**Onderzoek naar een Datalogger voor Validatie van Tijdwaarneming**

**1. Inleiding**

Voor dit onderzoek heb ik mij gericht op het ontwerpen en realiseren van een datalogger die geschikt is voor de validatie van de tijdwaarneming van verkeersregelinstallaties. Als leidraad heb ik onderzoeksvragen geformuleerd tijdens het opstellen van het plan van aanpak.

De eerste onderzoeksvraag was:  
"Wat is een datalogger en hoe werkt datalogging?"

Om deze vraag te beantwoorden, heb ik verschillende bronnen geraadpleegd en mijn eigen ervaring meegenomen in de analyse.

**2. Wat is een datalogger en hoe werkt datalogging?**

Een datalogger is een elektronisch apparaat dat gegevens verzamelt en opslaat over een bepaalde periode, vaak met behulp van ingebouwde of externe sensoren. Deze apparaten zijn doorgaans klein en draagbaar en worden gebruikt voor het monitoren van omgevingscondities zoals temperatuur, luchtvochtigheid, schokken, licht en kanteling.

Door gegevens automatisch en continu te registreren, geven dataloggers een nauwkeurig beeld van de omstandigheden waarin goederen of processen zich bevinden, zoals bij transport of industriële toepassingen.

**3. Onderzoeksvragen**

Om te bepalen welke specificaties de nieuwe datalogger moet hebben, heb ik de volgende onderzoeksvragen opgesteld:

Hoofdonderzoeksvraag:

* Welke hard- en software is het meest efficiënt en effectief voor een datalogger die nauwkeurig de tijdwaarneming kan loggen na een inkomend signaal?

Subvragen:

* Welke hardwarecomponenten zijn geschikt voor het detecteren en verwerken van een inkomend signaal met hoge nauwkeurigheid?
* Welke software-architectuur zorgt voor een efficiënte verwerking en opslag van tijdstempels?
* Wat zijn de prestatie-eisen voor een nauwkeurige tijdsvergelijking?
* Welke opslag- en verwerkingsmogelijkheden zijn nodig voor logging?
* Hoe kan de datalogger worden getest en gevalideerd op nauwkeurigheid?

**4. Vergelijking van bestaande dataloggers bij de politie**

Om een geschikte datalogger te ontwikkelen, heb ik onderzocht welke methodes en apparaten momenteel worden gebruikt binnen het team Forensische Opsporing van de politie Zeeland – West-Brabant. Momenteel zijn er twee systemen in gebruik:

1. Data Translation USB Data Acquisition (DAQ) Module (zie afbeelding 1)
2. Movilog BM22 (zie afbeelding 2)

Afbeelding met tekst, elektronica, Elektronische engineering, Elektrische bedrading

Door AI gegenereerde inhoud is mogelijk onjuist. Afbeelding met elektronica, Elektronisch apparaat, Draagbare spelcomputer, Spelcomputer

Door AI gegenereerde inhoud is mogelijk onjuist.

Afbeelding 1:DAQ module Afbeelding 2:Movilog BM22

De DAQ wordt gebruikt voor validatie van verkeersregelinstallaties, terwijl de BM22 wordt ingezet voor het meten van remvertraging van voertuigen, en specifiek voor het loggen van validatieritten met GPS.

Ik heb de specificaties van de beide dataloggers in een tabel gezet, zie hiervoor tabel A dat als bijlage is opgenomen.

**5. Evaluatie en Ontwikkelpunten van de huidige apparatuur**

Uit eigen ervaring heb ik vastgesteld dat beide apparaten zich in de praktijk hebben bewezen, maar ook ontwikkelpunten hebben.

Movilog BM22 - Sterk concept, maar met verbeterpunten:

* Ongunstige montagelocatie → De plaatsing op de (voor)ruit is onpraktisch, vooral omdat ruiten vaak beschadigd raken bij een aanrijding.
* Niet-oplosbare softwarefout → Een structurele softwarefout kon niet worden hersteld, waardoor een arbeidsintensieve herberekening noodzakelijk is.
* Onhandige voedingsaansluiting → De plus- en min-klemmen moeten direct op de accuklemmen worden bevestigd. Een interne batterij of alternatieve 12V-stekker zou gebruiksvriendelijker zijn.

**6. Eisen datalogger**

Gezien de sterke punten van de Movilog BM22, is dit apparaat in de basis geschikt, mits de aanpassingen van de ontwikkelpunten worden doorgevoerd.

