

THESIS

Air Filtration Control System (AFCS)

OPDRACHTGEVER

CERcuits

STAGEMENTOR

Ruben Mangelschots

STAGE -EN BACHELORPROEFCOÖRDINATOR (2024)

Filippo Bagnoli

BACHELORPROEF-COÖRDINATOREN (2025)

Anke Coomans, Hans Bartholomeus

Voorwoord

Als kind droomde ik ervan om op een dag uitvinder te worden. Ik zou machines en systemen bouwen die het leven van de mens aangenamer en makkelijker zouden maken. Gedurende mijn kindertijd – en nu mogelijks nog meer – werd mijn reeds hoge nieuwsgierigheid het meest geprikkeld door technologie, innovatie, gadgets, en alles wat hier wat bij aansloot. Over de jaren heen ben ik dit dus stelselmatig gaan opzoeken, waardoor ik uiteindelijk terechtkwam bij de IT Factory van Thomas More, meer specifiek bij het keuzetraject *Internet of Things* (IoT).

Voor mijn stageplek ging ik dan ook op zoek naar een bedrijf dat me interesseerde, maar vooral ook een stageopdracht aanbood die aansloot bij mijn kennis en vaardigheden. Tijdens mijn zoektocht herinnerde ik me het bedrijf genaamd CERcuits. Tijdens één van de lessen Embedded Devices Essentials had onze docent namelijk Dhr. Frederik Luppens – De CEO van dit bedrijf – uitgenodigd om de PCB-business in de kijker te zetten. Zijn bedrijf specialiseert zich in het ontwerpen en maken van keramische printplaten. Deze zijn bedoeld voor de ontwikkeling van chips en andere elektronica die ingezet worden in omgevingen met extreme temperaturen of waar de elektronica over het algemeen tegen een stootje moet kunnen.

Ik ging bij hen langs en werd enthousiast onthaald door Ruben Mangelschots, één van de werknemers van CERcuits. Hij gaf me een rondleiding doorheen het gebouw en vertelde wat hij er al had verwezenlijkt gedurende zijn werkperiode bij CERcuits. We wisselden contactgegevens uit en na goedkeuring van Dhr. Frederik Luppens werkte Ruben een stageopdracht voor mij uit en nam hij de rol van stagementor op zich. Deze opdracht bestond eruit een systeem te ontwikkelen dat de lasers - die bij het productieproces van de printplaten gebruikt werden - te monitoren en in functie hiervan een stofafzuigsysteem automatisch aan- of uit te zetten. Hoofdreden hiervoor was het jaarlijks energieverbruik en de daarbij horende factuur naar beneden krijgen. Het was een project dat uitdagend zou worden, maar mooi aansloot bij de automatisatie- en monitoringsopstellingen die ik reeds had ontwikkeld gedurende mijn opleiding.

Alvorens over te schakelen naar het uitgebreide realisatieverslag, had ik wel graag nog enkele mensen bedankt aangezien zij er mee verantwoordelijk voor geweest zijn om dit alles tot een goed einde te brengen. In de eerste plaats Ruben Mangelschots - mijn stagementor - voor zijn enthousiasme, de antwoorden op al mijn vragen en de vele ondersteuning die ik tijdens het verloop van de stage van hem gekregen heb. Daarnaast had ik ook graag Tim Christiaensen bedankt, voor zijn technische ondersteuning bij zowel het coderen als het bouwen van mijn systeem en voor de tips en tricks die ik van hem kreeg. Ook Dhr. Frederik Luppens verdient mijn dankbaarheid voor het vertrouwen dat ik kreeg en de mogelijkheid om kennis te maken met het bedrijfsleven in de IT-sector. Tot slot had ik graag mijn stagecoördinator Mr. Filippo Bagnoli, mijn bachelorproef-coördinatoren Mevr. Anke Coomans en Mr. Hans Bartholomeus ook mijn dank getoond voor de vele concrete, nuttige feedback en tussentijdse evaluaties. Ik ben tevreden dat ik met bovengenoemde mensen heb mogen samenwerken en kijk met veel plezier terug op deze leerrijke periode.

Inhoudsopgave

1. TECHNISCHE BESCHRIJVING	8
1.1. Startsituatie	8
1.2. Systeemvereisten	9
1.3. Hardware	11
1.3.1. Inleiding	11
1.3.2. Microcontrollers	11
1.3.3. Druksensor	12
1.3.4. Contactor	13
1.3.5. Optocoupler	14
1.3.6. Aansluitingen	15
1.3.7. Schema's	16
1.3.7.1. Globaal schema AFCS	16
1.3.7.2. Elektrisch schema	16
1.3.7.3. Schema plaatsing componenten	17
1.3.7.4. Schema onderzijde aftakdoos	17
1.3.7.5. Schema voorzijde aftakdoos	18
1.4. Energiebesparing	19
1.4.1. Inleiding	19
1.4.2. Resultaten	19
1.5. Software	20
1.5.1. Confluence	20
1.5.2. EasyEDA	20
1.5.3. Het programma zelf	21
1.5.3.1. Imports en initialisatie (1)	22
1.5.3.2. Imports en initialisatie (2)	23
1.5.3.3. Wi-fi-verbinding	24
1.5.3.4. Mapping potentiometer	24
1.5.3.5. Lezen van inputs	25
1.5.3.6. Schrijven van outputs	25
1.5.3.7. Lampfunctie (1)	26
1.5.3.8. Lampfunctie (2)	27
1.5.3.9. Lampfunctie (3)	28
1.5.3.10. Lasercheckfunctie	29
1.5.3.11. Blowerfunctie	30
1.5.3.12. Sorteervfunctie drukwaarden	31
1.5.3.13. Drukmonitoring	32

1.5.3.14. Setupfunctie (1)	33
1.5.3.15. Setupfunctie (2)	34
1.5.3.16. Loopfunctie (1)	35
1.5.3.17. Loopfunctie (2)	36
1.5.3.18. Loopfunctie (3)	37
2. TROUBLESHOOTING	38
3. VERBETERVOORSTELLEN	39
3.1. Noodstop	39
3.2. Internetverbinding	39
3.2.1. Online dashboard en datalogging	39
3.2.2. Predictive maintenance	39
3.2.3. Monitoring vanuit kantoor	40
3.3. Geoptimaliseerde temperatuurregeling	40
3.4. Eigen PCB	40
3.5. Uitgebreide statusweergave	40

Lijst met tabellen

Tabel 1: Signalisatie driekleurenlamp	10
Tabel 2: Weighted Decision Matrix Microcontrollers	11
Tabel 3: Specificaties blower	19
Tabel 4: Resultaten verbruik blower	19

Lijst met figuren

Figuur 1: Blower 746W	8
Figuur 2: Afzuigkap + afzuigbuis	8
Figuur 3: Druksensor - groot	12
Figuur 4: Contactor - groot	13
Figuur 5: Optocoupler - groot	14
Figuur 6: DIN-5 kabel groot	15
Figuur 7: AFCS Schematisch	16
Figuur 8: Elektrisch Schema AFCS	16
Figuur 9: Plaatsing componenten	17
Figuur 10: Onderzijde aftakdoos	17
Figuur 11: Voorzijde aftakdoos	18
Figuur 12: ESP32	45
Figuur 13: Optocoupler	45
Figuur 14: Relaisbord vier-kanaals	45
Figuur 15: Relaisbord 1-kanaals	46
Figuur 16: Voedingskabel 230V	46
Figuur 17: Dubbele voeding (24V & 5V)	46
Figuur 18: Differentiële druksensor	46
Figuur 19: Schakelaars (drie inputs)	47
Figuur 20: Driekleurenlamp + zoemer	47
Figuur 21: Contactor	47
Figuur 22: Potentiometer	47
Figuur 23: Aftakdoos	47
Figuur 24: Kabel type DIN 5	48
Figuur 25: DIN 5 Panel Mount	48
Figuur 26: Wartel	48
Figuur 27: Dubbele Jack (AUX) 5m	48
Figuur 28: Dubbele Jack (AUX) 10m	48
Figuur 29: Jack Panel Mount	49
Figuur 30: Stekkerhulskit	49
Figuur 31: Adereindhulzenkit	49
Figuur 32: Draaiknop	49
Figuur 33: WAGO-Klem	50

Figuur 34: Jumper Wire Kit	50
Figuur 35: Rubberen slang (6mm)	50
Figuur 36: Breakout Board ESP32	50
Figuur 37: Koperdraad (rood)	51
Figuur 38: Koperdraad (zwart)	51
Figuur 46: Werking Optocoupler	55
Figuur 47: Relaisbord vier-kanaals - groot	56
Figuur 48: Relais werking	56

Inleiding

Mijn stageopdracht bestond eruit een systeem te ontwerpen én te bouwen om de afzuiginstallatie van de productielijnen te automatiseren. Dit systeem zou het gebruik van de lasers monitoren en in functie hiervan de afzuiginstallatie automatisch aan- en uitzetten. Daarnaast diende het ook de status van de filterzakken te controleren en aan te geven wanneer deze vervangen moesten worden. Het creëren van dit systeem heeft als voornaamste doel de energiefactuur naar beneden krijgen, maar het dient mij ook wegwijs te maken in het proces om voor een bedrijf een product op maat te ontwikkelen van A tot Z. De naam die eraan werd toegekend luidt als volgt: Air Filtration Control System, kortweg AFCS.

Bij aanvang van mijn stage is de situatie de volgende: de bestaande afzuiginstallatie werkt, maar staat bijna permanent aan. Vaak is de laser nog operationeel of wordt er zelfs een nieuwe bewerking gestart tegen het einde van de werkdag. Zo winnen de werknemers bij CERcuits aan tijd, want dan is er de dag nadien een printplaat klaar tegen de ochtend.

Helaas kent dit ook enkele nadelen:

- De blowers zijn hierdoor ook nog operationeel nadat de bewerking is afgerond en blijven tot de ochtend nadien ingeschakeld, wat extra energieverbruik en dus een onnodig hoge energiefactuur betekent.
- Doordat deze blowers veel langer aanstaan dan nodig, verslijten deze sneller en heeft men dus op termijn ook meer kosten om deze te onderhouden en vervangen.

Naast het bouwen van deze systemen is het dus ook wenselijk om te achterhalen welke energiekost er op jaarbasis vermeden kan worden door de implementatie van mijn oplossing.

Hieronder vindt u het hele ontwikkelingsproces van zowel mezelf als IT'er, als van het systeem dat op dit moment operationeel is bij CERcuits. Het verslag is zo ingedeeld dat er eerst gesproken wordt over de startsituatie en vervolgens over de belangrijkste hardware en de verschillende opties die voor dit systeem in overweging zijn genomen. Er wordt toegelicht hoe er gekozen werd voor specifieke onderdelen en welk onderzoek hieraan voorafging. Het hardware-onderdeel wordt afgerond met de aansluitschema's van alle gebruikte componenten en het schema van de aftakdoos waar het gehele systeem in zit.

U vindt ook een gedetailleerde besparingsanalyse die gelinkt is aan het implementeren van vier systemen. De berekeningen hiervoor kan u ook steeds terugvinden in de bijlagen. De functionaliteiten van het systeem en de daarvoor geschreven code wordt uitvoerig toegelicht opdat het duidelijk is wat waar gebeurt in mijn programma. Enkele mogelijke problemen en de hiervoor gevonden oplossingen werden ook opgenomen in dit verslag. Dit geheel wordt afgesloten met enkele verbetervoorstellen voor AFCS.

1. Technische beschrijving

1.1. Startsituatie

Bij aanvang van de stage zijn er reeds vier productielijnen aanwezig, elk met drie lasers (twaalf lasers in totaal). Ook een zelfgebouwd systeem voor de stofafzuiging is al aanwezig en operationeel.

Dit systeem bestaat uit:



Figuur 1: Blower 746W

- Een industriële blower van 746 Watt inclusief filterzak



Figuur 2: Afzuigkap + afzuigbuis

- Een 3D-geprinte afzuigkap.
- Een afzuigbuis voor de afvoer van stofdeeltjes richting de filterzak.

Per productielijn is er dus één blower voorzien, die verbonden wordt aan de hand van afzonderlijke afzuigbuizen tot bij elke laser. Vooraleer men met een bewerking kan starten dient dit gehele systeem handmatig te worden gestart. Hiervoor opent men de klep van de noodstop en zet men de machine aan. De blower wordt gestart en alle stofdeeltjes zullen nu dus worden weggezogen zo lang de blower ingeschakeld is.

1.2. Systeemvereisten

Mijn opdracht bestaat eruit één systeem (AFCS) per productielijn te voorzien (vier in totaal) om de activiteit van de lasers te monitoren en in functie hiervan AFCS automatisch op en af te zetten. Het uiteindelijke doel van dit systeem is een verlaging van de energiefactuur horende bij het verbruik van deze vier blowers.

Dit systeem bevat drie verschillende standen:

- Uit-modus (O); de blower wordt handmatig uitgezet ongeacht de status van de lasers.
- Manuele modus (I); de blower wordt handmatig opgezet ongeacht de status van de lasers.
- Auto-modus(II); de blower wordt automatisch opgezet in functie van de status van de lasers.

Het systeem moet in modus II (Auto-modus) de volgende functionaliteiten bevatten:


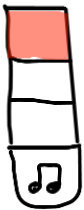




- Detecteren wanneer een laser wordt opgezet of reeds met een bewerking bezig is.
- De blower die zorgt voor de afzuiging automatisch opzetten indien minstens één van de lasers van desbetreffende productielijn actief is.
- Een timer starten wanneer alle aangesloten lasers inactief zijn.
- De blower van het systeem automatisch uitzetten wanneer de inactiviteit van lasers minstens vijftien minuten bedraagt.

Het systeem moet in alle modi (Auto, Uit & Manueel) deze functionaliteiten bevatten.

- De status van de filterzakken monitoren en aangeven wanneer deze vol zitten en vervangen moeten worden.
- Signaleren aan de hand van licht en geluid wanneer de status van de blower zal wijzigen. Dit wil zeggen: wanneer deze zal inschakelen of uitschakelen.

Ter signalisatie werd geopteerd voor een signaallamp met drie verschillende kleuren waar ook een zoemer ingebouwd zat. Dit zijn de opgenomen statussen en hun betekenis.

