Facultatea de Electronică,
Telecomunicații și
Tehnologia Informației

Denisa-Ionela PETRARU

PROIECT DE DIPLOMĂ

Smart tiltmeter using image sensors

SPECIALIZAREA ELECTRONICĂ APLICATĂ

Marchează dacă ai studiat în limba engleză: ÎN LIMBA ENGLEZĂ

☑

FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII
ȘI TEHNOLOGIA INFORMAȚIEI
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

2024

Str. George Barițiu, nr. 26-28, 400027, Cluj-Napoca, tel: 0264-401224, tel/fax: 0264-591689, etti.utcluj.ro

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA
FACULTATEA DE ELECTRONICĂ
TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI
Specializarea: Electronică Aplicată (în limba engleză)

Smart tiltmeter using image sensors

Proiect de diplomă

Absolvent,
Petraru Denisa-Ionela

Decan,
Prof.dr.ing. Ovidiu POP

Președinte comisie,
Prof.dr.ing. Dorin PETREU

2024

Această pagină a fost lăsată goală în mod intenționat.

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA
FACULTATEA DE ELECTRONICĂ
TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

Departamentul Electronică Aplicată

Titlul proiectului de diplomă:

Smart tiltmeter using image sensors

Descrierea temei:

This thesis paper showcases a tangible model for monitoring the exact position of a pendulum wire on x and y axes, fit to detect horizontal movements, meant to be used in tracking shifts in position of dams and otherwise large structures. The placement of the wire is detected based on the shadow it leaves behind over two image sensors disposed orthogonally. The electronic ensemble is constructed on a printed circuit board, responsible for connecting the sensors, the microcontroller and the means of communication. A proper control of the system is assured by programming the microcontroller peripherals. For increased precision, digital filtering methods are applied, a comparison between the median and the Savitzky-Golay filters being realized. The performance of the final product can be observed through a Python graphical user interface, capable to display the processed signal for a selected axis and automatically calculate the position of the pendulum.

Locul de realizare:

Universitatea Tehnică din Cluj-Napoca

Adaugă consultant: ☐

Data emiterii temei: 02.10.2023

Data predării temei: 10.07.2024

Absolvent,

Petraru Denisa-Ionela

Director departament,

Prof.dr.ing. Dorin PETREUȘ

Conducător,

Conf.dr.ing. Septimiu POP

Această pagină a fost lăsată goală în mod intenționat.

Absolvent: **Petraru Denisa-Ionela**
Conducător: **Conf.dr.ing. Septimiu POP**

SINTEZA PROIECTULUI DE DIPLOMĂ

Smart tiltmeter using image sensors

The presented thesis paper showcases a tangible model for tracking the exact position of a pendulum wire on x and y axes, fit to detect horizontal movements, meant to be used in tracking shifts in position of dams and otherwise large structures for maintaining their structural integrity. The placement of the wire is detected based on the shadow it leaves behind over two image sensors disposed orthogonally. The source of light for each axis is represented by a LED, whose divergent light rays are converted into parallel ones through a Fresnel lens. The electronic ensemble is constructed on a printed circuit board, responsible for connecting the sensors, the microcontroller and the means of communication. The control of the system is assured by programming the microcontroller peripherals, using a timer for periodically processing measurements, the internal analog comparator for digitalizing the signals, and the reception interrupt for serial data transmission once a specific character is received. The innovation brought by this prototype consists in the digital filtering methods applied, offering more precision. The performance of the final product can be observed through a Python graphical user interface, capable to display the processed signal for a selected axis and automatically calculate the position of the pendulum. The measurement results are observed to be consistent, with minimal errors.

Inclinometru inteligent utilizând senzori de imagine

Teza prezentată demonstrează un model tangibil pentru urmărirea poziției exacte a unui fir de pendul pe axele x și y, potrivit pentru detectarea mișcărilor orizontale, destinat utilizării în urmărirea schimbărilor de poziție ale barajelor și altor structuri mari pentru menținerea integrității lor structurale. Poziția firului este detectată pe baza umbrei lăsate de acesta peste doi senzori de imagine dispuși ortogonal. Sursa de lumină este reprezentată de un LED, ale cărui raze de lumină divergente sunt convertite în raze paralele printr-o lentilă Fresnel. Ansamblul electronic este construit pe o placă de circuit imprimat, responsabilă pentru conectarea senzorilor, a microcontrolerului și a mijloacelor de comunicație. Controlul sistemului este asigurat prin programarea perifericelor microcontrolerului, folosind un timer pentru procesarea periodică a măsurătorilor, comparatorul analogic intern pentru digitalizarea semnalelor și întreruperea de recepție pentru transmiterea serială a datelor odată ce un caracter specific este primit. Inovația adusă de acest prototip constă în metodele de filtrare digitală aplicate, oferind o precizie crescută. Performanța produsului final poate fi observată printr-o interfață grafică Python, capabilă să afișeze semnalul procesat pentru o axă selectată și să calculeze automat poziția pendulului. Rezultatele măsurătorilor sunt observate a fi consistente, cu erori minime.

Avizul conducătorului
Conducător,

Absolvent, Petraru

Contents

1 Rezumat în limba română

1.1 Introducere

Firul de pendul joacă un rol important în monitorizarea structurilor masive, fiind proiectat astfel încât să detecteze orice mică modificare față de poziția lor ideală în plan vertical. Conceptul de bază al sistemelor cu telependul este să urmărească deviația deplasării prin poziția firului de plumb perpendicular pe planul descris de sol. Această variație este menită să fie transpusă într-o formă de semnal electric care poate fi măsurată cu ușurință. Chiar dacă sistemul prezentat în această lucrare are scopul de a fi montat pentru mentenanța barajelor, acest tip de măsurătoare poate fi abordat într-o varietate extinsă de domenii, precum seismologia [1], monitorizarea erupțiilor vulcanice și a sistemului solar.

Inclinometrele de precizie sunt într-o continuă dezvoltare in ceea ce privește tehnologiile folosite si performanțele acestora. Primele astfel de dispozitive foloseau tehnicile inductive și capacitive, dar în literatură a ajuns să prindă amploare metoda optică, bucurându-se de atenție datorită imunității crescute față de factorii de mediu și a preciziei ridicate. Acest sistem este bazat pe preluarea informației printr-un senzor optic, care are ca scop cuantificarea cantității de lumină recepționată într-o formă de semnal electric măsurabil. In cazul dispozitivelor bazate pe aceasta tehnologie, poziția firului de plumb este determinată în funcție de umbra lăsată de acesta pe planul descris de senzor.

1.2 Configurație optică

Lumea optica se învârte în jurul unui element important, lumina. Pentru determinarea completă a întregului ansamblu optic folosit, începutul trebuie sa fie determinat de acest element, ți anume sursa de lumină folosită. In domeniul electronic, sursa uzuala de lumină este reprezentată de LED-uri, diode emițătoare de lumină.

Razele de lumină emise de aceste dispozitive au o tendință divergentă, care afecteaza inevitabil poziția umbrei firului. Pentru evitarea acestui fenomen, o altă forma de dispozitiv optic, si anume o lentilă. Lentilele Fresnel folosite au rolul de a transforma razele de lumină divergente în raze paralele, ideal perpendiculare pe senzorul de imagine folosit. Astfel umbra lăsată de pendul este una fixă, potrivită pentru a fi detectată.

Senzorul de imagine pe care se bazează acest prototip este unul de tip CCD. Aceste dispozitive reprezintă module electronice proiectate cu scopul de a transfera încărcările electrice, generate prin prezența luminii, preluate de fiecare celulă, în mod serial către un amplificator pentru ca tensiunile aferente să poată fi măsurate. Dispozitivul specific utilizat in aceasta aplicatie este un astfel de sensor liniar, cu pixelii dispuși în forma a două secțiuni orizontale continue. Datele sunt preluate de câte un amplificator separat pentru fiecare secție. Pentru ca semnalul să fie preluat în întregime, senzorul necesită o conexiune serială a pinilor.

Pentru funcționarea dispozitivului, anumite semnale trebuie generate si propagate pinilor aferenți. Aceste semnale sunt cel de ceas, cu o frecvență maxima de 8MHz, și intrarea serială, care indică începutul fiecărui set de date. Pentru generarea lor este folosit un sistem cu microcontroller, programat corespunzător.

Acest prototip propune măsurarea poziției pe cele doua axe ale planului, x și y. Rezultatul dorit este atins prin includerea a două sisteme formate din sursă de lumină, lentilă și senzor, pentru detectarea în ambele direcții a umbrei firului. Anumite erori pot fi cauzate de poziționarea eronată a elementelor, fiind de dorit o perpendicularitate perfectă între ele.

1.3 Proiectarea sistemului integrat

Întregul sistem este organizat sub formă de blocuri funcționale responsabile pentru câte un element major în funcționarea ansamblului. Acestea sunt dispuse sub formă de PCB (Printed Circuit Board), dispusă în Figura 1.1.

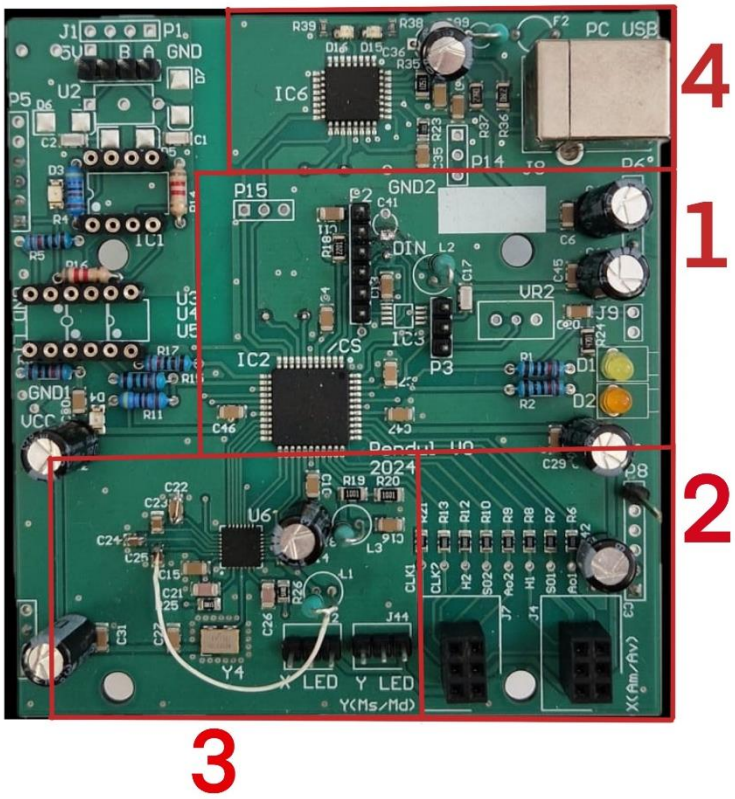

Figura 1.1. Blocurile funcționale ale circuitului

Primul bloc reprezintă ansamblul dedicat microcontrollerului, acesta fiind componenta principală, IC2. Dispozitivul este programat prin intermediul pinilor din conectorul P2, cu codul aferent creat in platforma Microchip Studio, scris în limbajul de programare C. Acest proiect este dependent de funcționalitățile perifericelor microcontrollerului, folosindu-se de timerele integrate, de comparatorul analogic intern pentru digitalizarea semnalului și de portul serial pentru comunicarea cu calculatorul. Pentru funcționarea lor adecvată, acestea trebuie inițializate corespunzător, prin programare la nivel de bit a regiștrilor de control aferenți. Functionalitățile lor sunt apelate prin întreruperi, generate odată cu realizarea unei anumite acțiuni. În cazul timerului, întreruperea se activează odata cu ajungerea numărătorii acestuia la valoarea setată precedent, iar pentru portul UART este folosită întreruperea de recepție.

Blocul al doilea este dedicat preluării și transmiterii semnalelor aferente celor doi senzori. Fiecare linie de semnal are câte o rezistență aferentă, propusă ca metodă de limitare a curentului. Semnalele sunt conectate cu anumiți pini ai microcontrollerului, care sunt corespunzător inițializați drept pini de intrare sau de ieșire, prin programarea biților corespunzători în regiștrii de control ai porturilor. De asemenea, aici sunt prezenți si pinii de control ai ledurilor, prin doi conectori cu câte trei pini, unul de masă, unul de alimentare și cel de control.

Al treilea bloc incorporează elementele necesare ansamblului de conversie analog digitală a semnalelor de ieșire ale senzorului. Acesta este un integrat de înaltă viteză, care funcționează pe doua canale pentru preluarea informației de pe ambii senzori. Datele sunt transmise paralel, prin intermediul a 8 biți de ieșire. Canalul de pe care sunt transmise datele poate fi observat din semnalul

de ieșire al dispozitivului, A/B, care indică nivel logic zero pentru canalul A și nivel logic unu pentru celălalt canal.

Blocul cu numărul patru asigură comunicarea întregului dispozitiv cu exteriorul, reprezentând o cale de conecție cu calculatorul. Portul USB asigură aceasta conexiune, iar circuitul integrat RS232BL realizează o conversie între protocolul de comunicare USB și UART, care este interpretat de microcontroller. Pachetele USB sunt convertite intern în semnalele de recepție și transmisie, RX și TX. Lucrează la o rată de baud de 9600 și este programat astfel încât la recepția caracterelor ce indică axele x sau y, să transmită serial sub formă de șir de date valorile ieșirii comparatorului pentru reprezentarea semnalului generat de senzor.

1.4 Programarea microcontrollerului

După asamblarea componentelor, urmează controlul lor prin intermediul microcontollerului. Acesta este programat conform automatului secvențial prezentat în Figura 1.2., conținând două bucle ce funcționează simultan. Bucla din partea stângă se ocupă de preluarea informației de pe cei doi senzori o dată pe secundă, primul senzor interceptat fiind cel responsabil pentru axa x, iar al doilea pentru axa y. Bucla din partea dreaptă asigură comunicarea serială, prin citirea caracterului recepționat, ca mai apoi să trimită șirul de valori binare ale pixelilor către calculator pentru a fi procesate. Pașii pentru inițializarea și programarea perifericelor necesare sunt prezentați treptat în secția anterioară, descriși pe blocuri funcționale.

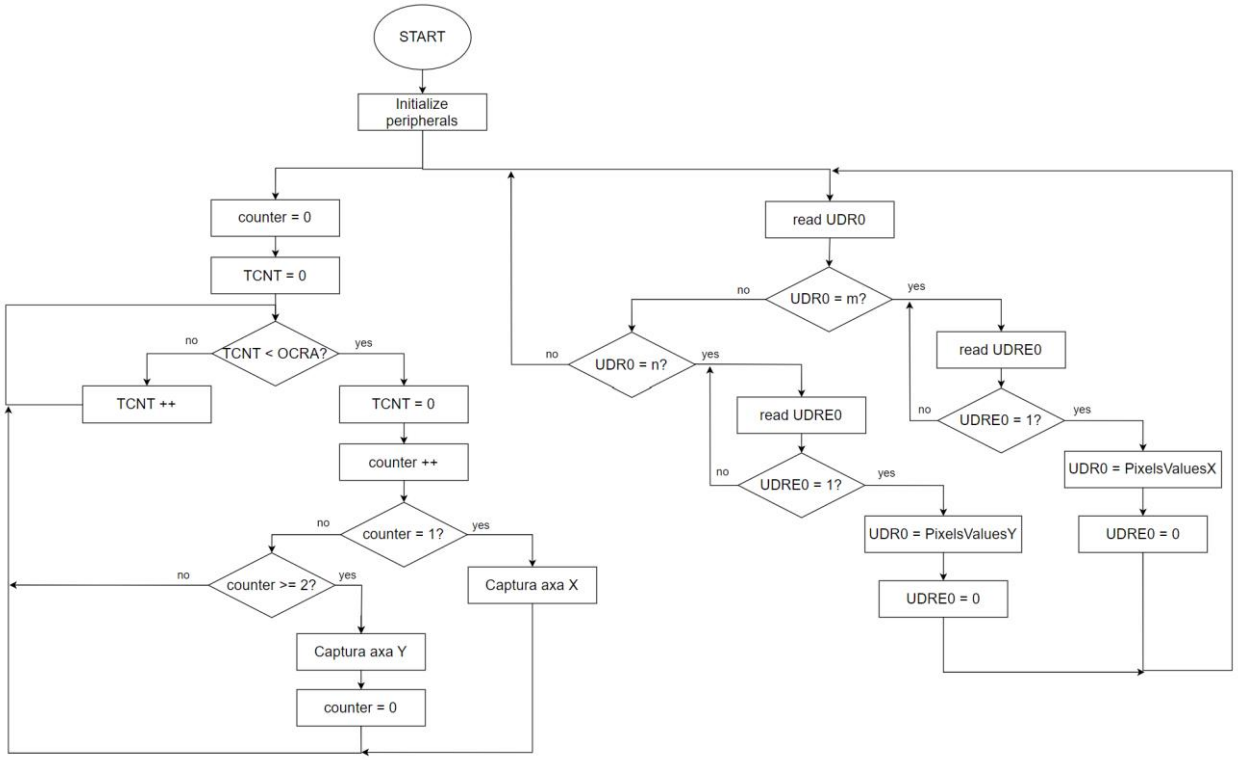

Figura 1.2. Automatul secvențial al programului

Aceste semnale sunt profund afectate de zgomot, necesitând filtrare pentru a putea citi cu acuratețe poziția firului. Filtrarea este realizată în primul rand prin comparatorul analogic, care scapă de agomotul cauzat de efectul de margine al blocurilor de pixeli, și in al doilea rând prin metode de prelucrare digitală a semnalului, aplicate printr-o interfață ghidata de utilizator realizata folosind limbajul de programare Python.

1.5 Procesare software

Metodele de filtrare alese sunt filtrul median și filtrul Savitzky-Golay, ambele fiind bazate pe metoda ferestrei glisante. Deși bazate pe aceleași principii, aceste modalități abordează tehnici diferite. Filtrul median face o aproximare bazată pe valoarea mediană dintr-o fereastră, valoare pentru care jumătate din numerele prezente în interval sunt mai mici și cealaltă jumătate mai mari. Filtrul Savitzky-Golay face o aproximare a unei funcții de grad superior.

Primele teste de procesare de semnal au fost realizate prin date preluate sub formă de fișier csv, generat de plăcile experimentale cu module de osciloscop integrate folosite, ADALM2000 and Digilent Electronics Explorer. Pentru integrarea sistemului în mediul Python, s-a folosit ulterior comunicarea serială prin portul USB. Platforma de programare Spyder, are capabilitatea de a trimite și recepționa date serial, utilizând librăria *serial*. Pentru selecția senzorului care urmează să fie citit, programul trimite anumite caractere specifice interpretate de microcontroller.

Pentru ușurința de manipulare a programului de către utilizator, a fost implementată o interfață grafică, ilustrată în Figura 1.3. Toate acțiunile necesare sunt realizate prin butoane, fiecare aducând un nou element vizual în fereastră. Butoanele de start citire ale fiecarei axe trimit caracterele necesare către microcontroller, și vin însoțite de o fereastră noua care indică începutul citirii datelor. Butoanele *Original* și *Filtered* genereaza grafice ale semnalului în funcție de dorința utilizatorului. Acestea pot genera o fereastră de eroare în cazul în care nici o citire nu a fost realizată.

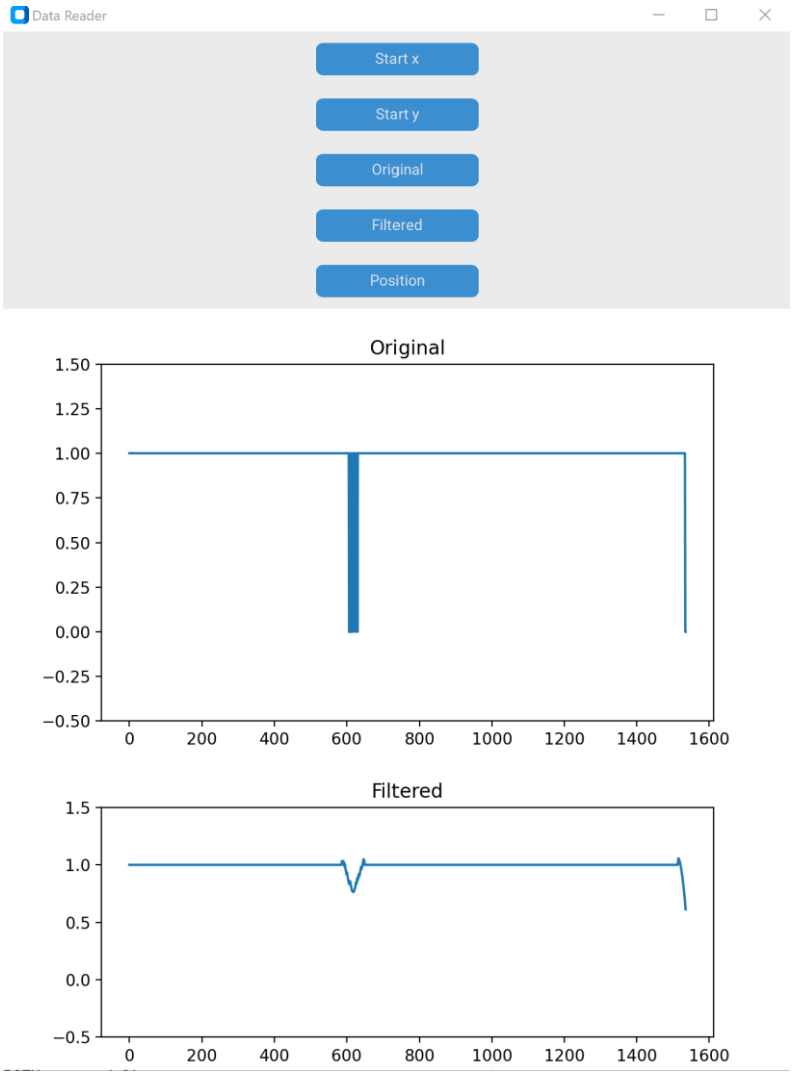

Figura 1.3. Prezentarea interfeței

Scopul acestei lucrări este reprezentat de ultimul buton, *Positon*, care determină poziția firului de pendul bazându-se pe forma semnalului filtrat. Aceasta este afișată într-o fereastră separată și este reprezentată în mm. În urma a multiple teste realizate pentru aceeași pozitie a obiectului, s-a putut observa consecvența măsurătorilor, care au o eroare maximă de 0.0635 mm, ce reprezintă lățimea unui pixel al senzorului.

1.6 Concluzii

Această lucrare descrie un prototip complet funcțional de determinare a poziției unui pendul, dedicat mentenanței barajelor. Acesta este realizat în trei pași majori, proiectarea circuitului pe un cablaj PCB, programarea microcontrollerului și procesarea semnalelor prin intermediul unei interfețe Python. Modelul creat este capabil să ofere măsurători consecvente, de o acuratețe ridicată.

Anumite îmbunătățiri pot fi aduse ulterior ansamblului electric, prin programarea blocului de conversie analog numerică. În ceea ce privește interfața grafică, aceasta poate fi dezvoltată pentru a prelua date continuu, fară a necesita apăsarea unui buton, si compararea acestora cu seturile preluate precedent, cu scopul de a indica erori cand măsurătorile ajung să fie puternic deplasate.

Acest sistem este doar începutul unei măsurători complete, urmând să fie integrat într-o întreagă rețea de senzori responsabili pentru mentenanța barajelor. Comunicarea în această rețea rămâne un factor de implementat în viitor.

2 Work planning

Activity number	Activity	Start date	End date
1	Theme choosing	13.10.2023	16.10.2023
2	Objectives definition	17.10.2023	23.10.2023
3	Bibliographic study	24.10.2023	16.11.2023
4	Sketching the implementation steps	16.11.2023	30.11.2023
5	Designing and testing the optical setup	01.12.2023	13.01.2024
6	PCB implementation	14.01.2024	31.01.2024
7	Programming the microcontroller	01.02.2024	20.02.2024
8	Python implementation	21.02.2024	17.03.2024
9	Documenting the teoretical fundamentals	18.03.2024	10.04.2024
10	Documenting the implementation	11.04.2024	01.05.2024
11	SSET documentation	02.05.2024	20.05.2024
12	Prorotype improvement	24.05.2024	15.06.2024
13	Finishing the documentation	16.06.2024	01.07.2024

3 State of the art

3.1 Introduction

Tiltmeters have a crucial role in detecting minuscule changes in the position of different objects. They are used for tracking the deviation of inclination through different measuring methods. The technique approached in this study is based on monitoring a pendulum hung perpendicular to the ground. The variation of movement is meant to be translated into a form of electrical signal. Tiltmeters have a wide range of applications, including seismology, monitoring volcano eruptions, human body motion [2], solar system and certain movements of large structures in general, such as historical heritages and dams. This prototype will be used for the last category, water dams.

