Aalto-yliopisto, Sähkötekniikan korkeakoulu
ELEC-D0301 Protopaja
2018

Raportti

Project #08
Test Sensor Package and Data Logger

Date: 26.8.2018

Jyri Korhonen
Antti Matikainen
Vladimir Kurazhev
Anton Dyomin

Information page

Students

Antti Matikainen 353980

Jyri Korhonen 529303

Anton Demin 585431

Vladimir Kurazhev 587345

Project manager

Jyri Korhonen

Sponsoring Company

Safera

Starting date

4.6.2018

Submitted date

31.8.2018

Tiivistelmä

Safera on yritys joka valmistaa keittiössä käytettävää turva laitteistoa jota käytetään keittiöissä tulipalojen välttämiseen. Esimerkiksi tilanteessa jossa paistinpannu on jätetty levylle ja se kuumenee aiheuttaen tulipalovaaran. Tällaisessa tilanteessa Saferan valmistama anturi voisi ottaa yhteyttää hellaan asennettuun yksikköön joka sulkee hellan. Safera valmistaa myös malleja jotka pelkästään hälyttävät jotta hellan käyttäjä huomaa vaaratilanteen ja korjaa sen. Jotta turva antureita voidaan käyttää on heillä oltava paljon dataa jonka perusteella he voivat määritellä mikä on ja ei ole turvallinen tilanne. Tähän tarvitaan niin lämpötila kun visuaalista dataa jota he mittaavat labrassaan.

Saferan antaman tehtävänannon mukaan projektin tarkoitus oli prototyypata ja luoda anturi paketti ja sen käyttöön liittyvät ohjelmat joita Safera voisi soveltaa laboratoriossaan. Tärkein osa anturi pakettia oli thermocouple anturi joka olisi helppo käyttöinen ja halvempi verrattuna heidän käyttämään kaupalliseen malliin. Projektin alussa käydyissä tapaamisissa totesimme että tämä thermocouple anturi on ns. minimum viable product projektille. Ensimmäinen idea projektille sisälsi myös TCP palvelin asiakas arkkitehtuurin joka mahdollistaisi Raspberry Pin käytön anturi keskuksena. Tämän laitteisto suunnittelun lisäksi projektiin kuului myös järjestelmän käyttöön liittyvät ohjelmistot.

Käytännössä kuitenkin projektin aikana jouduimme muuttamaan projektin scopea. Esimerkiksi suunnitelma käyttää Raspberry Pi:ta anturi hubina joka yhdistettäisiin koneeseen käyttämällä ethernettiä ja TCP palvelin asiakas arkkitehtuuria ei vaikuttanut realistiselta. Päätimme keskittyä pääasiassa thermocouple anturi järjestelmään ja yksinkertaisen tiedon tallennus järjestelmän luomiseen. Tämä johtui siitä että alkuperäinen suunnitelma osoittautui monimutkaisemmaksi kun oletimme. Käytännössä keskityimme siihen mikä oli projektille olennaisinta.

Abstract

Safera is a company which produces a variety of products which are intended to improve kitchen safety and to avoid situations which could potentially lead to domestic fires. For example, if a pan has been left on a hot stove the system could determine that the pan is reaching an unsafe temperature, as well as detecting the smoke, before communicating with a device connected between the wall socket and the stove in order to turn off the power. Other models are simpler and instead of turning off the power would simply alert the user so they will turn off the stove. In order to do this, they must have access to a wider variety of data from multiple cooking situations both safe and unsafe. Based on this data they can program their products in order to better determine when the power should be turned off or the user alerted. For this, they have a laboratory they use to take measurements of various sorts.

The initial information given to use at the start of the course indicated that they were hoping that the group could develop and prototype a more open source and better documented sensor solution as well as the software related to it. The most important part was a system for taking measurements from a thermocouple as the commercial solutions were often quite flawed and expensive. During meetings with the Safera representative it was determined that the thermocouple solution was the least viable product for the project. As long as it was comparable in accuracy as well as being cheaper, it'd be considered a success. In addition to this the project involved the software related to this functionality.

The original plan for the project involved using a Raspberry Pi as a sensor hub as well as implementing a TCP client server architecture allowing multiple Raspberry Pis to be connected to the base PC at once. However it was determined that this was likely to be unrealistic due to the amount of work involved in prototyping the software and sensors. It was decided that the project should focus on a solution where all the sensors are connected to the computer. This is due to it being determined that it was simply additional complications and we should focus on doing a single thing well instead of attempting to do everything.

