retrospective: timestamp, the power generated, the temperature and the humidity.

The default endpoint “/” is used for rendering the html page in charge with live data display while the “/statistics” endpoint is used for rendering the html page responsible for creating the daily retrospective.

```
def read_new_data_thread():
    while True:
        has_new_data, _, _ = select.select([client], [], [])
         new_data = client.recv(1024).decode()
         new_data_queue.put(new_data)
        data_available.set()
         time.sleep(1)

required_data = Readings.query.filter(db.func.date(Readings.read_time) ==
specified_date).order_by(Readings.read_time.asc()).all()
```

Code 6.Reading And Retrieving Data

5.2 FRONT-END IMPLEMENTATION

The front-end implementation used the following software tools:

- ChartJS library for displaying the data in a user friendly manner
- AJAX (Asynchronous JavaScript and XML) technique was used for updating the content of the webpage without refreshing it
- jQuery was used to simplify working with AJAX and handling different events

As mentioned previously, there are two main endpoints and for both of them was built an independent page according to their specific needs.

Firstly, the live data page was in dire need of some visual context of displaying the data and a charting library was chosen, in this case ChartJS [24]. There are three types of charts used from here:

- scatter chart
- polar area chart
- line chart

The scatter chart was used for supervsing the servomotor angle position, the polar area chart is responsible for displaying the LDRs data while the line charts are used for representing the current specific measurements: Voltage of the solar panel, the produced Ampers and most importantly, the power generated in milliWatts. As new data arrives, the charts are updated flawlessly and present a smooth transition to every new set of data.

While updating the current charts I implemented a method of having a shifting transition to new data to simulate the passing of time. Furthermore, I choose to display the last 9 seconds in the past and populate the charts with 0s at the beggining. While the data was updating, the following code section was activated:

```
function updateCurrentChart(chart, value) {
    chart.data.labels.pop();
    chart.data.datasets[0].data.pop();
    chart.data.labels.unshift(0);
    chart.data.datasets[0].data.unshift(value);
    for(let i = 1; i < chart.data.labels.length; i++) {
        chart.data.labels[i] = chart.data.labels[i] - 1;
    }
    chart.update();
}

function updateCurrentCharts(voltage, mAmps, mWatts) {
    updateCurrentChart(currentVoltChart, voltage);
    updateCurrentChart(currentAmpChart, mAmps);
    updateCurrentChart(currentWattChart, mWatts);
}
```

Code 7. Current Charts Updating

For having the time-flowing transition the dataset of each chart is updating in the following procedure:

1. remove the last element of the chart with pop()
2. add data at the beggining of the dataset with unshift(0)
3. shift the data to the right with 1 position (corresponding to 1 second)
4. update the corresponding table

The other two measurements that are being displayed are the temperature and humidity. I created a simple representation with the help of two icons from Flaticon, the humidity icon is [25] and the temperature icon is [26]. They were downloaded as free PNGs but had to be converted to SVG format and for that I used the following website https://www.pngtosvg.com.

The overall implementation for the live data webpage is found in the index.html file and also here can be found how the web page is updated. The script implements a setInterval() JavaScript [27] function that repeatedly calls the “/get_data” endpoint using a jQuery method for making an AJAX request to retrieve data from a server [28]. After the set of data is delivered, specific functions are called for updating each representation of those. At the end of the script is implemented a click event listener on a button that switches to the statistics webpage when activated.

6 SYSTEM PERFORMANCE

The system is quite responsive and runs smoothly on the specific Raspberry Pi 2 mode B hardware platform. It has a 900 MHz quad-core ARM Cortex-A7 CPU and it uses the Broadcom BCM2836 SoC (system-on-chip). When both applications are running, there is only about 6-7% CPU usage from those apps and at least half of it is required by the Flask web framework to run. This CPU usage is achievable only when using a remote connection for displaying the live data into the web browser application. In other words, displaying directly on the Raspberry Pi Firefox browser the specific data produces a heavy load on the hardware, with CPU reaching even 30% usage with a core being almost always at full load.