As shown in Figure 3.1, vertical tiltmeters consist of many elements, being somewhat of a large structure. The mean of measurement is presented as the 5th element, used for detecting the pendulum's position.

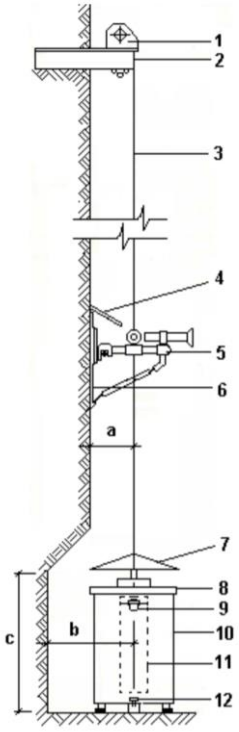

Figure 3.1. Schematic of a pendulum tiltmeter

One of the biggest companies producing such instruments is Sisgeo, the global leader of this market. As an illustration of the use of this system, it is to be mentioned that Sisgeo was responsible for monitoring the most important historical heritage in Rome, during the construction of the Rome metro lines going right underneath them. They attached distinct types of inclinometers to the Colosseum and the Temple of Venus.

3.2 Methods used in literature

Precision tiltmeters have been continuously growing in terms of technologies and performance, and their improvement is still ongoing. The technologies used are upgrading, from the capacitive

and the inductive methods, to the optical method, which is starting to stand out through higher precision and immunity to environmental conditions.

3.2.1 Inductive method

Tiltmeters based on the pendulum method, also known as telependulums, can be implemented with inductive transducers. Their purpose is to convert the wire displacement into frequency values.

Two coils need to be used for measuring the position on each axis. As shown in Figure 3.2, both coils are connected to an oscillator circuit and the microcontroller to measure the resonance frequency of the output signal. For protecting the coils from the electromagnetic influence of the steel wire, a paramagnetic target is placed on the pendulum, which affects inductance of the coils. The position on one axis is determined through a linearization function using the difference between the frequencies given by the previous step [3].

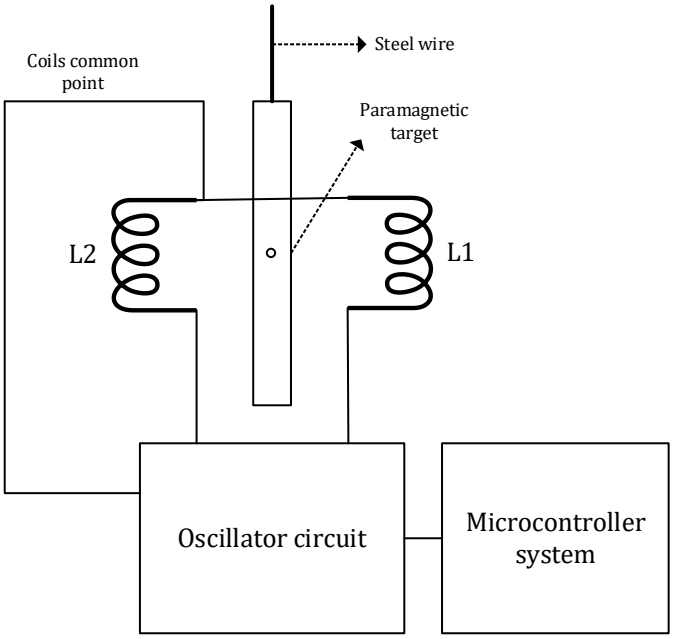

Figure 3.2. Inductive telependulum schematic

3.2.2 Capacitive method

Capacitive sensing is a widely used technique in different domains. Regarding tilt sensors, most of their mentions in literature are of capacitive type, having the possibility to obtain analog and linear outputs based on the tilt angle.

As a sub-category of the capacitive tiltmeters, capacitive fluidic sensing has been broadly studied in many literature works, although mostly using a structure that can only achieve single-axis sensing [4], [5]. The basics of this technique usually stand in using either parallel electrodes or annular coplanar electrodes, the latter having a better performance. The annular structure consists in replacing the conventional rectangular electrodes with concentric ones, achieving a larger sensing area. Between the concentric annular capacitive plates, two different dielectric materials are used, air and liquid. The change in the liquid's contact area with the capacitive plates is in relation to the tilt angle. In [5], the Hankel transform method was used, calculating the capacitance after dividing the conductive plates into circular filaments and accumulating each charge distribution. The circular capacitive channel presented in [4] contains two capacitors used in differential mode, for detecting

the direction of the tilt while also minimizing the parasitic effects. An accurate visual representation of the annular coplanar prototype is shown in Figure 3.3.

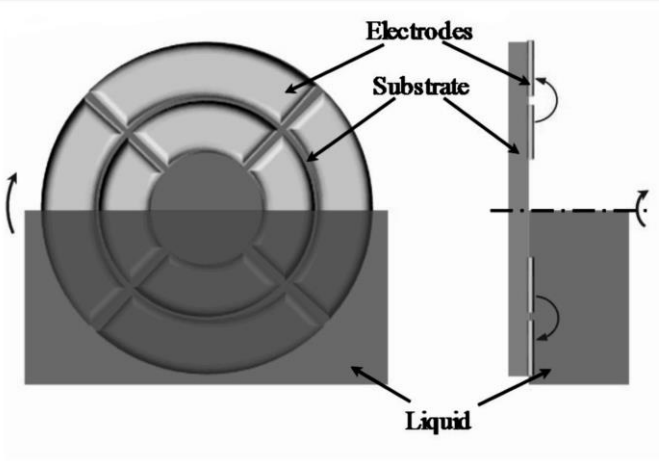

Figure 3.3. Annular Coplanar Electrodes Capacitive Sensor *[5]*

A different approach of capacitive fluidic sensing can be seen in [6], which proposes a two-axis concept. Using five sensing electrodes, two pairs of capacitors are generated. The change in capacitance is also given by the medium that surrounds the electrodes.

The capacitive technique is also used in [2], which is based on comb drive capacitors. The device concept uses suspension beams of different flexibility and a central mass that moves towards one of the electrodes according to gravity. The capacity is inversely proportional to the distance between the central mass and each electrode. The measurable variable in this case is the differential capacitance between the left and right comb capacitors, which is related to the tilt.

3.2.3 Optical method

There are various possibilities of measuring the tilt of objects using different optical means, some automatic, others that require human operators. The manual ones are based on an instrument named coordiscope, which has a fixed position and requires an operator to monitor the wire position by looking through small telescopes, mounted on both x and y axis.

On the other hand, certain units take automatic measurements, being connected to a data acquisition system based on a microcontroller. These systems are gaining a lot of attention in literature, due to their contactless means of measurement and high immunity to environmental noises. There is a broad range of sensing equipment that can be employed in such prototypes, from image sensors to optical fiber gratings.

A Fiber Bragg Grating (FBG) represents a periodic and permanent modification of the core refractive index of a photosensitive fiber through a lateral laser illumination [7]. These sensors are used for their sensitivity to external effects such as temperature, strain and pressure, given that these factors influence the Bragg wavelength. In systems focused on measuring inclination, FBG sensors are used for their capability to detect changes in strain. Such a prototype is shown in [8], where the fiber is connected at both ends with cylindrical floats and the entire system suspended in water. With inclination, the water surface, and the depth of the liquid in certain points, straining the fiber.

Another type of automatic optical system is based on image sensors, which have as purpose transforming the quantity of light rays detected into electrical signals. These sensors are mainly used in photography and videography related applications. The most used devices are the charge-coupled device (CCD) and the complementary metal oxide semiconductors (CMOS), both having their working principle based on pixels, while the main difference is the way they interpret the signal stored in each cell. The CMOS technology can assure random pixel access, as opposed to the CCD

sensor, which processes the information taken from a whole array of photodiodes through a charge amplifier.

3.3 Conclusion

After analyzing all the technologies previously mentioned, a decision was necessary. The principle of measurement used in this scientific paper is based on monitoring the shadow of a vertical pendulum by using CCD linear arrays. This type of device has a certain range of advantages that make them suitable for this application: the high response time, contactless measurement, and high immunity to noise.

The proposed system consists of an acquisition element of a larger network, suitable for automatic measurements. The core of this prototype is the microcontroller, which integrates all the elements used for communication, as well as data acquisition.

4 Theoretical fundamentals

4.1 Proposed tilt measurement method

As discussed previously, there are various methods throughout history and literature used in measuring tilt. Optical methods stand out through their contactless measurement, making them a new attraction in the industry. In this chapter, the chosen technology will be studied in more depth in order to understand the principles of this prototype.

The wanted result is an automatic model that senses the tilt of a pendulum wire using image sensors. Image sensors detect the shadow left by the wire with increased precision, in order to detect the smallest of changes in its position. This setup will two such sensors for measuring along both x and y axis.

4.1.1 Optical Setup

The optical world revolves around light. To start the setup idea, an appropriate source of light is needed. The simplest, yet the most used light source in electrical engineering is represented by nothing else but the LED.

The most known semiconductor device would be the diode, that basically represents a switch, either letting current pass in only one sense, while restricting its flow in the other. A light-emitting diode (LED) works based on the same principle, but generating light when turned on. The light emission needs a semiconductor die, made from certain materials that allow the energy released from the recombination of electrons with electron holes to be in the form of photons.

However, these light emitting devices generate divergent light rays, that are moving away from each other at certain angles. This represents a disadvantage for the prototype presented, making it more difficult to detect the exact position of the shadow. This problem is easily solved using another optical device, the Fresnel lens.

Fresnel lenses are used in a large range of applications, having many functionalities, from the most common one, light gathering, to magnification and projection in different illumination systems. This prototype will take advantage of their light collimation property, used to transform the divergent light rays into parallel ones, as presented in Figure 4.1.

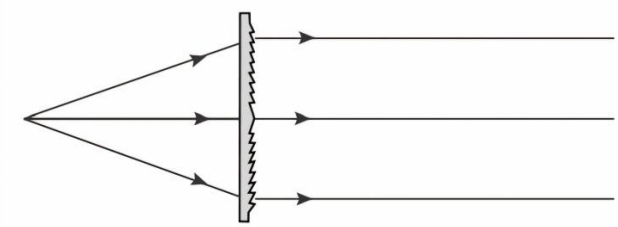

Figure 4.1. Light collimation using Fresnel lens *[9]*

As a result, the shadow of the pendulum wire will be detected with accuracy by generating small bursts of parallel right rays that are perpendicular on the opposite sensor. Nonetheless, some errors can appear, given that the angles between the lenses can have small deviations from the ideal ninety degree angle.

This prototype proposes measurement on both x and y axis, whereas needing a system with two perpendicular sensor arrays, the pendulum being orthogonal to the plane described by the sensors.

The main component used for this setup is the TSL1412S linear sensor array, consisting of two rows of photodiodes with a different charge amplifier circuitry. It consists of 1536 pixels, represented by photodiodes, divided in two sections. The light intensity sensed by each pixel induces

a certain photocurrent that is processed by the integration circuitry of the pixel's corresponding section, as shown in Figure 4.2.

This sensor provides certain functions, such as the hold operation, reset logic and output control. The hold function ensures of the simultaneous integration start and stop time of all pixels. The shift register assures the output control of each section, as well as the reset logic.

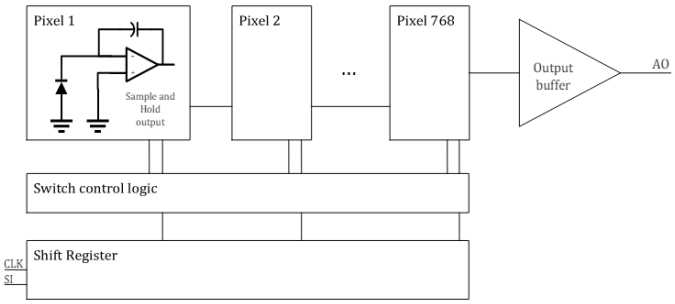

Figure 4.2. Block diagram of one section of TSL1412S

For the sensors to detect the shadow of the pendulum wire, a source of light is needed. As CCD sensors can be easily saturated by excessive exposure to light, only short light bursts would be fit for this application, result achieved by using an LED, programmed to be turned on after a preset time interval.

The proper integration of the optical setup into the system is realized through a smart system. The only thing that can make any prototype have its own "intelligence" is a microcontroller, that connects every part of the system, also assuring every component's synchronization with one another.

4.1.2 Principles of Optical Sensors

Image sensors, as optical sensors in general, quantify the attenuation of light rays into electrical signals that can be interpreted in diverse ways. The information retrieved by the sensor is conveyed to form an image. They are used in imaging devices, either digital or analog.

The classification of these sensors is divided into the following types: the CCD (charge-coupled device) and the complementary metal oxide semiconductor (CMOS) sensor (active-pixel sensor). Both are based on the same principle, retrieving information in the form of pixels, which consist of metal-oxide-semiconductors (MOS). The major difference between the two is represented by the way they interpret the signal from a given pixel. In contrast to the CC sensors, CMOS can assure random pixel access [10], having an incorporated analog to digital converter for each pixel column and a preamplifier for each pixel, as presented in Figure 4.3.

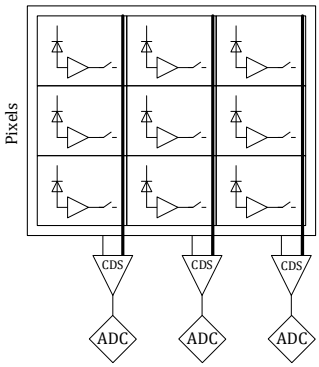

Figure 4.3. Schematic of a CMOS image sensor

However, CMOS sensors have some specific disadvantages that would not make them suitable for this application. Tiltmeters require a high precision, needing to measure very small alterations. This could be affected by their non-linear characteristic. Also, noise levels can be induced on every amplifier circuitry, undesirable for this project. After analysing both technologies, it is to be concluded that the more suitable option for this prototype is the CCD sensor.

4.1.3 CCD Sensors

Charge-coupled devices (CCD) are electronic modules designed to carry electronic charges. They were developed for data storage purposes by Willard Boyle and George Smith in the Bell Laboratories in 1969, later assuring them the Nobel Prize in Physics, in 2009 [11].

The internal photoelectric effect is what causes semiconductors to generate free electrons by exposure to photons, increasing their conductivity [11]. This phenomenon allows the quantification of light intensity into electric charge, that can be transported from every pixel cell through vertical and horizontal shift registers to a main analog to digital converter. The charge is shifted from neighbor to neighbor until reaching the final output buffer that serially processes the information.

Nonetheless, the sensor does not work independently, needing a certain power supply, reference to ground, and control signals for synchronization. For this reason, it must be integrated into a microcontroller system, that handles the data inputs and outputs.

4.2 Microcontroller Systems

A microcontroller is itself a system that incorporates a range of elements used for processing information. The main component of such a system is the microprocessor, an electronic component that handles information through various logical, arithmetic and transfer operations. The term microprocessor indicates the central processor unit (CPU) of a computer comprised in a single integrated circuit.

Besides the core, consisting in the CPU, a microcomputer contains a selection of diverse peripherals that aid in its functions [12]. The minimal ensemble that is usually encountered in such systems contains a data memory, a program memory and input-output modules, but most models contain additional features. Once assembled on a singular silicon chip, such a network is called a microcontroller, component used in every automatically controlled device.

There are many microcontroller types, different in speed, memory and certain other features. Many producers worldwide offer diverse technologies and architectures. Starting from the Intel Corporation, which created the first single-chip microprocessor, to STMicroelectronics, which offer one of the most popular microcontroller devices nowadays, the STM32. However, a lot of attention in the industry is granted to the Microchip products, which started out as Atmel, standing out through their collaboration with one of the most known experimental board producers, Arduino.

Atmel Corporation was a semiconductor manufacturer with expertise in microcontrollers before being acquired by Microchip Technology. They developed the AVR microcontroller family using the modified Harvard Architecture. This technology was created in Trondheim, Norway, in the Norwegian Institute of technology by two of their students [13].

4.2.1 Microcontroller Architecture

There are two main classes of architectures found in microcontrollers, Harvard and Von Neumann, that have different means of data processing. The Von Neumann architecture offers one bus for instructions as well as data, while in the Harvard architecture the two busses are separate, realizing simultaneous transfers. Conversely, in the pure von Neumann architecture can be slower, due to the need of fetching data and instruction at separate timings. The two memory types of the Harvard architecture have no need in sharing the same features. While data memory typically

requires a read-write memory, certain systems have instructions for pre-programmed tasks that are stored in read-only memory. As a result, a machine based on the Harvard architecture has different data and code address reserved spaces [14]. A visual representation of the Harvard architecture is shown in Figure 4.4.

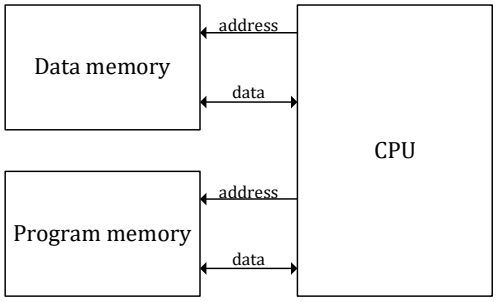

Figure 4.4. Block diagram of the Harvard architecture

A middle ground between the two models appeared in literature, the Modified Harvard architecture, containing features common to Harvard, as well as von Neumann. This architecture is classified into more types, based on the distinctive property that makes it different, also being split into two main categories, the so-called "Almost-Harvard" and "Almost-von-Neumann". The distinctive functions of the Almost-Harvard architecture are access instruction memory as data and the read instructions from data memory, whose names self-explain their functionality of partially combining the program and data memory in different ways. As for the Almost-von-Neumann architecture, it presents itself with cache-related particularities [15].

A variety of popular models implemented using the modified Harvard architecture is found in the AVR family, well known for their ATmega microcontrollers. The AVR architecture is based on the 8-bit RISC (Reduced Instruction Set Computing) microcontroller, having as features IO data space, EEPROM, SRAM and on-chip programmable memory, this family being the first in market to have on-chip flash storage [16]. The model used for this project is ATmega1284, having the structure presented in Figure 4.5.

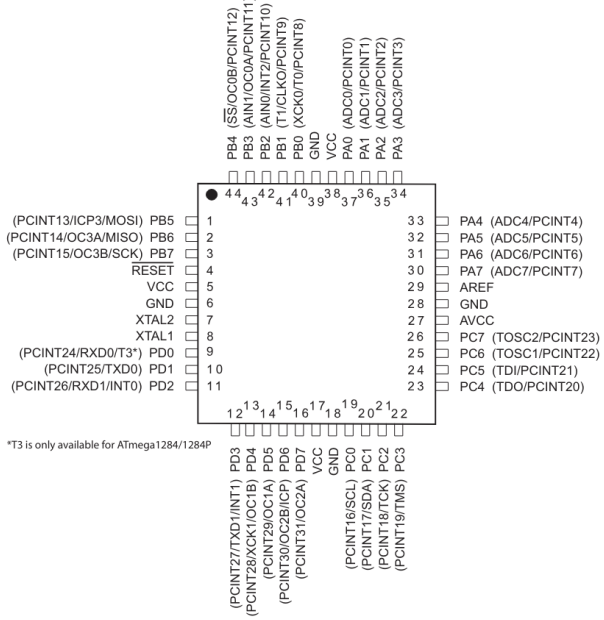

Figure 4.5. ATmega1284 *[17]*

It is equipped with four data ports, PA, PB, PC and PD, each having 8 parallel data bits. Some of them have secondary features, such as serial reception and transmission, and are responsible for

certain registers used in the programming of internal peripherals including the ADC, the comparator and many more. This device is the center piece of the whole embedded system, assuring the connection and control of all the other components.

4.2.2 Embedded System Design

Every embedded system is designed differently for any automatic project. The "brain" of each and every one of them is represented by the microcontroller, which is responsible for integrating all the blocks needed for the application. The microcontroller is also a mean for external communication, transmitting the information taken from the sensors to a user in various ways. Figure 4.6 describes briefly the blocks employed in this particular prototype.

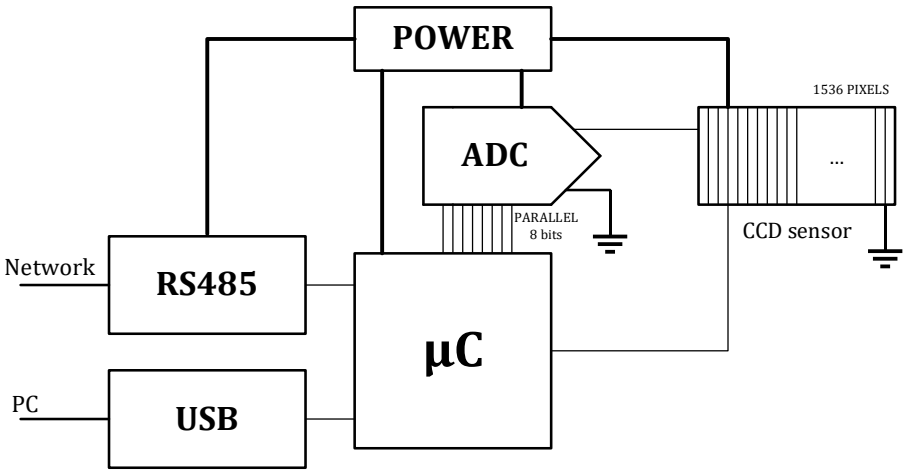

Figure 4.6. Block diagram of the embedded system

The integration of the CCD sensors into the application is realized through two different approaches. The microcontroller processes the analog output signals of the CCD sensors either on its own, through its integrated comparator, or with the help of an analog to digital converter. The information given by the sensor is in the form of analog voltages transmitted serially for each pixel in the array. As light charges a cell, the voltage transmitted increases, resulting in a voltage directly proportional to the quantity of light sensed by each pixel. The final output signal is represented by a series of low and high voltages, depending on where the shadow of the pendulum falls. However, for further processing, this signal needs to be digitalized.

As a first method of digitalization, as the signal mainly works on high and low voltages, with no more nuanced voltage levels of importance, the system employs the internal converter of the microcontroller. By comparing the output signal of the sensor with a constant voltage, the result will be exactly high or exactly low for each pixel.

In this case, the microcontroller needs to be programmed accordingly for the comparator to work in the wanted mode. The register responsible for maneuvering the comparator actions is ACSR (Analog Comparator Control and Status Register), an 8 bit register. It controls the enable function of the comparator, along with its interrupt and mode selection.

A more complex and accurate result can be obtained by using an ADC (analog-to-digital converter). An ADC generates at its output a value proportional to the input analog signal, denoted by {A}, as presented in Figure 4.7**Error! Reference source not found.**.

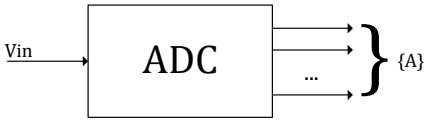

Figure 4.7. ADC succinct diagram

The output value of the ADC, {A} is calculated using the input voltage and the full-scale voltage and is illustrated only by positive numbers less than one. V_{FS} (Full Scale Voltage) represents the value needed at the input for the result to be exactly one. Equation (4.1) depicts how the digital value is calculated.

$$\{A\} = \frac{V_{in}}{V_{FS}} \tag{4.1}$$

The information is then converted into binary sequences of a certain number of bits, depending on the capacity of the converter.

Analog to digital converters have diverse features that can vary from a model to another, in terms of output data transmission, speed, supply voltage needed and many more. The internal converter of the microcontroller isn't fit for this application, having a smaller speed than the wanted one. A main characteristic needed for this prototype was the parallel output data transmission. The parallel technology consists in the chip having more output data pins, in order to transmit all the bits simultaneously, assuring faster transmission. Due to the necessity of integrating two sensors synchronously, the converter needs two channels, one for each sensor. After analyzing all the requirements, the converter chosen for this application if the MAX1193, with the features presented in Table 4.1**Error! Reference source not found.**.

Number of channels	2
Voltage supply	2.7V to 3.6V
Clock Frequency	Typically, 45MHz
Number of bits	8
Voltage reference	Internal/External

Table 4.1. Features of the MAX1193 ADC

It is integrated in a 28 pin thin QFN capsule, represented in Figure 4.8, having 2 pins for the negative and positive inputs of each channel, a channel data indicator that shows which channel is converted, 8 pins for the output bits, and two power logic bits. The power logic bits are the only ones that need to be programmed to work in the normal operating mode. This prototype works in the single-ended input mode, neglecting the negative inputs and with an external voltage reference.