Table of Contents / Sisällysluettelo

1. Introduction

Safera is a company specializing in stovetop sensor packages that are used to avoid potential domestic fires by identifying dangerous situations. Their product utilizes a variety of sensors in order to complete this task including smoke detectors and IR temperature sensors. In order for their product to be effective at its purpose it must be programmed to account for a variety of situations. In order to do this they require sensor data from a wide variety of situations that are both safe and unsafe. To gather this data they have both a small lab in their offices as well as a secondary lab built outside the building for experiments which might involve smoke. At the moment they use an in-house developed software alongside a commercial thermocouple to USB device as well as other commercial sensor solutions.

However the commercial solutions are not perfect. The primary issue with them is the expense involved. Each unit is capable of supporting multiple probes, however the units themselves are quite expensive at hundreds of euros each. In addition the placement of the cold-side compensation sensor (used to get an accurate temperature reading for the thermocouple) is poor leading to inaccuracies in measurement. In addition the drivers for them are often quite poor. For these reasons Safera was looking for a well documented open alternative that was comparable in price.

Another issue that this project was intended to solve was the fact that the commercial solutions are often poorly documented. Their drivers can be quite buggy and when the hardware is damaged it can be impossible to determine the cause of the flaw. This would mean that the only solution was to replace the entire device. In an ideal world it would be possible to simply repair the damaged part but due to the undocumented closed source nature of the commercial solutions this is simply not possible. Another aspect of this project was to ensure that the solution was well documented enough and designed in such a manner that this issue could be avoided.

2. Objective

The objective of the project was to create a sensor package that could be used to take a variety of measurements using thermocouple sensors as well as a camera, air particle counter and other potential tools that can be added in the future. The primary focus of the project was the thermocouple sensor package with the cameras and particle counters simply being commercially available modules. In addition to this the initial plan involved a TCP client/server architecture in order to allow for multiple sensor modules to be connected to a single computer. In addition to this the project involved programming data logging software as well as the software required for the RasPi and other components.

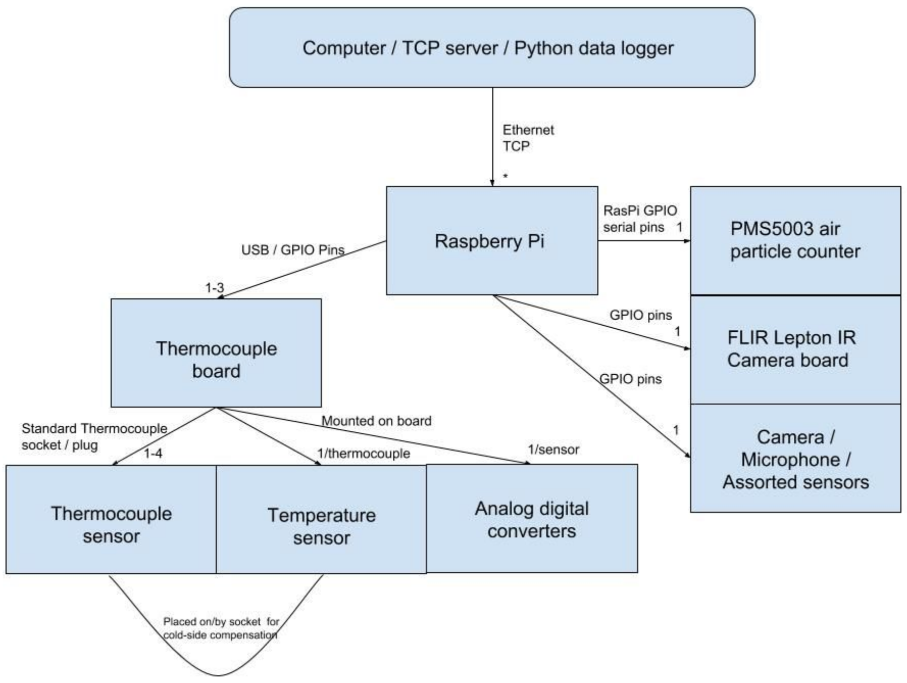

Computer / TCP server / Python data logger
Ethernet TCP
*
Raspberry Pi
RasPi GPIO serial pins 1
PMS5003 air particle counter
USB / GPIO Pins
1-3
GPIO pins 1
FLIR Lepton IR Camera board
GPIO pins
1
Camera / Microphone / Assorted sensors
Thermocouple board
Standard Thermocouple socket / plug
1-4
Mounted on board
1/thermocouple
1/sensor
Thermocouple sensor
Temperature sensor
Analog digital converters
Placed on/by socket for cold-side compensation

Figure 1. Initial proposed architecture for the sensor package

However, it was determined fairly soon into the project that the initial plan was on the extensive side. The Safera representative suggested that we could either do one thing well or do everything in a mediocre manner. Because of this, we determined that it'd be best to simply remove the TCP architecture aspect and connect all the sensors to the measurement PC via USB. In addition to this the IR camera option was determined to be of less importance as more of the man-hours were allocated towards the software and thermocouple solution. The members of the group were spread fairly evenly between working on the thermocouple solution, working on the software and working on integrating cameras. Chapters 3 and 4 will describe both aspects in further detail.