Tabel 1: Signalisatie driekleurenlamp

<ul style="list-style-type: none"> - Rode lamp: uit - Gele lamp: uit - Groene lamp: uit - Zoemer: uit 		Het systeem is uitgeschakeld, wordt niet voorzien van stroom.
<ul style="list-style-type: none"> - Rode lamp: aan - Gele lamp: uit - Groene lamp: uit - Zoemer: uit 		Het systeem wordt voorzien van stroom en bevindt zich in status nul (uit-modus). Rode lamp dient enkel ter info of het systeem ingeplugd is of niet.
<ul style="list-style-type: none"> - Rode lamp: aan - Gele lamp: uit - Groene lamp: aan - Zoemer: uit 		Het systeem wordt voorzien van stroom en bevindt zich in Auto-modus (II). De blower staat nog niet aan maar zal automatisch worden opgezet bij activiteit van minstens één van de aangesloten lasers.
<ul style="list-style-type: none"> - Rode lamp: pinkt (drie sec.) - Gele lamp: uit - Groene lamp: uit - Zoemer: aan (drie sec.) 		Statuswijziging blower. Wanneer de blower uitstaat en zal worden aangezet of de blower aan staat en zal worden uitgezet, zie en hoor je deze indicatie als waarschuwing.
<ul style="list-style-type: none"> - Rode lamp: uit - Gele lamp: uit - Groene lamp: aan - Zoemer: uit 		De blower is ingeschakeld. Het systeem kan zich in zowel Manuele modus (I) als Auto-modus (II) bevinden.
<ul style="list-style-type: none"> - Rode lamp: n.v.t. - Gele lamp: pinkt (drie keer per tien sec.) - Groene lamp: n.v.t. - Zoemer: uit 		Filterzak is vol en dient vervangen te worden. De gele lamp dient enkel ter indicatie voor de status van de filterzak en staat los van de status van de rest van het systeem. Ook al staat het systeem ingesteld op Uit-modus (O), de status van de filterzakken wordt nog steeds gemonitord.

1.3. Hardware

1.3.1. Inleiding

Bij de hardwareselectie werd steeds gezocht naar een balans tussen kostprijs, betrouwbaarheid en gebruiksgemak. De totale bestelkost, inclusief verzendkosten en beschikbaarheid, woog zwaarder door dan enkel de componentprijs. Soms werd gekozen voor iets duurdere onderdelen om dubbele verzendkosten te vermijden. Bij gelijkaardige prijzen kregen vertrouwde en eerder gebruikte componenten de voorkeur, wat de implementatie vereenvoudigde en de systeemzekerheid verhoogde.

1.3.2. Microcontrollers

Aangezien de microcontroller het belangrijkste van de gehele opstelling is, werd een vergelijkingsanalyse uitgevoerd. In de bijlagen vind je de vergelijkingstabel met alle belangrijkste kenmerken van elke optie die werden onderzocht. Hieronder bevindt zich de weighted decision matrix (WDM) die aangeeft welke criteria er belangrijk waren en in welke mate de microcontroller in kwestie hierop scoort. Er wordt telkens een score van één tot vijf toegekend voor elk criterium. Deze score wordt vermenigvuldigd met het gewicht dat erbij hoort en opgeteld om de totaalscore te berekenen.

Tabel 2: Weighted Decision Matrix Microcontrollers

Criterium	Score-uitleg	ESP32 Devkit	Arduino Uno	Raspberry Pi Pico	Raspberry Pi 4	Eigen PCB	Gewicht
Kost	1 = Duur 5 = Goedkoop	4	3	4	1	5	6
GPIO	1 = Weinig 5 = Veel	4	3	3	5	4	5
Complexiteit	1 = Moeilijk 5 = Eenvoudig	2	5	3	2	1	1
Documentatie	1 = Beperkt 5 = Uitgebreid	4	5	3	4	2	4
Bitgrootte	1 = 8-bit 5 = 64-bit	4	1	4	5	4	4
Wi-Fi / BT	1 = Afwezig 5 = Volledig	5	1	1	5	2	3
Totaalscore		93	65	73	84	81	

Conclusie: Op basis van de totaalscores is het duidelijk dat de ESP32 Devkit als beste keuze naar voren komt door de grote hoeveelheid GPIO's, de draadloze connectiviteit, de hoge rekenkracht en de lage kostprijs. Deze microcontroller vormde dan ook de basis voor AFCS.

1.3.3. Druksensor



Figuur 3: Druksensor - groot

Één van de systeemvereisten was het aangeven van de status van de filterzakken. Hiervoor werden twee soorten sensoren vergeleken; optische sensoren en druksensoren. Optische sensoren worden bijvoorbeeld gebruikt om luchtkwaliteit te meten doordat ze in staat zijn de concentratie aan deeltjes te detecteren, al is dat bij AFCS moeilijk haalbaar. De turbulentie in de filterzak zou voor een grote fluctuatie in metingen zorgen of er zou zich stof op de sensor kunnen ophopen. Door een gebrek aan licht in de filterzakken zouden metingen moeilijk verlopen en de constante trillingen zouden ook voor storing kunnen zorgen. Al snel werd duidelijk dat de beste manier om de filterstatus te detecteren via een drukmeting zou zijn.

In dat geval zijn er twee opties:

- Enkel de druk in de filterzak meten.
- Differentiële druk meten, m.a.w. zowel binnen als buiten de filterzak.

Aangezien de filterzakken uit stof bestaan en sommige ouder of meer vervuild zijn dan andere, ontstaat er variatie in de luchtstroming door elke individuele filter. Dit maakt het moeilijk om de druk op een consistente en representatieve manier te meten. Als we de druk binnenin een filterzak meten, kunnen we onregelmatige metingen krijgen door het Venturi-effect. In zones waar de lucht sneller stroomt (zoals de ingang van de druksensor), daalt de lokale statische druk. Hierdoor meet de druksensor via het buisje aan de ingang een lagere druk dan de werkelijke gemiddelde druk in de filterzak, wat de betrouwbaarheid van de meting aantast. Daarnaast is het moeilijk om elke keer op exact dezelfde manier te meten door variatie in filterzakken, de mate waarin ze de blower afsluiten, ...

Een differentiële drukmeting is betrouwbaarder dan een enkelvoudige drukmeting omdat ze het drukverschil meet vóór en na de filter, en dus rechtstreeks de vervuiling of weerstand van de filter aantoonst. Bij een enkelvoudige meting heb je geen referentiepunt en kunnen schommelingen veroorzaakt worden door externe invloeden zoals de blower of atmosferische druk. Hoewel beide meetmethoden gebruik maken van een slangetje in de filterzak — en dus gevoelig zijn voor het Venturi-effect of stof — blijft de differentiële meting stabiel, omdat het tweede meetpunt buiten de zak in een rustigere zone (zonder turbulentie) kan worden geplaatst. Hierdoor worden lokale verstoringen deels gecompenseerd en krijg je een realistischer beeld van de filtertoestand.

Om de maximale druk in een volle filterzak te meten, gebruikte ik een eenvoudige methode met een U-vormige waterslang. Deze methode is gebaseerd op de Wet van Pascal en laat toe om een drukverschil visueel af te lezen. In bijlage vind je het stappenplan hiervoor terug. De maximale druk in een volle filterzak ligt rond 200 Pascal. Aan de hand van deze info kwam ik uit bij deze sensor: **LWLP5000-5XD**. Deze heeft een bereik van -500 tot 500 Pascal en is makkelijk te gebruiken met microcontrollers.

1.3.4. Contactor



Figuur 4: Contactor - groot

Het gebruik van een standaard relais om een inductiemotor (zoals het geval is bij de blower) te schakelen is doorgaans geen verstandige keuze. Inductiemotoren trekken bij het opstarten een veel hogere stroom dan tijdens normaal gebruik. Deze hoge inschakelstroom kan ervoor zorgen dat het relais oververhit raakt of beschadigd wordt. Bovendien zijn inductiemotoren inductieve belastingen, wat betekent dat ze bij het uitschakelen hoge spanningspieken genereren. Die pieken veroorzaken vonkvorming tussen de contacten van het relais, wat leidt tot snelle slijtage, verminderde levensduur en in sommige gevallen zelfs doorslag of plakproblemen.

Daarnaast biedt een standaard relais geen enkele vorm van motorbeveiliging. Bij langdurige overbelasting zal het relais niet automatisch uitschakelen, waardoor de motor risico loopt op beschadiging of doorbranding. Om die redenen is het veel verstandiger om een contactor te gebruiken. Contactoren zijn specifiek ontworpen voor het schakelen van motoren en kunnen zowel de hoge inschakelstromen als de spanningspieken beter verwerken. Zo wordt niet alleen de betrouwbaarheid verhoogd, maar ook de veiligheid van het systeem gewaarborgd.

1.3.5. Optocoupler



Figuur 5: Optocoupler - groot

Om te detecteren of een laser op de productielijn actief was werd in de laserkast een signaal gemeten, afkomstig van de M4-controller. Deze controller beschikte over verschillende aansluitpinnen, waarvan sommige al in gebruik waren voor de aansturing van de laser. Eén van de vrije pinnen gaf echter een duidelijk spanningssignaal: 0V bij inactiviteit en 24V wanneer de laser actief was.

Aangezien de ESP32-microcontroller slechts signalen tot 3.3V verdraagt, kon het signaal van 24V niet rechtstreeks aangesloten worden zonder de microcontroller te beschadigen. Een ander risico dat mijn stagebegeleider liever wou vermijden was de invloed die de M4-controller zou ondervinden wanneer er kortsluiting zou ontstaan in het circuit dat verbonden was met de ESP32. Om dit op een veilige manier op te lossen, werd gebruikgemaakt van een optocoupler.

Een optocoupler is een component die twee elektrische circuits volledig van elkaar isoleert, maar toch in staat is een signaal over te brengen via licht. Wanneer er stroom loopt door de ingangskant van de optocoupler (vanuit het 24V-circuit), wordt een interne LED geactiveerd. Deze LED schijnt op een lichtgevoelige transistor aan de uitgangskant (het 3.3V-circuit), waardoor ook daar een elektrisch signaal ontstaat — volledig elektrisch gescheiden van het eerste circuit. Meer uitleg over de werking van de optocoupler vindt u terug in de bijlagen.

Dankzij deze scheiding wordt de ESP32 beschermd tegen de hoge spanningen van het industriële circuit, terwijl ze toch correct en veilig kan detecteren of de laser actief is. Ook de lasers zijn op hun beurt beschermd tegen beschadiging. Er werd gekozen voor een optocoupler met twee kanalen, zodat een tweede kanaal beschikbaar was als back-up mocht het eerste kanaal defect raken.

1.3.6. Aansluitingen

Om het gehele systeem van stroom te voorzien, was er een voeding nodig die twee verschillende spanningsniveaus kon leveren: 24V en 5V. De 24V-uitgang was nodig om de contactor en de signaallamp met ingebouwde zoemer correct te laten functioneren. De 5V-uitgang wordt gebruikt voor het voeden van de ESP32 en de relaisbordjes. Voor het aansluiten van deze voeding was een drie-aderige voedingskabel nodig, voldoende dik om het systeem veilig van stroom te voorzien. Een drie-aderige kabel bevat drie afzonderlijke draden: blauw (neuter of N), bruin (lijningang of L) en een aardingsdraad (\perp). Na overleg met mijn stagementor bleek een kabelkerndikte van 0,5 mm² voldoende. Het werd uiteindelijk een standaard voedingskabel met een iets grotere kerndikte van 0,75 mm². Deze bood een extra veiligheidsmarge en leek me daarom een verantwoorde keuze. Bovendien koos ik ervoor nieuwe kabels te gebruiken om mogelijke slijtage of beschadiging bij oudere kabels te vermijden.

Om de binnenkomende 230V-spanning ook te gebruiken voor de contactor, en niet enkel voor de spanningsbron, werd de kabel afgetakt met behulp van Wago-klemmen. Deze klemmen maken het mogelijk om op een veilige en overzichtelijke manier meerdere verbindingen te maken zonder te solderen.



Figuur 6: DIN-5 kabel groot

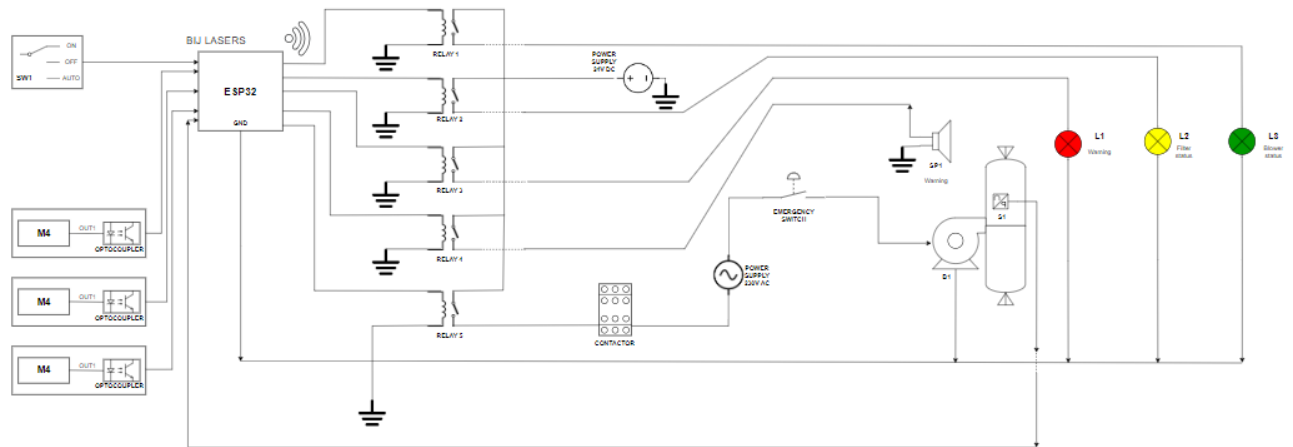
Voor de verbinding tussen AFCS (in de aftakdoos) en de signaallamp koos ik voor een DIN 5-kabel. Deze vijfaderige kabel verbindt respectievelijk de rode, gele en groene lampen, de zoemer en de grond. DIN-kabels hebben het voordeel dat ze stevig en betrouwbaar vastklikken, maar toch makkelijk vervangbaar blijven. Mocht de kabel ooit beschadigd raken of de lengte veranderen, kan eenvoudig een nieuwe kabel aangesloten worden zonder te moeten solderen.

Voor de signaaloverdracht van de optocoupler in de laser naar AFCS gebruikte ik een dubbele jack-kabel. De getwiste aderpennen in deze kabel beperken de gevoeligheid voor elektromagnetische interferentie, wat cruciaal is in een omgeving met veel elektrische apparatuur en al zeker wanneer je signalen meet. Net als bij de DIN-kabels koos ik ook hier bewust voor een vervangbare verbinding, om flexibiliteit en onderhoudsgemak te garanderen.