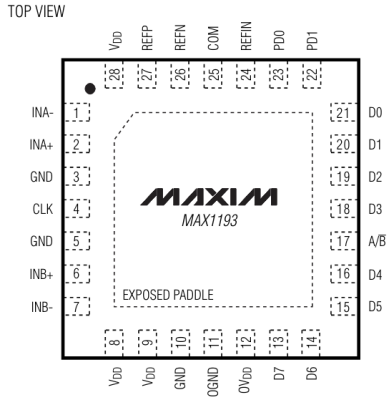

Figure 4.8. Top View of MAX1193 *[18]*

After the microcontroller processes the information from the sensors, it needs to send it to the user through certain means. There are diverse communication protocols that can be encountered in such systems.

4.2.3 Communication Protocols

Industrial applications often demand data transmission from system to system over long distances, called communication protocols. They are of crucial importance in modern automatic systems, having a set of conventions and rules for transmitting and receiving messages from a component to another, in this way assuring coordination and communication throughout the entire ensemble [19].

Embedded systems can employ a large range of communication standards, resulting in a lot of options to decide from depending on the requirements of every application. Some examples worth mentioning are the Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Universal Asynchronous Receiver-Transmitter (UART), Universal Serial Bus (USB) and the Recommended Standard (RS) family.

Starting with the oldest form of communication in embedded systems, the UART is maybe the simplest and most used of all of them. Its implementation is based on asynchronous serial communication, which implies the lack of any type of clock signal, usually employed for the transmitter and receiver to be in sync with one another. To compensate for the asynchronism, data transmission is realized along two wires, while adding start and stop bits for discerning incoming data. This technology requires a single master and a single slave, resulting in one of its greatest disadvantages, the fact that it cannot support multiple systems.

The USB is popular in the computer world, being encountered in every such device through one form or another, as a form of connection between two devices. It is similar to the UART, realizing a form of asynchronous serial communication. It is designed for complex transfers of high-capacity and increased speed, while still standing out through its simplicity and low cost. It is fit for the prototype presented in this paper, having the capability of easily connecting the system to any computer for data processing purposes.

The RS family contains a variety of members, each differing in certain features. All of them are based on serial communication, and the most well-known ones are RS232, RS422 and RS485. A comparison between the three is presented in Table 4.2. Out of all of them, RS485 stands out through the increased number of devices that can be connected, both receivers and transmitters. As this application consists in integrating the prototype into a much bigger acquisition system, this protocol represents a great communication method.

	RS-232	**RS-422**	**RS-485**
Devices number	1 transmitter 1 receiver	1 transmitter 10 receivers	32 transmitters 32 receivers
Maximum distance	15 m	1.2 km	1.2 km
Communication configuration	Full Duplex	Full duplex, Half duplex	Full duplex, Half duplex
Cable type	Single ended	Single ended multi-drop	Single ended multi-drop

Table 4.2. Comparison between properties of the RS protocols (Sonnenberg citation)

Through the FT232BL module, a conversion between USB and UART protocols is realized. This project takes advantage of both their qualities, using the USB as a power supply and the UART for simplicity in communication. The IC contains a specific protocol engine that handles all the necessary layers of the USB, converting data through the RXD (Receive Data) and TXD (Transmit Data) pins, and, reversely, into USB packets fit for the USB host.

In conclusion, the protocols employed in this application are the USB, UART and the RS485. The USB assures a power supply for the ensemble, which is combined with the simplicity of the UART protocol. The RS485 protocol is the mean of integrating this prototype into a larger network of sensors and components. However, any way of communication has its defects and errors that can

be prevented. The output signal of the sensor is observed to usually be very noisy, regardless of the transmission method. Filtering techniques can be applied for reading the sensors as accurately as possible.

4.3 Filtering techniques

Filtering represents the removal of noise, or else unwanted frequency components from a signal, and is considered itself a specific technique of signal processing. Digital signal processing can be used in a variety of engineering domains, from telecommunications up to biomedical engineering. The first step in any processing maneuver is the representation of the original signal, as to better analyze the type of noise encountered in order to choose the right filtering method.

The signal generated by the sensors in this project is mainly disturbed by bursts of high frequency random noise. Besides, some other type of alternating noise can appear caused by the separation of pixel structures. For the proper filtering of these forms of noise, linear filtering variations are approached, based on the sliding window method, due to the previously mentioned difference between pixel structures, that represent intervals of fixed length.

Linear filtering in its initial form consists in computing the average of all the values found inside the chosen window. For linearization, each value is replaced by the average computed, creating an interval of constant amplitude. However, this comes with great disadvantages when dealing with high peak noise, creating erroneous signal levels. This problem is solved using the following techniques.

4.3.1 Median Filtering

Median filtering was implemented as an improvement of the linear filtering [20], being fundamental in maintaining signals with slow variations while also dampening noise. Similar to linear filtering, it replaces the values in the sliding window with one number computed by a certain algorithm, in this case, the mean value, also called the median value. The median value represents the number for which there are an equal number of larger and smaller values through the interval.

Median filtering can be applied on more than one dimension, being used in image processing ($d = 2$) as well as in signal processing ($d = 1$). Where f is a function formed by equispaced samples, degraded by noise, we have the set of values Y_n defined by equation (4.2).

$$Y_n(i) = f\left(\frac{i}{n}\right) + \sigma Z_n(i) \tag{4.2}$$

Here, Z_n is the distributed white noise and σ the noise level, $\sigma > 0$. The original signal that needs to be recovered is represented by the f function. By fixing a window size $l > 0$, median filtering can be applied as described in equation (4.3).

$$M_l[Y_n](i) = Median\{Y_n(j): j \in W[n, l](i)\} \tag{4.3}$$

$W[n, l](i)$ indicates a discrete window of centered at $i \in \{1, \dots, n\}$ of radius nl. It is described by equation (4.4).

$$W[n, l](i) = \{j \in \{1, \dots, n\}: \|j - i\| \leq nl\} \tag{4.4}$$

Certain problems in the case of median filtering can be raised due to the difference between the computed values for each window, creating visible edges.

4.3.2 Savitzky-Golay Filtering

Savitzky-Golay filters, for short SG filters, were introduced in 1964 by Savitzky and Golay [21]. They accomplish signal smoothing as well as noise reduction. It stands out through increasing the signal to noise ratio while maintaining the useful signal uncompromised.

As it is also based on the sliding window method, it requires the selection of $2m + 1$ equidistant points, with the center positioned at $N = 0$. They are employed in representing polynomials of chosen degree n. Using this group of data, the result is given by calculating the least square polynomial.

The least square polynomial method consists in calculating the coefficients of a polynomial of a chosen degree. When each abscissa point is substituted into the equation, the square of the differences between the observed number, y, and the computed number, f, is minimum through the whole observation process, all the error being assumed in the ordinate, while the abscissa stays ideal. However, finding the coefficient only results in the best value at just the central point of the set.

The final purpose is to find the best mean square fit through the $2m + 1$ consecutive values of a polynomial of nth degree, having the from presented in equation (4.5). The derivatives of order x of the function are represented as in equation (4.6). The point $i = 0$ is considered the central point ($i = \{-m, \dots, m\}$). The value at this point for any derivative is given by equation (4.7). As previously mentioned, the least square method implies that the sum of squares of differences between the calculated f_i and the observed y_i is minimum throughout the interval. the descriptive equation of the criterion is illustrated in equation (4.8). With respect to the general b_{nx} the relation from equation (4.9) or (4.10) is obtained. x is the index representing the number of the equation running from 0 to n. Due to the independence of b regarding i, the obtained result is depicted in equations (4.11) and (4.12). It is to be noted that $S_{x+k} = 0$ for odd values of $x + k$. The set of $n + 1$ equations can be conclusively in two sets, one for odd values of k and one for even, simplifying the final computations.

$$f_i = \sum_{k=0}^{k=n} b_{nk} i^k = b_{n0} + b_{n1} i + b_{n2} i^2 + \cdots + b_{nn} i^n \tag{4.5}$$

$$\frac{d^x f_i}{d i_x} = x!\, b_{nx} \tag{4.6}$$

$$\left(\frac{d^x f_i}{d i_x}\right)_{i=0} = x!\, b_{nx} = a_{nx} \tag{4.7}$$

$$\frac{\delta}{\delta b_{nk}} \left[\sum_{i=-m}^{i=m} (f_i - y_i)^2 \right] = 0 \tag{4.8}$$

$$2 \sum_{i=-m}^{i=m} \left[\left(\sum_{k=0}^{k=n} b_{nk} i^k \right) - y_i \right] i^x = 0 \tag{4.9}$$

$$\sum_{i=-m}^{i=m} \sum_{k=0}^{k=n} b_{nk} i^{k+x} = \sum_{i=-m}^{i=m} y_i i^x \tag{4.10}$$

$$\sum_{k=0}^{k=n} b_{nk} \sum_{i=-m}^{i=m} i^{k+x} = \sum_{k=0}^{k=n} y_i i^x = F_k \tag{4.11}$$

$$\sum_{k=0}^{k=n} b_{nk} S_{x+k} = F_k \tag{4.12}$$

4.3.3 Comparative Analysis

Even if the two methods work based on similar principles, they are distinguished by certain features, including speed, shape preservation and performance in general. The main quality that is wanted in this prototype is the speed of processing, in order to observe the processed data almost simultaneously, so the position of the wire is described accurately through time.

The SG filter has a petter performance in preserving the curvatures of the signal, which are missing in this application, due to the digitalization of the signal that made it constructed of sharp edges between low and high. Its process is also challenging relative to the median filter, it implies a larger computation time, which is not wanted.

To conclude, the two methods will be compared throughout the experimental results, although the final product will be based on median filtering, due to its better fit to the project's requirements. Both techniques are to be applied through software mechanisms, implemented using the Python programming language. Python has turned out to be a powerful tool in signal processing, as well as having the ability to communicate serially with the microcontroller through the USB port, which represents a great asset for this prototype.

With the setup complete and the signal as accurate as possible, the hardware assembly is needed. Due to the large quantity of components needed and the need for a stable signal integrity, the prototype is fit to be assembled on a Printed Circuit Board.

4.4 Printed Circuit Board (PCB) Design

The schematic presented in the previously depicted Figure *4.6* needs to be implemented in a compact electrical form, a printed circuit board. The board is designed using the Altium software interface, making it possible to sync the electrical schematic with the PCB assembly itself, while also offering more detailed settings regarding assembly rules and constraints.

4.4.1 PCB Design Fundamentals

A printed circuit board (PCB) consists in a flat plate made of insulating materials that is designed to have a pattern of conducting material for interconnecting its components.

When first designing a PCB, one should take into consideration the approximate number of components and the dimension restrictions to come up with the initial idea of the board shape. Further decisions are to be made regarding the number of layers and their purpose. The most important parameters to take in consideration when starting a design are the electrical boundaries of the system, such as maximum currents and voltages, shielding considerations, signal types and so on [22].

After deciding on the components used, the first practical step to make in the creation of a PCB system is creating their software model. As mentioned in the introduction, the project will be split into schematic and layout parts. Regarding the schematic, all components must have all the pins found in their datasheet with the adequate label and name. The pins will then be connected to their

equivalent in the PCB layout, called a footprint. The footprint needs to follow the exact dimensions of the components, including case dimensions, pads and pitch values.

After completing the details of each component, they need to be connected through an electrical schematic. It is meant to show the component blocks and how connections should be made, both inside the block between its components and outside of it with the other blocks on the board. at this level, no physical details of the components and the shape of the board are needed, just strictly the electrical connections made, the values of the components.

After a brief analysis of the components' positions, the designer needs to think about how the connections will be made, to help decide on the board stackup. A decision needs to be made regarding the number of layers the PCB must have, depending on how crowded the PCB layout is, inducing the need to route on multiple layers and on the impedance characteristics of the circuit. Every PCB has an even number of layers, separated by dielectrics, and each layer can be associated to a power, ground or supply.

Before starting on designing the board, certain rules and constraints need to be set, regarding spacing between components, traces and pads. This step assures that there appear no short circuit errors that can affect the functioning of the prototype.

A proper board shape needs to be set, depending on the possible protection casing and the dimension restrictions of the prototype. After properly setting the board, the component layout can be assembled on the top and bottom layer, depending on the requirements. This will properly show how the components fit on the board and how they should be placed so that they are connected as easily as possible. If certain drill holes are necessary, this is the step in which they should be placed.

The connections from the schematic need to be implemented on the PCB through routing. This represents adding traces of conducting material on the appropriate layers. For passing from one layer to another, vias are used. There are certain rules that need to be followed for a good routing, the most common one being the trace angle, that is meant to be of 45 degrees due to issues that can appear in 90 degrees angles.

Before finalizing the project, the designer can add certain labels and identifiers, meant to show the name of the prototype, the design date and any other notation needed. With this final step, the prototype is ready for a design rule check, and after the error evaluation and fixing, the necessary files can be sent to production.

4.4.2 Fabrication and Assembly

In the fabrication process, each layer is fabricated separately through a photoimaging process. A stencil is used to predefine the copper areas that will remain intact on each layer. From the Gerber files, stencils for the inner layers of the board are created, containing all their copper features.

On every copper layer a photoresist material is applied, material that will harden once exposed to UV rays. Through the stencils, light is applied on the sides of the coated copper layers, that will follow in the hardening of the copper tracks that need to remain. After the tracks remain protected by the photoresist layer, the layer is submerged in a solution meant to etch the surplus copper, removing it. The photoresist layer also needs to be removed from the now formed track, leaving behind only the useful copper on the exposed dielectric material. To be prepared for the multilayer lamination, the copper needs to be roughened, in order to provide a surface fit for sticking to the next layer [23].

The layers need to be inspected individually before assembling the stackup, ensuring the lack of mechanical and electrical defects. At high temperature, the layers are compressed together in the lamination process until bonding. Once these steps are finished and cured, it is ready for plating and drilling. A similar photoresist method is applied in order to coat the outer layer copper. A solder mask needs to be added to protect the copper on the outer layers. The final touches are applied through surface finishes and printing of the silkscreen.

Soldering is the process of attaching components to the printed circuit board. In automatic soldering, the surface mounted devices (SMD) components need a paste mask stencil for assembly.

The exposed SMD pads are covered in solder paste that will harden in the oven, adhering to the components once heated in the oven. The through hole components are to be soldered by hand.

With the components placed, the product is finished for proper testing. A visual test is first require, in order to observe possible visible shortcuts or misplacements. The functionality of the product is to be tested in multiple ways.

5 Implementation

5.1 Circuit design

The circuit of the prototype is implemented based on the previously described Figure *4.6.* This section is meant to describe its functional blocks and connections and how they were implemented, with the appropriate software and hardware tools.

5.1.1 Sensor configuration

As mentioned previously in paragraph 4.1.1., this prototype employs the CCD sensor array TSL1412S, formed by two pixel sections of 768 photodiodes disposed horizontally, presented in Figure 5.1. The center-to-center pixel spacing of 63.5 μm represents the accuracy of the sensor, and gives the total range of the sensor, of 97,5 mm. The pendulum's range of movement is encased within a surface of approximately 100 square centimeters, as a plane described by the two perpendicular sensors used for both measurement axes. These dimensions are used in converting the information given by the sensor, which Is represented into the form of pixel numbers, into a physical distance expressed in centimeters.

Figure 5.1. TSL1412S [24]

For the proper integration of the sensor into the system, its electrical properties need to be analyzed, starting from its datasheet [24]. The first step in realizing the suitable connections of the sensors consists in studying its pins and signals they have assigned, found in Figure 5.2, in order to understand its working principle.

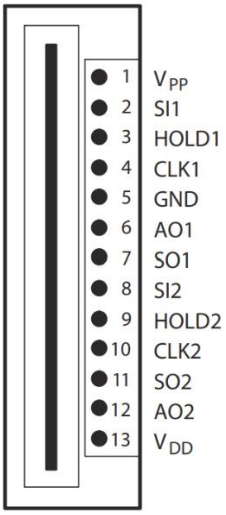

Figure 5.2. TSL1412S Pin Assignments [24]

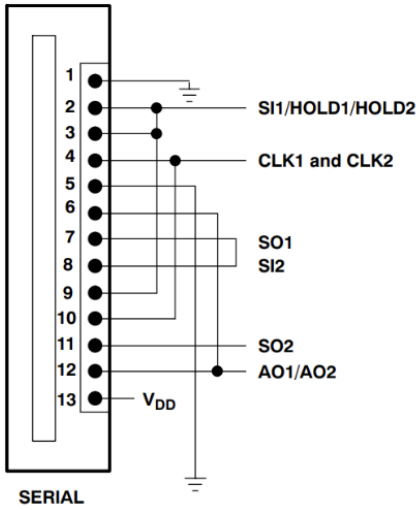

Figure 5.3. TSL1412 Serial Operational Configuration [24]

A reading cycle starts by applying a logic 1 on SI (Serial Input). The HOLD signal is generated once with the rising edge of the SI, by connecting SI1 and HOLD1 together. A separate output signal is generated for both photodiode sections, SO1 and SO2, on the 768th clock rising edge. For this implementation, the whole output signal needs to be generated once, result achieved through the serial connection presented in Figure 5.3.

This configuration is achieved by manual soldering through small wires, so the signal are not perturbed by large quantities of noise over the length of the connector. After the hardware assembly is finished, it needs to be checked for possible short circuits. For the system to work, the proper signal pulses need to be applied accordingly to the pins.

5.1.2 Sensor integration

After the control waveforms of the sensors are properly generated, the output signal needs to be properly digitalized in order to be handled by the microcontroller. This result can be achieved through two different approaches, first using the internal comparator of the microcontroller and afterwards by employing a high-speed ADC.

For the analog comparator option, a certain configuration must be implemented, in order to compare the output signal of the sensor with a fixed bandgap voltage reference. For this, the internal ADC of the microcontroller is turned off, and the pins responsible for the converter peripheral, the pins of Port A, depicted in Figure 5.4. They can be employed either as input or output pins, in this implementation being used as both. The signals generated by the microcontroller are the SI and CLK signals. The bits set as inputs are PA0, PA1, PA3 and PA4, on which the output waveforms of the sensors will be propagated. The SO signals are represented through a high level that indicates the end of each half-frame.

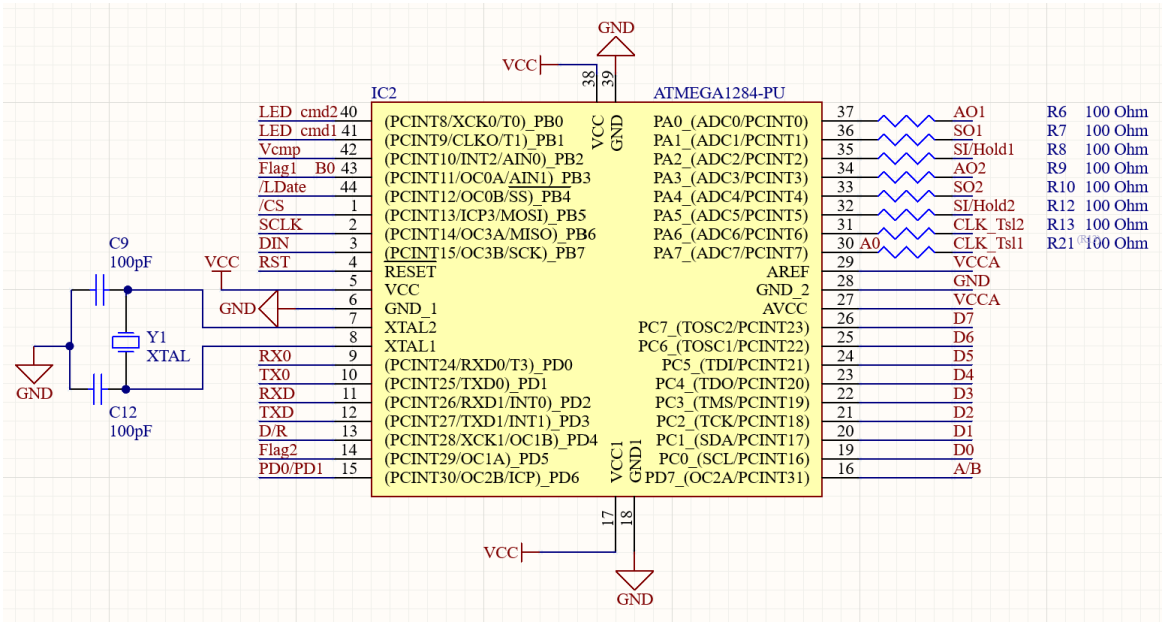

Figure 5.4. Microcontroller schematic connections

The chosen ADC is an 8-bit converter with two channels, used to read both channels alternatively. The pins of Port C are used specifically for this purpose, being connected to the output data pins of the ADC. The A/B signal is an output signal of the ADC, showing which channel is being read at the moment. The PD0 and PD1 pins control the power logic, which needs to be set to normal operation by applying logic one to both.

The voltage supply of the converter is set to the voltage supply of the whole circuit, VCC, equal to 3.3V. For better accuracy, it is filtered through an inductor and an arrangement of parallel capacitors, forming VCCA1.

For the proper operation of the ADC, external elements need to be used. The device has an internal voltage reference of 0.512V, which is too low for this application, the whole circuit using a power supply of 3.3V. To set the voltage reference externally, the REFIN pin needs to be connected to ground through a 100nF capacitance, as shown in Figure 5.5. The voltage reference is defined as the difference between the REFP and REFN pins in the unbuffered external reference mode. In this operation mode, REFP, REFN and COM are bypassed through a 0.33µF capacitor. For the proper voltage range of this prototype, REFP is set to 3.3V and REFN to ground, for a difference matching the voltage supply.

As it is needed to work in single ended mode, with no negative voltages the negative pins of both inputs are set to ground. The output signals of the sensors are connected to the positive pins, INA+ and INB+. The resistances allocated for each input isolate the signals from the capacitive input to prevent unwanted oscillations.

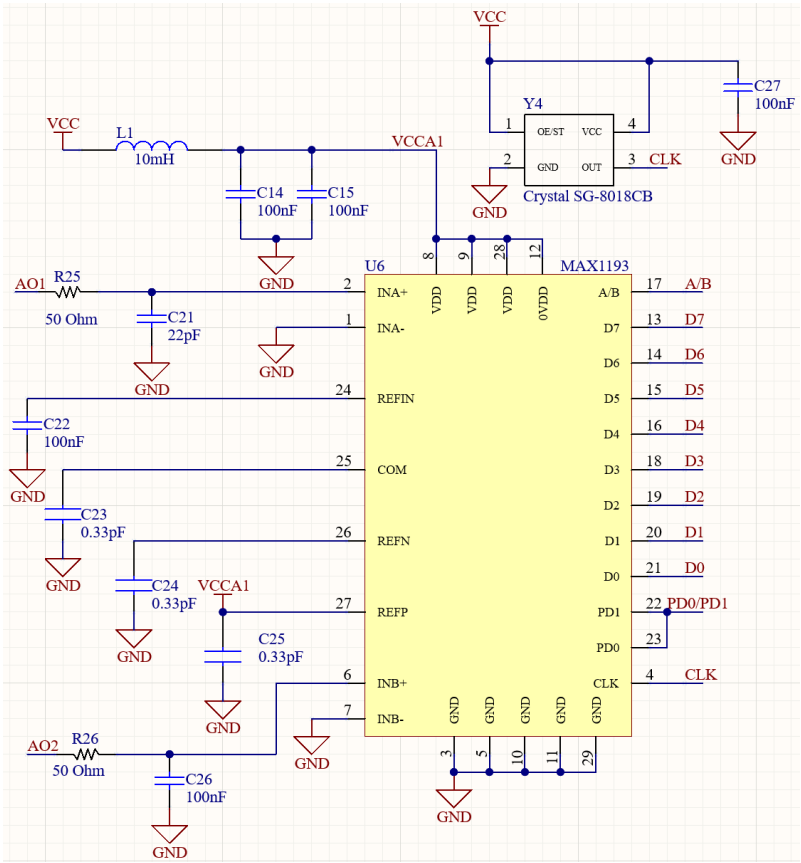

Figure 5.5. ADC schematic connections

5.1.3 USB integration

The RS232BL integrated circuit is used in the USB bus powered configuration, assuring that its associated components get their power from the USB bus. The J8 connector, found in Figure 5.6, is a USB type B connector, where pin 1 is supplied with 5V from the port, pins 2 and 3 are the differential USB data lines, and the rest are ground connections. On the VCC line of pin 1, protection elements are added, the ferrite bead F2 and the R99 resettable polyfuse (PTC), .