3. Thermocouple Sensor Solution

Figure 2. Picture representing koala. It is adorable and makes up for the fact our project isn't very visual and we don't really have pictures to show of it.

The thermocouple sensor solution was the primary focus of the project and in a way a way to judge the project's success or failure. The aim of the thermocouple sensor solution is to prototype a sensor capable of outputting thermocouple sensor temperature readings to a computer over USB. In addition the thermocouple sensor solution includes a temperature sensor placed next to the connector to allow for accurate measurements for the coldside compensation. In addition to this the board includes analog-digital converters as well as a microcontroller to act as an interface.

Thermocouples consist of two conductors with different electrical characteristics (U.A. 2018.). When the point at which the two conductors are connected is exposed to a hot or cold temperature (depending on the type of thermocouple) a voltage is produced that can be measured. This voltage is also dependent on the temperature allowing for the temperature to be calculated. Another important aspect for accurate measurements using thermocouples is that the voltage is based on the difference in temperature between the two junctions at which the wires are connected. To get an accurate reading one must know the temperature of the cold side utilizing a process known as Cold Side Compensation (Laurila, H. 2017). Some methods involve submerging the cold side in an ice bath while more practical solutions utilize a temperature sensor in order to measure the temperature of the cold side.

Once the hot junction has been placed at the site of the measurement and the voltage generated by the thermocouple has been recorded, the second stage of the process involves recording the temperature of the cold junction at that time. Utilizing either a table of standards or a mathematical formula the cold-side temperature can be turned into voltage and added to the recorded voltage. This voltage can then be converted into a temperature using the same formula or table of standards. Mathematically this can be expressed by the following formula:

$E = E_N(t_{U1}) - E_N(T_r)$, where E is the measured voltage, E(T_U1) is the voltage generated by the hot end and E(T_r) is the voltage generated by the cold end.

Formula 1. Calculating a temperature from a Thermocouple reading. (Laurila, H. 2017)

The thermocouple solution used in the project utilizes a DS18B20+T&R digital temperature sensor in order to allow for cold-side compensation without requiring the cold junction to be at a specific temperature. This allows for more accurate measurements of the temperature. In order to allow for the probes to be easily replaced, the solution uses a standardized thermocouple plug which allows for easy replacement of components. The voltage measurements from the probe are converted to a digital form by a AD7793BRUZ analog to digital converter. The measurements taken by the sensors are then transferred to a microcontroller acting as an interface that allows the sensor to be connected to a computer. These components are mounted to a circuit board designed by the project team. The microcontroller will then output the package containing the readings (either before or after conversion from voltage reading to temperature) over USB to the computer where it can be read by the software.

The designed circuit board consists of socked for thermocouple, AD7793, low pass filter and DS18B20 temperature sensor. Research about minimizing noise and general circuit board design took much more time than anticipated. One side of the board was divided between analog and digital ground planes connected by 0 ohm resistor. Power supply for the AD7793 was decoupled with 0.1μF and 10 μF capacitors and filter included 0.1μF decoupling for both wires.

Although initial designs focus on utilizing a single probe, there is nothing that stops a multiprobe version from being built based on the design in the future. Because the project focuses on the prototyping phase the choice was made to focus on a single probe at first. It was simply one of the many choices made due to the short timespan of the course. A potential issue with a multiprobe solution is that readings are intended to be taken once per second. If too many probes are connected at once and the calculations to convert the thermocouple readings to a temperature are done in an inefficient manner the output might be delayed. This can, in turn, be solved with optimization of the code as well as choosing a microcontroller that is powerful enough to perform the required calculations.