Om verbindingen te leggen tussen de ESP32 en de andere componenten werd er ook gebruik gemaakt van een ESP32-breakout board. Dit bevat schroefterminals waar je kabels in kan vastschroeven. Op deze manier hoeft er niet gesoldeerd te worden en zijn de verbindingen semipermanent en betrouwbaar.

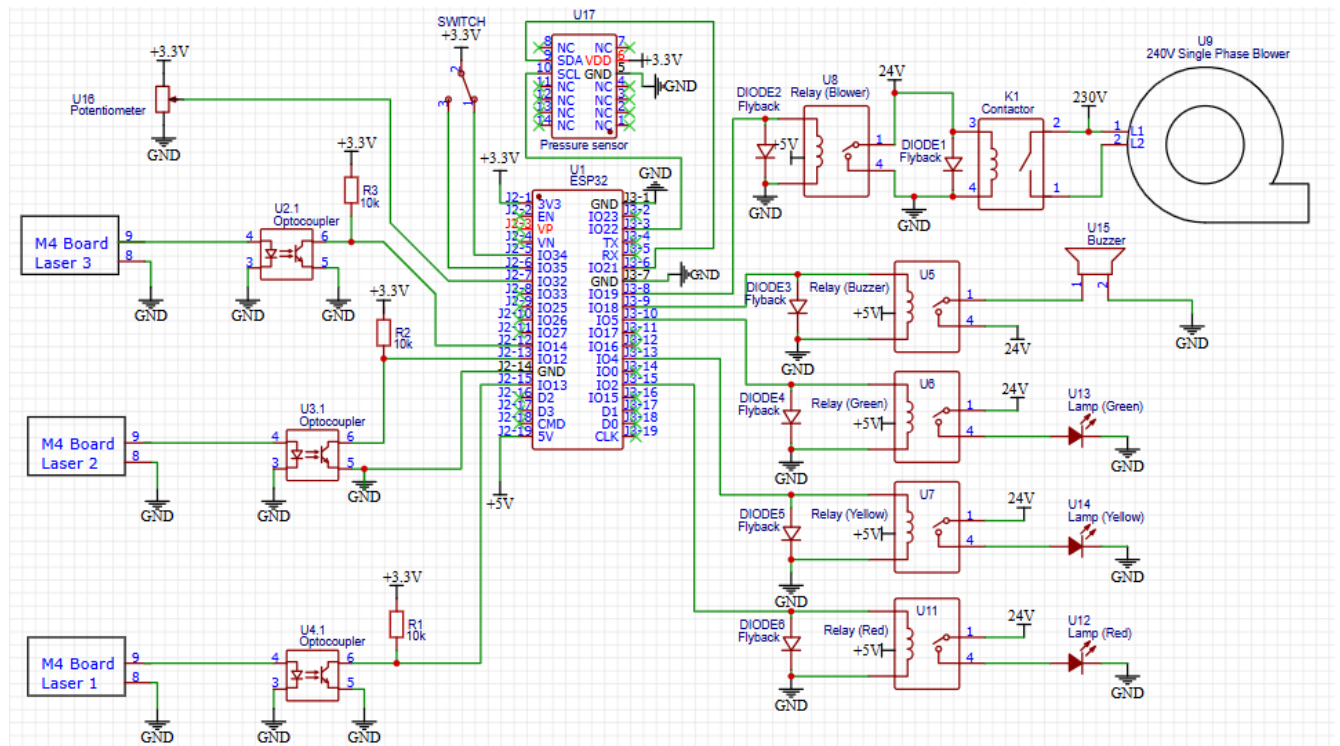
1.3.7. Schema's

1.3.7.1. Globaal schema AFCS



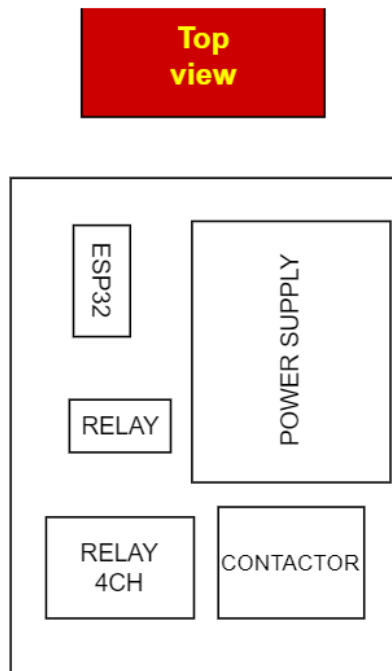
Figuur 7: AFCS Schematisch

1.3.7.2. Elektrisch schema



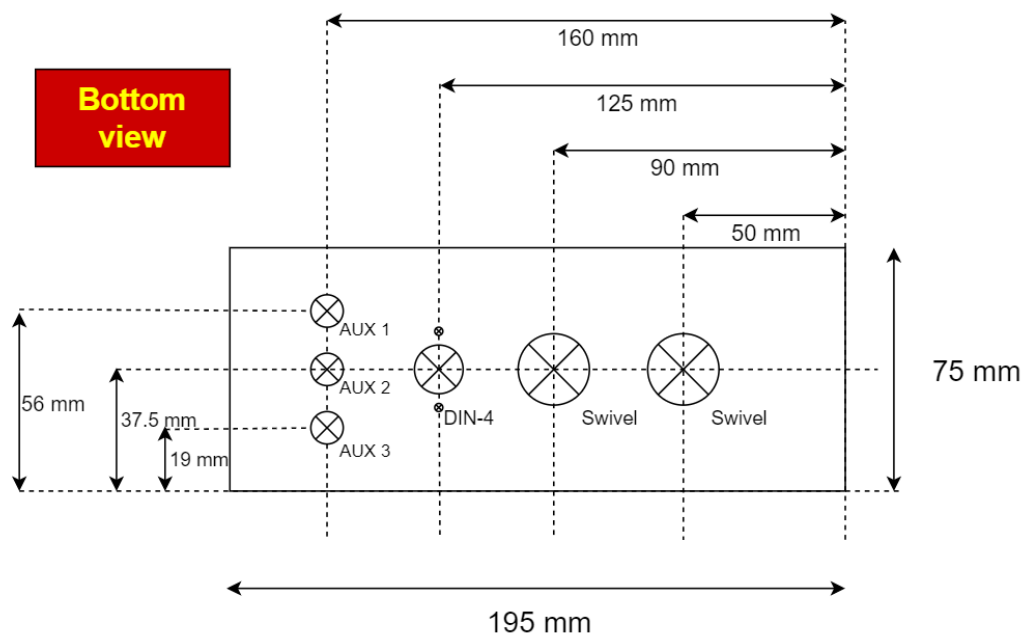
Figuur 8: Elektrisch Schema AFCS

1.3.7.3. Schema plaatsing componenten



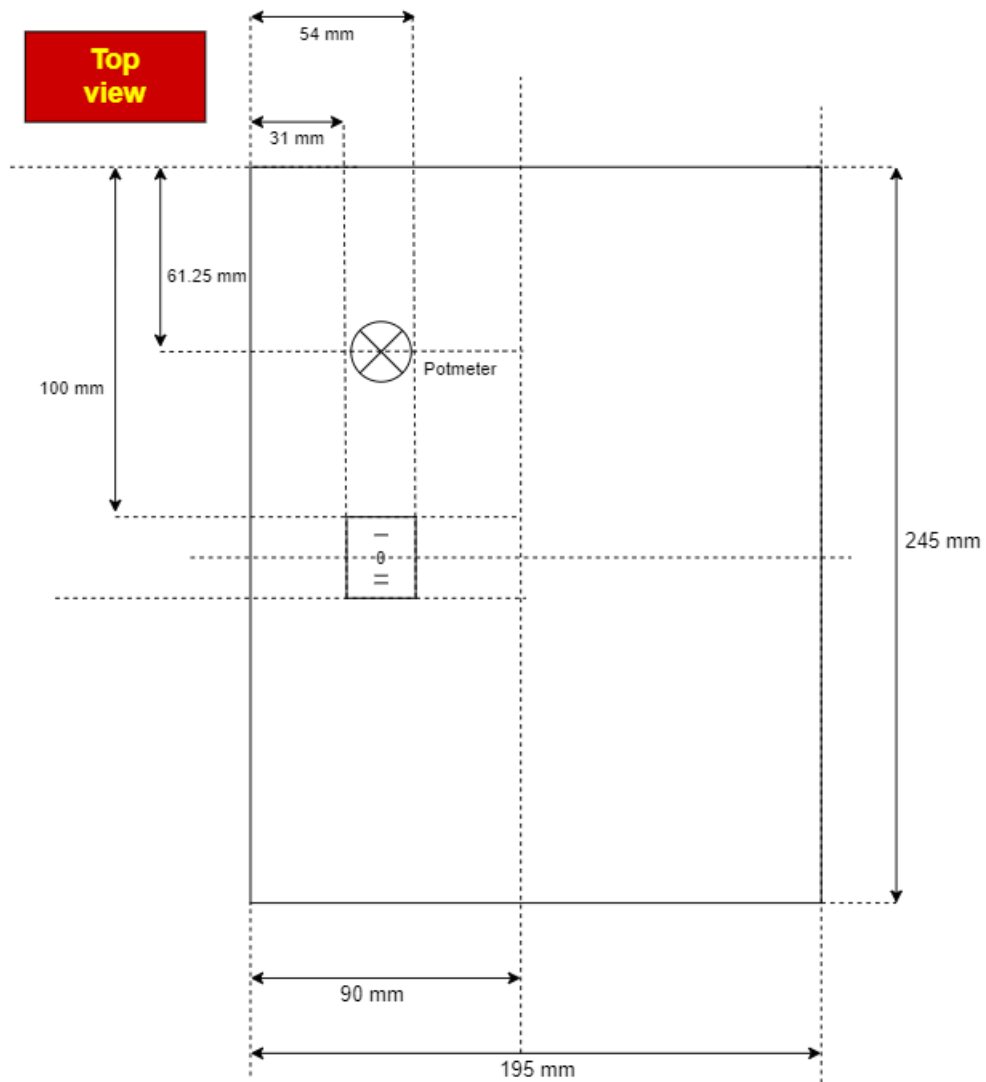
Figuur 9: Plaatsing componenten

1.3.7.4. Schema onderzijde aftakdoos



Figuur 10: Onderzijde aftakdoos

1.3.7.5. Schema voorzijde aftakdoos



Figuur 11: Voorzijde aftakdoos

1.4. Energiebesparing

1.4.1. Inleiding

Hieronder vindt u de gegevens terug die een inschatting maken van de hoeveelheid uitgespaarde energie en de daarbij horende energiekost door implementatie van vier systemen. De berekeningen die hebben geleid tot deze resultaten vindt u terug in de bijlagen.

1.4.2. Resultaten

Tabel 3: Specificaties blower

Blower	
Spanning (U)	230 Volt
Stroom (I)	4.8 Ampère
Vermogen (U*I)	746 Watt
Verbruik	0.746 kWh

Tabel 4: Resultaten verbruik blower

Verbruik blowers	
Totaal verbruik (4 blowers)	2.984 kWh
Laatst mogelijke startuur bewerking	18:00 u
Gemiddelde duur bewerking	3:00 u
Laatst mogelijke einduur bewerking	21:00 u
Onnodig ingeschakelde duur	11u
Onnodig verbruik per dag	32.824 kWh
Onnodig verbruik per week	164.12 kWh
Gemiddeld aantal weekdays per maand	22
Onnodig verbruik per maand	722.128 kWh
Gemiddeld aantal werkdagen/jaar	256
Onnodig verbruik per jaar	8402.944 kWh
Elektriciteitsprijs 2024	0.33€/kWh
Onnodige kost per jaar	2772.97 €

Uit de verbruiksanalyse blijkt dat de blowers in hun huidige gebruikspatroon tot wel elf uur per dag onnodig ingeschakeld blijven. Dit leidt tot een jaarlijks energieverlies van 8403 kWh, wat overeenkomt met een vermijdbare kost van €2772,97 per jaar. Deze cijfers onderstrepen het belang van het Air Filtration Control System (AFCS), dat gericht is op het automatisch uitschakelen van de blowers buiten de effectieve werktijd. Door deze optimalisatie kan een aanzienlijk deel van dit energieverlies — en de bijhorende kosten — worden uitgespaard.

1.5. Software

1.5.1. Confluence

Platform waar bij CERcuits reeds gebruik van gemaakt wordt. Het is een verzamelplaats voor documentatie, datasheets, schema's, instructies voor bepaalde processen, enz. Het kan ook dienst doen om (sprint)planningen en to-dolijsten te creëren. Tijdens de stage diende het om dagelijkse activiteiten en vooruitgang wat betreft de stage beknopt bij te houden, nuttige links op te slaan en bestellijsten te samen te stellen. Schema's maken was mogelijk via een ingebouwd tekenprogramma.

1.5.2. EasyEDA

Voor het ontwerpen van het elektrisch schema werd gebruikgemaakt van EasyEDA, een gratis softwarepakket dat eerder aan bod kwam tijdens mijn opleiding. Een belangrijk voordeel van deze software is de uitgebreide ingebouwde componentenbibliotheek, die vrijwel elk gangbaar elektronisch onderdeel bevat. Hierdoor kan snel en overzichtelijk een volledig elektrisch schema worden opgebouwd.

Hoewel er geen eigen printplaat werd ontworpen of besteld, werd het schema intensief gebruikt als leidraad bij het soldeerwerk. Daarnaast bood het schema een waardevolle visuele ondersteuning tijdens technische besprekingen en overlegmomenten met collega's. EasyEDA bleek zo niet alleen nuttig tijdens de ontwerpfase, maar ook in de praktische uitvoering van het systeem.

1.5.3. Het programma zelf

Het programma voor het AFCS-systeem werd volledig geschreven in de programmeertaal van Arduino, maar niet via de klassieke Arduino IDE. In plaats daarvan werd gekozen voor Visual Studio Code in combinatie met PlatformIO, voornamelijk vanwege de gebruiksvriendelijkheid en het bredere aanbod aan beschikbare libraries ten opzichte van de standaard Arduino-omgeving.

Tijdens het project werd ook de manier van programmeren aanzienlijk verbeterd. Onder begeleiding van een collega werd geleerd hoe 'propere' en gestructureerde code kan worden geschreven, met als resultaat een hogere leesbaarheid en schaalbaarheid. Hierbij werd inspiratie gehaald uit principes van objectgeoriënteerd programmeren.