The USBDP (USB Data Plus) and USBDM (USB Data Minus) are used for differential USB communication, allowing the exchange of information between the computer and the device. Being

a differential pair, signal integrity is improved over longer distances, compared to single ended signaling. Their difference represents either a logical one, when the Plus is greater than the Minus, or logical zero otherwise.

The input differential data is converted internally into UART compatible data, through the TX pin. Conversely, information sent on the RX pin is transformed into USB packets. This allows for effective communication between different interface standards. LEDs are used for indicating that the transmission and reception of data are properly realized.

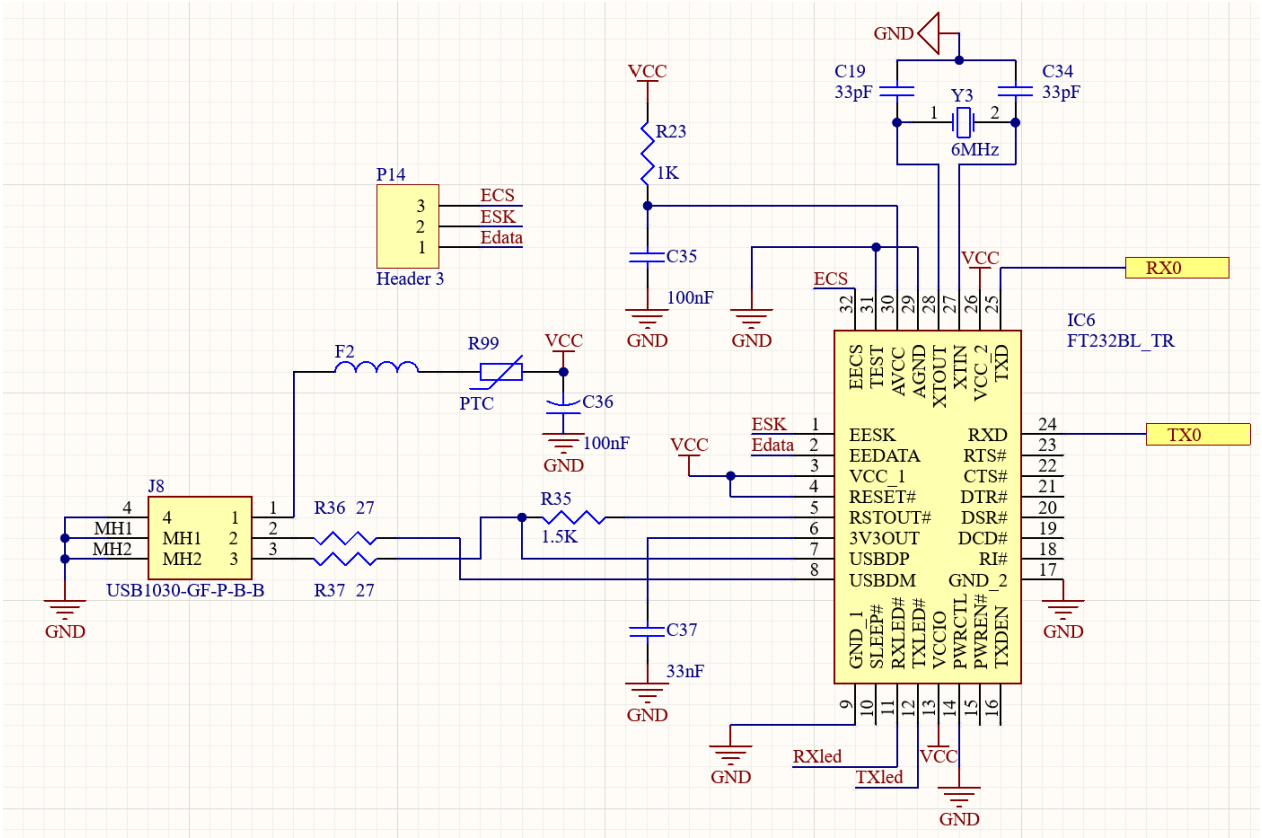

Figure 5.6. USB schematic connections

5.2 PCB design principles

After the circuit schematic is completed, it needs to be implemented on the actual board. It does not have any dimension requirements, so the board shape is chosen as a 9 cm by 9 cm square, so almost all components fit on the top size. However, certain components are also placed on bottom for efficiency.

5.2.1 Component placement

The main component blocks need to be placed appropriately for their function. As the microcontroller connects all of them together, it will be the center piece of the assembly, placed on the top side.

The sensors are connected to the board through 6 pin headers, displayed on 2 columns of 3. The LEDs responsible for generating a proper source of light are also connected through headers, needing only 3 pins. This connector ensemble is placed on the upper edge of the board, to assure a connection through wires as short as possible, to avoid noise.

A test point is placed for every signal attributed to the sensors, represented by a through hole circular via with a 1.2mm diameter, to fit the test pins. As the wires on any board have a certain

parasitic capacitance, adding a resistor on each signal line creates a low pass filter used to remove possible high frequency, while also limiting the current flow.

The ADC block is placed close to the sensors, to have traces as small as possible between the sensor's output and the ADC input pins. Each capacitor is placed as close as possible to the IC in order to remove the noise right before the signals enter the pins.

Decoupling and bypass capacitors need to be installed on every power line, serving as filtering (low-pass) components. Both capacitors eliminate noise of high frequencies carried from the input signal by providing a low impedance track, offering a discharge path to ground. Decoupling capacitors come in an electrolytic case, having smaller values in the range of nF.

5.2.2 Polygons

Both the top and bottom layers are covered in ground polygons, for high immunity to external noise. However, the plane follows certain spacing rules, to distance it from the other traces. Even with the proper rule settings, some ground fringes can remain between traces, creating possible antennae. One example of removing such elements is shown in Figure 5.7 through the red dotted lines.

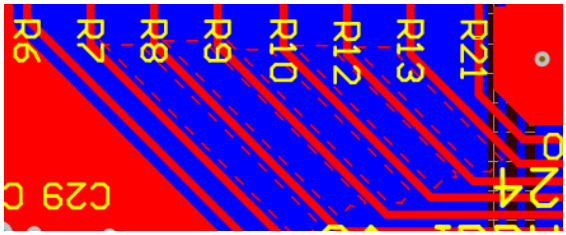

Figure 5.7. Removal of ground plane induced antennae

USB ensemble's oscillator, as well as the one assigned to the microcontroller, are placed on the bottom side of the PCB, along with a bottom solder polygon on one of its halves to absorb noise, represented as the pink rectangular shape is highlighted in Figure 5.8.

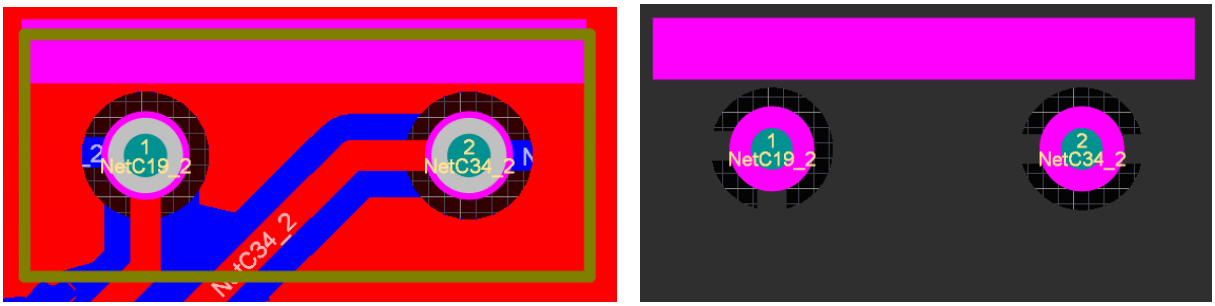

Figure 5.8. Solder polygon of the oscillators

5.2.3 Vias

Ground vias are placed throughout the board to avoid ground bouncing, a type of noise induced when there are different voltages for the PCB ground and the die package ground. Vias assure connections from the top to the bottom of the board, creating the same voltage along the entire plane. They are also placed to connect floating ground patches or on plane edges to properly connect each fringe to ground.

The oscillator assigned to the ADC is properly guarded, as shown in Figure 5.9, in order to prevent its high frequency to interfere with the other components. Through this via ensemble, the noise is rapidly absorbed to the ground.

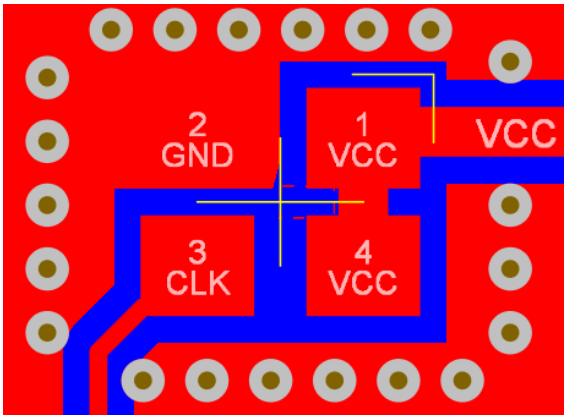

Figure 5.9. Guarding of the ADC's oscillator

5.3 Programming the microcontroller

The programming of the microcontroller is achieved through the Microchip Studio application, in C programming language. The program is split into headers used for grouping the functions based on their purpose.

All the port pins and peripherals need to be properly initialized before realizing any actions, through functions created in the *uC_Periferic_Initialisation* header. The functions responsible for the image sensor are found in TSL1412S.h. The other operations are memorized in a general header called *Functions*. The header that can be found inside each one of the others is *UsedVariable.h*, that contains all the fixed values needed for the application.

5.3.1 Ports initialization

Each pin of every port needs to be properly initialized depending on its purpose. Output pins are to be set to logic one and input pins to logic zero. These attributions are made in the DDR register (Data Direction Register) responsible for each port.

Other settings regarding the port initialization are made in the PORT register assigned to each port. By driving its bits to logic one, it enables their internal pull-up, actually driving their voltage to VCC.

```
DDRB |=((1<<1)|(1<<0));
PORTB |=((1<<1)|(1<<0));
```

Figure 5.10. Code for port initialization

The proper settings are achieved as shown in the code from Figure 5.10.

5.3.2 Signal generation

The signals that need to be generated for the proper function of the sensor are the CLK and SI, according to Figure 5.11. The clock signal needs to have an appropriate frequency based on the sensor's limits and the system's needs. A frequency as high as possible is wanted due to the high accuracy required for this project, each period being vital for an accurate measurement. The SI needs to be set to logical 1 once after every 1537 clock cycles, in order to integrate every pixel in the array.

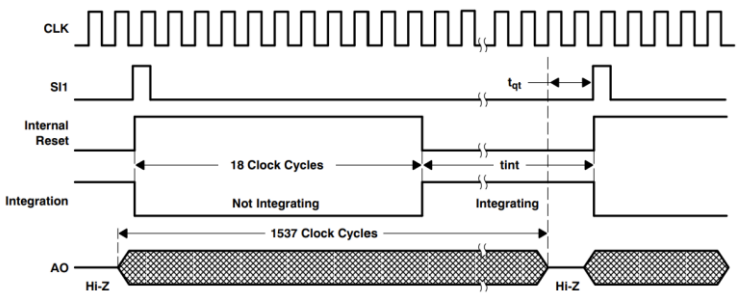

Figure 5.11. Timing Waveforms

The actual signal is generated on a chosen pin from one of the ports by toggling it, with delays induced by *nop* functions in order to form the low and high levels. In order to achieve this result, a separate function is created, hintingly called *signals*, responsible for the generation of all the waveforms needed for the sensor's function, as well as reading its output signal. The logical interconnection of the events in this function are presented in Figure 5.12.

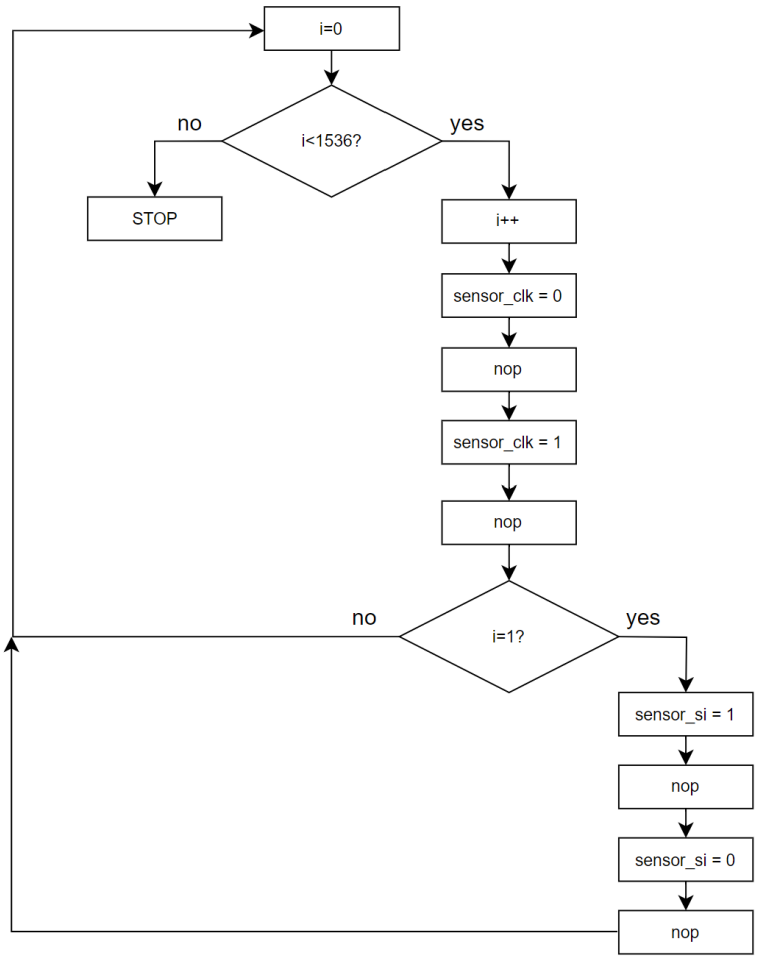

Figure 5.12. Logic diagram of the "signals" function

For generating the SI waveform, a bit is toggled only on the first iteration, for indication the start of every pixel frame that is processed. After passing through every pixel, a total of 1536, the process starts all over for processing the next frame.

The signals need to be propagated on their attributed pins of the microcontroller. Before doing so, the ports and pins need to be properly initialized. Everything is implemented in the *signals*

function, shown in Figure 5.13, where the *sensor_clk* and *sensor_si* are the pins on which the signals are propagated.

```
void signals(){
    char local_var = 0;
    unsigned int counter_clk = 0;
    Port_sensor |= (1<<sensor_clk);
    asm("nop");
    Port_sensor &= (~(1<<sensor_clk));
    asm("nop");
    Port_sensor |= (1<<sensor_si);
    asm("nop");
        for (counter_clk=0; counter_clk < (integration_periods+5); counter_clk++ )
        {
            Port_sensor |= (1<<sensor_clk);
            asm("nop");
            Port_sensor &= (~(1<<sensor_si));
            asm("nop");
            Port_sensor &= (~(1<<sensor_clk));
            asm("nop");
        }
}
```

Figure 5.13. Code for signal generation

5.3.3 Comparator

The output signal is generated using the internal analog comparator of the microcontroller, which compares the input values from the positive pin AIN0 and the negative pin AIN1. To utilize this feature of the microcontroller, the internal ADC needs to be turned off. The output can be seen on the ACO bit (Analog Comparator Output) of the ACSR register (Analog Comparator Control and Status Register). From this register, the comparator is also enabled through setting the ACD bit (Analog Comparator Disable) to logic one.

In this application a fixed reference voltage is wanted for the comparison with the sensor's output signal. This is achieved through writing logic one to the ACBG bit (Analog Comparator Bandgap Select), which generates a bandgap reference voltage of 1V1 that replaces the positive input of the comparator. For selecting the negative input, the ADC multiplexer is used. This feature needs the ACME bit (Analog Comparator Multiplexer Enable) from the ADCSRB register (ADC Control and Status Register B) to be set to logic one. Every setting is implemented using the code from Figure 5.14.

```
void AnalogComparator_Initialisation( char channel){
    Comparator_Enable;//ACSR &= (1<<ACD);
    //ACSR |= (1<<ACBG); // AIN0=referinta 1.1v AIN1 for sensor
    ADC_off; //ADC off
    ADMUX = channel;
    ADCSRB |=(1<< ACME);
    _delay_ms(5);
}
```

Figure 5.14. Code for initializing the analog comparator

The data comparing step needs to be introduced into the function responsible for reading the pixel values. The information is stored into an array, where the pixel value is represented by the value of the ACO bit for each position.

5.3.4 Timer

The timer is used for reading the pixel values at a certain time interval, chosen to be one second. It works in Clear Time on Compare Match (CTC) mode, where the OCR1A register is used to

manipulate its resolution, which represents the stop value of the counting process. In this mode, the counter is set to 0 once its value (TCNT1) reaches the number set in OCR1A, allowing great control over the frequency.

The register is set using the formula displayed in equation (5.1), where N is the prescaler factor, represented by a power of 8 from 1 to 1024. N is set in the TCCR1B register, through the first 3 bits, CS10, CS11, and CS12, called clock select bits. This register is also used for setting the CTC top value to OCR1A by putting the 4th bit, WGM12 (Wave Generation Mode) to logic 1.

$$f_{OC1A} = \frac{f_{clk_{IO}}}{2 * N * (1 + OCR1A)} \tag{5.1}$$

For achieving the wanted value of one second, we need a frequency of one Hertz. $f_{clk_{IO}}$ is the clock frequency of the microcontroller, in this application 16MHz. OCR1A is calculated to be 31250 for a prescaler factor of 256, value that can be represented on the 16 bits of the register.

```
void Timer_1_Initialisation_0_1S(void){
    TCCR1A = 0;
    TCCR1B = ((1<<WGM12)|(1<<CS12)|(0<<CS11)|(0<<CS10));  //CTC foscil/256;
    OCR1A = 31250;
    TIMSK1 |= (1<<OCIE1A);
}
```

Figure 5.15. Code for initializing the timer

After properly initializing the timer, as presented in Figure 5.15., it is put into function by calling the timer interrupt. The interrupt is enabled by writing logic 1 to the OCIE1A bit in the TIMSK1 (Timer/Counter1 Interrupt Mask) register.

The interrupt is written in the main program as a function that is automatically called at the set period of time. A counter will memorize the number of iterations, and for each one a sensor will be read, first the x axis sensor, next the y axis sensor.

5.3.5 USART communication

For the proper function of the USART routine, it needs to be initialized accordingly to the specifications of this projects. The first object to be controlled is the UCSR0B register (USART Control and Status Register B). The RXCIE0 bit (Receive Complete Interrupt Enable) is set to logic one in order to trigger an interrupt when a byte is received. On the other hand, the TXCIE0 bit (Transmit Complete Interrupt Enable) is set to logic zero, disabling this function. Both the receiver and transmitter circuitry are enabled through setting the RXEN0 (RX enable) and TXEN0 (TX enable).

Another setting needs to be realized regarding the USART Baud Rate. This is achieved through setting UBRR0L and UBBR0H registers as the low and high levels of the baud rate. The wanted result is a baud rate of 9600, which needs to be converted according to Equation (5.2), where F_{OSC} represents the clock frequency of the microcontroller, 16MHz. The computed result for UBBR0H is 103.

$$UBRR = \frac{F_{osc}}{16 * baud\ rate} - 1 \tag{5.2}$$

The processing of information through the USART is realized inside of a specific interrupt, the Interrupt Service Routine (ISR) for the USART0 RX vector, *ISR(USART0_RX_vect)*. After the needed variables are initialized, the first step is to read the received data from the USART data register (UDR0).

The data received by the serial port represents the command that the microcontroller should execute. Commands are given in the form of certain characters, that decide which data needs to be transmitted, the values taken from the x or y axis. This process is realized through a switch case statement based on the received data. One such case is presented in Figure 5.16.

```
case 'm': //x axis memory
    for (data=0;data <MaXLenghtPixel; data++)
    {
        Local_vr = UCSR0A &  0x20;
        while(Local_vr ==0) Local_vr = UCSR0A & 0x20;
        UCSR0A |= 0x20;
        UDR0 = (PixelsValue_X[data]+48); //conversion from hex to ascii
    }
break;
```

Figure 5.16. Code for the USART interrupt

After every iteration, the buffer of the serial communication needs to be emptied. This is done through the 5th bit of the UCSR0A register (USART Control and Status Register A), that holds various flags and status bits related to serial communication. When the UDRE0 bit (USART Data Register Empty) nis set, UDR0 is prepared to accept new data.

5.3.6 Sequential automata of the entire program

The whole program can be split into two logical loops that are iterated continuously. The first loop is responsible for capturing the data once every second using timer 1 and can be seen on the left side of Figure 5.17. On the other hand, the right side represents the process of receiving and transmitting information to the computer, in order to actually see the results captured by its opposite.

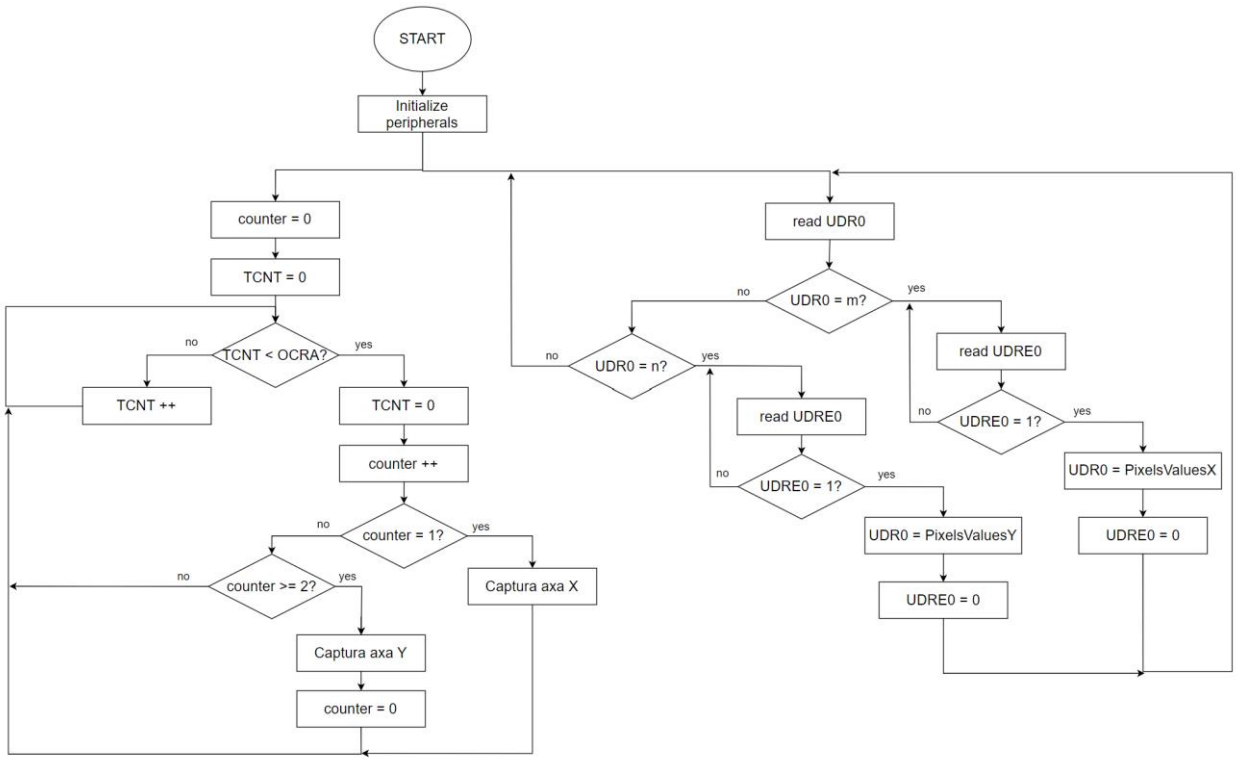

Figure 5.17. Sequential Automata of the entire program

5.4 Python implementation

The output data is processed using Python programming language, within the open source environment Spyder. The signal is interpreted from a csv (comma-separated values) file exported from Scopy, using certain libraries and plotting techniques.

The first step in implementing the whole interface is analyzing the possible methods for signal processing. This is realized though an experimental test using just the sensor connected to a microcontroller. Instead of the proper pendulum wire, a pen is used for experimental purposes, generating somewhat of a wider shadow.