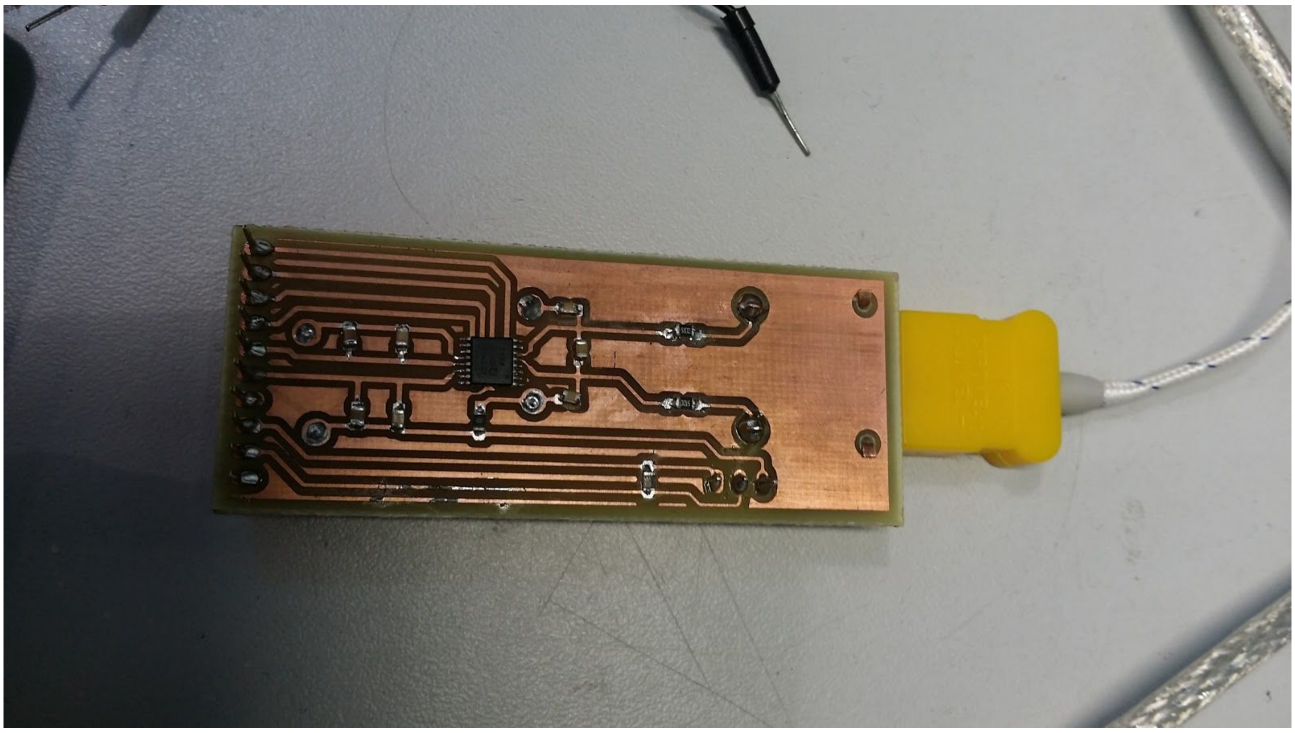


Figure 2. Image of the Thermocouple circuit board

4. Datalogger Software

The second part of the project was to program a data-logging software to be used alongside the sensor package. It was determined that the software would be built from scratch for several reasons. The first of these is that the code for the software currently used by Safera is based on .Net which none of the group members were familiar with. The second reason was that one of the stipulations of the course was that all code would be released under the MIT license, which poses an issue as Safera stated that all of their software would be given to use only if a NDA was signed. Because of this it was mutually decided that starting from scratch was beneficial to both parties.

The decision to utilize Python instead of .Net in writing the datalogger software was made during one of the initial meetings for the project group as it was determined that it was the programming language that every member of the group had at least some familiarity with. It also had the benefit of being multi platform allowing for the proposed software to be used both on linux and windows based machines. The flexibility of the language was also determined to be useful for rapid prototyping and development. Python also had multiple development environments as well as existing libraries for user interfaces and serial port related operations. This gives it an advantage over lower level languages as it would most likely be significantly more challenging.

The software can be split into a handful of large components or modules each of which will be covered by its own sub-topic. The largest two of these are the user interface and the serial port functionality. Then there are file operations and coldside-compensation modules which are significantly smaller and are used by the first two for a variety of purposes. These descriptions are on the brief side, but the code itself is commented which should elaborate more on its function and the purposes of files, functions and specific lines of code.

4.1. User Interface

The user interface for the datalogger software was coded using the TkInter package built into Python. During planning for the user interface several options were considered such as PyQT and Pyside. The issue with many of them, however, was that the licenses proved potentially bothersome due to either requiring commercial licenses or open licenses with restrictions. Because of this it was decided that TkInter provided the solution least likely to pose problems at a future date. One of the potential downsides of TkInter is the limited and rudimentary appearance of the UI however this was determined to not be a concern.

TkInter is an event based interface that is designed to be quite simple. The system is entirely event based meaning that instead of the programmer defining a loop containing UI functionality such as clicking they instead first define UI elements and events triggered by button presses and elements of the UI. The functionality of the UI such as saving data and other functions are handled by call-back functions tied to these UI elements. This is a limiting factor but one that can be worked around.