The system was tested according to its purpose, meaning as a demonstration project. All the functionalities are working correctly apart from the power module, which has a slight flaw. Obviously, the entire system is not fully error-proof. For example, a core piece of hardware, meaning the current sensor, could not be calibrated successfully to achieve high precision readings. It was calibrated for reading at most 16V and 400mA, but the readings did not seem to reflect the exact reality of the current generated from the solar cell. Some factors that might have helped this negative outcome, such as:

- Interferences
- Long cables
- Insulation was not the best
- Sensor not being original hence failing at computing tiny details
- Or the sensor simply was not calibrated accordingly.

The temperature and humidity have the values mentioned in the datasheet and fit in the error bound while also reflecting reality.

The turret rotation mechanism captures light in the lower half of the spectrum of visible light, thus enabling stimulation of the LDRs with devices as simple as a small flashlight or a phone flashlight, which is more common. The proper stimulation the system awaits is that which has a delay of at least 1s between every new interaction meaning, if the system is tried to be inquired with multiple light position in this span of time, it will only deliver for the latest.

The overall system works as expected and the manipulation of data is performed in a thread-safe manner. It is also versatile since it can be accessed through the LAN by any device.

7 CLOSING REMARKS

7.1 PROJECT EVALUATION

The system uses a general method of acquisition of 1 second mainly because of software related concerns since using PWM signals can become very cumbersome. The application uses a library designed to interact with GPIOs for transmitting PWM signals with the help of a Python package as mentioned already. I encountered several issues with using this RPi GPIO package and the only solution was to restrict the application to using a 1 second delay in servomotor movement and in almost every data acquisition process. Some of the observered problems are:

- Random excessive jitter
- 360° single-way servomotor rotation even though they should be aable to move in a range of 180° only
- jitter for minor changes of PWM
- small jitter worsening over small timestamp and conducting to system not being usable
- premature failure
- standard speed of rotation being relatively high

Other software related issues were met when trying to install RabbitMQ for sending data from the C++ main application to the Flask application or when trying to install Redis Database for the same purpose. Those applications would have enabled a greater scalability factor.

Another problem that might restrict the use of this Raspberry Pi 2 from higher-end software usage is the fact that it is a rather old device and it reached end of life and end of support. Although there are third-party communities that may provide support, they are usually hard to find. As example, I tried using several WLAN adapters and searched through forums and github repositories, I could not find something that eventually worked. I even tried with 2 totally different WLAN adapters that are specified to be “plug and play” but I`ve received newer versions that were not qualified for that anymore. At the end, the salvation came from prof. Răzvan Cioargă who happened to have had encountered the same problem and has a sollution.

I also encountered hardware issues about the micro-SD card that holds all the important data. The file system got corrupted and I had to reformat the card. This happened 3 or 4 times in the first two weeks of working on the Raspberry Pi. This also acted as a brutal reminder to use a versioning control system for the code, so I used GitHub.

It is also worth mentioning that I did not create any kind of test for the solution and tested and refined the system by re-running it with every new change.

7.2 PROPOSAL FOR FURTHER IMPROVEMENT

The movement of servomotors is 100% dependable on this PWM signal and I tried using other libraries to address this issue like wiringPi and pigpio, but to no avail. To improve the system signifiantly, direct register access should be used for controlling the registers along with building a very accurate pulse signal. This modification would allow the system to be even more responsive. For another signifiant improvement, a small DC motor can be used. Some of the benefits are:

- The control of rotation speed
- Enabling of more precise adjustments
- Higher torque: meaning less needed energy for maintaining a position under a load
- More durable and more robust
- Lower price
- Better handling for little rotations

Also, the servomotors and DC motors work based on the principles of electromagnetism thus, creating an electromagnetic field around them which can negatively affect other hardware components. Improved insulation, different power source and a higher distance between the motors and hardware components are some of the good advices for integrating those into a system without producing any harm.

The project could also benefit from a better and larger solar panel since the used current sensor allows current readings of maximum 26V and it is possible to even implement a battery re-charging station.

Having these modules working would greatly improve the efficiency of the system and would get it closer to a real-world working device even though it implies a relatively long period of development and testing.

A software related problem remained, unfortunately, not solved and this is the ending of the program. The program starts as desired but for stopping it, no actual mechanism was implemented and the main way of stopping the program remains as described in the User Guide Manual attached at Appendix 1. The C++ program implement classes destruction and thread stopping explicitly but those are not called in a way that is part of the program flow. One implementation that can fix this issue is making the C++ program handle a SIGINT or a custom signal whenever it acklowledges it.