5.4.1 Data processing

Before visualizing the waveforms, the data given in the form of a table in the *csv* file needs to be processed. After a first analysis of the rows and columns present in the file, the useful information is detected. From Figure 5.18. some rows of additional information can be detected, which must be ignored. The points plotted graphically are the ones given by the columns referring to time and the channels.

```
;Scopy version,f4beeb1
;Exported on,Thursday January 11/01/2024
;Device,M2K
;Nr of samples,1600
;Sample rate,1e+08
;Tool,Oscilloscope
;Additional Information,
Sample,Time(S),CH1(V),CH2(V)
0,-9.2e-07,-0.0123419,-0.0581034
1,-9.1e-07,-0.0413919,-0.0581034
2,-9e-07,-0.0123419,-0.0435784
3,-8.9e-07,-0.0268669,-0.0726284
4,-8.8e-07,0.031233,-0.0726284
5,-8.7e-07,-0.0123419,-0.0726284
```

Figure 5.18. Sample of a csv file extracted using Scopy

Python integrates multiple data analysis libraries with various features and functions. A fast and flexible tool found within them is the *pandas* library, used in diverse applications regarding information manipulation. It has powerful capabilities in reading and writing data, while also providing more complex functions. This application uses the features of the pandas *read_csv* function, which assures multiple options regarding the columns that need to be used, the rows to be ignored, possible comments and the type of separator found in the document. This action is depicted in Figure 5.19.

```
# Read the data into a DataFrame, skipping comment lines and using the correct delimiter
data = pd.read_csv('test eexplorer 1.csv',usecols=[0, 1], sep=',', comment='#', encoding='utf-8-sig')
```

Figure 5.19. Code for data processing from .csv file

After the appropriate processing of the file, the data needs to be plotted accordingly. The used library is *matplotlib*, tool that contains all the required options of graph text settings, legend appending and so on. The necessary graphs of the original and filtered signals are plotted according to Figure 5.20.

```
# Plot the graph
fig = plt.figure()
ax1 = Subplot(fig, 111)
fig.add_subplot(ax1)
ax2 = Subplot(fig, 212)
fig.add_subplot(ax2)
ax1.plot(data[data.columns[0]], data[data.columns[1]])
ax2.plot(data[data.columns[0]], result_med)
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Channel 1 (V)')
ax1.set_title('Original signal',fontsize = 'xx-large')
ax1.legend(['Savgol Filter', 'Median filter'], fontsize = 'xx-large')
ax2.set_xlabel('Time (s)')
ax2.set_ylabel('Channel 1 (V)')
ax2.set_title('Graph from Data')
fig.show()
```

Figure 5.20. Code for graph plotting

5.4.2 Serial communication

Python has an integrated library used for serial communication with certain devices, suggestively called serial. An object denoted with *ser* is responsible for handling all the necessary functions for this operation. It is set to work on the proper COM to which the board is connected, at a baud rate of 9600.

The previously described object is used for sending and receiving data. The microcontroller was programmed to execute different instructions when certain characters are sent serially. When the microcontroller receives the character “m”, it sends the data describing the waveform of the sensor responsible for the x axis, doing the same for the y axis once the “n” character is received, following the flowchart from Figure 5.21. The characters are sent within separate functions, *start_reading_x* and *start_reading_y*, which call the reception function *read_data.*

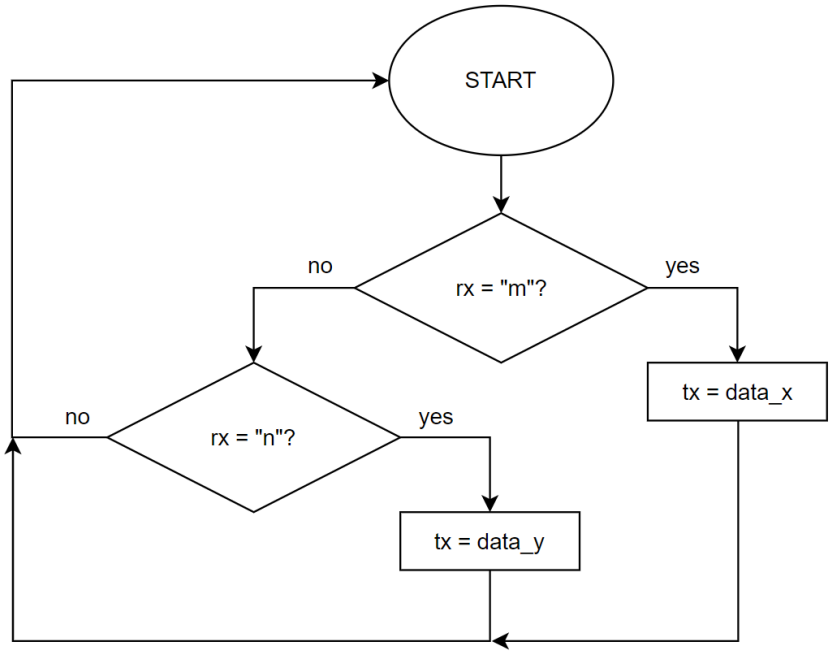

Figure 5.21. Flowchart of data transmission/reception

The data is received in the form of bytes, which need to be decoded by the utf-8 standard into a string. This needs to be once again transformed into an array of integers for proper manipulation.

All of these operations are realized using functions of the numpy library, which is a tool specialized in data handling. This set of data conversion and filtering is realized using the code from Figure 5.22.

```
def read_data():
    global y_final
    global result_sav #filtered result
    y = ser.readline()
    print("number of characters received", len(y))
    print("Output type: ", type(y))
    print(f"Received non-numeric data: {y}")

    # Decode the bytes object to string and strip any leading/trailing whitespace
    y_str = y.decode('utf-8').strip()
    print("Output type: ", type(y_str))

    # Convert the string into a list of characters
    y_list = list(y_str)
    print("Output type: ", type(y_list))

    # Convert the list of characters to a list of integers (0s and 1s)
    y_final = np.array([int(char) for char in y_list])
    print("Output type: ", type(y_final))
    print("Final result", y_final)

    # Calculate filtered data
    result_sav = savgol_filter(y_final, 40, 2)
    print("Filtered data:", result_sav)
```

Figure 5.22. Code for data conversion and filtering

Once the data is properly received, everything resumes to the whole purpose of this paper, determining the position of the wire. The signal is practically split into small sections generated by each pixel, having either a low or high value. This process can affect the quality of the measurement through an unwanted margin effect, inducing false values throughout the actual shadow of the wire. This issue can be solved through filtering, as presented in the previous section, 5.4.1., that creates a proper signal on which the position can be measured.

After the process is finished, the serial port needs to be properly closed, or it can cause errors on the next iteration. This is realized in the *on_closing* function, that is responsible for both closing the serial port and destroying the root of the user interface.

5.4.3 Calculating the position of the pendulum

Even for a really thin object, the shadow can be extended over more than one pixel. For this reason, the actual position of the pendulum is considered as the middle of the interval over which the shadow is detected. The zero values indicate the presence of shadow, so the result will be found inside one interval of zeroes. However, there can be noise induced zeroes throughout the signal, resulting in the need to find the widest such interval. This can be achieved through following the steps in Figure *5.23*. The interval of maximum length is found by passing once through the entire array and comparing the interval lengths given by the stop and start position. The result is updated only for the greatest interval found.

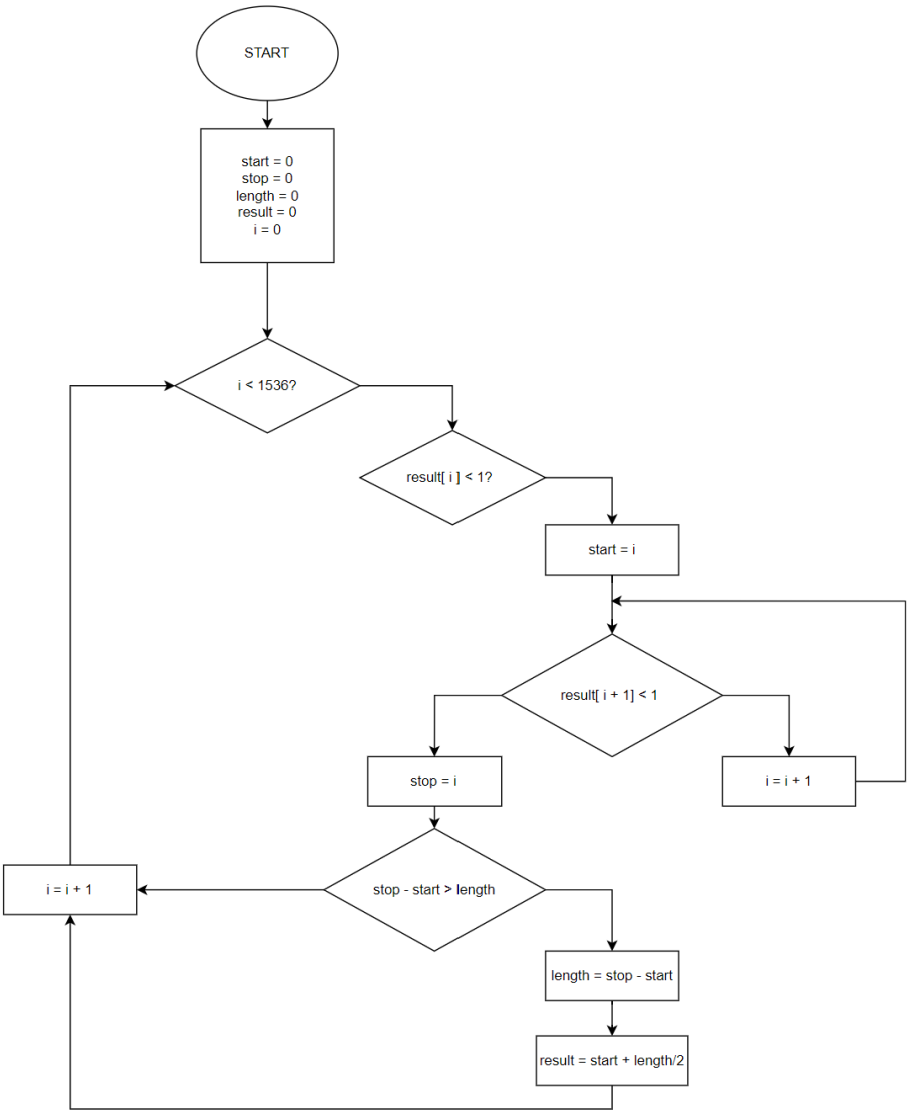

Figure 5.23. Function that calculates the position of the pendulum

However, the result given directly from the array expresses the number of the pixel on which the shadow can be found. As the application is meant to measure a position, the result needs to be expressed in the metric system, based on the dimensions of the sensors. By multiplying the initial result with 0.0635, the width of a pixel, the final position is expressed in centimeters. This process is entirely implemented in the *calculate_position* function, presented in Figure 5.24.

```
def calculate_position():
    if result_sav is not None:  # Check if result_sav has been initialized
        interval_start = 0
        interval_stop = 0
        interval_length = 0
        result = 0
        i = 0

        while i < 1535:
            if abs(result_sav[i] - 1) > 0.1:
                interval_start = i
                while abs(result_sav[i] - 1) > 0.1 and i < 1535:
                    i = i + 1
                interval_stop = i
                if interval_stop - interval_start > interval_length:
                    interval_length = interval_stop - interval_start
                    result = (interval_length / 2 + interval_start) * 0.0635
            i = i + 1
        messagebox.showinfo("Info", "The position is: " + str(result) + " mm")
```

Figure 5.24. Code for the function *calculate_position*

The presented functions will not be called continuously, but only by the choice of the user, through the interface. In user guided interfaces in general, the actions are taken either by typing certain text pieces, or simply by the press of a button. For the fluidity of the process, certain buttons will have attributed the data sending through the serial port, the graph plotting and displaying the position of the wire.

5.4.4 Guided User Interface

After all the functions are implemented, they are all inserted within an interface. A root is created from the *ctk* library (Custom Tkinter), representing the entire interface page. A series of buttons are implemented, responsible for calling every function mentioned. The logical process of the whole prototype is presented in Figure 5.25.

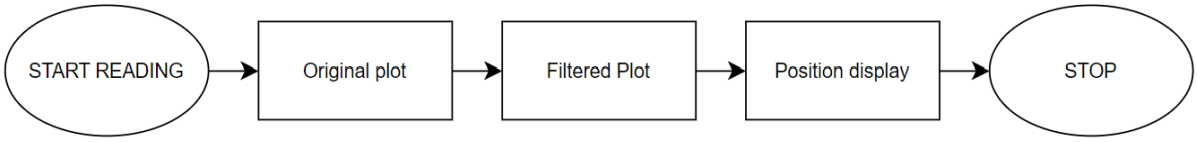

Figure 5.25. Logical process of the interface

The buttons used for starting the reading process of each axis come with a pop-up window showing that the reading was successful or if there were any errors in the process. The *Original* and *Filtered* buttons deal with plotting the graphs within a defined *canvas* widget, taken from the previously mentioned library. They also have an error pop-up in case no axis was selected for measuring. Once the *Position* button is pressed, an information window shows the taken measurement.

6 Experimental results

6.1 PCB design

The final prototype of this project is shown in Figure 6.1. this entire assembly needed a level of dexterity and detail that couldn't have been achieved entirely manually, so it was realized in an electronics factory, through a stencil created to resemble every pad of the board.

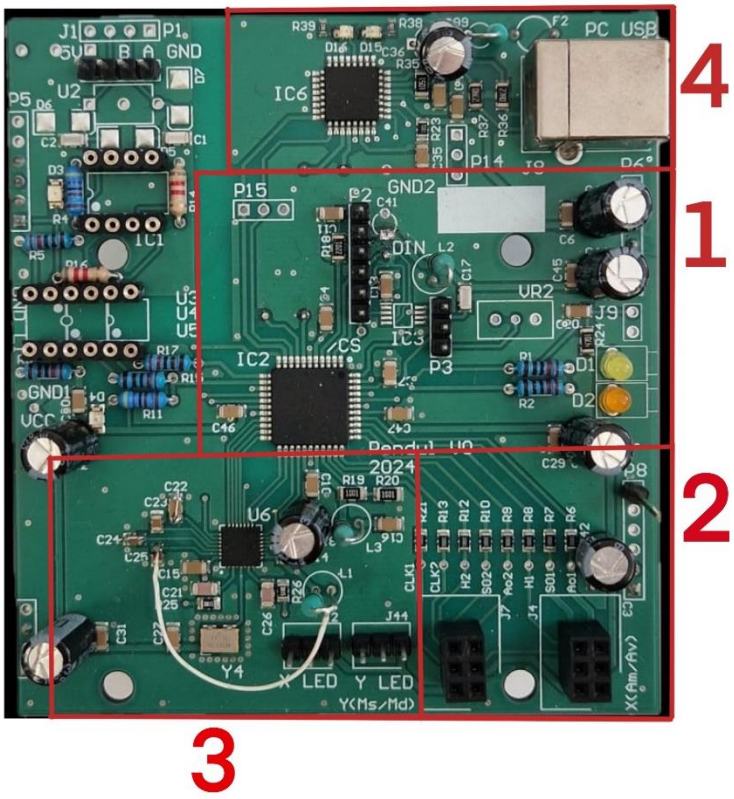

Figure 6.1. Blocks arrangement on the PCB

The first block represents the microcontroller assembly, with IC2 being the microcontroller, P2 being the connector used for programming it. The D1 and D2 LEDs responsible for indicating if the transmission and reception are realized.

The second block comprises the connections necessary for the sensors, J4 being used for the x axis sensor and J7 for the y axis sensor. Their signals are indicated by silkscreen notations next to their afferent test point.

Block number 3 represents the ADC and its necessary components. Here, a design error was solved manually through an external wire meant to properly connect the reference voltage. This block also encases the pins used to control the sources of light of the entire setup, the x and y LEDs.

The last block, block 4, contains the entire USD setup, with the RS232BL integrated circuit represented as IC6. Due to lack of space, the ferrite bead F2 needed to be placed on the bottom side of the board. Another set of LEDs is found here, indicating when the RX and TX signals are triggered.

The stability to external noise of the entire board is assured through the ground polygons present on both sides. After certain cutouts to remove possible antennae zones, the final result is depicted in Figure 6.2.

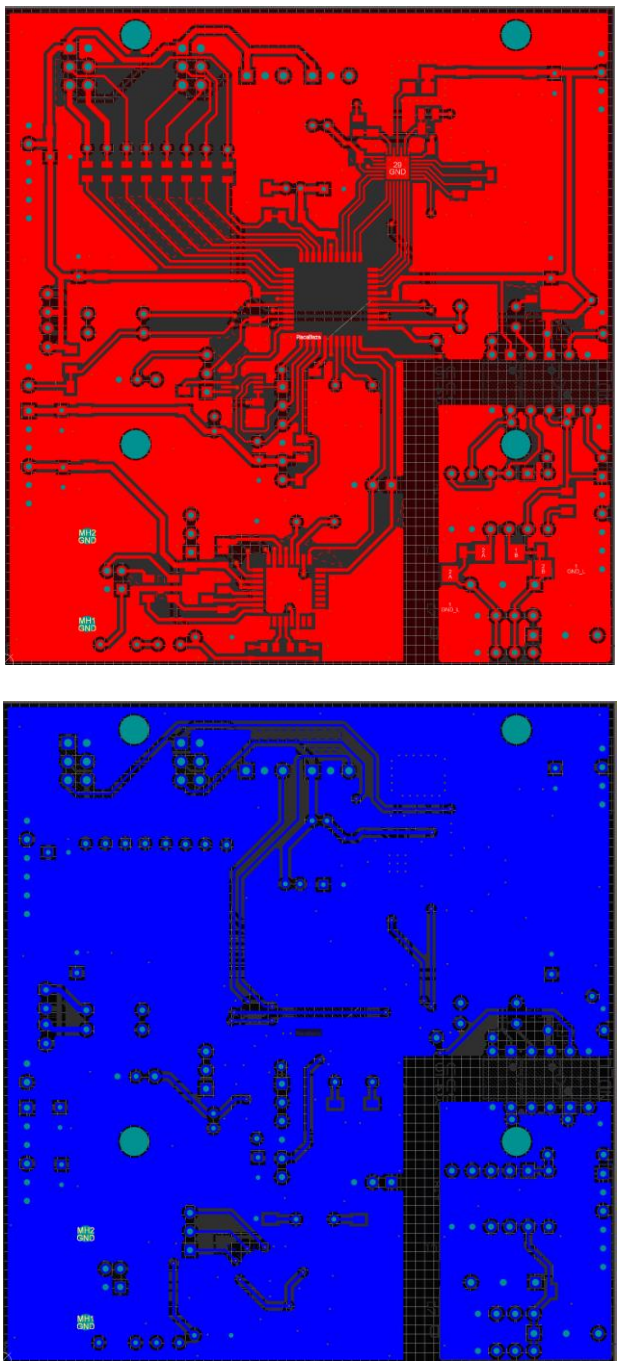

Figure 6.2. Top (red) and bottom (blue) ground polygons

Once the board was properly tested for any possible short circuits or mistakes in the assembly, for deeper observation of its functionality the microcontroller needed to be properly programmed.

6.2 Signal acquisition using oscilloscope modules

The acquisition of the programmed signals was first realized through the ADALM2000 Evaluation Board, an active learning module designed by Analog Devices. This module is used for its two-channel USB digital oscilloscope. The waveforms can be observed through the Scopy software toolset, compatible with Analog Devices products.

Using the mentioned methods, the obtained results are displayed in Figure 6.3, showing the first five periods of the waveforms. The clock signal is shown in purple, along with its measurements in

the upper left corner of the image. It can be seen that the wanted frequency of 3.5MHz is achieved. The SI signal is displayed in the color orange, its rising edge indicating the start of the integration process.

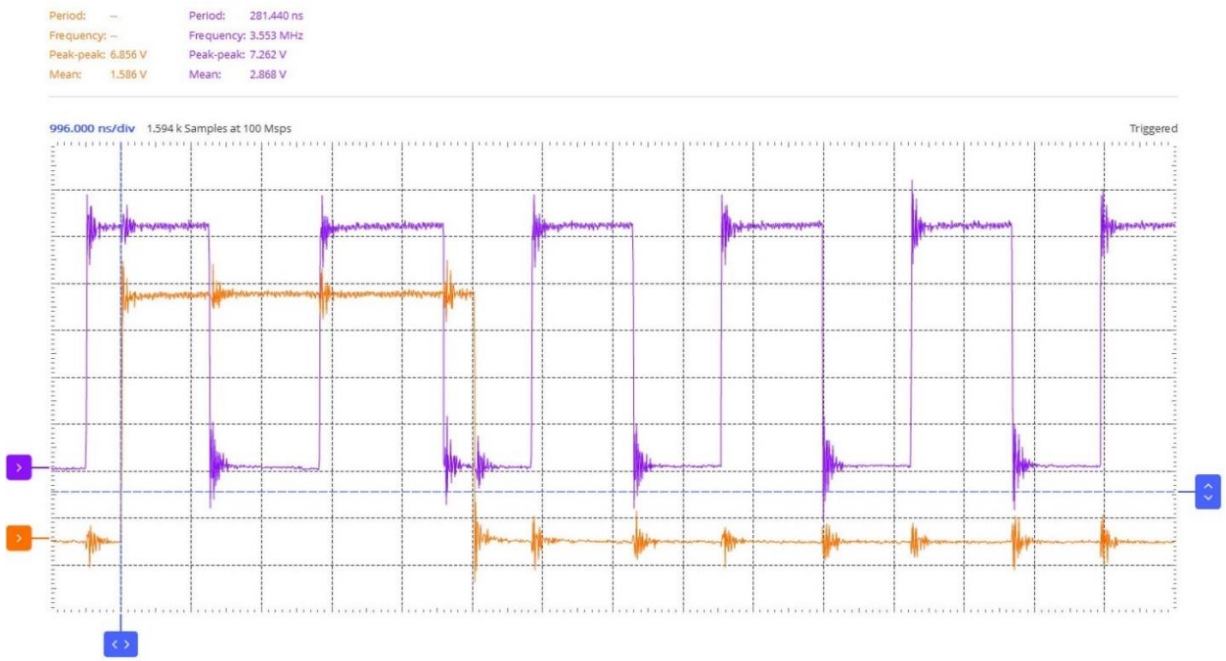

Figure 6.3. Signal acquisition using ADALM2000 in the Scopy interface

For a higher accuracy of the signal, some tests are also realized through the Electronics Explorer Board, produced by Digilent. This device has a larger bandwidth than the ADALM2000, fit for measuring higher frequencies. This board communicates with the computer through the Waveforms interface, capable to display the data taken from the board's oscilloscope module. In Figure 6.4, the noise given by the edge effect can be easily observed through the periodical thin lines, problem that will be solved by the internal comparator of the microcontroller.

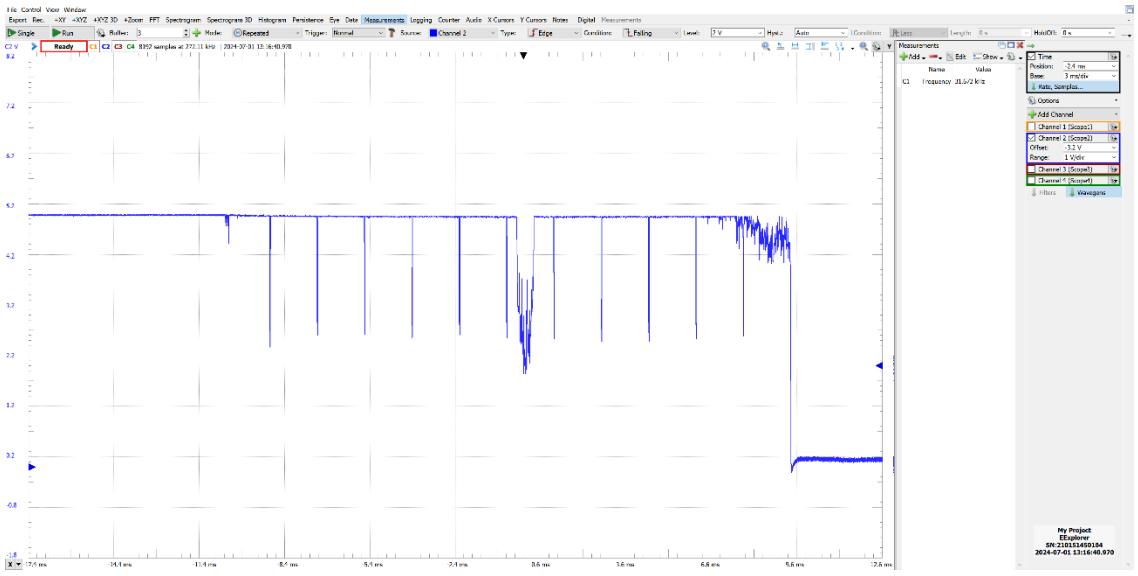

Figure 6.4. Output signal of the sensor using Digilent EExplorer

After observing multiple sets of data on oscilloscope modules integrated within different experimental boards, the final results are to be implemented in the Python software environment. The final plots are represented through an interface, by the choice of the user.