The main function containing the UI functionality is quite simple. The first segment contains an object declaration which represents the user interface itself. The initialization function for this object defines and places all the major UI elements that make up the program. The choice to use an object for this was made as it allowed for the various variables to be better accessed by various functions. It also helped with other minor matters such as the structure of the code. These callback functions that're tied to the various buttons make up the entire functionality of the UI from the re-initialize function to the functions to start and stop recording to the comment functionality. A text-box was also included for diagnostics information.

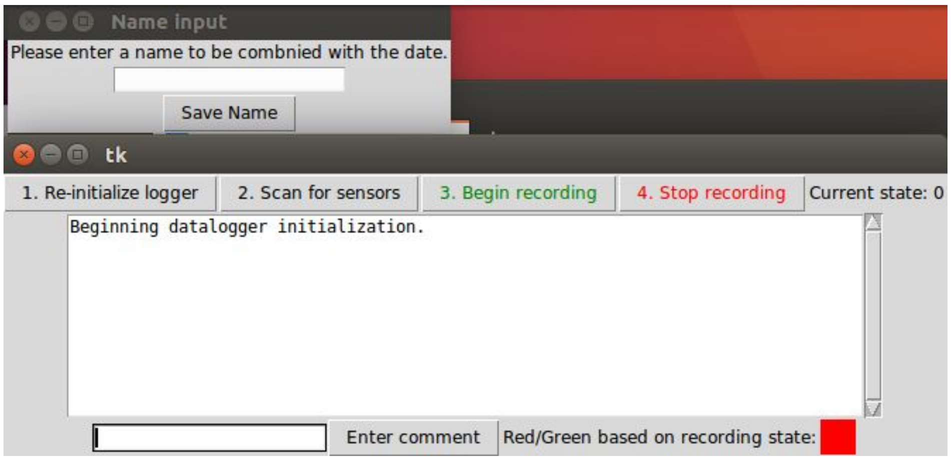
TkInter contains several methods of controlling the positioning of UI elements. The one used for the project user interface is the grid functionality. Essentially it allows several grids, in this case 2, to be defined. In these grids the relative positioning of the buttons is easily defined by giving them a position in the cells of the grid simply by defining the x and y coordinates of the cells. Objects such as the text box can also be made to take up multiple cells of the grid.

One of the primary issues, however, with TkInter is that due to the event based nature of the user interface it is challenging to have a timed event which reads from the serial ports once a second. A while loop running inside a function called by one of the user interface elements would simply lead to the user interface freezing as the primary process can't return to the TkInter main loop. This means that a work-around for this must be utilized in order to accomplish the desired functionality.

The first of these is the less desirable of the two options. TkInter includes a "after" method which can be used to schedule events. The method takes as inputs a function and a delay after which the function is to be called. This can used to have a function that calls itself after a second in order to read from the serial ports roughly once a second. Shutting down the loop can be done simply by toggling an object variable which is checked before a new event is scheduled. In practice however this is quite a flawed solution. Due to the fact that it schedules the event after a second after the iteration of the function the function will be completed after a second and an unknown delay instead of once a second. This is also a very unwieldy and in many ways unelegant solution. However there is a chance that it might be the only one that will work in practice.

The second option is to use the multiprocessing library in python in order to create a subprocess that calls a specific function. This function is then passed a pipe which can be used for communication between the main and sub processes. This has the benefit of allowing the subprocess to call a loop without negatively affecting the main process. The pipe also allows for the sub process to be easily ended when it is time to stop the recording. This secondary process can also time events in a more effective manner via other libraries. There is, however, an issue with this solution as well. Because this subprocess solution utilizes pipes for inter-process communication Python IO functionality must be utilized to read the messages. This is where the problems arise. The pipe receive function is what is referred to as blocking I/O. This means that if there is nothing in the pipe to receive the function will simply wait until there is. This means that we can't simply stick a check for the pipe at the beginning of the sensor reading loop. With Python3 there is the asyncio module which can be used for non-blocking I/O however this has the problem of being a fair deal more challenging to utilize.

TL;DR: Tkinter was used for the user interface. The functionality is largely built into the App object as object variables can be used to pass information between functions. The object constructor defines all of the UI elements as well as the functions tied to them before before calling the function to create the name input box. After this the program enters the Tkinter main loop which handles much of everything else. The buttons call functions to enter the name of the experiment (to initialize and create the file and folder), scan for connected serial ports and to begin and end recording. The code is quite extensively documented.



Please enter a name to be combnied with the date.