Another piece of improvement can be found in the web server the systems uses. As specified by Flask when running the system, the used WSGI (Web Server Gateway Interface) is not a production-ready. This can be replaced with servers that are more secure and stable and which are more suited for real-word usage such as eventlet, Gunicorn, uWsgi or gevent as specified by the official flask documentation for self hosting. Although I tried using eventlet, I quicky ran into errors that would render my second application un-runnable, hence I dropped the improvement.

7.3 SYNTHESIS OF CONTRIBUTIONS

The personal contributions for this project can be sorted in the next categories and are as follows:

- System sollutions such as:
  - Choosing components (based on price and performance)
  - Ordering components
  - Setting up development environment (bootable microSD, installing all the necessaries)
  - Compiling and installing WLAN Linux kernel driver
  - Setting up LAN access for the Flask application
- Packaging
  - Conceptual design
  - Assembly
- Hardware
  - Design
  - Implementation on a breadboard
  - Soldering on a dedicated board
- Software
  - Individual application design
  - System design
  - Implementation

7.4 PERSONAL ASSESMENT

The completion of the project brings me joy as it works 99.9% as desired. It is the first poject I realize that has so many fields of development and at times, I got overwhelmed. Nonetheless, I managed to get over every wall that was encountered, being it a miss-calculation of the packaging, a burnt hardware component or a software limitation and come up with a new solution. I am profoundly grateful of the support I had when times were hard and meeting the deadline seemed impossible. I`ve learned again, not to give up.

APPENDIX

APPENDIX 1. INSTRUCȚIUNI DE UTILIZARE (MANDATORY IN ROMANIAN / OBLIGATORIU ÎN LIMBA ROMÂNĂ)

1 SETĂRI INIȚIALE

Pentru alimentarea sistemului va fi nevoie de un cablu micro USB care este capabil să producă o tensiune de 5V si curent de 2A. Pentru alimentare, cablul va fi inserat prin gaura inferioară a cutiei în soclul de alimentare al Raspberry Pi-ului.

Pentru folosirea conexiunii la distanță precum, verificați instrucțiunile de la următorul capitol.

Pentru folosirea unui cablu HDMI se poate folosi aceeași gaură ca și cea de alimentare.

Pentru verificarea integrității componentelor se poate înlătura capacul din spate desfăcând șuruburile din părțile laterale, pozitionate în rândul de sus.

Suplimentar, va fi nevoie de un mouse sau o tastatura conectate. Un set de tastatură și mouse wireless sunt recomandate pentru a evita desfacerea excesiva a carcasei.

2 CONEXIUNI

Conexiuni Hardware

În cazul în care sunt întâlnite defecte hardware va fi nevoie de verificarea placuței din interior. Pentru verficări se poate folosi un multimetru.

Pentru adăugări, eventualele modificări vor fi efectuate cu grijă și munca va fi testată incremental.

Conexiunea la internet

Pentru a avea o conexiune de internet, se poate folosi un cablu de internet normal sau un stick USB adaptor pentru WLAN.

Se pot folosi urmatoarele adaptoare (testate deja):

- TP-LINK TL-WN821N v4 funcționează ”plug & play”
- Tenda U12 funcționează cu drivere instalate de aici: https://github.com/morrownr/8812au-20210629

Conexiunea SSH

Pentru folosirea unei conexiuni SSH va fi nevoie ca rețeaua folosită să fie identică cu cea a Raspberry Pi.

Pentru a putea porni o sesiune SSH, se va descărca (dacă este cazul) anumite programe ce pot indeplini acest lucru pe computerul gazdă, ca de exemplu:

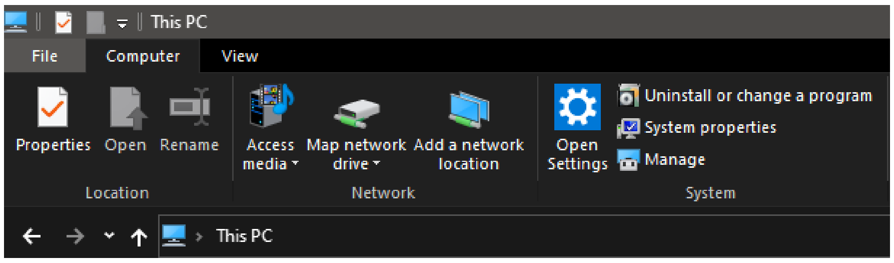
a. Putty (https://www.putty.org)
b. MobaXterm (https://mobaxterm.mobatek.net)