Dankzij deze aanpak is de code niet alleen functioneel, maar ook beter overdraagbaar. Toekomstige ontwikkelaars kunnen hierdoor eenvoudiger instappen in het project, wat de onderhoudbaarheid en uitbreidbaarheid van het systeem ten goede komt.

Onderstaande principes werden tijdens de stage aangeleerd en zullen voortaan structureel toegepast worden bij het ontwikkelen van software:

- Het programma wordt opgedeeld in zo klein mogelijke functionele eenheden. Voor elke specifieke functionaliteit van het systeem wordt een afzonderlijke functie voorzien. Dit vergemakkelijkt het opsporen van fouten en zorgt ervoor dat problemen lokaal kunnen worden aangepakt.
- Elke functie wordt beperkt tot één duidelijke verantwoordelijkheid.
- Functies worden zo ontworpen dat ze onafhankelijk zijn van elkaar.
- Lees- en schrijfacties worden bij voorkeur vermeden binnen functies. In plaats daarvan worden statussen bijgehouden. Dit laat toe om de code ook zonder aangesloten hardwarecomponenten te testen.
- Aan het begin van elke programmacycclus wordt één centrale functie gebruikt om alle inputs van het systeem in te lezen.
- Aan het einde van de cyclus wordt één functie gebruikt om alle outputs uit te sturen.
- 'Blocking functions' worden vermeden. Dit zijn stukken code die de verdere uitvoering van het programma tijdelijk blokkeren.

Typische voorbeelden hiervan zijn:

- Wachten tot een bepaalde tijd verstreken is
- Wachten tot een variabele een bepaalde grenswaarde heeft bereikt
 - While-lussen worden bij voorkeur niet gebruikt

Door deze aanpak wordt voorkomen dat functies elkaar onbedoeld beïnvloeden of blokkeren.

Verder in dit verslag zijn ook screenshots van de AFCS-code terug te vinden, telkens aangevuld met toelichting bij het bijbehorende codefragment.

1.5.3.1. Imports en initialisatie (1)

```
1  #include <Arduino.h>
2  #include <WiFi.h>
3  #include "time.h"
4  #include <DFRobot_LWLP.h>
5
6  // Define the relay pins
7  #define RED_PIN 2
8  #define YELLOW_PIN 4 // for all the other systems
9  #define GREEN_PIN 5
10 #define BUZZER_PIN 18
11 #define BLOWER_PIN 19
12 #define LASER_PIN_1 13
13 #define LASER_PIN_2 27
14 #define LASER_PIN_3 14
15 #define MANUAL_PIN 35 //switch position I
16 #define AUTO_PIN 34 //switch position II
17 #define POTENTIO_PIN 32
```

De eerste vier regels van de code bevatten allemaal ‘include’ statements. Deze statements voegen bepaalde **libraries** toe die het mogelijk maken om **specifieke functies** en/of **syntax** te kunnen gebruiken.

Regel één importeert de **Arduino** library, hierdoor kunnen we in deze specifieke programmeertaal coderen. Regel twee voegt een **Wi-Fi**-bibliotheek toe. Deze werd toegevoegd om de mogelijkheid tot internetverbinding op de ESP32 open te houden. Als derde werd de ‘**time**’ library toegevoegd om **timers** en dergelijke te kunnen gebruiken. Tot slot de toevoeging van de DFRobot-library die hoorde bij de **druksensor**. Hierdoor konden functies worden gebruikt die de meetdata van de sensor konden verwerken en presenteren.

Regel 7 tot 17 definiëren de nummers van de gebruikte GPIO pinnen van de ESP32 en hun bijhorende naam.

1.5.3.2. Imports en initialisatie (2)

```
25
26 //initializing variables
27 int redStatus = 0;
28 int yellowStatus = 0;
29 int greenStatus = 0;
30 int buzzerStatus = 0;
31 int blowerStatus = 0;
32 int laserStatus[3] = {0, 0, 0};
33 //int laser_1_Status[3] = {0, 0, 0};
34 int manualStatus = 0;
35 int autoStatus = 0;
36 const char *ntpServer = "pool.ntp.org"; //
37 DFRobot_LWLP lwlp; // initialize pressure sensor
38 DFRobot_LWLP::sLwlp_t data; // initialize data from pressure sensor
39 float pressure = 0.0;
40 int potentionValue = 0;
41 float maxPressure = 0.0;
42 unsigned long previousMillis;
43 unsigned long currentMillis;
44 const long blinkInterval = 500;
45 const long blinkDuration = 3100;
46 const long blowerOffDelay = 900000;
47 const long pressureDuration = 10000;
48 const long pressureInterval = 600;
49 int count = 0;
50 int pressureCount = 0;
51 int warningFlag = 0;
52 float pressureArray[256];
53 uint8_t pressureIndex = 0;
54 int pressureLength = 256;
```

Hier worden **variabelen** die we later in het programma zullen gebruiken **geïnitieerd**. Initialiseren wil zoveel zeggen als: aangegeven welk datatype er bij deze variabele hoort (bv. **Integer**, **character**, ...) en welke standaardwaarde ze eventueel dienen te hebben wanneer het programma voor de eerste keer zal worden gerund. Zo zie je dat de statussen van alle lampen, zoemer, blower en lasers aanvankelijk op nul worden gezet. Nul betekent hier hetzelfde als 'uitgeschakeld, inactief'. Lijn 34 en 35 beschrijven de variabelen die de modus van AFCS zullen aangeven. De variabele voor de druksensor en de variabele waarin de meetdata hiervan terecht zal komen worden ook gecreëerd. Daarna worden de waarden voor de **gemeten differentiële druk**, **maximale druk** die gemeten mag worden vooraleer de filterzak als vol beschouwd wordt en de variabele waarin de data van de **potentiometer** (hoeveel deze opengedraaid is) zal worden opgeslagen. De variabelen **previousMillis** en **currentMillis** zullen timestamps, een **tijd in milliseconden** stockeren. Het gebruik hiervan wordt wat later in dit document duidelijk aan de hand van de toepassing ervan in het programma. Daarna worden er nog enkele **constanten** aangemaakt. Dit zijn – zoals de naam het zegt – zaken die constant of onveranderd blijven in heel het programma. Ze worden gebruikt als waarden ter vergelijking, meer specifiek om te checken of het aantal verstreken milliseconden gelijk is aan een bepaalde vaste waarde. Lijn 49 en 50 zijn **counters**, deze zaken worden gebruikt om bij te houden hoeveel keer iets al gebeurd is in het programma. Regel 52 tot 54 beschrijven de 'opslagplaats' waarin de geregistreerde drukmetingen terecht zullen komen. De grootte ervan en de index wordt meegegeven, om te weten hoeveel waarden er kunnen worden opgeslagen en om te kunnen zoeken naar een meting met specifieke index (nummer in het rijtje) uit de hoop metingen.

1.5.3.3. Wi-fi-verbinding

```
57 //Functions
58
59 // Initialize WiFi
60 void initWiFi() {
61     WiFi.mode(WIFI_STA);
62     WiFi.begin(ssid, password);
63     Serial.print("Connecting to WiFi ..");
64     while (WiFi.status() != WL_CONNECTED) {
65         Serial.print('.');
66         delay(1000);
67     }
68     Serial.println(WiFi.localIP());
69 }
```

Deze functie heeft als doel **verbinding** op te zetten tussen de **ESP32** en een bestaand **Wi-Fi**-netwerk. Dit werd toegevoegd in de code om de gemeten data op te sturen en op te slaan in de cloud ter analyse of voor eventueel secundair gebruik. Dit onderdeel is uiteindelijk niet opgenomen in het finale systeem. Zie verbetervoorstellen onderaan dit document voor meer uitleg hierover.

1.5.3.4. Mapping potentiometer

```
71 // Potentiometer mapping
72 float floatMap(float x, float in_min, float in_max, float out_min, float out_max) {
73     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
74 }
75
```

De potentiometer in het systeem diende om de **grenswaarde** met betrekking tot de differentiële **drukmeting** handmatig af te stellen. Zo konden potentiële **verschillen tussen de afzonderlijke systemen** worden opgevangen. Vervolgens kon na een periode van bijstellen een accurate **maximum drukwaarde** worden ingesteld die kon worden gebruikt als trigger voor het monitoringsysteem van de filterzakken. Eens die ingestelde drukwaarde overschreden werd voor het systeem in kwestie, zou de gele lamp aangeven dat de filterzak vol zat. Voorwaarde om dit te doen werken was dat de gemeten spanning over de potentiometer, die recht evenredig is aan de overeenkomstige weerstand op dat moment, zou worden gekoppeld aan een waarde tussen nul en 500. Deze functie 'mapt' de potentiometerwaarden programmatorisch aan waarden die kunnen worden geïnterpreteerd en gebruikt verder in het programma.

1.5.3.5. Lezen van inputs

```
76 //Read all inputs
77 void Digital_IN(){
78     laserStatus[0] = !digitalRead(LASER_PIN_1); // Read the status of Laser 1
79     laserStatus[1] = !digitalRead(LASER_PIN_2); // Read the status of Laser 2
80     laserStatus[2] = !digitalRead(LASER_PIN_3); // Read the status of Laser 3
81     manualStatus = digitalRead(MANUAL_PIN);
82     autoStatus = digitalRead(AUTO_PIN);
83     data = lwlp.getData();
84     pressure = abs(data.pressure);
85     potentioValue = analogRead(POTENTIO_PIN);
86 }
```

Deze functie heeft als doel **alle inputs** van de ESP32 te **lezen** aan het begin van de cyclus van het programma. Zo wordt er eerst geverifieerd of de pinnen horende bij de lasers hoog zijn, wat wil zeggen dat de desbetreffende laser actief is. Deze waarden worden bijgehouden in een array genaamd laserStatus. Deze array kan dan verder in het programma worden gebruikt om gekoppeld te worden aan bijhorende acties (lampen inschakelen, relais inschakelen, ...). De status van de schakelaar wordt uitgelezen om te weten in welke modus het systeem zich bevindt. Wat lager wordt de data van de druksensor (lwlp) geanalyseerd om te weten wat de huidige druk is binnenin de filterzak. Tot slot wordt de 'rauwe' data van de potentiometer binnengehaald (een spanningsniveau tussen nul en 3.3V vertaald naar een integer dat daarmee correspondeert).

1.5.3.6. Schrijven van outputs

```
88 //Control all outputs
89 void Digital_OUT(){
90     digitalWrite(YELLOW_PIN, yellowStatus);
91     digitalWrite(RED_PIN, redStatus);
92     digitalWrite(GREEN_PIN, greenStatus);
93     digitalWrite(BUZZER_PIN, buzzerStatus);
94     digitalWrite(BLOWER_PIN, blowerStatus);
95 }
96
```

Deze functie **schrijft alle outputs weg**. Deze functie wordt bovenaan het programma geplaatst om conflicten m.b.t. variabelen die lager worden aangeroepen te vermijden. Logischerwijs wordt ze wel pas helemaal onderaan uitgevoerd, vlak voor de cyclus is afgerond. Vervolgens kunnen alle inputs weer worden gelezen en begint alles opnieuw. Zoals je kan zien heeft AFCS slechts **5 outputs (de gele, rode & groene lamp, de zoemer en de relais die de contactor voor de blower schakelt)**.

1.5.3.7. Lampfunctie (1)

```
97 // Control lamp with different possible cases
98 void lamp(int lampOption) {
99     switch (lampOption) {
100
101         // Turn everything off (at startup)
102         case 1:
103             redStatus = 0;
104             yellowStatus = 0;
105             greenStatus = 0;
106             break;
107
108         // Warning signal before turning blower on
109         case 2:
110             static unsigned long blinkTimerStart = 0;
111             blinkTimerStart = millis();
112             while (millis() - blinkTimerStart <= blinkDuration) {
113                 buzzerStatus = 1;
114                 currentMillis = millis();
115                 if (currentMillis - previousMillis >= blinkInterval && count <= 6){
116                     previousMillis = currentMillis;
117                     count += 1;
118                     redStatus = !redStatus;
119                     Digital_OUT();
120                 }
121             }
122             warningFlag = 0;
123             buzzerStatus = 0;
124             count = 0;
125             break;
126     }
```

Deze functie heeft als doel alle mogelijke toestanden van de driekleurenlamp + zoemer te beschrijven. Hiervoor werd een zogenaamde switch-case functie gebruikt. Het is een manier om de variabele **lampOption** te testen tegen meerdere mogelijke waarden (**hier één tot en met negen**), en op basis daarvan verschillende codeblokken uit te voeren. Deze aanpak maakt het geheel wat overzichtelijker dan wanneer je bijvoorbeeld opeenvolgende if-statements en else-statements zou gebruiken. Na elke case volgt er een **break**. Hiermee geven we aan dat we aan het einde van de **case** gekomen zijn.. Op deze screenshot zie je de eerste twee mogelijke **cases**.

Case 1: Initialisatie

Hier wordt het systeem in een **neutrale of opstarttoestand** gezet. Hierbij worden alle statuslampen uitgeschakeld: de rode, gele en groene indicaties worden op nul gezet, waardoor er geen visuele signalen meer worden weergegeven. Deze toestand wordt gebruikt tijdens het opstarten van het systeem of bij een volledige reset.

Case 2: Waarschuwing

Deze case activeert een **waarschuwingssignaal** voorafgaand aan het inschakelen van de blower. Er wordt een timer gestart die de duur van de waarschuwingsfase bepaalt (**blinkDuration**). Tijdens deze periode klinkt de buzzer continu en knippert de rode LED op vaste intervallen (**blinkInterval**). Elke keer dat het interval verstrijkt, wordt de LED-status omgeschakeld en de wijziging doorgevoerd via de functie **Digital_OUT()**, die de bijhorende relais aanstuurt.

Na afloop van de waarschuwingsduur worden de tijdelijke indicatoren, zoals de buzzerstatus, het waarschuwingsvlaggetje en de knippertelling, terug op nul gezet. Zo is het systeem klaar om verder te gaan, bijvoorbeeld met het starten van de blower.