Current state: 0

```
Beginning datalogger initialization.
```

Red/Green based on recording state: 🟥

Figure 3: Screenshot of the datalogger user interface

4.2. Serial Port Operations

The serial port operations covers all functionality that interacts with the sensors connected to the computer by USB or serial port adapter. It is called by a forked process from the main function because of limitations of the user interface. Because of how TkInter runs its own loop and is entirely event driven the readings can't be taken automatically on a timer. Due to this a call-back function has to fork to a process that reads from the sensors before writing the readings to a file. Python includes a multiprocessing library with functionality that allows for this. The interactions with the serial ports themselves are accomplished using the PySerial library which includes a wide variety of useful functionality.

Due to the use of PySerial opening serial ports for communication is quite simple. PySerial includes a tool under serial.tools.list_ports.comports which can be called in a function to return a list of serial ports connected to the computer. These ListPortInfo objects also contain further information about the serial port such as the name and path, just the name, description as well as other IDs and hardware information. These can be used to identify the devices which can be quite useful when determining how the data read from the ports must be handled. Although this tool can also be used from the command line, for the purposes of this project calling it as a function is the most appropriate solution.

Opening and reading from the serial port is quite simple. Opening the serial port can be done quite simply by creating a variable with the serial.Serial object, giving the address of the port as an input variable. In addition it is possible to define the baudrate and timeout as well as other variables. By default the baudrate is 9600, meaning that in most situations no change is necessary. Using the variable it is possible to write to, read from and close the port. Reading is done by simply using the Readline function.

For the datalogger program created for this project the listPorts.comports tool is used to retrieve a list containing the sensors which can then be scanned for the relevant ones based on a variety of variables. From here, the recording function can open serial ports which it will then query once per second for the readings. These readings are then handled, converted to temperature in the case of the thermocouple, before being passed on to the file writing function. While on C or C++ working with serial ports would be quite challenging, the PySerial library handles much of the work in for the purposes of this project.

TL;DR: PySerial was used for serial port operations. A tool that comes built in called listPorts.comports is used to gather a list of connected serial devices so that the user can select the desired ones to take readings from. Aside from this, standard serial port flush and read operations that come with the package are used. The read operation used reads until a newline character is detected (although alternatives are also available). The data read from the serial port is then parsed and written to file.

Uses standard PySerial functionality as Python has no other method of utilizing serial ports. PySerial is open source under an simple license included in the git Licenses file.

4.3. File Operations

The file operations segment is quite simple. It is a module containing the functions for writing formatted strings to the relevant files. The code is quite simple containing little beyond checks that the inputs are valid and the operations required for the file operations. They were made into a separate module as both the comment functionality and sensor functionality work by writing into formatted text files. Because of the choice of using text files it was determined that simple python file operations would suffice instead of a library to handle, for example, excel data. These text files can then later on be parsed with other tools and data visualization solutions.

Takes in the name/filepath of the file to be written to and the string to be written as inputs. Uses standard python file IO functionality. Overall, a simple implementation.

4.4. Thermocouple Operations

The thermocouple operations code is functionally a wrapper around an open sourced thermocouple calculation library. The library includes functionality for calculating either the temperature or voltage of a thermocouple based on the other. It also includes functionality for a variety of thermocouple types. As the library is open source, it was determined that re-producing the functionality would be an unnecessary delay and the fact that the library includes support for multiple types of thermocouple aids in future expandability. The formulas and numbers used are based on those available from the National Institute of Standards and Technology. Due to the fact that the library was published under the MIT license, there're no potential issues with licensing to take into account.

Essentially, the thermocouple operations written simplify the process of using the open source library, performing the calculations required for coldside compensation. Library available at: https://github.com/andygock/Thermocouple

4.5. Room for Improvement

The datalogger software created for the project functions as requested, however, it is far from perfect. As with any project or product, there is room for improvement that could be achieved with another iteration. In the case of the datalogger software, it became clear that several choices made at the beginning of the project were less than ideal. For example, the decision to use Tkinter was made under the assumption that Safera wanted to avoid potential issues with licensing. In practice, however, when speaking with the representative later on during the course of the project it became clear that this assumption was false. Because of the rather rudimentary nature of Tkinter, replacing it with something such as QT in future iterations would be advisable. QT is both more advanced in terms of available functionality as well as being visually cleaner and more modern. Another change that could be made is to utilize the new Python asyncio module. It would, work far better for the purposes of the software while also being sleeker in terms of execution. There're also other small mistakes and optimization changes that could have been made had there been enough time.