Aangezien het huidige concept grotendeels functioneel en bewezen effectief is, zijn er geen redenen om hiervan af te wijken. Daarnaast ben ik gebonden aan een aantal landelijk vastgestelde normen (FO-norm), die voorschrijven dat validatie moeten worden uitgevoerd met een datalogger die tijdmetingen uitvoert op 10 kHz.

**7. Beantwoording onderzoeksvragen**

Ik heb als eerste de Subvragen beantwoord, want de Subvragen helpen bij de opbouw van het volledige antwoord op de hoofdvraag. Ik heb hiermee structuur aangebracht door van klein naar groot te werken. Eerst details (hardware/software), dan grotere lijnen (prestaties/logging) en uiteindelijk de validatie.

**7.1 Hardware keuze**

**Eerste sub onderzoeksvraag:**

*Welke hardwarecomponenten zijn geschikt voor het detecteren en verwerken van een inkomend signaal met hoge nauwkeurigheid?*

Ik heb de alle benodigde hardware componenten in een tabel gezet, met daarbij de eisen en de huidige hardware die is gebruikt voor de BM22 en de DAQ, zie hiervoor tabel B die als bijlage is toegevoegd.

**Microcontroller**

Voor de selectie van geschikte microcontrollers heb ik eerst onderzocht welke merken en typen veel worden gebruikt in de automotive industrie en waarom juist deze microcontrollers worden gekozen.

In de automotive sector worden microcontrollers van Infineon, NXP, Texas Instruments en Renesas veel toegepast. Deze microcontrollers worden geselecteerd op basis van de volgende criteria: ondersteuning voor RTOS, deterministische reactietijd, ISO 26262 ASIL-B certificering, ondersteuning voor automotive communicatieprotocollen zoals CAN communicatie. Die toepassingen zijn voor mij niet allemaal noodzakelijk, op de ondersteuning voor RTOS na. Hierdoor wijk ik af van de traditionele en verplichte eisen die gesteld worden aan de microcontrollers.

Ik heb aan de microcontroller de volgende voor het project belangrijke eisen gesteld: CPU snelheid en architectuur, geheugen (RAM & FLASH), RTOS, I/O, SPI/I2C/UART, DMA en voldoende ondersteuning van commodity’s en standaard bibliotheken. Om het enigszins een beetje overzichtelijk te houden heb ik de volgende microcontrollers in een tabel gezet en beoordeeld op de eisen:

* Arduino Nano ESP32-S3;
* Teensy 4.1;
* STM32F4;
* NXP KL25Z;
* Rasp Berry PI.

Zie hiervoor bijlage C dat achterin dit document is opgenomen.

**Definitieve keuze: Arduino Nano ESP32-S3**

Uiteindelijk heb ik gekozen tussen twee veelzijdige microcontrollers: de Arduino Nano ESP32-S3 en de Teensy 4.1. De specificaties van beide microcontrollers liggen dicht bij elkaar, maar de doorslaggevende factoren waren mijn bekendheid met het Arduino-ecosysteem en de aanwezigheid van een dual-core processor. Daarom is mijn keuze gevallen op de Arduino Nano ESP32-S3 (zie afbeelding 3).

Hoewel de Teensy 4.1 een krachtige microcontroller is met uitgebreide mogelijkheden, vond ik de beschikbare tijd voor mijn afstudeerproject te beperkt om hiermee aan de slag te gaan. Daarom heb ik gekozen voor de optie waarbij ik sneller productief kan zijn.

De datasheet van de Arduino Nano ESP32-S3 is als bijlage toegevoegd.

Omdat de tijdwaarneming van de datalogger belangrijk is, heb ik uitgezocht wat de verwerkingstijd is door de Arduino Nano ESP32-S3. GPIO Interrupts (zoals een knop indrukken of een sensor-trigger) hebben meestal een latentie van 5-10 µs, afhankelijk van de systeembelasting. Deze vertraging is zo goed als te verwaarlozen.

Afbeelding met elektronica, Elektronisch onderdeel, Stroomkringonderdeel, Passief stroomkringonderdeel

Door AI gegenereerde inhoud is mogelijk onjuist.

Afbeelding 3: De Arduino Nano ESP32.