6.3 Python results

The first tests of the prototype were not realized in ideal conditions, being highly affected by possible saturation. The first measurable target was not the pendulum wire, but a pen used for experimental purposes. As observed in Figure 6.5., the output signal of the sensor is deeply affected by high peak random noise, making it very difficult to detect the actual position of the object.

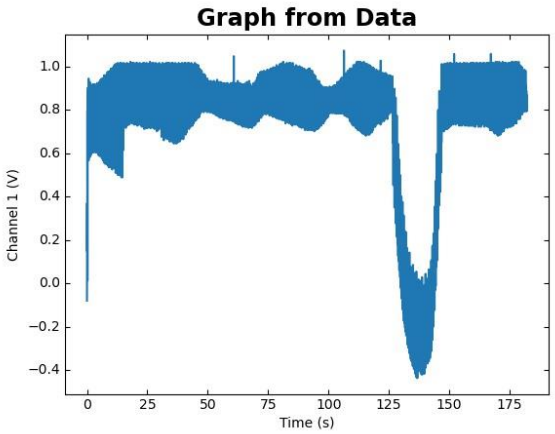

Figure 6.5. Original plot of the first sensor iteration

The waveform needs to be properly filtered through digital signal processing methods, as mentioned in section 4.3. The adequate filtering methods are applied through specific Python functions, imported from the scipy library. This tool contains fundamental algorithms, giving extension of applying mathematical formulas. The results of the two filters applied can be observed in Figure 6.6.

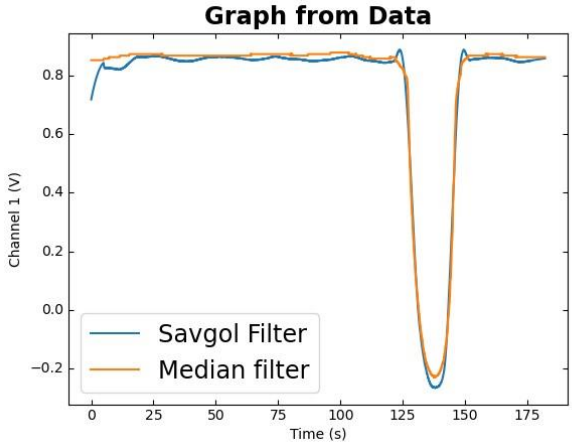

Figure 6.6. Filtered signals of the first sensor iteration

After the proposed experiment is finished, the PCB ensemble is connected serially to the computer for direct communication. The binary data taken from the internal comparator of the microcontroller is sent to the computer, where certain Python functions are applied.

In the final program prototype, the data is not processed from a csv file, but from an array of characters transmitted serially by the microcontroller. The graphs are plotted in separate functions, *plot_original* and *plot_filtered*.

The interface shows up quickly through the push of the run button of the program. The first page to show up can be observed in Figure 6.7, with all the previously implemented buttons.

Figure 6.7. Start page of the interface

The first step in taking the measurement is pressing the start reading button, either for x or y axis, which send the necessary characters through the serial port in order for the appropriate data to be sent back. Once this step is made, a pop-up window will appear indicating that everything worked accordingly. This is represented in Figure 6.8.

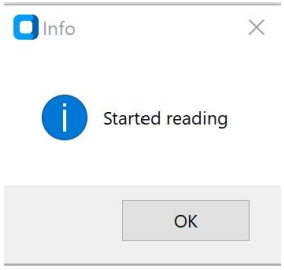

Figure 6.8. Start reading pop-up

The next step is to plot the original signal given directly by the computer in the form of binary values through the serial port. An iteration of this function is displayed in Figure 6.9.

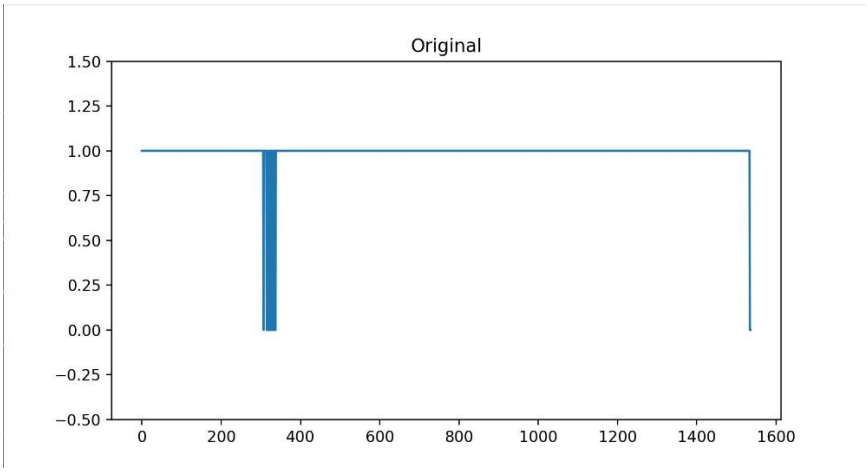

Figure 6.9. Graph of the original signal

If no start button is pressed, an error message will pop up, shown in Figure 6.10. this functionality is achieved through just a few lines of code, presented in Figure 6.11. The *if* statement verifies if there is any information stored in the data vector, *y_final*.

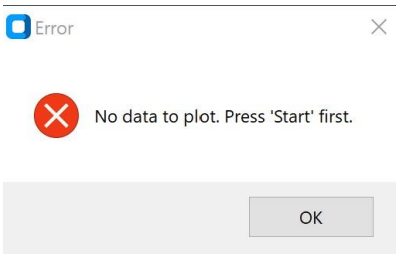

Figure 6.10. Error pop-up

```
if y_final is None:
    messagebox.showerror("Error", "No data to plot. Press 'Start' first.")
    return
```

Figure 6.11. Code generating error message

This signal can also be processed through filtering methods by the choice of the user. By pushing the Filtered button, another graph will show up, underneath the original one, as depicted in Figure 6.12. after this process,

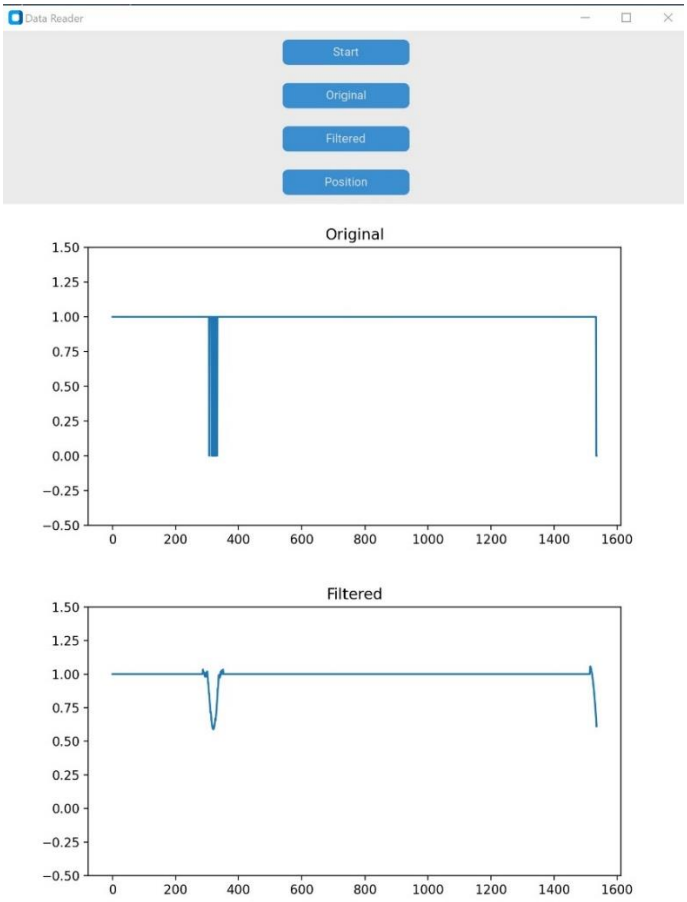

Figure 6.12. Original + Filtered plot

The position is calculated using the previously described *calculate_position* function once the Position button is pressed. It also contains a function responsible for creating an information pop-up window that shows the calculated result, illustrated in Figure 6.13.

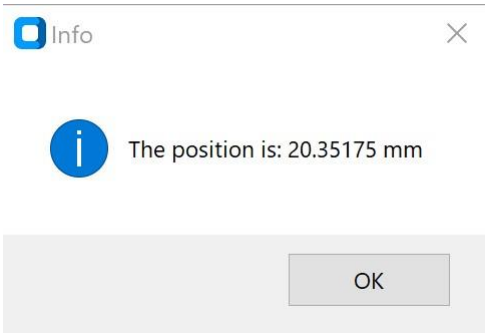

Figure 6.13. Position pop-up

After multiple iterations, the results are observed to be consistent, with a maximum error of 63.5 µm, as observed in Table 6.1.

Table 6.1. Multiple iterations of the same position

Iteration number	1	2	3	4	5	6	7	8
Position value [mm]	20.3517	20.3517	20.3517	20.2882	20.3517	20.3517	20.3517	20.4152

7 Conclusions

As it could be observed, this paper depicts the niches of a fully working measurement setup used in the maintenance of water dams. For the final implementation of the inclinometer prototype presented, different engineering domains blended together, starting from notions of circuit and PCB design, continuing with microcontroller programming and concluding with finer filtering details. All of the techniques are implemented in diverse software environments and employing different programming languages.

This project stands out through the wide palette of technologies used and introducing somewhat new combinations of measurement and filtering blends. As stated in the third section of this work, it employs a technology that gains a lot of attention recently, the optical measurement method. To finely introduce the optical elements into the electrical setup, CCD sensors were employed. The entire setup is controlled by a microcontroller, taking advantage of its various peripherals, from which the timers, the serial ports and the analog comparator were highlighted. The final prototype is implemented on a printed circuit board, arranged on four blocks, the "brain" represented by the microcontroller, the "eyes" depicted through the optical ensemble, and the "ears and mouth" used for receiving and transmitting information serially in the USB to UART conjunction. However, to be trustful, every information is processed, in this case, through filtering methods implemented using python programming language. This ensures the quality of the signal, resulting in the quality of the measurements taken.

As dam measurement systems are of utmost importance, affecting the daily lives of the population surrounding great masses of water, there is no room for errors or mistakes when creating such a prototype. However, there is always room for improvement, regardless of the type of project. This prototype has a series of improvements itself that can be adopted in the near future. Regarding the PCB assembly, a solution was found, but yet developed, for the better communication between the ADC and a microcontroller, caused by the mismatch in their clock frequency. This can be solved changing the oscillator crystal of the ADC block to one identical to the microcontroller's. In concern to the software interface implementation, new methods can be applied to achieve continuity using live updates and measurements of the signal itself, without the need for pressing any button. Once the module is mounted, functions can be applied to alert other elements of a greater system of sudden changes in position, discussed briefly through the RS485 block, that is yet to be properly functional.

Nonetheless, this prototype is the beginning of a much greater purpose, following to be interconnected to multiple devices in a larger network of dam measurements. Even if there are a few steps left towards its perfection, it represents a high accuracy measurement device implemented with reduced costs comparing it to the other prototypes present on the market, still having an increased quality.

8 References

[1] J. Xia *et al.*, "Portable vertical pendulum tiltmeter development and application test," *Acta Geodaetica et Geophysica*, vol. 54, no. 2, pp. 287–300, Jun. 2019, doi: 10.1007/s40328-019-00258-4.

[2] L. Zhao and E. M. Yeatman, "Micro capacitive tilt sensor for human body movement detection," in *IFMBE Proceedings*, 2007. doi: 10.1007/978-3-540-70994-7_34.

[3] L. Viman, M. Dabacan, I. Ciascai, and S. Pop, "Embedded microcontroller system for reading inductive telependulum," in *ISSE 2007 - 30th International Spring Seminar on Electronics Technology 2007; Conference Proceedings: Emerging Technologies for Electronics Packaging*, 2007. doi: 10.1109/ISSE.2007.4432893.

[4] B. Salvador, A. Luque, and J. M. Quero, "Microfluidic capacitive tilt sensor using PCB-MEMS," presented at the Proceedings of the IEEE International Conference on Industrial Technology, 2015. doi: 10.1109/ICIT.2015.7125596.

[5] J. Guo, P. Hu, and J. Tan, "Analysis of a segmented annular coplanar capacitive tilt sensor with increased sensitivity," *Sensors (Switzerland)*, vol. 16, no. 1, 2016, doi: 10.3390/s16010133.

[6] T. D. Dinh *et al.*, "Two-axis tilt angle detection based on dielectric liquid capacitive sensor," in *Proceedings of IEEE Sensors*, 2016. doi: 10.1109/ICSENS.2016.7808708.

[7] D. Kinet, P. Mégret, K. W. Goossen, L. Qiu, D. Heider, and C. Caucheteur, "Fiber Bragg grating sensors toward structural health monitoring in composite materials: Challenges and solutions," *Sensors (Switzerland)*, vol. 14, no. 4, 2014, doi: 10.3390/s140407394.

[8] C. R. Chao, W. L. Liang, and T. C. Liang, "Design and testing of a 2d optical fiber sensor for building tilt monitoring based on fiber bragg gratings," *Applied System Innovation*, vol. 1, no. 1, 2018, doi: 10.3390/asi1010002.

[9] "Advantages of Fresnel Lenses." Accessed: Jun. 21, 2024. [Online]. Available: https://www.edmundoptics.com/knowledge-center/application-notes/optics/advantages-of-fresnel-lenses/

[10] H. Helmers and M. Schellenberg, "CMOS vs. CCD sensors in speckle interferometry," *Optics & Laser Technology*, vol. 35, no. 8, pp. 587–595, Nov. 2003, doi: 10.1016/S0030-3992(03)00078-1.

[11] "CCD sensors." Accessed: Jun. 19, 2024. [Online]. Available: https://www.vision-doctor.com/en/camera-technology-basics/ccd-sensors.html

[12] D. Ibrahim, "Microcomputer systems," in *Arm-Based Microcontroller Multitasking Projects*, 2021. doi: 10.1016/b978-0-12-821227-1.00001-3.

[13] "AVR microcontrollers," *Wikipedia*. Jun. 06, 2024. Accessed: Jun. 21, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=AVR_microcontrollers&oldid=1227599727

[14] R. Pawson, "The Myth of the Harvard Architecture," *IEEE Annals of the History of Computing*, vol. 44, no. 3, 2022, doi: 10.1109/MAHC.2022.3175612.

[15] "Modified Harvard Architecture: Clarifying Confusion," IT Hare on Soft.ware. Accessed: Jun. 22, 2024. [Online]. Available: http://ithare.com/modified-harvard-architecture-clarifying-confusion/

[16] "AVR Microcontroller Architecture," TechniCodes. Accessed: Jun. 25, 2024. [Online]. Available: http://technicodes.weebly.com/avrarch.html

[17] "ATmega1284." Accessed: Jun. 25, 2024. [Online]. Available: https://www.microchip.com/en-us/product/atmega1284

[18] "MAX1193 Datasheet and Product Info | Analog Devices." Accessed: Jun. 23, 2024. [Online]. Available: https://www.analog.com/en/products/max1193.html

[19] "Communication Protocols In System Design," GeeksforGeeks. Accessed: Jun. 23, 2024. [Online]. Available: https://www.geeksforgeeks.org/communication-protocols-in-system-design/

[20] E. Arias-Castro and D. L. Donoho, "Does median filtering truly preserve edges better than linear filtering?," *Ann. Statist.*, vol. 37, no. 3, Jun. 2009, doi: 10.1214/08-AOS604.

[21] Abraham. Savitzky and M. J. E. Golay, “Smoothing and Differentiation of Data by Simplified Least Squares Procedures.,” *Anal. Chem.*, vol. 36, no. 8, pp. 1627–1639, Jul. 1964, doi: 10.1021/ac60214a047.

[22] D. Kaufman, “PCB Design Steps & Complete Guide | Cirexx,” Cirexx International. Accessed: Jun. 24, 2024. [Online]. Available: https://www.cirexx.com/pcb-design-steps/

[23] “The PCB Assembly Process.” Accessed: Jun. 25, 2024. [Online]. Available: https://education.altium.com/courses/1624787/lectures/37274286

[24] “TSL1412S ams - Linear ICs - Distributors, Price Comparison, and Datasheets | Octopart component search,” Octopart. Accessed: Jun. 26, 2024. [Online]. Available: https://octopart.com/tsl1412s-ams-55596344

9 Annexes

9.1 C code for programming the microcontroller

9.1.1 Functions.c

```
#include "Functions.h"
#include "UsedVariable.h"
#include "Functions.h"
#include "TSL1412S.h"
#include "uC_Periferic_Initialisation.h"
#include "stdlib.h"
#include <avr/wdt.h>
#define F_CPU 16000000UL
#include <util/delay.h>

unsigned char display_Chars_12[12]={'x','=','2', '5', '6' ,',', '9','9','D','C', 0x0D,
0x0A};

        uint8_t Index_Receptie_PC =0;
        uint8_t Index_Receptie_PC_OLD =0 ;



//input parameter
//unsigned char cmd  comanda de intrare x sau y ,
//unsigned long int valoare valoarea de trimis pe UDR0
//out void      cmd=|___|oEcrLf
//terminal apare x=921.230E  ASCII string  10+crlf

void Send_Value (unsigned char cmd, unsigned long int valoare)
{
       volatile unsigned char Local_vr = 0, Local_vr1 = 0, *point_display;
       volatile unsigned long int Diplay_Val=valoare;

              for (Local_vr=0;Local_vr <12; Local_vr++)  display_Chars_12[Local_vr] =48;


       if ((cmd =='x') ||(cmd =='y'))
       {
              display_Chars_12[0] =cmd;
              display_Chars_12[1] ='=';
              display_Chars_12[5] =0x2E;  //','

                           if (cmd =='x'){
                                  Diplay_Val = WirePosition.mm_X;
                                  Local_vr = (WirePosition.error_Xmm);//error x
error_Xmm
                            }
                           if (cmd =='y'){
                                 Diplay_Val = WirePosition.mm_Y;
                                 Local_vr = (WirePosition.error_Ymm);
                           }
              point_display =((display_Chars_12) + 8);
        point_display = itoa(Local_vr, point_display, 16);
              if ((Local_vr & 0xF0)==0)  display_Chars_12[9] = 48;

              display_Chars_12[10] =0x0A;
              display_Chars_12[11] =0x0D;
```

```
                        //Diplay_Val = valoare;
                        for (Local_vr=7;Local_vr > 1;Local_vr--)
                        {
                                if (Diplay_Val !=0)
                                {
                                        if (Local_vr!=5)//poz lui ','
                                        {
                                                display_Chars_12[Local_vr] =
(Diplay_Val%10)+48;
                                                Diplay_Val = Diplay_Val/10;
                                        }
                                }
                                else break;
                        }

                        for (Local_vr1=0;Local_vr1 <12; Local_vr1++)
                        {

                                Local_vr = UCSR0A &  0x20;
                                while(Local_vr ==0) Local_vr = UCSR0A & 0x20;
                                UCSR0A |= 0x20;
                                UDR0 = display_Chars_12[Local_vr1];
                        }
        }
        else  // afisez datele din memorie
        {
                for (Local_vr1=0;Local_vr1<80;Local_vr1++)
                {
                        //Local_vr = UCSR0A &  0x20;
                        //while(Local_vr ==0) Local_vr = UCSR0A & 0x20;
                        //UCSR0A |= 0x20;
                        //UDR0 = (INEL_DATE[data_int])>>8;
                        //
                        //Local_vr = UCSR0A &  0x20;
                        //while(Local_vr ==0) Local_vr = UCSR0A & 0x20;
                        //UCSR0A |= 0x20;
                        //UDR0 = INEL_DATE[data_int];
                }
        }
}
//////////////////////


////////////////////////////////////////////////////////////////////

unsigned char Transmission_function(unsigned char CMD)
{
    volatile uint16_t LocalData=0;
       uint8_t  *PointerSTR  ;

       //trec pe transmisie uC -> PC
       ///////////////////////*

       RS485_uC_Transmit;
       _delay_ms(50);//delay_ms(100);
       ////////////////////////

       BuffRx_PC.Antet = Antet_Transmisie_uC_To_PC; //uC
       BuffRx_PC.Adres_uC = Adr_uC; //uC
       BuffRx_PC.Cmd_PC = CMD; //uC  Stare_uC
       //BuffRx_PC.Adr_Trad = Reg_Adr_Trad; //uC  Stare_uC

       Index_Receptie_PC = 0;
```

```
            if(( CMD == CitesteVal_uC_x ) && (BuffRx_PC.Adr_Trad ==AXA_x))  //SelX   0A
            {
                  LocalData = WirePosition.mm_X;
                  BuffRx_PC.ValoareH = (LocalData & 0xff); //Temp LOW
                  LocalData = LocalData >> 8;
                  BuffRx_PC.ValoareL = LocalData; //Temp High
                  BuffRx_PC.OctetE = WirePosition.error_Xmm; //
        }
            if(( CMD == CitesteVal_uC_y ) && (BuffRx_PC.Adr_Trad ==AXA_y))  //SelY  0A
            {
                  LocalData = WirePosition.mm_Y;
                  BuffRx_PC.ValoareH = (LocalData & 0xff); //Temp LOW
                  LocalData = LocalData >> 8;
                  BuffRx_PC.ValoareL = LocalData; //Temp High
                  BuffRx_PC.OctetE = WirePosition.error_Ymm; //
            }
      ////////////XXXXX comanda instant
                  if(( CMD == Citire_Instant_x ) && (BuffRx_PC.Adr_Trad ==AXA_x))
//SelX   0A
                  {
                        LocalData = WirePosition.mm_X;
                        BuffRx_PC.ValoareH = (LocalData & 0xff); //Temp LOW
                        LocalData = LocalData >> 8;
                        BuffRx_PC.ValoareL = LocalData; //Temp High
                        BuffRx_PC.OctetE = WirePosition.error_Xmm; //
                  }
                  if(( CMD == Citire_Instant_y ) && (BuffRx_PC.Adr_Trad ==AXA_y))
//SelY  0A
                  {
                        LocalData = WirePosition.mm_Y;
                        BuffRx_PC.ValoareH = (LocalData & 0xff); //Temp LOW
                        LocalData = LocalData >> 8;
                        BuffRx_PC.ValoareL = LocalData; //Temp High
                        BuffRx_PC.OctetE = WirePosition.error_Ymm; //
                  }

      ////////////////////////////////////
            if(( CMD == CitesteMasTrad_uC) && ( BuffRx_PC.Adr_Trad ==AXA_x ) )  //citire
X
            {
                  LocalData = WirePosition.X_mm_mediu;
                  BuffRx_PC.ValoareH = (LocalData & 0xff); //Temp LOW
                  LocalData = LocalData >> 8;
                  BuffRx_PC.ValoareL = LocalData; //Temp High
                  BuffRx_PC.OctetE = WirePosition.errorXmediu; //
            }
            if(( CMD == CitesteMasTrad_uC) && ( BuffRx_PC.Adr_Trad ==AXA_y ) )  //citire
Y
            {
                  LocalData = WirePosition.Y_mm_mediu;
                  BuffRx_PC.ValoareH = (LocalData & 0xff); //Temp LOW
                  LocalData = LocalData >> 8;
                  BuffRx_PC.ValoareL = LocalData; //Temp High
                  BuffRx_PC.OctetE = WirePosition.errorYmediu; //
            }

      if( CMD == RESET_uC)
      {
            //Actiune = Actiune | Start_masura;  //urmeaza peste 16ms masuratoarea
      }

   LocalData = ((BuffRx_PC.Antet + BuffRx_PC.Adres_uC + BuffRx_PC.Cmd_PC +
BuffRx_PC.Adr_Trad + BuffRx_PC.ValoareH + BuffRx_PC.ValoareL + BuffRx_PC.OctetE) & 0xFF);
```

```
	BuffRx_PC.Reg_SC = (~LocalData) +1;

	//transmisie_PC efectiv
	ON_LED1;
	PointerSTR =  &BuffRx_PC.Antet;
	for ( Index_Receptie_PC = 0; Index_Receptie_PC < 8; Index_Receptie_PC ++)
	{
				LocalData = UCSR1A &  0x20;
				while(LocalData ==0) LocalData = UCSR1A & 0x20;
				UCSR1A |= 0x20;
		UDR1 = *PointerSTR ;
		PointerSTR++;
	}
					LocalData = UCSR1A &  (1<< TXC1);
					while(LocalData ==0) LocalData = UCSR1A &  (1<<
TXC1);
					UCSR1A |= 0x20;

	Index_Receptie_PC = 0;
	_delay_ms(10);//obtional
	RS485_uC_Receptie;
    OFF_LED1;
	return 0;
}
//////////////////////////////////////////////////////////////////

 void VerificPachete_Intirziate(void){
	uint16_t static Contor_local = 0 ;   //se incrementeaza la 4ms

		if (Index_Receptie_PC_OLD != Index_Receptie_PC)
		{
			Index_Receptie_PC_OLD = Index_Receptie_PC;
			Contor_local = 0;
		}
		//if ((Index_Receptie_PC_OLD == Index_Receptie_PC))
		else
			{
					Contor_local ++;
					if (Contor_local >= 10) //40ms x 20  163ms 325ms
//9  DACA SUNT INTRERUPERI IN RECEPTIE  (cond 2 == -> >=)
					{
						Index_Receptie_PC =0;
						Index_Receptie_PC_OLD =0;
						Contor_local = 0;
						RS485_uC_Receptie; // am completat in 2020
//obs la oasa se bloca la datele de la alt uC  (cond 2)
						wdt_reset();
					}
			}

}

// delay la start bling led de 4 ori marcheaza pornirea aplicatiei
void Semnalizare_Pornire(void)
{ uint8_t Repet_blink=0;
	//
	for(Repet_blink=0;Repet_blink < 4; Repet_blink++)
	{    wdt_reset();
		_delay_ms(300);//delay_ms(500);
		ON_LED1;
		 wdt_reset();
		_delay_ms(300);//delay_ms(500);
		OFF_LED1;
	}
```

```
}
//////////////////////////////////////

void SalvezDate( uint8_t AXA){
	volatile static uint8_t ContorX=0, ContorY=0;

	switch (AXA)
	{
		case 'x':
				//if (WirePosition.error_Xmm ==0)
				//{
					FirX.PozitiePixel[ContorX] = WirePosition.mm_X;
					FirX.PozitieError[ContorX] = WirePosition.error_Xmm;
					ContorX++;
					if (ContorX >= SaveDataLenght) ContorX =0;
				//}

		break;

		case 'y':
				//if (WirePosition.error_Ymm ==0)
				//{
					FirY.PozitiePixel[ContorY] = WirePosition.mm_Y;
					FirY.PozitieError[ContorY] = WirePosition.error_Ymm;
					ContorY++;
					if (ContorY >= SaveDataLenght) ContorY =0;
				//}
		break;
	}
}
//////  CALCULEZ MEDIA ULTIMELOR 32 DE DATE //ELIMIN DATELE ERONATE
	uint8_t MedieDate( uint8_t AXA){

		volatile uint8_t index_date=0,ErrorCount=0,DataCount=0;
		volatile uint32_t LocalDataMedie=0;
		volatile uint16_t *PointData ;
		volatile uint8_t *PointError ;

		 //for ( index_date=0; index_date < SaveDataLenght;index_date++)
		 //{
			//FirX.PozitiePixel[index_date] = 8000;
			//FirX.PozitieError[index_date] =0;
			////if ((index_date> 10) && (index_date<22))
FirX.PozitieError[index_date] =2;
		 //}
		 // incar tebelul cu masuratori in pointer
		switch(AXA){
			case 'x':
				PointData = (uint16_t) &FirX.PozitiePixel[0];
				PointError =(uint16_t) &FirX.PozitieError[0];
			break;
			case 'y':
				PointData =(uint16_t)&FirY.PozitiePixel[0];
				PointError =(uint16_t) &FirY.PozitieError[0];
			break;
			default:
								PointData =
(uint16_t)&FirX.PozitiePixel[0];
								PointError =
(uint16_t)&FirX.PozitieError[0];
		}
 // calculez media datelor cu eroare zero
		 LocalDataMedie = 0;
		 for ( index_date=0; index_date < (SaveDataLenght);index_date++)
```

```
                {
                        if ((*(PointError+index_date)) == 0)
                        {       LocalDataMedie+= *(PointData+index_date);
                                DataCount++;
                        }
                         //PointError++;
                         //PointData++;
                }
//verific cite date au fost cu eroare 0;
                if (DataCount >= (SaveDataLenght-4))
                {
                                if (AXA == 'x') {
                                        WirePosition.X_mm_mediu = LocalDataMedie/DataCount;    //
media valorilor
                                        WirePosition.errorXmediu = 0;                          //
media valorilor
                                        }
                                if (AXA == 'y') {
                                        WirePosition.Y_mm_mediu = LocalDataMedie/DataCount;
                                        WirePosition.errorYmediu = 0;
                                }
                        return OK;
                }
                else
                {
                                if (AXA == 'x')
                                {
                                        WirePosition.X_mm_mediu = WirePosition.mm_X;  //ultima
valoare
                                                if (DataCount >= (SaveDataLenght/2))
WirePosition.errorXmediu = MasuratoareIstabila;  //daca sunt peste 16 instabil
                                                else WirePosition.errorXmediu =
WirePosition.error_Xmm;  //daca sub 16 copiez eroarea
                                }
                                if (AXA == 'y')
                                {
                                        WirePosition.Y_mm_mediu = WirePosition.mm_Y;  //ultima
valoare
                                                if (DataCount >= (SaveDataLenght/2))
WirePosition.errorYmediu = MasuratoareIstabila;
                                                else WirePosition.errorYmediu =
WirePosition.error_Ymm;
                                }
                        return NotOK;
                }
                return NotOK;
        }
```