5. Camera Recording and Other Sensors

The camera and other sensors were something of a side objective for the project. During one of the initial meetings with Safera and the project group it was proposed that various sensors could be tested and integrated with the Safera measurement system. To this end, two members of the project group worked on this task. To this end, they researched various camera solutions, eventually settling on a standard webcam as well as various air particulate counter sensors.

6. Reflection of the Project

6.1. Reaching objective

Fairly early on it was determined that the project plan was rather extensive and that in reality we would not have time to complete everything required. After this realization, during one of the weekly meetings we discussed what we believed to be achievable and what parts of the plan should be skipped. We concluded that the plan of using a TCP client server architecture was excessively complex and that it would be better to simply focus on the sensors and related software assuming they would be connected to the computer instead of to a hub. This was determined to be a more realistic goal to the project than the one initially proposed. After this meeting the project plan was revised, at least the one the group was working off of, in order to reflect this change.

Although the project scope was reduced to one that was much more achievable, that does not mean there were no delays. In addition to delays due to miscommunication, out of course obligations and other such matters, there were several times when progress hit a brick wall due to a problem that could not be fixed in a timely manner. Towards the end of the project these delays simply built up, leading to the group being unable to complete the project.

The reasons for these delays stem from a variety of factors. The first of these is that the research done at the beginning of the project was insufficient. This led to issues with both the software and hardware. The one that led to the project goal not being met was related to the SPI bus of the ADC. Another factor that led to this was that the group had forgotten that Safera had offered assistance.

6.2. Timetable

The timetable estimated during the initial planning stage of the project proved to be quite optimistic. Because we did not know how to properly estimate how long it would take to complete various tasks it was difficult for us to give a more accurate estimate. Originally the plan stated that the software and hardware aspects would, at the latest, be integrated almost a week before the gala. In practice the group was working right up until the last moment.

6.3. Risk Analysis

The risk analysis performed for the project proved to be quite accurate. During the initial planning stage of the project, when asked to analyze potential risks to the project the risks the group believed most serious were project delays, budget overflow issues or a team member being unavailable. While the group faced absolutely no issues with the budget and some issues with team members being unavailable, the group correctly judged that the largest risk to the project came from delays and missed deadlines. It was also ranked as the highest probability and the most critical in terms of issues it could cause.

During the initial project plan the group proposed that delays could be avoided by having regular meetings as well as by providing assistance to those working on work packages which were not making any progress. During the project this was, however, not nearly as easy as the group had initially thought. Because of how group members were split, each working on their respective work package, it proved difficult to provide assistance. Due to this split, each member had more extensive information of their own subject and only basic knowledge of the rest. Because the issues faced by group members were often quite specific or related to an advanced topic, bringing another group member up to the same level of knowledge regarding the work package that was facing problems would've only caused further delays. This was an aspect of the issue that the group had not originally foreseen.

7. Discussion and Conclusions

As a preface to this chapter, the Protocamp project website available at http://protopaja.aalto.fi/protopaja-2018/safera-2018/ contains a more freeform article reflecting on mistakes made during the project and what can be learned from them. It is less formal than the report, however covers much of the same ground as this chapter.

Even though the project failed to meet the definitions for success set by the group during the initial planning stages, it served to teach members of the group several important lesson. It is something of a given that the project team learned skills related to their work packages. On top of that, however, the team members learned practical skills regarding group work as well as some of the realities of situations that might arise. In addition to this, the team developed a good idea of how to react to various types of situations that a project might encounter.

I can be easy to simply say "In this situation, do this", but in practice, it can be hard to put contingency plans into effect. This is something that the group learned the hard way, but that also guarantees that the lesson will stick. For example, if the project requires each group member to work on a specific aspect but one falls behind, how do you deal with that situation? Even in the group's initial plan it was stated that members of the team would assist. In practice, however, this is not always the case. There is a saying which states "What one programmer can do in a month, two programmers can do in two months." From the experiences gathered during this project, the team has, learned ways of mitigating this issue. For example, maintaining a centralized knowledge base can allow for quicker transition between topics. At the beginning of the project none of the group members realized this was an option or something that might have to be considered.

In addition to this, the group learned about how various project management practices function in a practical situation. In addition to this, there was discussion of how the project could've been managed more effectively. Another aspect that the team learned about is the importance of thorough research. Learning from this experience has better prepared the team for future projects.

For a more detailed perspective, please refer to the post-mortem article mentioned at the beginning of the chapter.