1. Software-ul va fi instalat si configurat (daca e nevoie) iar apoi se ruleaza
2. Dupa rulare se va seta Host Name sau adresa IP.
   - Hostname: raspberrypi
   - IP: se poate afla folosind comanda: `hostname -I`
3. Setare username: corona
4. Setare parolă: corona

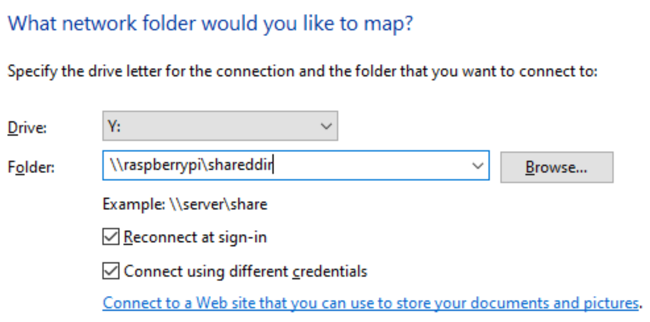
Conexiune Samba

Pentru a folosi un Samba share, adică pentru a putea vedea directorul cu software-ul care ruleaza pe un computer gazdă ce ruleaza Windows, acesta trebuie mapat unui drive. Aceasta se realizează în urmatorii pași.

1. Deschide Windows File Explorer şi selectează Map Network Drive



2. Configurează conexiunea Samba


What network folder would you like to map?

Specify the drive letter for the connection and the folder that you want to connect to:

Drive: Y:

Folder: \\raspberrypi\shareddir

Example: \\server\share

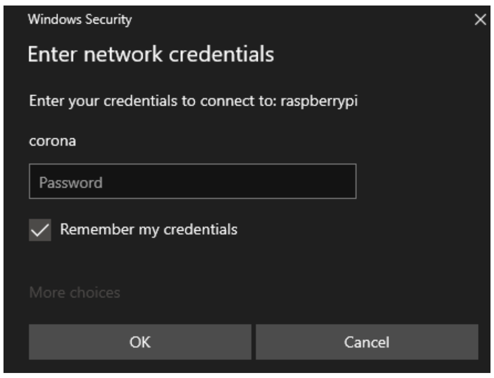
☑ Reconnect at sign-in

☑ Connect using different credentials

Connect to a Web site that you can use to store your documents and pictures.

3. Introdu informatiile de identificare

Username: corona
Password: toby (foloșeste pentru schimbare: sudo smbpasswd -a corona)


Windows Security

Enter network credentials

Enter your credentials to connect to: raspberrypi

corona

Password

☑ Remember my credentials

Apasa OK iar drive-ul ar trebui să apară mapat în file explorer. În cazul în care hostname-ul *raspberrypi* nu funcționeaza, folosește ip-ul asignat al Raspberry Pi în rețea.

3 UTILIZAREA SISTEMULUI

Pentru rularea programului e nevoie de 2 conexiuni shell diferite. Acestea se pot deschide in Putty.

1. Prima conexiune se va realiza:
    a. Navighează in fișierul aplicației 1 folosind comanda urmatoare:
        i. `cd /home/corona/shared/aaa/SunTracker/appl`
2. Pentru a doua conexiunea shell:
    a. Navighează in fișierul proiectului
        i. `cd /home/corona/shared/aaa/SunTracker`
    b. Ruleaza comanda urmatoare
        i. `source env/bin/activate`
    c. Navighează in fișierul aplicației
        i. `cd webserver`

După aceasta, proiectul se va porni executând:

1. Pentru pornirea primei aplicații:
    a. în primul shell se va rula: `./appl`
2. Pentru porniea aplicației secundare:
    a. `python3 app.py`

După pornirea celor două aplicații se va folosi un web browser pentru a naviga la adresa returnată la pornirea celei de-a doua aplicații sau se va introduce direct in browser urmatoarele:
`raspberrypi.local:5000`

Pentru oprirea sistemului se va utiliza `CTRL + C` de 2 ori consecutiv in shell-ul al doilea.