9.1.2 Functions.h

```
#ifndef __FUNCTIONS_H
#define __FUNCTIONS_H

#ifdef __cplusplus
extern "C" {
        #endif
        //////////////////////////____ Sensor TSL1410R pins declarations
////////////////////////////////////////////////
        //SI must go low before the rising edge of the next clock pulse.
        // SI -----|______
        // clk    _____|-----
        #include <avr/io.h>
```

```
        #include "UsedVariable.h"

        extern uint8_t display_Chars_12[12];

         extern uint8_t Index_Receptie_PC;
         extern uint8_t Index_Receptie_PC_OLD ;
        //extern uint8_t Reg_O_Er;
        //extern uint8_t Reg_Cmd_PC;
        //extern uint8_t Reg_SC;
        //extern uint8_t Reg_Adr_Trad;
        //



        void Send_Value (unsigned char cmd, unsigned long int valoare);
        unsigned char Transmission_function(unsigned char CMD);
        void VerificPachete_Intirziate(void);
        void Semnalizare_Pornire(void);
        void SalvezDate( uint8_t AXA);
        uint8_t MedieDate(uint8_t AXA);
               ////////////////////////////////////
        //#endif
        #ifdef __cplusplus
}
#endif

   #endif /* CHEADER_H_INCLUDED */
```

9.1.3 Main.cpp

```
#include <avr/io.h>
#include <avr/interrupt.h>


#include "UsedVariable.h"
#include "uC_Periferic_Initialisation.h"
#include "Functions.h"
#include "TSL1412S.h"
#include <avr/wdt.h>
#define F_CPU 16000000UL
#include <util/delay.h>
unsigned char CharData =0;
unsigned char PixelsValue_X[]="";
unsigned char PixelsValue_Y[]="";

uint8_t Actiune = 0;


//////////////
int main(void)
{
    /* Replace with your application code */

        Timer_1_Initialisation_0_1S();  //thred citire sensor

        Port_initialisation();         //initializare port
        RX_TX_0_Initialisation();      //initializare RX0

        Timer_0_Initialisation_4mS();   //thred determinare pachete intirziate
        Initializare_TX_RX_1_RS485();        //initializare RX1 RS485
```

```
              //watchdog timer is a simple countdown
              cli();
              wdt_reset();
              WDTCSR |= (1<<WDCE) | (1<<WDE);
         WDTCSR = ((1<<WDE) | (1<<WDP2) | (1<<WDP1) | (1<<WDP0));  //0.5s
              ///////// end watchdog///////
              Semnalizare_Pornire();
              RS485_uC_Receptie;
              Index_Receptie_PC = 0;
              Index_Receptie_PC_OLD=0;
              sei();
       //wdt_reset();
    while (1)
    {

    }//end while
} //end main
//ISR Rx
ISR(USART0_RX_vect) {

       //afisez 99999 = 999.99*100;  folosesc 2 zecimale
       //intreruperea de receptie seiriala
       uint16_t data=0;
       uint8_t Local_vr=0;
       ON_LED2;// testez rutina rx

       data = UDR0; //citesc comanda receptionata

       switch(data)
       {
              case 'x': //rezultat ADC
              Send_Value('x', WirePosition.N_X*6.35);
              break;
              case 'e'://Volt medie
              Send_Value('e', WirePosition.error_X);
              break;
              case 'm':// memoria x
                     for (data=0;data <MaXLenghtPixel; data++)
                     {
                            Local_vr = UCSR0A &  0x20;
                            while(Local_vr ==0) Local_vr = UCSR0A & 0x20;
                            UCSR0A |= 0x20;
                            UDR0 = (PixelsValue_X[data]+48); //conversie din hexa in ascii
                     }
              break;
              case 'n':// memoria y
                     for (data=0;data <MaXLenghtPixel; data++)
                     {
                            Local_vr = UCSR0A &  0x20;
                            while(Local_vr ==0) Local_vr = UCSR0A & 0x20;
                            UCSR0A |= 0x20;
                            UDR0 = (PixelsValue_Y[data]+48);
                     }
              //break;
       }

       OFF_LED2;
}
////////////// end  USART0_RX_vect  ////

// interrupt of 0.1 sec
ISR(TIMER1_COMPA_vect)
{
       volatile static uint16_t timerCounter=0;
```

```
	wdt_reset();
	timerCounter ++;
	if (timerCounter == 20)  //10
	{	//timerCounter=0;
		ON_LED1;//
		//wdt_reset();
			Captura_AxaX('x');
		OFF_LED1;
	}//end
	if (timerCounter >= 40)  //20
	{	ON_LED2;//
		//wdt_reset();
		timerCounter=0;
			Captura_AxaY('y');
		OFF_LED2;
	}//end
}//end isr
////////////////////

ISR(TIMER0_OVF_vect)
{
	uint16_t static Counter_1sec=0;
	wdt_reset();
	Counter_1sec++;

				if (Counter_1sec == 489)
				{
					//Trimite_Valoare('x',WirePosition.mm_X
);//WirePosition.mm_X  denisa
				}
				if (Counter_1sec >= 976)
				{ Counter_1sec =0;

	//Trimite_Valoare('y',WirePosition.mm_Y);//WirePosition.mm_X  denisa
				}


	//verific pachete intirziate
	VerificPachete_Intirziate();

	//calculez X/Y medie
	//raspund la comanda de pe rs485
			switch(Actiune)
			{
				case 'x':
					MedieDate('x');
					Actiune=0;
				break;

				case 'y':
					MedieDate('y');
					Actiune=0;
				break;
				case 'X':
					WirePosition.X_mm_mediu = WirePosition.mm_X;  //ultima
valoare
					WirePosition.errorXmediu = WirePosition.error_Xmm;
//daca sunt peste 16 instabil  //valoare instant X
					Actiune=0;
				break;
				case 'Y':
					WirePosition.Y_mm_mediu = WirePosition.mm_Y;  //valoare
instant Y
```

```
                                        WirePosition.errorYmediu = WirePosition.error_Ymm;
//valoare instant Y
                                        Actiune=0;
                                break;
                                default:
                                Actiune = 0;
                        }


}

ISR(USART1_RX_vect)   //receptie RS485
{
        volatile uint8_t Data_RX = 0;

         Data_RX = UCSR0A;
      if((Data_RX & ((1<<DOR1) |(1<<FE1) |(1<UPE1))) == 0) //start1//am receptie un
caracter
      {
              Data_RX = UDR1;

              Index_Receptie_PC++;
              UDR0 = Index_Receptie_PC;


              //if (Index_Receptie_PC <= 5) Index_Receptie_PC++;
              //else Index_Receptie_PC=1;


                      if((Index_Receptie_PC == 1) && (Data_RX != Antet_Receptie_uC))
Index_Receptie_PC = 6;

                      if (Index_Receptie_PC <= 5)   //start2  //trebuie sa tratez daca nu
vin intr-un interval de timp !!!!!
                      {
                              if (Index_Receptie_PC == 1) BuffRx_PC.Antet = Data_RX;

                              if((Index_Receptie_PC == 2) && (Data_RX != Adr_uC)) {
                                      Index_Receptie_PC = 6;
                              }
                  if (Index_Receptie_PC == 2) BuffRx_PC.Adres_uC = Data_RX;

                              if((Index_Receptie_PC >= 3) && (Index_Receptie_PC <=5))
                              {// start3


                                      if (Index_Receptie_PC == 3)  BuffRx_PC.Cmd_PC = Data_RX;
                                      if (Index_Receptie_PC == 4)  BuffRx_PC.Adr_Trad =
Data_RX;
                                      if (Index_Receptie_PC == 5)
                                      {
                                              BuffRx_PC.Reg_SC = Antet_Receptie_uC + Adr_uC +
BuffRx_PC.Cmd_PC + BuffRx_PC.Adr_Trad + Data_RX;
                                              Index_Receptie_PC = 0;

                                                      if (BuffRx_PC.Reg_SC == 0) {

                                                              if (( BuffRx_PC.Cmd_PC ==
CitesteVal_uC_x ) && (BuffRx_PC.Adr_Trad == AXA_x )) Actiune ='x';
                                                              if (( BuffRx_PC.Cmd_PC ==
CitesteVal_uC_y ) && (BuffRx_PC.Adr_Trad == AXA_y )) Actiune ='y';

                                                              if (( BuffRx_PC.Cmd_PC ==
Citire_Instant_x ) && (BuffRx_PC.Adr_Trad == AXA_x )) Actiune ='X';
```

```
                                                        if (( BuffRx_PC.Cmd_PC ==
Citire_Instant_y ) && (BuffRx_PC.Adr_Trad == AXA_y )) Actiune ='Y';


       Transmission_function(BuffRx_PC.Cmd_PC);


                                                       }

                                  }
                             } //end start3
                         }//end start2
                  } //end start1
       //else Index_Receptie_PC = 0;

       //if (Index_Receptie_PC > 5) Index_Receptie_PC = 0;//deocamdata

   }
```