1.5.3.8. Lampfunctie (2)

```
127 // Indicate that filter bag is full
128 case 3:
129     static unsigned long pressureTimerStart = 0;
130     pressureTimerStart = millis();
131     greenStatus = blowerStatus;
132     while (millis() - pressureTimerStart <= pressureDuration) {
133         currentMillis = millis();
134         Serial.print("pressure count = ");
135         Serial.println(pressureCount);
136         if (currentMillis - previousMillis >= pressureInterval && pressureCount <= 5){
137             previousMillis = currentMillis;
138             pressureCount += 1;
139             yellowStatus = !yellowStatus;
140             Digital_OUT();
141         }
142     }
143     pressureCount = 0;
144     break;
145
146 // Indicate the status of the blower
147 case 4:
148     if (blowerStatus == 1) {
149         greenStatus = 1;
150     }
151     if (blowerStatus == 0 && autoStatus == 0) {
152         greenStatus = 0;
153     }
154     break;
155
156 case 5:
157     redStatus = 0;
158     break;
```

Case 3: Filterzak is vol

Wanneer case 3 wordt geactiveerd, wordt dit geïnterpreteerd als een indicatie dat de **filterzak vol** is. Als eerste wordt een statische variabele `pressureTimerStart` gedefinieerd en bijgewerkt met de huidige tijd via `millis()`. Deze variabele dient als startpunt van een timer. Daarna wordt de status van de blower (`blowerStatus`) toegekend aan `greenStatus` (**blower aan = groene LED aan**). Vervolgens wordt een `while`-lus gestart die blijft draaien zolang de huidige tijd (`millis()`) zich binnen de ingestelde tijdsduur (`pressureDuration`) bevindt. Elke keer dat de ingestelde tussenperiode (`pressureInterval`) is verstreken én zolang het aantal tellingen (`pressureCount`) niet hoger is dan 5, wordt de telling verhoogd, de status van een gele indicatie (`yellowStatus`) gewisseld (aan/uit), en een functie genaamd `Digital_OUT()` aangeroepen — waarschijnlijk om de gewijzigde status door te voeren naar de buitenwereld, bijvoorbeeld via een LED of relais. Na afloop van de `while`-lus wordt de druk-telling terug op nul gezet en verlaat het programma deze case.

Case 4: Status van de blower

In case 4 wordt de **status van de blower gecontroleerd**. Als `blowerStatus` gelijk is aan één (blower is aan), wordt `greenStatus` ook ingeschakeld. Als daarentegen de blower uit staat (`blowerStatus == 0`) én het systeem niet in automatische modus werkt (`autoStatus == 0`), wordt `greenStatus` uitgeschakeld. Op deze manier wordt de groene indicatie alleen geactiveerd wanneer de blower actief is, tenzij er automatische besturing is.

Case 5: Uitschakelen van rode LED

In case 5 wordt eenvoudigweg enkel **redStatus op nul** gezet waardoor de rode lamp wordt uitgeschakeld.

1.5.3.9. Lampfunctie (3)

```
159
160     case 6:
161         yellowStatus = 0;
162         break;
163
164     case 7:
165         greenStatus = 0;
166         break;
167
168     case 8: // Off mode
169         redStatus = 1;
170         break;
171
172     case 9: // Auto mode
173         redStatus = 1;
174         greenStatus = 1;
175         break;
176     }
177 }
```

Case 6: Uitschakelen van gele LED

In case 6 wordt eenvoudigweg **yellowStatus op nul** gezet waardoor énkél de gele lamp wordt uitgeschakeld.

Case 7: Uitschakelen van groene LED

In case 7 wordt eenvoudigweg **greenStatus op nul** gezet waardoor énkél de groene lamp wordt uitgeschakeld.

Case 8: OFF-modus

Deze case zal de **OFF-modus van het systeem** aanduiden door énkél de rode LED aan te zetten. Dit geeft aan dat het systeem voeding krijgt, maar uit staat.

Case 9: AUTO-modus

Deze case zal de **AUTO-modus van het systeem** aanduiden door de rode én de groene LED aan te zetten. Dit geeft aan dat het systeem klaar is om te detecteren wanneer een laser start met een bewerking en als gevolg de blower aan te schakelen.

1.5.3.10. Lasercheckfunctie

```
179 // Check if at least one of the lasers is active
180 bool isAnyElementHigh(int laserStatus[])
181 {
182     //to rework add array
183     for (int i = 0; i < 3; ++i)
184     {
185         if (laserStatus[i] == HIGH)
186         {
187             return true; // Return true if at least one element is HIGH
188         }
189     }
190     return false; // Return false if none of the elements are HIGH
191 }
192
```

Deze functie `isAnyElementHigh()` **controleert of er ten minste één laser actief** is. De functie neemt een array van integers (`laserStatus[]`) als invoer, waarin de statussen van drie lasers worden opgeslagen. Elke waarde in de array stelt de status van één laser voor, waarbij `HIGH` betekent dat de laser actief is.

Binnen de functie wordt een `for`-lus gebruikt om door elk van de drie elementen in de array te lopen (`i` van nul tot 2). Voor elk element wordt gecontroleerd of de waarde gelijk is aan `HIGH`. Zodra dit het geval is, geeft de functie onmiddellijk `true` terug. Dit betekent dat er minstens één laser actief is, en verdere controle is dan niet meer nodig.

Als de lus alle drie de elementen heeft gecontroleerd zonder een `HIGH` tegen te komen, dan wordt `false` teruggegeven. Dat betekent dat geen van de lasers actief is. Kort samengevat: deze functie beantwoordt de vraag "Is er ten minste één laser aan?" met `true` of `false`.

1.5.3.11. Blowerfunctie

```
193 // Control the blower
194 void blower(int laserStatus[]) {
195     static unsigned long blowerTimerStart = 0; // Variable to store the start time for warning blink
196
197     if (isAnyElementHigh(laserStatus)) { // If one of the LASER_PINS is high
198         blowerStatus = 1;
199         blowerTimerStart = 0; // Reset the timer
200     } else { // If all LASER_PINS are low
201         if (blowerTimerStart == 0) { // If timer is not running
202             blowerTimerStart = millis(); // Start the timer
203             Serial.print("blowertimerstart = ");
204             Serial.println(blowerTimerStart);
205         }
206         else {
207             if ((millis() - blowerTimerStart) >= (blowerOffDelay - blinkDuration) && blowerStatus == 1) { // If blower is about to turn off
208                 warningFlag = 1;
209             }
210             if (millis() - blowerTimerStart >= blowerOffDelay) { // If blowerOffDelay time has passed
211                 blowerStatus = 0;
212                 warningFlag = 0;
213                 blowerTimerStart = -1; // Reset the timer
214             }
215         }
216     }
217 }
```

Deze functie bepaalt **wanneer de blower moet worden ingeschakeld of uitgeschakeld** op basis van de status van drie lasers. De functie gebruikt een **timer** om de blower nog even te laten draaien nadat alle lasers zijn uitgeschakeld, met een waarschuwingssignaal kort vóór het uitschakelen.

De functie begint met een statische variabele `blowerTimerStart`, die bijhoudt wanneer het aftellen naar het uitschakelen van de blower moet beginnen.

Daarna wordt **gecontroleerd of minstens één van de lasers actief is** via de hulpfunctie `isAnyElementHigh(laserStatus)`. Als dit het geval is, wordt de blower ingeschakeld (`blowerStatus = 1`) en wordt de timer gereset naar nul. Hierdoor blijft de blower actief zolang minstens één laser een hoog signaal geeft.

Als echter **alle lasers inactief zijn**, wordt er gekeken of de timer nog niet is gestart (waarde 0). In dat geval wordt de **timer gestart** door de huidige tijd (`millis()`) op te slaan in `blowerTimerStart`. Dit betekent dat de blower nog niet direct wordt uitgeschakeld, maar eerst een soort uitloophase ingaat. Als de timer wél al actief is (dus blower draait nog even door na het uitvallen van de lasers), dan wordt gecontroleerd of de ingestelde vertragingstijd bijna is verstreken. Als dit het geval is én de blower nog actief is, dan wordt het waarschuwingsvlaggetje (`warningFlag`) aangezet. Dit triggert het waarschuwingssignaal uit de lampfunctie om aan te geven dat de blower binnenkort stopt. Tot slot wordt gecontroleerd of **de volledige uitlooptijd** (`blowerOffDelay`) verstreken is. Als dat zo is, wordt de **blower uitgeschakeld**, de waarschuwingsvlag wordt weer gereset, en de timer wordt op -1 gezet om aan te geven dat hij niet meer loopt.

1.5.3.12. Sorteervfunctie drukwaarden

```
219 int sort_desc(const void *cmp1, const void *cmp2)
220 {
221     // Need to cast the void * to int *
222     int a = *((int *)cmp1);
223     int b = *((int *)cmp2);
224     // The comparison
225     return a > b ? -1 : (a < b ? 1 : 0);
226     // A simpler, probably faster way:
227     //return b - a;
228 }
```

De functie `sort_desc()` wordt gebruikt om **twee elementen met elkaar te vergelijken**, specifiek bedoeld voor het sorteren in aflopende (dalende) volgorde.

In het AFCS-programma wordt ze specifiek gebruikt voor het sorteren van de gemeten drukwaarden in de filterzak.

1.5.3.13. Drukmonitoring

```
230 void monitorPressure(float pressure, float maxPressure, int yellowStatus) {
231
232     pressureArray[pressureIndex] = abs(data.presure);
233     Serial.print(data.presure);
234
235     pressureIndex++;
236     Serial.println("Starting median calculation");
237     qsort(pressureArray, pressureLength, sizeof(pressureArray[0]), sort_desc); Serial.print("Pressure = ");
238     float median = pressureArray[int(pressureLength/2)];
239     Serial.print("Median value = ");
240     Serial.println(median);
241     Serial.println(pressureIndex);
242
243     if (median > maxPressure ) {
244
245         lamp(3); // Turn on the yellow LED
246     }
247     if (median < maxPressure && yellowStatus == 1) {
248         lamp(1); // Turn off the yellow LED
249     }
250 }
```

De functie `monitorPressure` **controleert de luchtdruk in het systeem** en beslist op basis daarvan of de gele LED (waarschuwingsschijf) moet worden in- of uitgeschakeld. Bij elke aanroep van de functie wordt eerst de huidige drukwaarde (`data.presure`) opgeslagen in een array `pressureArray`, waarbij alleen de absolute waarde wordt gebruikt. Daarna wordt `pressureIndex` met één verhoogd, zodat de volgende drukmeting op een nieuwe plaats in de array wordt opgeslagen. Meteen daarna begint de **berekening van de mediaan**: de array met drukmetingen wordt gesorteerd met behulp van `qsort()` in aflopende volgorde via een aangepaste sorteermethode `sort_desc` (zie functie hierboven). Zodra de array gesorteerd is, wordt de mediaan berekend als het middelste element in de reeks, namelijk op de index `pressureLength / 2`. Deze waarde wordt afgedrukt voor diagnose. **De mediaan wordt gebruikt omdat die robuuster is tegen uitschieters dan het gemiddelde.**

Daarna worden op basis van deze mediaan twee controles uitgevoerd:

- Als de **mediaan groter** is dan de toegelaten maximale druk (`maxPressure`), dan wordt de functie `lamp(3)` aangeroepen om de **gele LED in te schakelen** als visueel waarschuwingssignaal.
- Als de **mediaan kleiner** is dan `maxPressure` én de gele LED op dat moment nog aan staat (`yellowStatus == 1`), dan wordt `lamp(1)` aangeroepen om **de LED weer uit te schakelen**.

1.5.3.14. Setupfunctie (1)

```
254 void setup() {
255
256     // Initialize relay pins as OUTPUT
257     pinMode(YELLOW_PIN, OUTPUT);
258     pinMode(RED_PIN, OUTPUT);
259     pinMode(GREEN_PIN, OUTPUT);
260     pinMode(BUZZER_PIN, OUTPUT);
261     pinMode(BLOWER_PIN, OUTPUT);
262     pinMode(LASER_PIN_1, INPUT);
263     pinMode(LASER_PIN_2, INPUT);
264     pinMode(LASER_PIN_3, INPUT);
265     pinMode(MANUAL_PIN, INPUT);
266     pinMode(AUTO_PIN, INPUT);
267     delay(500);
268     // Initially turn off all relays
269     digitalWrite(YELLOW_PIN, HIGH);
270     digitalWrite(RED_PIN, HIGH);
271     digitalWrite(GREEN_PIN, HIGH);
272     digitalWrite(BUZZER_PIN, HIGH);
273     digitalWrite(BLOWER_PIN, LOW);
274     delay(500);
275
276     digitalWrite(YELLOW_PIN, LOW);
277     digitalWrite(RED_PIN, LOW);
278     digitalWrite(GREEN_PIN, LOW);
279     digitalWrite(BUZZER_PIN, LOW);
```

In de setup()-functie worden **alle pinnen geconfigureerd** en het systeem in een bekende starttoestand gezet. Allereerst worden de relais- en actuatorpinnen als uitvoer ingesteld met pinMode(..., OUTPUT): de gele, rode en groene LED-relais (YELLOW_PIN, RED_PIN, GREEN_PIN), de zoemer (BUZZER_PIN) en de blower (BLOWER_PIN). De drie laser-inputs (LASER_PIN_1, _2 en _3) en de twee standen-schakelaars voor handmatig en automatisch (MANUAL_PIN en AUTO_PIN) worden met pinMode(..., INPUT) als ingangen geconfigureerd.

Na een korte vertraging van 500 ms volgt de eerste “soft start” om alle relais in een uit-stand te brengen. Omdat de relais waarschijnlijk actief-laag geschakeld zijn, wordt met digitalWrite(..., HIGH) elke uitgang in de hoog-stand gezet, waardoor de relais uitschakelen. Vervolgens wordt na nog eens 500 ms met digitalWrite(..., LOW) elke uitgang teruggezet naar laag—dit is bedoeld om de relais kort te activeren en dan definitief uit te schakelen, of om een **hardware-reset** te forceren. Op deze manier begint het systeem altijd met alle relais en actuatoren in de uit-stand, klaar voor de hoofdloop om de status van de sensoren en schakelaars te gaan volgen.

1.5.3.15. Setupfunctie (2)

```
285 //Start wifi connection
286 //initWiFi();
287 //configTime(0, 0, ntpServer);
288
289 // Start Serial communication
290 Serial.begin(9600);
291 int N = 0;
292 while (lwlp.begin() != 0 or N<=10) {
293     digitalWrite(BUZZER_PIN, !digitalRead(BUZZER_PIN));
294     N++;
295     delay(500);
296 }
297
298 }
```

Dit stukje code komt uit de setup()-functie en handelt de **opstart van de seriële communicatie én de initialisatie van de druksensor** (lwlp.begin()) af. Eerst wordt met Serial.begin(9600) de seriële communicatie gestart met een baudrate van 9600. Dit is belangrijk voor het verzenden van gegevens naar bijvoorbeeld de seriële monitor op een computer. Daarna wordt de integer N geïnitieerd op 0. Vervolgens start een while-lus die twee voorwaarden combineert: lwlp.begin() != 0 (de sensor faalt bij opstart) of N <= 10 (de pogingsteller is nog niet over de limiet heen). De bedoeling is om maximaal tien pogingen te doen om de druksensor correct te initialiseren.