Dacă se intamplă ca sistemul să nu pornească, se va opri execuția sistemului iar apoi se repornește.

Pentru ca datele să fie afișate corespunzător, este recomandat folosirea browserului Firefox (în cazul conexiunii directe la Raspberry Pi) sau a oricărui browser modern de pe un calculator în cazul conexiunii de la distanță, anume SSH (remote). În cazul conexiunii directe, puterea de procesare va fi masiv impactată de rularea browserului.

Dacă se dorește rularea pe un anumit hostname sau port, folosiți această pagină pentru mai multe detalii https://flask.palletsprojects.com/en/2.3.x/quickstart/.

Pentru a putea qqqeed

Notă: modelul folosit Raspberry Pi 2 Model B v1.1 este un model invechit si modelele noi sunt mult mai performante.

BIBLIOGRAPHY

[1] "Installing Python Packages." [Online]. Available: https://packaging.python.org/en/latest/tutorials/installing-packages/. [Accessed: April 1, 2023].

[2] A. Kumar, S. K. Gawre, M. Sarkar, and S. Gosula, “A real-time comparative data analysis of different types of solar panels during partial shading with distinct tilt angles,” *2018 15th IEEE India Council International Conference (INDICON)*, 2018. doi:10.1109/indicon45594.2018.8987115.[Accessed: June 19, 2023]

[3] Y. Liu, C. Li, C.-W. Lo, and H. Jiang, “Autonomous passive light tracking utilizing single-wall carbon nanotube enhanced opto-thermo-mechanical elastomer actuators,” Jun. 2011, doi: https://doi.org/10.1109/transducers.2011.5969812, pp. 2726. [Accessed: June 19, 2023]

[4] Bhagwan Deen Verma, Anurag Gour, and Dr. Mukesh Pandey, “A review paper on Solar Tracking System for Photovoltaic Power Plant,” *International Journal of Engineering Research and*, vol. V9, no. 02, 2020. doi:10.17577/ijertv9is020103, Figure 3.[Accessed: June 19, 2023]

[5] "Solar Tracker Projects," Instructables,. Available: https://www.instructables.com/search/?q=solar%20tracker&projects=all. [Accessed: March 12, 2023]

[6] "DIY Miniature Solar Tracker," Instructables, Available: https://www.instructables.com/DIY-Miniature-Solar-Tracker/ .[Accessed: March 12, 2023]

[7] Md. N. Reza, Md. S. Hossain, N. Mondol, and Md. A. Kabir, “Design and implementation of an automatic single axis solar tracking system to enhance the performance of a solar photovoltaic panel,” 2021 International Conference on Science &amp; Contemporary Technologies (ICSCT), 2021. doi:10.1109/icsct53883.2021.9642557. [Accessed: June 19, 2023]

[8] "Thingiverse - Digital Designs for Physical Objects," [Online]. Available: https://www.thingiverse.com/thing:708819. [Accessed: May 19, 2023].

[9] "Linux man page - socket(2)," man7.org. [Online]. Available: https://man7.org/linux/man-pages/man2/socket.2.html. [Accessed: May 4, 2023].

[10] Flask, "Flask Documentation," [Online]. Available: https://flask.palletsprojects.com/en/2.3.x/. [Accessed: April 1, 2023].

[11] cplusplus.com, "thread - C++ Reference," [Online]. Available: https://cplusplus.com/reference/thread/thread/. [Accessed: April 24, 2023].

[12] cppreference.com, "std::mutex - cppreference.com," [Online]. Available: https://en.cppreference.com/w/cpp/thread/mutex. [Accessed: April 24, 2023].

[13] SourceForge, "Raspberry-GPIO-Python - Documentation," [Online]. Available: https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/. [Accessed: April 22, 2023].

[14] Pimylifeup.com, "Raspberry Pi Humidity Sensor using the DHT22," [Online]. Available: https://pimylifeup.com/raspberry-pi-humidity-sensor-dht22/. [Accessed: April 27, 2023].

[15] AllDataSheet.com, "MCP3008-I/P Datasheet - Microchip," [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/194718/MICROCHIP/MCP3008-I/P.html. [Accessed: April 30, 2023].

[16] Emcraft Systems, "Accessing SPI Devices in Linux," [Online]. Available: https://www.emcraft.com/som/stm32f7-240/accessing-spi-devices-in-linux. [Accessed: May 7, 2023].