9.1.4 TLS1412S.c

```
#include "TSL1412S.h"
#include "UsedVariable.h"
#include "uC_Periferic_Initialisation.h"
#include "Functions.h"

//sensor signals
uint8_t  TSL_SI_HOLD_x = 2;        // bit 2
uint8_t TSL_SO_x = 1;                     // bit 1
uint8_t TSL_AO_x = 0;                     // bit 6
uint8_t TSL_SI_HOLD_y = 5;         // bit 4
uint8_t TSL_SO_y = 4;                     // bit 5
uint8_t TSL_AO_y = 3;                     // bit 3
uint8_t TSL_CLK_y = 6;                    // bit 7
uint8_t TSL_CLK_x = 7;                    // bit 7

//sensor signals end

//uint8_t TSL_Port_initalization(uint8_t TSL_SI_HOLD,uint8_t TSL_SO ,uint8_t TSL_AO ){
//volatile  uint8_t TSL_local_data=0;
 ////reset port
//
//
 //
//TSL_local_data = (TSL_DDR_uC & (~(1<<TSL_SI_HOLD)| (1<<TSL_SO) | (1<<TSL_CLK) |
(1<<TSL_AO)));
//TSL_local_data |= ((1<<TSL_SI_HOLD)| (0<<TSL_SO) | (1<<TSL_CLK) | (0<<TSL_AO));
                  ////TSL_SI_HOLD=out  TSL_SO =int   TSL_CLK = out    TSL_AO=z analog
//TSL_DDR_uC = TSL_local_data;
       //
//TSL_local_data = (TSL_PORT_uC & (~(1<<TSL_SI_HOLD)| (1<<TSL_SO) | (1<<TSL_CLK) |
(1<<TSL_AO)));
//TSL_local_data |= ((0<<TSL_SI_HOLD)| (0<<TSL_SO) | (1<<TSL_CLK) | (0<<TSL_AO));
//TSL_PORT_uC = TSL_local_data;
////check  TSL_SO = low
////wait for pin TSL_SO = to be low
//TSL_nop();
//
//
 //
        //TSL_local_data = (TSL_PinPort_uC & (1<<TSL_SO)) ;
             //if ((TSL_local_data) != 0 ) return NotOK; //0x10
```

```
            //else return OK;//if SO=0 return OK;TSL_SO
//}

//xxxxxxxxxxxxxxx
uint8_t TSL_Port_initalization(uint8_t TSL_SI_HOLD,uint8_t TSL_SO ,uint8_t TSL_AO ,
uint8_t TSL_clk){
       volatile  uint8_t TSL_local_data=0;
       //reset port



       TSL_local_data = (TSL_DDR_uC & (~((1<<TSL_SI_HOLD)| (1<<TSL_SO)  | (1<<TSL_clk) |
(1<<TSL_AO))));
       TSL_local_data |= ( (1<<TSL_SI_HOLD)| (0<<TSL_SO) | (1<<TSL_clk) | (0<<TSL_AO) );
       //TSL_SI_HOLD=out  TSL_SO =int   TSL_CLK = out    TSL_AO=z analog
       TSL_DDR_uC = TSL_local_data;//directie

       TSL_local_data = (TSL_PORT_uC & (~((1<<TSL_SI_HOLD) | (1<<TSL_SO) | (1<<TSL_clk) |
(1<<TSL_AO))));
       TSL_local_data |= ((0<<TSL_SI_HOLD)| (0<<TSL_SO) | (1<<TSL_clk) | (0<<TSL_AO));
       TSL_PORT_uC = TSL_local_data;
       //check  TSL_SO = low
       //wait for pin TSL_SO = to be low
       TSL_nop();

       TSL_local_data = (TSL_PinPort_uC & (1<<TSL_SO)) ;
       if ((TSL_local_data) != 0 ) return NotOK; //0x10
       else return OK;//if SO=0 return OK;TSL_SO
}
//xxxxxxxxxxxxxxx


//xxxxxxxxxxxxxxxxxx
uint8_t  ReadPozitionfromTSL(uint8_t TSL_SI_HOLD,uint8_t TSL_SO, uint8_t TSL_clk, uint8_t
XorY){
       volatile uint16_t clock_index =0;
       volatile uint8_t  RP_local_data =0;

       //test SO low
                           //RP_local_data = TSL_PinPort_uC;//(TSL_PinPort_uC &
(1<<TSL_SO)) ;
                           //RP_local_data &= (1<<TSL_SO) ;  // ar trebui sa fie 0 daca
nu-i mai astept ALTA DATA


       //RP_local_data =
       TSL_PORT_uC |=((1<<TSL_clk));// clk=1;          0____/----1
       TSL_nop();
       TSL_PORT_uC &=(~(1<<TSL_clk));// clk=0;          1----\____0
       TSL_nop();
       TSL_PORT_uC |=(1<<TSL_SI_HOLD);// SI=1;;      __0_______/----1
       TSL_nop();
       TSL_nop();
       TSL_nop();

       for (clock_index =0;clock_index < (TSL_pixel_lenght + 300 ); clock_index++)  //10
cresc integrasion time
       //for (clock_index =0;clock_index < (TSL_pixel_lenght-1); clock_index++)  //10 cresc
integrasion time
       {
              TSL_PORT_uC |=((1<<TSL_clk));// clk=1;          0____/----1
              TSL_nop();
              TSL_nop();
              TSL_nop();
```

```
                TSL_nop();
                TSL_nop();

                TSL_PORT_uC &=(~((1<<TSL_SI_HOLD)));// SI=0;;      1----\_________0
                TSL_nop();

                TSL_PORT_uC &=(~(1<<TSL_clk));// clk=0;          1----\____0
                TSL_nop();
                //TSL_nop();   //one ore more; to have an certain settling time of min120ns
                ////////////////////
                //Red analog out of sensor or pixel value
                if (clock_index < TSL_pixel_lenght) ReadPixelValuefromTSL(clock_index, XorY);
                ////////////////////
                //if(clock_index == (TSL_pixel_lenght-1))//TSL_pixel_Half_lenght
                if((clock_index >= (TSL_pixel_lenght-5)) && (RP_local_data
==0))//TSL_pixel_Half_lenght  //testez inainte cu 5 pixeli de final daca vine SO
                {
                        TSL_nop();
                        RP_local_data = TSL_PinPort_uC;//(TSL_PinPort_uC & (1<<TSL_SO)) ;
                        RP_local_data &= (1<<TSL_SO) ;

                }
                TSL_nop();
        }
        //////end for
        //last value for clk and SI hold 0;
        TSL_PORT_uC |=((1<<TSL_SI_HOLD));// TSL_SI_HOLD=1;          0____/----1
        TSL_nop();
        TSL_PORT_uC |=((1<<TSL_clk));// TSL_SI_HOLD=1;          0____/----1
        TSL_nop();
        TSL_nop();
        TSL_nop();
        TSL_nop();


        TSL_PORT_uC &=(~((1<<TSL_SI_HOLD)));// TSL_SI_HOLD=0;          1----\____0
        TSL_nop();
        TSL_DDR_uC = 0xE4;
        TSL_PORT_uC = 0x1B;
        //TSL_PORT_uC = 0x1B;//(~(1<<TSL_clk));// TSL_CLK=0;;      1----\_________0
        //                          TSL_PORT_uC &=(~((1<<TSL_SI_HOLD)|(1<<TSL_CLK)));//
SI=Clk=0;;      1----\_________0

                            if ((RP_local_data) != 0 ) return OK; // if SO=1 return
OK;TSL_SO
                            else return NotOK;

}



void TSL_nop(void){
                            asm ("nop");
                            asm ("nop");
                            asm ("nop");
                            asm ("nop");
                            asm ("nop");
                            asm ("nop");
                            asm ("nop");
                            asm ("nop");
                            asm ("nop");
                            asm ("nop");
        }
```

```
//////////////////////   end -nop //////////////////////

//////////////////////   ReadPixelValuefromTSL //////////////////////
//  valoarea pizelului "PixelPosition" este incarcata in sirul PixelsValue pe pozitia
PixelPosition
//
void  ReadPixelValuefromTSL(uint16_t PixelPosition,uint8_t XorY)
{
       //PixelsValue[PixelPosition] =PixelPosition;
       if (XorY =='y')//trebuie inversat pt ca pe scanarea lui x citesc de fapt y si invers
       {
                                    //if ((PixelPosition > 100) && (PixelPosition < 150))
PixelsValue_Y[PixelPosition] =0;
                                    //else  PixelsValue_Y[PixelPosition] =2;

              if ((ACSR & (1<<ACO)) !=0) PixelsValue_Y[PixelPosition] =0;//0;//0;
              else PixelsValue_Y[PixelPosition] =1;//1;//1;
       }
       if (XorY =='x')
       {
                                    //if ((PixelPosition > 500) && (PixelPosition < 550))
PixelsValue_X[PixelPosition] =0;
                                    //else  PixelsValue_X[PixelPosition] =1;
              if ((ACSR & (1<<ACO)) !=0) PixelsValue_X[PixelPosition] =0;//0;//0;
              else PixelsValue_X[PixelPosition] =1;//1;//1;
       }


}
//////////////////////   end -ReadPixelValuefromTSL //////////////////////

////input = pointer to the string  1280 lenght
////Return wire position (the middle wire position) as index pixel of 63um
1280*63um=80.640mm
uint8_t DetermineWirePosition( uint8_t XorY){
       volatile uint16_t WireWidth=0;
    volatile uint16_t ShadowWidth=0;
       volatile uint8_t count_SmalShadow=0,count_LargeShadow=0;
       volatile uint16_t OldPixelsOrder=0, PixelsOrder=0;
       volatile uint8_t * PointerPixelArray;

       //RESETEZ POZITIA SI EROAREA CORESPUNZATOARE DIRECTIE CITITE
       if (XorY == 'x')
       {      WirePosition.mm_X = 0;
              WirePosition.N_X = 0;
              WirePosition.error_X = 0;
              PointerPixelArray = PixelsValue_X; //pointer la situl lui X
       }
       if (XorY == 'y')
       {   WirePosition.mm_Y = 0;
              WirePosition.N_Y = 0;
              WirePosition.error_Y = 0;
              PointerPixelArray = PixelsValue_Y; //pointer la situl lui Y

       }
//////////////////////////////////////////////////////
//-> INCEP DETERMINAREA POZITIEI
              PixelsOrder = 0;
              //for (PixelIndex =0;PixelIndex < TSL_pixel_lenght ; PixelIndex++)
              while(( PixelsOrder) < (MaXLenghtPixel-20))  //nu citesc pina la ultimi
pixeli s-ara fie zero   1111 fir 11000
              {
                     if ((*(PointerPixelArray +PixelsOrder))< 1 )  // valoare pixel mai mic
de 1/ pixel value low that 1
```

```
                    {
                OldPixelsOrder = PixelsOrder;
                            WireWidth=0;//fir size=0
                            while (((*(PointerPixelArray +PixelsOrder))< 1) && (WireWidth <
MaxWireWidth) && (PixelsOrder < MaXLenghtPixel)) // valoare pixel mai mic de 1
                            {
                                    PixelsOrder++;  //increment pointer
                                    WireWidth ++;  //determin dimensiune umbrei /shadow with
                                    ShadowWidth++;//determin dimensiune umbrey
totale+eroare/shadow with with

                            }
                                    if ((WireWidth > NoisWireWidth) && (WireWidth <
MinimWireWidth))
                                    {   count_SmalShadow++;  // riplu mic
                                       if (XorY == 'x') WirePosition.error_X =
smal_shadow;
                                       if (XorY == 'y') WirePosition.error_Y =
smal_shadow;
                                       WireWidth = 0;
                                    }//end if min
                                    if (WireWidth > MaxWireWidth)
                                    {   count_LargeShadow++;
                                       if (XorY == 'x') WirePosition.error_X =
large_shadow;
                                       if (XorY == 'y') WirePosition.error_Y =
large_shadow;
                                       return NotOK;
                                    }//end if max
                                    if ((WireWidth >= MinimWireWidth) && ((WireWidth <=
MaxWireWidth)))
                                    {   count_LargeShadow ++; //riplu mare
                                       if (XorY == 'x')
                                       {      WirePosition.N_X = (OldPixelsOrder
+(WireWidth>>1)-1);
                                              WirePosition.error_X = OK_shadow;
                                       }
                                       if (XorY == 'y')
                                       {   WirePosition.N_Y = (OldPixelsOrder
+(WireWidth>>1)-1);
                                              WirePosition.error_Y = OK_shadow;
                                       }
                                       WireWidth =0;

                                    }  //end if compute position
                        }//end if valoare pixel mai mic de 1
                        PixelsOrder++;
                }//end while  // AM TERMINAT DE DETERMINAT POZITIA

// OARE AM DETERMINAT CEVA TESTEZ POTITIA RESPECTIV EROAREA PINA AICI
                        if(( XorY == 'x') && (ShadowWidth !=0) && (WirePosition.N_X == 0) &&
(WirePosition.error_X == 0))
                        {    WirePosition.error_X=LipsaFir;
                              return NotOK ;
                        }
                        if(( XorY == 'y') && (ShadowWidth !=0) && (WirePosition.N_Y == 0) &&
(WirePosition.error_Y == 0)){
                               WirePosition.error_Y=LipsaFir;
                              return  NotOK;
                        }
//////////////////////////////////////////   IES CU LIPSA FIR

                        if ((ShadowWidth >(MaxWireWidth+(MaxWireWidth/2))) ||(count_SmalShadow
>40))
```

```
                        {
                                if (XorY == 'x') WirePosition.error_X = Noise_shadow;
                                if (XorY == 'y') WirePosition.error_Y = Noise_shadow;
                                return  NotOK;
                        }

                        if ((ShadowWidth >(MaxWireWidth+(MaxWireWidth/2)))
||(count_LargeShadow >10))
                        {
                                if (XorY == 'x') WirePosition.error_X = numerous_shadow;
                                if (XorY == 'y') WirePosition.error_Y = numerous_shadow;
                                return  NotOK;
                        }
                        else return  OK      ;
}
//input parameter [PixelOrder] wire position in order number[1-1280],axis x or y  [xy]
//return true if calculation is OK NotOK if error
//load x, y value of wire position in [WirePosition.mm_X] [WirePosition.mm_Y]

uint8_t CalculezPozitieFirMM( uint16_t PixelOrderCalc, uint8_t xyClac)
{
        volatile uint8_t error_in;

        switch(xyClac){
                        case 'x':
                                error_in = WirePosition.error_X;
                        break;
                        case 'y':
                                error_in = WirePosition.error_Y;
                        break;
                        default:
                        return NotOK;
                        }
        //error_in = 0;
        //if ( (error_in ==0) && (PixelOrder > MinimWireWidth) && (PixelOrder <
(MaXLenghtPixel - MaxWireWidth)))  //[15  1580-63]
   if ((error_in != smal_shadow) && (PixelOrderCalc > MinimWireWidth) && (PixelOrderCalc <
(MaXLenghtPixel - MaxWireWidth)))  //[15  1580-63]

        {               switch(xyClac){
                                        case 'x':
                                                WirePosition.mm_X = PixelOrderCalc*Pixelsixe_uM;
                                                WirePosition.error_Xmm = (error_in );
                                        break;
                                        case 'y':
                                                WirePosition.mm_Y = PixelOrderCalc*Pixelsixe_uM;
                                                WirePosition.error_Ymm = (error_in );
                                        break;
                                        }
                        return OK;
        }
        else
        {
                switch(xyClac){
                        case 'x':
                                        WirePosition.error_Xmm = LipsaFir;
                        break;
                        case 'y':
                                        WirePosition.error_Ymm = LipsaFir;
                        break;
                return NotOK;
                }
        }
        return NotOK;
```

```
}

// void  function
//initializare comp, initializare port TSL, led on Citire, calcul + tx uder0
void Captura_AxaX(uint8_t x_cmd){
	volatile uint16_t local_sterg=0;

		WirePosition.error_X = 0; //clear flag error
		WirePosition.error_Xmm = 0;
		WirePosition.N_X = 0;

// sterg citirile anterioare
		for(local_sterg =0;local_sterg < TSL_pixel_lenght; local_sterg++)
{PixelsValue_X[local_sterg] = 0;}

			AnalogComparator_Initialisation(Channel_AO_x); //admux0Channel_AO_x
			ON_LED_x;

			//TSL_Port_initalization(TSL_SI_HOLD_x, TSL_SO_x , TSL_AO_x, TSL_CLK_x
);
			if(TSL_Port_initalization(TSL_SI_HOLD_x, TSL_SO_x , TSL_AO_x,
TSL_CLK_x ) !=NotOK)
			{
				//ReadPozitionfromTSL(TSL_SI_HOLD_x, TSL_SO_x, TSL_CLK_x, 'x');
				//ReadPozitionfromTSL(TSL_SI_HOLD_x, TSL_SO_x, TSL_CLK_x, 'x');
				//ReadPozitionfromTSL(TSL_SI_HOLD_x, TSL_SO_x, TSL_CLK_x, 'x');
denisa

				//ReadPozitionfromTSL(TSL_SI_HOLD_x, TSL_SO_x, TSL_CLK_x, 'x');
				//WirePosition.N_X = 500;
				if( ReadPozitionfromTSL(TSL_SI_HOLD_x, TSL_SO_x, TSL_CLK_x,
'x') !=NotOK)
				 {	OFF_LED_x;
					OFF_LED_y; //adaugat 12.23

				   Comparator_Disable;
					Comparator_MuxDisable;
				   DetermineWirePosition('x');
					//if(
	DetermineWirePosition(PixelsValue,x_cmd)!=NotOK)
					//{ON_LED2;

					CalculezPozitieFirMM(WirePosition.N_X,'x');
				 }
			    else WirePosition.error_X = TSL_dont_answar;
			}
			else WirePosition.error_X = TSL_init_faild;//{//TSL initialization
failed


		local_sterg = PINC;
		local_sterg &= (1<<LED_X);  // am ajuns aici su nu am sters ledul ii bai
		if (local_sterg !=0)
		{
			OFF_LED_x;
			WirePosition.error_Xmm = ErrorLed;
		}
	//WirePosition.mm_X = 500;
	//Trimite_Valoare('x',WirePosition.mm_X);//WirePosition.mm_X
	//salvez datele pentru filtrare suplimentara
	SalvezDate('x');
}

//initializare comp, initializare port TSL, led on Citire, calcul + tx uder0
```

```
void Captura_AxaY(uint8_t y_cmd){
	volatile uint16_t local_sterg=0;

	WirePosition.error_Y = 0; //clear flag error
	WirePosition.error_Ymm = 0;
	WirePosition.N_Y = 0;

	for(local_sterg =0;local_sterg < TSL_pixel_lenght; local_sterg++)
{PixelsValue_Y[local_sterg] = 0;}

		AnalogComparator_Initialisation(Channel_AO_y); //admux0Channel_AO_y
		ON_LED_y;

				//TSL_Port_initalization(TSL_SI_HOLD_y, TSL_SO_y , TSL_AO_y,
TSL_CLK_y );
				if(TSL_Port_initalization(TSL_SI_HOLD_y, TSL_SO_y , TSL_AO_y,
TSL_CLK_y ) !=NotOK) //TSL_Port_initalization(TSL_SI_HOLD_y, TSL_SO_y , TSL_AO_y );
				{
					//ReadPozitionfromTSL(TSL_SI_HOLD_y, TSL_SO_y,
TSL_CLK_y, 'y');
					//ReadPozitionfromTSL(TSL_SI_HOLD_y, TSL_SO_y,
TSL_CLK_y, 'y');  denisa
					if( ReadPozitionfromTSL(TSL_SI_HOLD_y, TSL_SO_y,
TSL_CLK_y, 'y') !=NotOK) //ReadPozitionfromTSL(TSL_SI_HOLD_y, TSL_SO_y);
					{
						OFF_LED_y;
						OFF_LED_x; //adaugat 12.23
						Comparator_Disable;
						Comparator_MuxDisable;
						DetermineWirePosition('y');
						//if(
	DetermineWirePosition(PixelsValue,x_cmd)!=NotOK)
						//{ON_LED2;
						CalculezPozitieFirMM(WirePosition.N_Y,'y');
					}
					else WirePosition.error_Y = TSL_dont_answar;
				}
				else WirePosition.error_Y = TSL_init_faild;//{//TSL
initialization failed

						local_sterg = PINC;
						local_sterg &= (1<<LED_Y);  // am ajuns aici su
nu am sters ledul ii bai
						if (local_sterg !=0)
						{
							OFF_LED_y;
							WirePosition.error_Ymm = ErrorLed;
//defapt este eroare pe Y
						}
				//Trimite_Valoare('y',WirePosition.N_Y );//WirePosition.mm_X
				SalvezDate('y');
   }
```

9.1.5 TSL1412S.h

```
#ifndef __TSL1412S_H
#define __TSL1412S_H

#ifdef __cplusplus
extern "C" {
	#endif
```

```
//////////////////////____ Sensor TSL1410R pins declarations
////////////////////////////////////////////////
//SI must go low before the rising edge of the next clock pulse.
// SI -----|______
// clk   _____|-----
#include <avr/io.h>
//sensor signals
extern uint8_t  TSL_SI_HOLD_x;           // bit 2
extern uint8_t TSL_SO_x ;                // bit 1
extern uint8_t TSL_AO_x ;                // bit 6
extern uint8_t TSL_SI_HOLD_y ;           // bit 4
extern uint8_t TSL_SO_y ;                // bit 5
extern uint8_t TSL_AO_y ;                // bit 3
extern uint8_t TSL_CLK_y ;               // bit 7
extern uint8_t TSL_CLK_x ;               // bit 8

//sensor signals end

#define TSL_pixel_lenght 1536 //1280
#define TSL_pixel_Half_lenght 768 //640
//wire dimension in pixels
#define NoisWireWidth 5
#define MinimWireWidth 11
#define MaxWireWidth 63
#define Pixelsixe_uM 6.35//63.5 //um


#define TSL_PORT_uC PORTA
#define TSL_DDR_uC DDRA
#define TSL_PinPort_uC PINA



//////////////////////____ End Sensor TSL1410R pins
////////////////////////////////////////////////

uint8_t TSL_Port_initalization(uint8_t TSL_SI_HOLD,uint8_t TSL_SO ,uint8_t TSL_AO, uint8_t
TSL_clk );
uint8_t  ReadPozitionfromTSL(uint8_t TSL_SI_HOLD,uint8_t TSL_SO, uint8_t TSL_clk, uint8_t
XorY);
void  ReadPixelValuefromTSL(uint16_t PixelPosition,uint8_t XorY);
uint8_t DetermineWirePosition( uint8_t XorY);
uint8_t CalculezPozitieFirMM( uint16_t PixelOrderCalc, uint8_t xyClac);
void Captura_AxaX(uint8_t x_cmd);
void Captura_AxaY(uint8_t y_cmd);

void TSL_nop(void);

//#endif
#ifdef __cplusplus
}
#endif

   #endif /* CHEADER_H_INCLUDED */
```

9.1.6 uC_Periferic_Initialisation.c

```
#include <avr/io.h>
#include "uC_Periferic_Initialisation.h"

void Port_initialisation(){
```

```
  DDRB |=((1<<1)|(1<<0));//led 0, 1  flag_LED
  PORTB |=((1<<1)|(1<<0));;//led 0, 1 flag_LED

  DDRD |=(1<< PORTD4);    //Enable RS485
  RS485_uC_Receptie;    //Enable RS485
}


 void AnalogComparator_Initialisation( char channel){

        Comparator_Enable;//ACSR &= (1<<ACD);  //enable Compar;
        //ACSR |= (1<<ACBG); // AIN0=referinta 1.1v AIN1 for sensor
        ADC_off; //ADC off
        ADMUX = channel;
        ADCSRB |=(1<< ACME);
        _delay_ms(5);
        ;}

 void Timer_1_Initialisation_0_1S(void){
        TCCR1A = 0;
        TCCR1B = ((1<<WGM12)|(1<<CS12)|(0<<CS11)|(0<<CS10));  //CTC foscil/256;
        OCR1A = 31250;
        TIMSK1 |= (1<<OCIE1A);
 }


 void RX_TX_0_Initialisation(void){
       UCSR0B  = 0b10011000;
       UBRR0H = 0;    //
       UBRR0L = 103;  //9600biti/sec
 }

 void Timer_0_Initialisation_4mS (void){

        TCCR0B = (1 << CS02);// 0x05;  //0.01636,//   /
        TCNT0=0x00;
        TIMSK0 |= (1<<TOIE0);
 }

 void Initializare_TX_RX_1_RS485 (void){
        //transmisie/receptie 1200biti pe secunda
        UCSR1A = 0x00;
        //UCSRB = 0b10011000;
        UCSR1B=0x98;
        UCSR1C=0x86;
        UBRR1H=0x03;
        UBRR1L=0x40;
 }
```

9.1.7 uC_Periferic_Initialisation.h

```
#ifndef __uC_Periferic_Initialisation_H
#define __uC_Periferic_Initialisation_H

#ifdef __cplusplus
extern "C" {
       #endif
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
```

```
//flag seemnalizare
       #define ON_LED1 PORTB &= 0x0FE
       #define OFF_LED1 PORTB |= 0x01
       #define ON_LED2 PORTB &= 0x0FD
       #define OFF_LED2 PORTB |= 0x02

//LED_X/Y
    #define LED_X 0
       #define LED_Y 1
       #define ON_LED_x PORTC |= 0x01
       #define OFF_LED_x PORTC &= 0xFE
       #define ON_LED_y PORTC |= 0x02
       #define OFF_LED_y PORTC &= 0xFD


         #define ADC_off  ADCSRA &= (~((1<<ADEN) |(1<<ADSC)))
         #define Channel_AO_x 0
         #define Channel_AO_y 3
         #define Comparator_Enable ACSR &= (~(1<<ACD))  //enable Compar;
         #define Comparator_Disable ACSR |= (1<<ACD)
         #define Comparator_MuxDisable ADCSRB &= 0xB0 //(~(1<<ACME))



 //#define Start_masura 0x80;// IS CU SAU LOGIC
// #define Start_Transmisie 0x40;


 #define RS485_uC_Receptie PORTD |= (1<<PORTD4)
 #define RS485_uC_Transmit PORTD &=(~(1<<PORTD4))
 ////////////////////////////////////////
void Port_initialisation();
void   AnalogComparator_Initialisation( char channel);
void Timer_1_Initialisation_0_1S(void);
void RX_TX_0_Initialisation(void);
void Initializare_TX_RX_1_RS485(void);
void Timer_0_Initialisation_4mS (void);

//#endif
#ifdef __cplusplus
}
#endif

   #endif /* CHEADER_H_INCLUDED */
```

9.1.8 UsedVariable.h

```
#ifndef __UsedVariable_H
#define __UsedVariable_H

#ifdef __cplusplus
extern "C" {
       #endif
//////////////////////////////////////////////////////////////////////////
#include "TSL1412S.h"

#include <avr/wdt.h>
//#define     wdt_reset()  ;// __asm__ __volatile__ ("wdr")

#define OK 1
#define NotOK 0
```

```
#define SaveDataLenght 32


 //////////////  RX1   ////////////////////////
 //#################    ADRESA MICROCONTROLLERULUI   ###################33
 #define Adr_uC  0x71//;FD    //0x71, 0x72; 0x73
 //#########################################################33
 #define AXA_x  0
 #define AXA_y  1

 //#################    CMD MICROCONTROLLERULUI   ###################33
 #define RESET_uC 0x00

 #define CitesteMasTrad_uC 0x05 //cit
 #define CitesteVal_uC_x 0x0A  //sel
 #define CitesteVal_uC_y 0x0A  //sel

 #define Citire_Instant_x 0x0B
 #define Citire_Instant_y 0x0B
 //#define CitesteVal_uC_3 0x03
 //
 //#define CitesteVal_M_1 0x19
 //#define CitesteVal_M_2 0x20

 #define Antet_Receptie_uC 0x25
 #define Antet_Transmisie_uC_To_PC 0x21
 #define Selectie_manuala 0x05


struct FiltruPixel{
       uint16_t PozitiePixel[SaveDataLenght];
       uint8_t PozitieError[SaveDataLenght];
       } FirX, FirY ;



struct Position{
       uint16_t N_X;
       uint8_t error_X;
       uint16_t N_Y;
       uint8_t error_Y;
       uint16_t mm_X;
       uint8_t error_Xmm;
       uint16_t mm_Y;
       uint8_t error_Ymm;
       uint16_t X_mm_mediu;
       uint8_t errorXmediu;
       uint16_t Y_mm_mediu;
       uint8_t errorYmediu;
       } WirePosition;

       struct Receptie{
              uint8_t Antet;
              uint8_t Adres_uC;
              uint8_t Cmd_PC;
              uint8_t Adr_Trad;
              uint8_t ValoareH;
              uint8_t ValoareL;
              uint8_t OctetE;
              uint8_t Reg_SC;
       } BuffRx_PC;

       //////////////
       //error pixel position message
```

```
        #define OK_shadow 0x00  // ok
        #define smal_shadow 0x40 // fir pre subtire
        #define large_shadow 0x41 // umbra prea mare
        #define numerous_shadow 0x42 // umbre multiple
        #define Noise_shadow 0x43  //umbra zgomotoaza
        #define TSL_dont_answar 0x44//0x06  // senzor nu raspunde
     #define TSL_init_faild 0x45  // initializare esuata
     #define ErrorLed 0x46   // led defect
        #define MasuratoareIstabila 0x03  // masuratoare instabila
        //error pixel position MM message
        //#define Error_mm_outOfRange 0xC0  //depasire domeniu
        #define LipsaFir 0x35  //Fir lipsa

////////////////////////////////////////////////////////////////////////
#define MaXLenghtPixel TSL_pixel_lenght//1536//1280

extern unsigned char CharData;
extern unsigned char PixelsValue_X[TSL_pixel_lenght];
extern unsigned char PixelsValue_Y[TSL_pixel_lenght];

//extern

extern  uint8_t Actiune;

//////////////////////////////////////

//////////////////////////////////////
//#endif
#ifdef __cplusplus
}
#endif

   #endif /* CHEADER_H_INCLUDED */
```

9.2 Python code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 11 16:04:39 2024

@author: Denisa
"""

import customtkinter as ctk
from tkinter import messagebox
from tkinter import ttk
import serial
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from scipy.signal import savgol_filter
import numpy as np
import time

# Initialize serial connection
ser = serial.Serial('COM9', baudrate=9600, timeout=2)
time.sleep(0.2)
```

```
# Global variable to store the data
y_final = None
result_sav = None  # Initialize result_sav globally

# Function to send 'm' to Arduino
def start_reading_x():
    start_char = 'm'
    ser.write(start_char.encode())  # Send the start character
    read_data()
    messagebox.showinfo("Info", "Started reading x axis")

def start_reading_y():
    start_char = 'n'
    ser.write(start_char.encode())  # Send the start character
    read_data()
    messagebox.showinfo("Info", "Started reading y axis")

def read_data():
    global y_final
    global result_sav #filtered result
    y = ser.readline()
    print("number of characters received", len(y))
    print("Output type: ", type(y))
    print(f"Received non-numeric data: {y}")

    # Decode the bytes object to string and strip any leading/trailing whitespace
    y_str = y.decode('utf-8').strip()
    print("Output type: ", type(y_str))

    # Convert the string into a list of characters
    y_list = list(y_str)
    print("Output type: ", type(y_list))

    # Convert the list of characters to a list of integers (0s and 1s)
    y_final = np.array([int(char) for char in y_list])
    print("Output type: ", type(y_final))
    print("Final result", y_final)

    # Calculate filtered data
    result_sav = savgol_filter(y_final, 40, 2)
    print("Filtered data:", result_sav)

    #calculate_position()

# Function to plot original data
def plot_original():

    if y_final is None:
        messagebox.showerror("Error", "No data to plot. Press 'Start' first.")
        return

    fig, ax = plt.subplots()
```

```
        ax.plot(y_final)
        ax.set_title("Original")
        ax.set_ylim(-0.5, 1.5)  # Ensure 0 is below 1 on the y-axis

        # Embed the plot in the tkinter window
        canvas = FigureCanvasTkAgg(fig, master=plot_frame)
        canvas.draw()
        canvas.get_tk_widget().pack(side=ctk.TOP, fill=ctk.BOTH, expand=1)

    # Function to plot filtered data
    def plot_filtered():
        if y_final is None:
            messagebox.showerror("Error", "No data to plot. Press 'Start' first.")
            return

        fig, ax = plt.subplots()
        ax.plot(result_sav)
        ax.set_title("Filtered")
        ax.set_ylim(-0.5, 1.5)  # Ensure 0 is below 1 on the y-axis

        # Embed the plot in the tkinter window
        canvas = FigureCanvasTkAgg(fig, master=plot_frame)
        canvas.draw()
        canvas.get_tk_widget().pack(side=ctk.TOP, fill=ctk.BOTH, expand=1)


    def calculate_position():
        if result_sav is not None:  # Check if result_sav has been initialized
            interval_start = 0
            interval_stop = 0
            interval_length = 0
            result = 0
            i = 0

            while i < 1520:
                if abs(result_sav[i] - 1) > 0.05:
                    interval_start = i
                    while abs(result_sav[i] - 1) > 0.1 and i < 1535:
                        i = i + 1
                    interval_stop = i
                    if interval_stop - interval_start > interval_length:
                        interval_length = interval_stop - interval_start + 1
                        result = (interval_length / 2 + interval_start) * 0.0635
                i = i + 1
            messagebox.showinfo("Info", "The position is: " + str(result) + " mm")  # Convert result to
string



    # Function to close the serial connection and exit
    def on_closing():
        ser.close()
```

```
    root.destroy()

# Create the main window
root = ctk.CTk()
root.title("Data Reader")

# Add buttons
start_button_x = ctk.CTkButton(root, text="Start x", command=start_reading_x)
start_button_x.pack(pady=10)
start_button_y = ctk.CTkButton(root, text="Start y", command=start_reading_y)
start_button_y.pack(pady=10)

original_button = ctk.CTkButton(root, text="Original", command=plot_original)
original_button.pack(pady=10)

filtered_button = ctk.CTkButton(root, text="Filtered", command=plot_filtered)
filtered_button.pack(pady=10)

position_button = ctk.CTkButton(root, text="Position", command=calculate_position)
position_button.pack(pady=10)

# Create a frame to hold the plot
plot_frame = ctk.CTkFrame(root)
plot_frame.pack(fill=ctk.BOTH, expand=1)


# Bind the close event
root.protocol("WM_DELETE_WINDOW", on_closing)

# Start the GUI event loop
root.mainloop()
```