Binnen de lus:

- De zoemer (BUZZER_PIN) wordt aan- en uitgezet bij elke iteratie door zijn huidige status om te keren met !digitalRead(BUZZER_PIN). Dit resulteert in een knipperend geluidssignaal dat aangeeft dat de sensorinitialisatie nog bezig is of faalt.
- N++ verhoogt het aantal pogingen.
- delay(500) zorgt ervoor dat er een halve seconde tussen de pogingen zit.

Deze logica probeert de sensor tot maximaal 10 keer op te starten en waarschuwt visueel/auditief met de buzzer als het niet lukt. Het geeft de gebruiker zo een signaal dat er een opstartprobleem is.

1.5.3.16. Loopfunctie (1)

```
302 void loop() {
303
304     currentMillis = millis();
305     Digital_IN();
306     float maxPressure = floatMap(potentialValue, 0, 4095, 0, 500); // map the measured potentiometer
307
308     // Things to do when the switch is set to auto mode (II)
309     if (autoStatus == 1) {
310         if (blowerStatus != 1) {
311             lamp(9);
312         }
313         Digital_OUT();
314         if (blowerStatus == 0 && isAnyElementHigh(laserStatus)) {
315             lamp(2);
316         }
317         if (warningFlag == 1) {
318             lamp(2);
319             warningFlag = 0;
320         }
321         // Continuously run the blower function
322         blower(laserStatus);
323         monitorPressure(pressure, maxPressure, yellowStatus);
324         lamp(4);
325     }
326 }
```

In deze loop()-functie van een Arduino-programma worden **metingen en acties uitgevoerd op basis van de huidige systeemstatus**. Allereerst wordt met millis() de huidige tijd in milliseconden opgeslagen in de variabele currentMillis. Vervolgens wordt Digital_IN() aangeroepen om alle digitale invoersignalen in te lezen. Daarna wordt met floatMap() de gemeten waarde van een potentiometer (potentialValue) omgerekend van het bereik 0-4095 naar een drukwaarde tussen nul en 500, die wordt opgeslagen in maxPressure. Dit is de grenswaarde voor de drukmeting in de filterzak.

De code **controleert** daarna **of het systeem in automatische modus staat** (autoStatus == 1). Als dat zo is, volgen verschillende acties.

- Ten eerste wordt gecontroleerd of de blower actief is (blowerStatus != 1); als dit niet het geval is, wordt lamp(9) geactiveerd om aan te geven dat het systeem in auto-modus staat. Daarna wordt Digital_OUT() uitgevoerd om alle uitgangen aan te sturen.
- Als de blowerstatus op nul staat (dus uit) én er is een hoge status op een lasersensor (isAnyElementHigh(laserStatus)), wordt lamp(2) geactiveerd. Eveneens, als er een waarschuwingsvlag (warningFlag) actief is, wordt ook lamp(2) geactiveerd en de vlag weer op nul gezet.

Daarna draait het programma continu de **blower-functie** (blower(laserStatus)) om **op basis van de lasersensor de blower aan te sturen**. Ook wordt de **druk bewaakt met monitorPressure()**, waarbij de actuele druk, de maximale druk, en de status van een gele indicator worden meegegeven. Tot slot wordt lamp(4) aangestuurd, wellicht als algemene status- of controlelamp.

1.5.3.17. Loopfunctie (2)

```
327 // Things to do when the switch is off
328 if (manualStatus == 0 && autoStatus == 0) {
329     lamp(8);
330     Digital_OUT();
331     if (blowerStatus == 1 ) {
332         lamp(2);
333     }
334     blowerStatus = 0;
335     lamp(4);
336     monitorPressure(pressure, maxPressure, yellowStatus);
337 }
```

Dit gedeelte van de code beschrijft wat er moet gebeuren wanneer **het systeem in een volledig uitgeschakelde of inactieve toestand** verkeert. Als deze conditie waar is, wordt als eerste lamp(8) aangestuurd om aan te geven dat het systeem in uit-stand staat. Vervolgens wordt Digital_OUT() aangeroepen om. Daarna controleert de code of de blower nog actief is (blowerStatus == 1). Als dat het geval is, wordt lamp(2) geactiveerd — als waarschuwing dat de blower nog actief was en zal worden uitgezet. Vervolgens wordt de blower expliciet uitgeschakeld door blowerStatus = 0. Tot slot wordt lamp(4) ingeschakeld, misschien om de status van deze systeemtoestand aan te geven, en wordt monitorPressure() aangeroepen om de druk te blijven bewaken.

1.5.3.18. Loopfunctie (3)

```
338
339 //Things to do when switch is set to manual mode (I)
340 if (manualStatus == 1) {
341     if (blowerStatus == 0) {
342         lamp(2);
343     }
344     lamp(1);
345     blowerStatus = 1;
346     monitorPressure(pressure, maxPressure, yellowStatus);
347     lamp(4);
348
349 }
350 Digital_OUT();
351 }
```

In dit deel van de code wordt beschreven wat het systeem moet doen wanneer **de schakelaar is ingesteld op de handmatige modus** (`manualStatus == 1`). Dit betekent dat de gebruiker zelf de blower aanzet. Dient als een fallback-methode moest er iets mislopen met de auto-modus. Als de blower nog niet actief is (`blowerStatus == 0`), wordt `lamp(2)` ingeschakeld, als waarschuwing of indicatie dat de blower wordt geactiveerd. Daarna wordt `lamp(1)` ingeschakeld, wat aangeeft dat de handmatige modus actief is. De blower wordt vervolgens aangezet door `blowerStatus = 1`. Daarna wordt de druk bewaakt via de functie `monitorPressure()`, waarbij de huidige druk, de ingestelde maximale druk en de status van een gele indicatorlamp worden meegegeven. Tot slot wordt `lamp(4)` geactiveerd. Na deze handmatige handelingen wordt buiten de if-structuur nog `Digital_OUT()` aangeroepen, wat zorgt voor het actualiseren van de digitale uitgangen op basis van de nieuwe status.

2. Troubleshooting

In dit hoofdstuk worden de voornaamste problemen besproken die tijdens het project werden ondervonden, samen met de bijbehorende oplossingen. Deze toelichting dient als referentie om gelijkaardige fouten in de toekomst te vermijden — zowel bij de ontwikkeling van een nieuw systeem als bij de aanpassing van een bestaand systeem aan specifieke noden.

Probleem	Oplossing
Afbreken van koperdraadjes	Gebruik geen harde draden (die in een bepaalde vorm blijven staan) maar gebruik zachte draden. Zachte draden zijn minder onderhevig aan metaalmoeheid.
Loskomen van solderingen	Gebruik zoveel mogelijk aansluitingen via schroefterminals.
Loskomen van draadjes uit schroefterminals	Gebruik adereindhulzen die je vastknijpt op het uiteinde van een kabeltje met de bijgeleverde krimptang. Draai de schroefterminal helemaal open met behulp van een kleine schroevendraaier, schuif de adereindhuls erin en draai de schroefterminal goed vast. Door deze hulzen te gebruiken, wordt het uiteinde van een kabeltje wat dikker en kan je deze beter vastklemmen in de terminal.
Resetten van ESP32	Elektromagnetische storing. Plaats contactor en ESP32 verder uit elkaar.
Foute spanningsniveaus	<p>Aansluitingen nakijken</p> <ul style="list-style-type: none">- Zijn schroefterminals goed aangedraaid?- Zijn solderingen proper gebeurd, zonder barsten, zijn er geen onopgevulde gaatjes?- Zijn draden op de juiste plek aangesloten (geen 3.3V aan 5V en vice versa)? <p>Onderdelen nakijken</p> <ul style="list-style-type: none">- Geven ze meetwaarden die liggen binnen de verwachtingen? Indien niet is dit onderdeel mogelijk defect.

3. Verbetervoorstellen

3.1. Noodstop

In de huidige opstelling werd de noodstop uit het circuit gehaald om de aansluiting van de contactor aan de blower te vergemakkelijken. Om het systeem veiliger te maken zou het wenselijk zijn om terug een noodstop te introduceren. Voorwaarde is wel dat deze noodstop dan de mogelijkheid verschaft om uitgelezen te worden via de ESP32. Zonder de mogelijkheid om de laatste status op te vragen weet AFCS niet of de noodstop ingeschakeld is of niet na een reset.

3.2. Internetverbinding

De ESP32-microcontroller beschikt over een ingebouwde wifi-module, wat het mogelijk maakt om verbinding te maken met een lokaal netwerk of het internet. Van zodra het systeem verbonden is met het internet, ontstaan er tal van interessante uitbreidingsmogelijkheden op het vlak van monitoring, automatisering en gebruiksgemak.

3.2.1. Online dashboard en datalogging

Een online dashboard kan worden opgezet om in real-time gegevens te visualiseren en op te slaan. Dit biedt de mogelijkheid tot centrale opvolging van meerdere systemen of locaties.

Mogelijke gegevens die kunnen worden bijgehouden zijn:

- Het totaal aantal gepresteerde uren van de blower
- Het gemiddeld aantal bewerkingscycli per dag/week/maand
- De gemiddelde bewerkingstijd van een PCB
- Aantal meldingen i.v.m. volle filterzakken
- De gemiddelde vervangtijd van filterzakken

3.2.2. Predictive maintenance

Door gegevens over tijd, gebruik en meldingen te analyseren, kan men trends detecteren die wijzen op aankomend onderhoud. Dit maakt het mogelijk om preventief in te grijpen vóórdat een storing optreedt.

Mogelijkheden hiervoor zijn:

- Automatische meldingen bij afwijkende drukmetingen (filterverstopping)
- Waarschuwingen bij verhoogd stroomverbruik van de blower
- Onderhoudsherinneringen op basis van gebruiksduur

3.2.3. Monitoring vanuit kantoor

Een centraal controlepaneel in de bureauruimte kan toelaten om de status van het AFCS-systeem te controleren zonder fysiek naar de ruimte te gaan. Visuele weergave van filterstatus en systeemstatus kunnen hier interessant zijn. De mogelijkheid om het systeem handmatig op afstand uit te schakelen of te resetten kan ook een meerwaarde zijn.

3.3. Geoptimaliseerde temperatuurregeling

Door de gegenereerde warmte van het lasersysteem slim te benutten, kan de verwarming van de ruimte efficiënter verlopen.

- In de winter kan restwarmte van het afzuigstelsel (blower) worden gerecupereerd om de ruimte bij te verwarmen.
- Bij een teveel aan warmte kan warme lucht worden afgevoerd naar de aangrenzende trainingshal, die vaak ondertemperatuur heeft.
- Belangrijk hierbij is dat de afgevoerde lucht eerst wordt gefilterd om stofdeeltjes te verwijderen. Aangezien dit een extra filterkost met zich meebrengt, moet er worden overlegd wie deze kosten op zich neemt (bv. eigenaar van de trainingshal).

Een geautomatiseerde regeling kan op basis van temperatuurmetingen en gebruikstoestand van de blower beslissen wanneer welke ventilatierichting wordt ingeschakeld.

3.4. Eigen PCB

Het ontwikkelen van een op maat gemaakte printplaat biedt de mogelijkheid om het huidige prototype te professionaliseren. Hierdoor wordt het systeem goedkoper, compacter, betrouwbaarder en eenvoudiger reproduceerbaar.

Enkele voordelen hiervan zijn:

- Vaste en overzichtelijke aansluitpunten voor alle componenten
- Verbeterde signaalintegriteit door geoptimaliseerde layout
- Minder foutgevoelige bedrading en meer mechanische stabiliteit

3.5. Uitgebreide statusweergave

De bestaande signaallamp met en zoemer kan uitgebreid worden met extra indicatiefuncties om foutmeldingen of systeemevents visueel te communiceren. Zo kunnen bijvoorbeeld alle drie de lampen samen pinken om foutindicaties te geven.

Voorbeelden van foutindicaties:

- Knipperpatroon 1x: fout bij druksensor
- Knipperpatroon 2x: Wi-Fi-verbinding onderbroken met ESP32

Deze foutcodes kunnen ook gekoppeld worden aan interne logs of externe meldingen (bv. via internet).

Besluit

De stageopdracht bij CERcuits had als hoofddoel het ontwikkelen van een geautomatiseerd systeem dat het gebruik van de afzuiginstallatie optimaliseert in functie van de laseractiviteit. Dit systeem, het Air Filtration Control System (AFCS), werd succesvol ontworpen, gebouwd en geïmplementeerd op vier productielijnen. Zoals vooropgesteld in de inleiding, was het voornaamste doel om het energieverbruik — en dus de bijhorende kosten — aanzienlijk te verlagen, zonder de workflow van de werknemers te verstoren.

Uit de energieberekening blijkt dat AFCS een aanzienlijke besparing van ongeveer 8400 kWh per jaar oplevert, goed voor een vermeden kost van 2772.97 euro op jaarbasis. Bovendien wordt de levensduur van de blowers verlengd doordat ze enkel actief zijn wanneer nodig, wat op termijn ook onderhouds- en vervangingskosten reduceert. Aangezien 4 systemen samen 711.48 euro kosten is de return on investment na het eerste jaar 2061.49 euro.

Naast de technische realisatie betekende dit project voor mij ook een waardevolle persoonlijke ontwikkeling. Ik kreeg de kans om het volledige ontwikkeltraject van een product te doorlopen — van behoefteanalyse en hardwareselectie tot softwareontwikkeling en probleemoplossing. Er was tijdens de stage een goede balans tussen zelfstandigheid en begeleiding, wat essentieel was voor mijn leerproces. Ik had de vrijheid om zelfstandig keuzes te maken en het systeem uit te breiden of te verbeteren na overleg met Ruben, mijn stagementor. Onze regelmatige feedbackmomenten hielpen bij het optimaliseren van de bestellijst, het verbeteren van de code en het afleveren van een robuuster eindproduct. In de laatste fase van mijn stage heb ik ook veel bijgeleerd van Ruben en andere collega's — en hopelijk kon ik op mijn beurt ook hen af en toe iets bijbrengen.

Het AFCS bewijst dat een goed doordachte automatisering niet alleen ecologische en economische voordelen biedt, maar ook het productieproces kan optimaliseren. Dit verslag bundelt dan ook niet alleen het technische resultaat, maar weerspiegelt eveneens mijn evolutie als IT'er binnen een bedrijfscontext. De opgedane kennis en ervaring geven me het vertrouwen om in de toekomst soortgelijke uitdagingen aan te gaan. Ik zie het dan ook als een waardevolle eerste stap richting een carrière waarin ik duurzame en slimme technologieën wil blijven ontwikkelen die een tastbare impact hebben op de werkvloer.