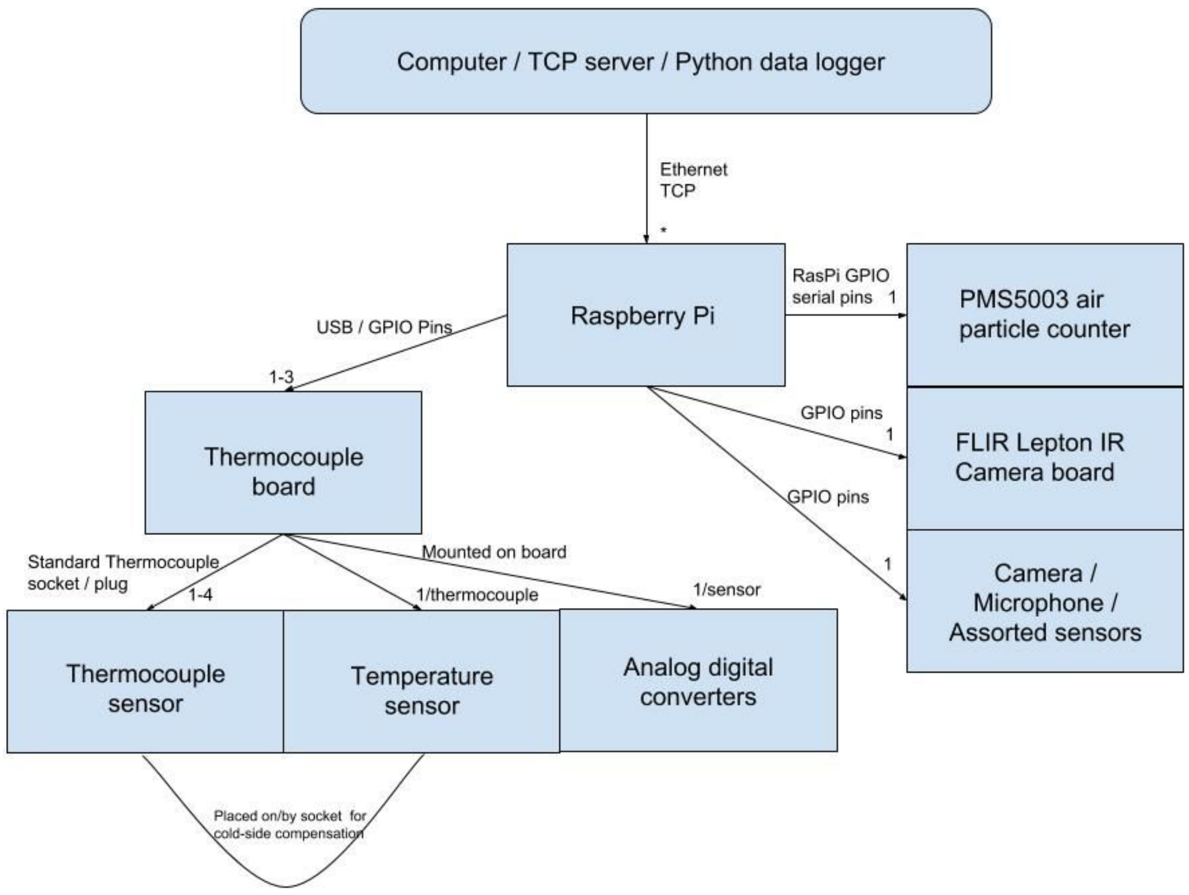
List of Appendices / Liitteet

- Safera Group Project Plan (Not attached, can be found on mycourses and wiki)
- Updated Structure Model (Attached at the end)
- Project Budget and Purchases (Not attached for reasons. Can be accessed by course personnel on wiki.)
- Meeting Notes (Not attached, available on the wiki)

References

- Laurila, Heikki. 2017. *Thermocouple Cold (Reference) Junction Compensation*. Beamex. Accessed on 24.07.2018. Accessed at: {https://blog.beamex.com/thermocouple-cold-junction-compensation}
- Unknown Author. 2018. *Using Thermocouples in Temperature Measurements*. Omega. Accessed on 25.07.2018. Accessed at: {https://www.omega.com/prodinfo/thermocouples.html}.
- Thermocouple library by Andy Gock found at: https://github.com/andygock/Thermocouple
- PySerial found at: https://pythonhosted.org/pyserial/

Appendix 1:


Computer / TCP server / Python data logger
Ethernet
TCP
*
Raspberry Pi
RasPi GPIO
serial pins 1
PMS5003 air particle counter
USB / GPIO Pins
1-3
Thermocouple board
GPIO pins
1
FLIR Lepton IR Camera board
GPIO pins
1
Camera / Microphone / Assorted sensors
Standard Thermocouple socket / plug
1-4
Mounted on board
1/thermocouple
1/sensor
Thermocouple sensor
Temperature sensor
Analog digital converters
Placed on/by socket for cold-side compensation