[17] cleste.ro, "PANOU SOLAR MINI 5V 1.1 W 220MAH," [Online]. Available: https://cleste.ro/panou-solar-mini-5v-1-25-w-100mah.html. [Accessed: April 7, 2023]

[18] Texas Instruments, "INA219: High-Side Measurement, Shunt and Bus Voltage Monitor," [Online]. Available: https://www.ti.com/lit/ds/symlink/ina219.pdf. [Accessed: May 5, 2023].

[19] Stack Overflow, "Adafruit INA219 module - Shunt voltage values and calibration," [Online]. Available: https://stackoverflow.com/questions/39713799/adafruit-ina219-module-shunt-voltage-values-and-calibration. [Accessed: May 7, 2023].

[20] SparkFun Electronics, "DHT22 Digital Temperature and Humidity Sensor," [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf. [Accessed: May 15, 2023].

[21] "Flask-SQLAlchemy," Pallets Projects. [Online]. Available: https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/. [Accessed: May 1, 2023].

[22] Python Software Foundation, "select — Waiting for I/O completion," Python 3 Documentation, [Online]. Available: https://docs.python.org/3/library/select.html. [Accessed: May 19, 2023].

[23] SQLAlchemy, "Query — SQLAlchemy 1.4 Documentation," [Online]. Available: https://docs.sqlalchemy.org/en/14/orm/query.html#sqlalchemy.orm.Query.filter. [Accessed: May 9, 2023]

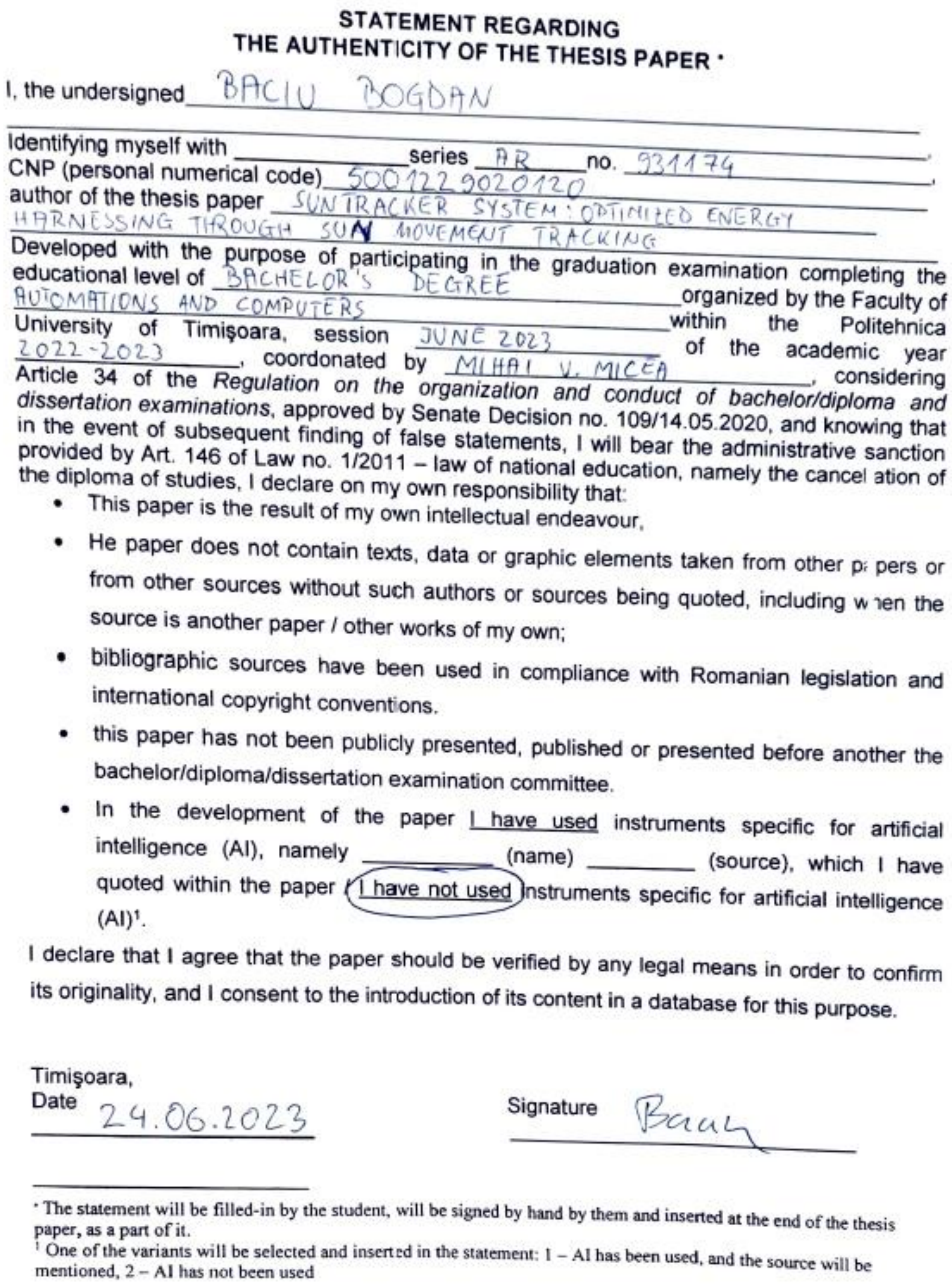
[24] Chart.js, "Simple yet flexible JavaScript charting for designers & developers," [Online]. Available: https://www.chartjs.org/. [Accessed: May 10, 2023].