Bibliografie

- 123-3D.nl. (z.d.). *Mean Well voeding 24V 52.8 W 2.2 A gesloten chassis LRS-50-24*. Geraadpleegd van <https://www.123-3d.nl/MeanWell-Mean-Well-voeding-24V-52-8-W-2-2-A-gesloten-chassis-LRS-50-24-i1699-t13129.html>
- All3DP. (z.d.). *ESP32 vs Arduino: Differences*. Geraadpleegd van <https://all3dp.com/2/esp32-vs-arduino-differences/>
- Amazon. (z.d.). *Power Bridge Rectifier MDS100A Three Phase*. Geraadpleegd van <https://www.amazon.com/Power-Bridge-Rectifier-MDS100A-Three-Phase/dp/B07GZR8JC7>
- Arduino AG. (z.d.). *Arduino Uno Rev3. Bits & Parts*. Geraadpleegd van https://www.bitsandparts.nl/product/ARDUINO_UNO_REV3
- Boardor. (z.d.). *Understanding microcontrollers: 51, Arduino, ESP32, STM32 and Raspberry Pi*. Geraadpleegd van <https://boardor.com/blog/understanding-microcontrollers-51-arduino-esp32-stm32-and-raspberry-pi>
- Build Electronic Circuits. (z.d.). *Transistor as a switch*. Geraadpleegd van <https://www.build-electronic-circuits.com/transistor-as-a-switch/>
- Cadence. (z.d.). *The Venturi Effect and the Bernoulli's Principle*. Geraadpleegd van <https://resources.system-analysis.cadence.com/blog/msa2022-the-venturi-effect-and-bernoullis-principle>
- Circuit Digest. (z.d.). *Interfacing Dust Sensor with Arduino*. Geraadpleegd van <https://circuitdigest.com/microcontroller-projects/interfacing-dust-sensor-with-arduino>
- CloudTweaks. (2024, maart). *Choosing the Right Microcontroller*. Geraadpleegd van <https://cloudtweaks.com/2024/03/choosing-the-right-microcontroller/>
- DFRobot. (z.d.). *Differential Pressure Sensor $\pm 500\text{Pa}$ SKU SEN0343*. Geraadpleegd van https://wiki.dfrobot.com/Differential_Pressure_Sensor_%C2%B1500pa_SKU_SEN0343
- DFRobot. (z.d.). *Water Level Pressure Sensor*. Geraadpleegd van <https://www.dfrobot.com/product-2096.html>
- Dwyer Omega. (z.d.). *An Introduction to Venturi Flow Meters, Flow Nozzles, and Segmental Wedge Elements*. Geraadpleegd van <https://www.dwyeromega.com/en-us/resources/venturi-meter>
- Edmund Optics. (z.d.). *The Correct Material for Infrared (IR) Applications*. Geraadpleegd van <https://www.edmundoptics.com/knowledge-center/application-notes/optics/the-correct-material-for-infrared-applications/>
- Electronics Tutorials. (z.d.). *Three Phase Rectification*. Geraadpleegd van <https://www.electronics-tutorials.ws/power/three-phase-rectification.html>
- HBM. (z.d.). *Pressure Transducers & Pressure Sensors*. Geraadpleegd van <https://www.hbm.com/en/0297/pressure-transducers-pressure-sensors-product-overview/>
- Instructables. (z.d.). *Building an Electronics Enclosure*. Geraadpleegd van <https://www.instructables.com/Building-an-Electronics-Enclosure/>
- Instructables. (z.d.). *How to Interface With Optical Dust Sensor*. Geraadpleegd van <https://www.instructables.com/How-to-Interface-With-Optical-Dust-Sensor/>
- Instructables. (z.d.). *How to Read MPX5010 Differential Pressure Sensor With Arduino*. Geraadpleegd van <https://www.instructables.com/How-to-Read-MPX5010-Differential-Pressure-Sensor-W/>
- Kiwi Electronics. (z.d.). *Raspberry Pi 4 Model B – 8GB*. Geraadpleegd van <https://www.kiwi-electronics.com/nl/raspberry-pi-4-model-b-8gb-10032>
- LED24.nl. (z.d.). *Voedingsadapter 24V 72W*. Geraadpleegd van <https://www.led24.nl/voedingsadapter-24v-72w.html>

- Li, Y., Wang, X., & Zhang, L. (2021). Development and performance detection of higher precision optical sensor for coal dust concentration measurement. *Optics and Lasers in Engineering*, 139, 106514. <https://doi.org/10.1016/j.optlaseng.2020.106514>
- Lucid. (z.d.). *Weighted Decision Matrix*. Geraadpleegd van <https://lucid.co/blog/weighted-decision-matrix>
- Omvormer.nu. (z.d.). *230V naar 24V AC-DC adapter voedingen*. Geraadpleegd van <https://www.omvormer.nu/omvormers/ac-dc-adapter-voedingen/230v-naar-24>
- Random Nerd Tutorials. (z.d.). *Raspberry Pi Pico W Pinout: GPIOs*. Geraadpleegd van <https://randomnerdtutorials.com/raspberry-pi-pico-w-pinout-gpios/>
- Raspberry Pi Foundation. (z.d.). *Raspberry Pi 4 Model B – 8GB*. Kiwi Electronics. Geraadpleegd van <https://www.kiwi-electronics.com/nl/raspberry-pi-4-model-b-8gb-10032>
- Tameson. (z.d.). *Transformer Three Phase*. Geraadpleegd van <https://tameson.com/pages/transformer-three-phase>
- TechTarget. (z.d.). *What Is Transistor-to-Transistor Logic (TTL)?*. Geraadpleegd van <https://www.techtarget.com/whatis/definition/transistor-to-transistor-logic-TTL>
- Tettra. (2024, juni 10). *Confluence vs Sharepoint: What's Best for Knowledge Management?*. Geraadpleegd van <https://tettra.com/article/confluence-vs-sharepoint/>
- The Verge. (2024, november 25). *Raspberry Pi Pico W adds Wi-Fi to the \$6 microcontroller*. Geraadpleegd van <https://www.theverge.com/2024/11/25/24305455/raspberry-pi-pico-2-w-wi-fi-wireless-microcontroller>
- Wikipedia. (z.d.). *Bernoulli's principle*. Geraadpleegd van https://en.wikipedia.org/wiki/Bernoulli%27s_principle
- Wikipedia. (z.d.). *ESP32*. Geraadpleegd van <https://en.wikipedia.org/wiki/ESP32>
- Wikipedia. (z.d.). *Luchtdruk*. Geraadpleegd van <https://nl.wikipedia.org/wiki/Luchtdruk>
- Wikipedia. (z.d.). *Venturi effect*. Geraadpleegd van https://en.wikipedia.org/wiki/Venturi_effect
- Wikipedia. (z.d.). *Water*. Geraadpleegd van <https://nl.wikipedia.org/wiki/Water>
- Wikipedia. (z.d.). *Wet van Pascal*. Geraadpleegd van https://nl.wikipedia.org/wiki/Wet_van_Pascal
- Yao, K., Lin, Q., Jiang, Z., Zhao, N., Peng, G.-D., Tian, B., Jia, W., & Yang, P. (2019). Design and Analysis of a Combined Strain–Vibration–Temperature Sensor with Two Fiber Bragg Gratings and a Trapezoidal Beam. *Sensors*, 19(16), 3571. <https://doi.org/10.3390/s19163571>
- YoungCapital. (z.d.). *Werkdagen per maand 2024*. Geraadpleegd van <https://www.youngcapital.nl/carriere/werkleven/werkdagen#werkdagen-per-maand-2024>
- YouTube. (z.d.). *Differentiaaldruksensor uitleg*. Geraadpleegd van <https://www.youtube.com/watch?v=UrqPxwsPWGk>
- YouTube. (z.d.). *MPX5010 Sensor uitlezen*. Geraadpleegd van <https://www.youtube.com/watch?v=kDkusNay2ZQ>
- YouTube. (z.d.). *Transistor als schakelaar*. Geraadpleegd van https://www.youtube.com/watch?v=l_b4oHwepI
- HBM Machines. (z.d.). *HBM 100 Profi Stofafzuiginstallatie*. Geraadpleegd van <https://www.hbm-machines.com/nl/p/hbm-100-profi-stofafzuiginstallatie?cr=jou9#specifications>




Bijlagen





VERGELIJKING MICROCONTROLLERS






Eigenschap	Arduino Uno	ESP32 Devkit V1	Raspberry Pi Pico	Raspberry Pi 4
Microcontroller	ATmega328P	ESP32	RP2040	Broadcom BCM2711
Architectuur	8-bit AVR	32-bit Xtensa LX6	32-bit ARM Cortex-M0+	64-bit ARM Cortex-A72
Kloksnelheid	16 MHz	Tot 240 MHz	Tot 133 MHz	1.5 GHz
Flashgeheugen	32 KB	Tot 4 MB	2 MB	Afhankelijk van SD-kaart
SRAM	2 KB	520 KB	264 KB	Afhankelijk van RAM-model (1-8 GB)
EEPROM	1 KB	512 Bytes	Geen	Afhankelijk van opslag
GPIO-pinnen	14 digitaal, 6 analoog	25	26	40
Analoge ingangen	6	Tot 18	3	Geen
PWM-kanalen	6	Tot 16	16	Afhankelijk van software
Communicatie	UART, SPI, I ² C	UART, SPI, I ² C, Wi-Fi, Bluetooth	UART, SPI, I ² C, USB	USB, Ethernet, Wi-Fi, Bluetooth
USB-ondersteuning	Ja	Ja	Ja	Ja
Wi-Fi / Bluetooth	Nee	Ja	Nee	Ja
Spanning	5V	3.3V	3.3V	5V via USB-C
Prijsindicatie	€15–€20	€5–€10	€4–€6	€35–€90






BILL OF MATERIALS (BOM)





Hieronder vind je een overzicht van de benodigde materialen voor het bouwen van één AFCS.


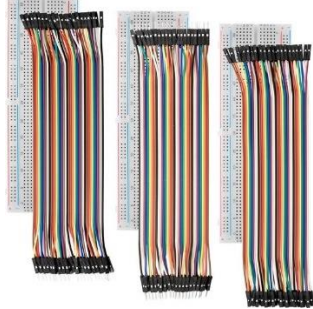

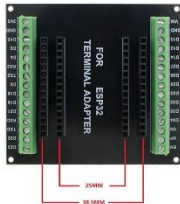
Naam	Foto	Prijs/stuk (€)	Aantal	Totaal (€)
ESP32 Devkit	 <p><i>Figuur 12: ESP32</i></p>	8.55	1	8.55
Optocoupler twee-kanaals	 <p><i>Figuur 13: Optocoupler</i></p>	3.00	3	9.00
Relaisbord vier-kanaals	 <p><i>Figuur 14: Relaisbord vier-kanaals</i></p>	7.00	1	7.00

Relaisbord 1-kanaals	 <p><i>Figuur 15: Relaisbord 1-kanaals</i></p>	2.25	1	2.25
Voedingskabel 230V	 <p><i>Figuur 16: Voedingskabel 230V</i></p>	4.00	1	4.00
Voeding 24V + 5V	 <p><i>Figuur 17: Dubbele voeding (24V & 5V)</i></p>	19.32	1	19.32
Differentiële druksensor LWLP5000-5XD	 <p><i>Figuur 18: Differentiële druksensor</i></p>	39.73	1	39.73

Schakelaar drie inputs (2-pack)	 <p><i>Figuur 19: Schakelaars (drie inputs)</i></p>	4.89	1	4.89
Signaallamp drie kleuren + zoemer	 <p><i>Figuur 20: Driekleurenlamp + zoemer</i></p>	17.68	1	17.68
Contactor Siemens 3RT1015-1BB41	 <p><i>Figuur 21: Contactor</i></p>	57.75	1	57.75
Potentiometer 10k	 <p><i>Figuur 22: Potentiometer</i></p>	0.50	1	0.50
Aftakdoos	 <p><i>Figuur 23: Aftakdoos</i></p>	19.70	1	19.70

DIN 5 Kabel (3m)	 <p><i>Figuur 24: Kabel type DIN 5</i></p>	5.79	1	5.79
DIN 5 connector Inbouw Vrouwelijk	 <p><i>Figuur 25: DIN 5 Panel Mount</i></p>	1.09	2	2.18
Wartel	 <p><i>Figuur 26: Wartel</i></p>	0.63	1	0.63
Kabel (5m) Dubbele Jack 3.5 mm	 <p><i>Figuur 27: Dubbele Jack (AUX) 5m</i></p>	5.29	2	10.58
Kabel (10m) Dubbele Jack 3.5 mm	 <p><i>Figuur 28: Dubbele Jack (AUX) 10m</i></p>	10.99	1	10.99

Jack connector Inbouw Vrouwelijk	 <i>Figuur 29: Jack Panel Mount</i>	1.09	3	3.27
Stekkerhulskit + tang	 <i>Figuur 30: Stekkerhulskit</i>	28.91	1	28.91
Adereindhulzenkit + tang	 <i>Figuur 31: Adereindhulzenkit</i>	19.99	1	19.99
Draaiknop	 <i>Figuur 32: Draaiknop</i>	0.50	1	0.50

Verbindingsklem WAGO (50st)	 <p><i>Figuur 33: WAGO-Klem</i></p>	16.15	1	16.15
Jumper wires	 <p><i>Figuur 34: Jumper Wire Kit</i></p>	9.99	1	9.99
Rubberen slang ø 2 mm binnen ø 6 mm buiten	 <p><i>Figuur 35: Rubberen slang (6mm)</i></p>	2.75/m	2m	5.50
Breakout bord ESP32	 <p><i>Figuur 36: Breakout Board ESP32</i></p>	3.90	1	3.90

Koperdraad (10m) Rood	 <p><i>Figuur 37: Koperdraad (rood)</i></p>	1.95	1	1.95
Koperdraad (10m) Zwart	 <p><i>Figuur 38: Koperdraad (zwart)</i></p>	1.95	1	1.95
Totaalprijs AFCS				177.87 *

* Totale kost van één systeem volgens de BOM bedraagt 235.62 euro. Het bedrijf bezat nog vier contactoren waardoor 57.75 euro uitgespaard kon worden en het totaalbedrag op 177.87 euro uitkwam.

BEREKENINGEN ENERGIEBESPARING

Info blower:

- Spanning: 230V
- Stroom: 4.8A
- Vermogen: 746 Watt
 - 0.746 kWh

Verbruik vier blowers:

- $4 * 0.746 \text{ kWh} = 2.984 \text{ kWh}$

Gemiddelde maximale duur van een bewerking = 3u

Laatste bewerking wordt opgezet ten laatste om: 18u

- ➔ $\text{Bewerking klaar om } (18 + 3 =) 21\text{u}$
- ➔ $\text{Onnodig ingeschakelde duur} = 21\text{u tot } 8\text{u 's morgens} = 11\text{u}$

$\text{Onnodig verbruik blowers/dag} = 2.984 \text{ kWh} * 11\text{h} = 32.824 \text{ kWh}$

$\text{Onnodig verbruik blowers/week} = 32.824 \text{ kWh} * 5 \text{ (dagen)} = 164.12 \text{ kWh}$

- $\text{Gemiddeld aantal weekdays/maand} = 22$

$\text{Onnodig verbruik blowers/maand} = 32.824 * 22 \text{ (dagen)} = 722.128 \text{ kWh}$

- $\text{Gemiddeld aantal weekdays/jaar} = 256$

$\text{Onnodig verbruik blowers/jaar} = 32.824 * 256 = 8402.944 \text{ kWh}$

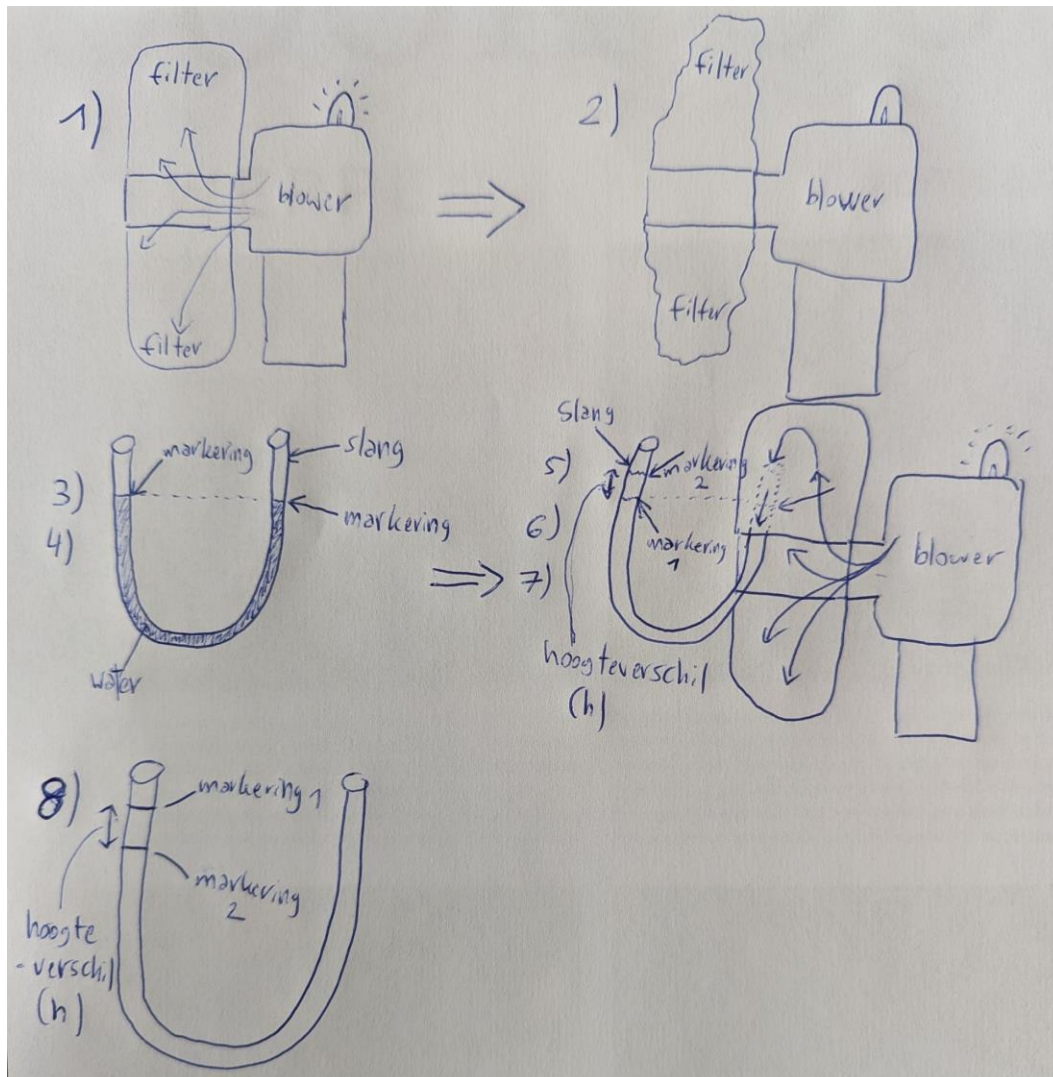
$\text{Gemiddelde prijs kWh 2024} = 0.33 \text{ EUR}$

$\text{Onnodige kost blowers/jaar} = 8402.944 * 0.33 = 2772.97 \text{ Euro}$

STAPPENPLAN DRUKMETING IN FILTERZAK:

1. Wacht tot de filterzak volledig vol is
2. Zet de blower uit
3. Vul een transparante slang gedeeltelijk met water
4. Houd de slang in U-vorm zodat het water in balans komt
→ Markeer het waterniveau aan beide zijden
5. Plaats één kant van de slang in de filterzak
6. Zet de blower weer aan en wacht tot de zak volledig opgeblazen is
7. Markeer opnieuw het waterniveau aan beide zijden
8. Meet het hoogteverschil tussen de twee waterniveaus
9. Bereken de druk met de formule: $p = \rho \cdot g \cdot h$
 - ρ = dichtheid van water (1000 kg/m^3)
 - $g = 9.81 \text{ m/s}^2$
 - h = hoogteverschil in meters
10. Dit was het resultaat in mijn geval:

$$p = 1000 \text{ kg/m}^3 \cdot 9.81 \text{ m/s}^2 \cdot 0.02 \text{ m} = 196.2 \text{ Pascal}$$

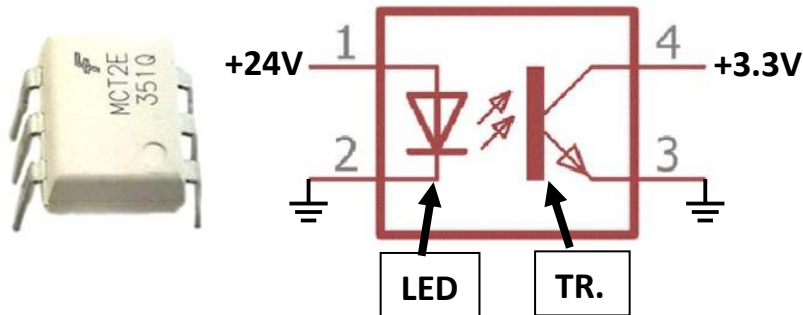


AANSLUITSCHEMA ESP32

Terminal op ESP32 (Breakout Board)	Aangesloten op
3V3	Rij één op het perfboard. Op deze rij worden alle zaken aangesloten die 3.3V vereisen. Dit zijn: <ul style="list-style-type: none"> • Potentiometer (+Pool). • Middelste contact van schakelaar (0) • VCC van druksensor
GND	Rij twee op het perfboard. Op deze rij worden de GND-pinnen van deze componenten aangesloten.
D2	CH1 van 4-kanaals relaisbord. Rode lamp.
D4	CH2 van 4-kanaals relaisbord. Gele lamp.
D5	CH3 van 4-kanaals relaisbord. Groene lamp.
D18	CH4 van 4-kanaals relaisbord. Zoemer.
D19	CH1 van 1-kanaals relaisbord. Blower.
D21	SDA-pin Druksensor.
D22	SCL-pin Druksensor.
D13	Plek X op perfboard. Verbinden met V1 van optocoupler horende bij Laser één op perfboard (gebruik hiervoor de dubbele Jack). Verbinden met pulldown weerstand op perfboard.
D27	Plek Y op perfboard. Verbinden met V1 van optocoupler horende bij Laser twee op perfboard (gebruik hiervoor de dubbele Jack).. Verbinden met pulldown weerstand op perfboard.
D14	Plek Z op perfboard. Verbinden met V1 van optocoupler horende bij Laser drie op perfboard (gebruik hiervoor de dubbele Jack). Verbinden met pulldown weerstand op perfboard.
D32	Middelste pin Potentiometer.
D35	Pin Manuele modus. Aansluiten aan de (I) zijde van de schakelaar.
D34	Pin Auto-modus. Aansluiten aan de (II) zijde van de schakelaar.
Vin	5V. Rechtstreeks op de voeding of aan een 5V ingang van één van de relaisbordjes op voorwaarde dat deze op hun beurt verbonden zijn aan 5V van de voeding.

EXTRA UITLEG ONDERDELEN

OPTOCOUPLER (2-KANAALS)



Figuur 39: Werking Optocoupler

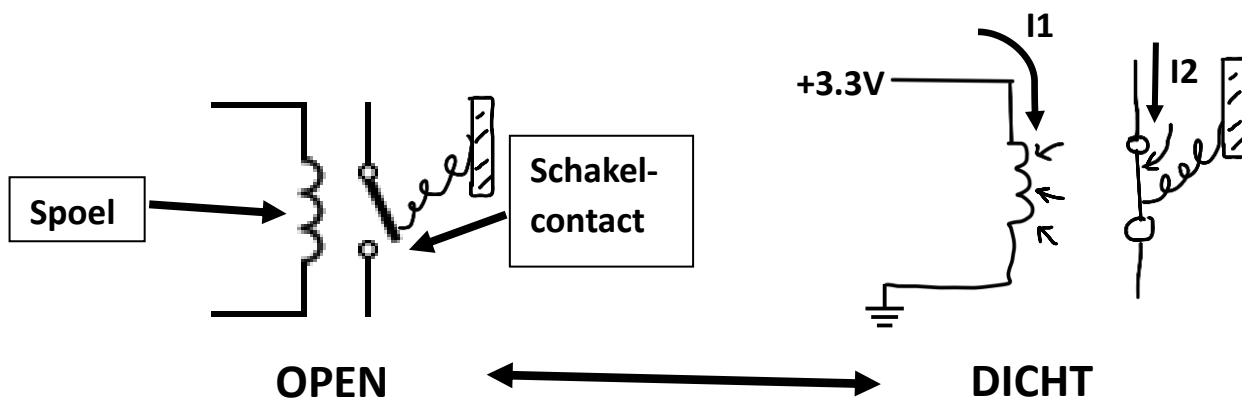
Aan de linkerzijde zien we aan de bovenkant een spanningsniveau van 24 Volt. Dit is de kant van de laser, hier gaan we een signaal meten dat de rest van het systeem nodig heeft als trigger om de blower op te zetten. Zoals je kan zien in de schematische voorstelling bevindt er zich een LED. Wanneer er spanning wordt aangelegd aan de linkerzijde zal er stroom door de LED lopen en zal deze dus oplichten.

Aan de rechterzijde bevindt er zich een lichtgevoelige transistor (TR.) die enkel en alleen stroom zal geleiden als er licht op valt. Er kan geen stroom rechtstreeks van de ledzijde naar de transistorzijde vloeien omdat er een elektrisch isolerend doorzichtig materiaal tussen zit (kan ook gewoon lucht zijn maar meestal is dit een soort kunststof). Wanneer de lichtgevoelige transistor licht opvangt, begint er stroom te vloeien aan de rechterzijde. Aangezien dit een circuit is dat een maximale spanning heeft van 3.3 volt, is dit geen probleem voor onze ESP32. Op deze manier kan het signaal van de laserzijde worden doorgegeven aan de ESP32 zonder fysiek contact te maken en weet de ESP32 dus wanneer de laser actief is.

RELAISBORD (VIER-KANAALS & EEN-KANAALS)



Figuur 40: Relaisbord vier-kanaals - groot



Figuur 41: Relais werking

In een relais zit een spoel en een schakelcontact. Een spoel ziet er wat uit als een staafje waar heel veel koperdraad is rond gewikkeld. Het schakelcontact ziet er wat uit als een metalen deur die altijd openstaat. Vaak is deze verbonden aan een veer zodat ze altijd blijft openstaan. Wat gebeurt er nu als je spanning (+ 3.3V boven en 0V beneden) aanbrengt op de spoel? Zoals je kan zien begint er dan een stroom I1 van boven naar beneden door de spoel te lopen. Deze stroom zorgt ervoor dat er een magnetisch veld wordt opgewekt in de spoel (drie pijltjes naast de spoel). De schakelaar (de metalen 'deur') die open bleef staan door de veer, wordt nu zo hard aangetrokken door de spoel dat de veer de deur niet meer kan tegenhouden. Hierdoor sluit de schakelaar (slaat de deur dicht) en kan er een stroom I2 in de tweede stroomkring (tweede kamer) van boven naar beneden stromen. Deze kan op haar beurt bijvoorbeeld één van de lampen aanzetten. Wanneer de spanning wegvalt, valt het magnetisch veld van de spoel weg en word de schakelaar (deur) niet meer dichtgetrokken. De veer trekt de schakelaar weer open en de lamp die aanstond wordt weer uitgezet omdat er geen stroom meer vloeit.

Deze relaisbordjes vereisen een voedingsspanning van 5V – die los staat van het schakelcircuit. Deze zal geleverd worden door een externe voeding (die we ook al nodig hebben ter voeding van de ESP32). Ze vereisen ook een kleine schakelstroom, die geleverd kan worden door de ESP32, om het schakelcontact te openen of te sluiten. De relaisbordjes zijn dus aan de ingang aangesloten op een schakelcircuit van 3.3V, samen met de ESP32. Aan de uitgang kan een stroomkring aangelegd worden die een hoger spanningsniveau heeft, tot maximaal 30 Volt. Dit is voldoende om zowel de lampen, de zoemer als de contactor te schakelen. Zij vereisen allen een spanningsniveau van 24 Volt. Op deze manier kan de ESP32 dus zaken aan- of uitzetten die een spanningsniveau nodig hebben dat hoger ligt dan 3.3 Volt.



CONTACT

Harold De Ridder | Student
r0720137@student.thomasmore.be
de.ridder.harold@gmail.com

Tel. + 32 483 11 20 61

VOLG ONS

www.thomasmore.be
fb.com/ThomasMoreBE
#WeAreMore

THOMAS
MORE