[25] Flaticon, "Humidity icon" [Online]. Available: https://www.flaticon.com/free-icon/humidity_1574227?term=humidity&page=1&position=73&origin=search&related_id=1574227. [Accessed: May 15, 2023].

[26] Flaticon, "Temperature icon" [Online]. Available: https://www.flaticon.com/free-icon/temperature_2387835?term=thermometer&page=1&position=35&origin=search&related_id=2387835. [Accessed: May 15, 2023].

[27] "setInterval() Method in JavaScript," Programiz.com. [Online]. Available: https://www.programiz.com/javascript/setInterval. [Accessed: April 28, 2023].

[28] "jQuery.getJSON() | jQuery API Documentation," jQuery. [Online]. Available: https://api.jquery.com/jquery.getjson/. [Accessed: April 28, 2023].



**STATEMENT REGARDING
THE AUTHENTICITY OF THE THESIS PAPER ***

I, the undersigned BACIU BOGDAN

Identifying myself with ____________ series AR no. 931174, CNP (personal numerical code) 5000122020120, author of the thesis paper SUNTRACKER SYSTEM: OPTIMIZED ENERGY HARNESSING THROUGH SUN MOVEMENT TRACKING

Developed with the purpose of participating in the graduation examination completing the educational level of BACHELOR'S DEGREE AUTOMATIONS AND COMPUTERS organized by the Faculty of within the Politehnica University of Timişoara, session JUNE 2023 of the academic year 2022-2023, coordonated by MIHAI V. MICEA, considering Article 34 of the *Regulation on the organization and conduct of bachelor/diploma and dissertation examinations*, approved by Senate Decision no. 109/14.05.2020, and knowing that in the event of subsequent finding of false statements, I will bear the administrative sanction provided by Art. 146 of Law no. 1/2011 – law of national education, namely the cancel ation of the diploma of studies, I declare on my own responsibility that:

- This paper is the result of my own intellectual endeavour,
- He paper does not contain texts, data or graphic elements taken from other papers or from other sources without such authors or sources being quoted, including when the source is another paper / other works of my own;
- bibliographic sources have been used in compliance with Romanian legislation and international copyright conventions.
- this paper has not been publicly presented, published or presented before another the bachelor/diploma/dissertation examination committee.
- In the development of the paper I have used instruments specific for artificial intelligence (AI), namely ____________ (name) __________ (source), which I have quoted within the paper / [✓ I have not used] instruments specific for artificial intelligence (AI)[1].

I declare that I agree that the paper should be verified by any legal means in order to confirm its originality, and I consent to the introduction of its content in a database for this purpose.

Timişoara,
Date 24.06.2023

Signature [illegible]

* The statement will be filled-in by the student, will be signed by hand by them and inserted at the end of the thesis paper, as a part of it.
[1] One of the variants will be selected and inserted in the statement: 1 – AI has been used, and the source will be mentioned, 2 – AI has not been used