**Afstandssensor**

Voor de afstandssensor heb ik gezocht naar een vergelijkbaar model als het exemplaar dat momenteel wordt gebruikt in de DAQ-module. Op basis van die functionele eisen heb ik de volgende minimale technische specificaties opgesteld:

* Aansluiting: M12, 4-polig.
* Beschermingsklasse: Minimaal IP67.
* Meetbereik: Minimaal 1 meter.
* Uitgangssignaal: digitaal (PNP of NPN).
* Montage: Moet in een bepaalde hoek geplaatst kunnen worden.
* Kosten: Maximaal 300 euro (om het enigszins betaalbaar te houden)
* Schakelfrequentie minimaal 10 kHz;
* Aanspreektijd minimaal 0,1 ms.

De huidige sensor Leuze PRK 96K/N-1380-46 heeft een schakelfrequentie van 1.000 Hz en een aanspreektijd van 0,5 ms. De DAQ logt met 10 kHz, wat betekent dat er tien keer zo vaak wordt gemeten als de sensor kan schakelen. Dit is overbodig, omdat de sensor niet sneller dan 1.000 metingen per seconde kan verwerken. Hierdoor worden veel metingen opgeslagen zonder dat de sensorwaarde daadwerkelijk verandert. Door mij werd besloten om dit uit te leggen bij de collega’s die zich met deze materie bezig houden. Mij rest verder dat ik een datalogge ga bouwen die in 1 kHz meet, wat min of meer werd gedaan, totdat ik tegenbericht krijg.

De belangrijkste vereiste is dat de sensor betrouwbaar moet reageren op een reflectiepaaltje, ongeacht de lichtomstandigheden – zowel bij fel zonlicht als in volledige duisternis. Om de juiste technologie te selecteren, heb ik radar, rood licht, infrarood en ultrasoon met elkaar vergeleken. Hieruit blijkt dat radar de meest geschikte optie is. Radar is echter te complex en te duur, waardoor de keuze valt op infrarood en/of rood licht technologie als de meest efficiënte en haalbare oplossing.

Rood licht (620-750 nm) is zichtbaar en wordt gebruikt in optische sensoren waar visuele controle nuttig is. Infrarood licht (>750 nm) is onzichtbaar, minder gevoelig voor omgevingslicht en beter voor detectie van donkere of matte oppervlakken. Rood licht is ideaal voor nauwkeurige reflectiedetectie, terwijl infrarood beter werkt in omgevingen met veel zichtbaar licht of warmtebronnen. Praktisch gezien zijn beide opties goed. De afstandssensor die nu wordt gebruikt is een roodlichtsensor en er zijn nooit storingen of problemen geweest. Roodlicht is goedkoper dan infrarood en er is eer keuze als het gaat om sensoren die werken op roodlicht. Hierom heb ik gekozen voor roodlicht.

Ik heb gekozen voor een afstandssensor die werkt op basis van het Time-of-Flight (ToF)-principe, waarbij de sensor berekent hoelang een puls onderweg is van uitzending tot detectie. Hierdoor kan ik de maximale detectieafstand instellen en valse detecties minimaliseren, een belangrijke eis van de stakeholders. Deze sensoren kunnen nauwkeurig worden afgesteld om alleen objecten binnen een bepaalde afstand te detecteren. Voor optimale meetnauwkeurigheid moeten zowel de sensor als het reflectieobject waterpas worden gemonteerd, zodat de laserstraal optimaal wordt teruggekaatst en meetfouten door verkeerde hoeken worden voorkomen.

Het aanbod aan afstandssensoren is zeer uitgebreid, waardoor een gerichte selectie noodzakelijk was om de keuze eenvoudiger en overzichtelijker te maken. Daarom heb ik een selectie van vier sensoren gemaakt en deze beoordeeld aan de hand van de gestelde eisen:

* Leuze PRK 96K/N-1380-46
* Leuze PRK25C.D1/4P-200-M12
* ifm O5D150
* sick wl250-2p2431
* Pepperl+Fuchs ML100-55/103/115
* Omron E3Z-R81
* Banner Q20FFLPQ
* Keyence PZ-G51N

Zie hiervoor bijlage D tabel 1 dat achterin dit document is opgenomen.

Uit deze vergelijking komen de Leuze PRK25C.D1/4P-200-M12 en de SICK wl250-2p2431 als beste keuze naar voren. Deze sensoren voldoen bijna aan alle gestelde eisen en hebben bovendien een stevige, robuuste behuizing, wat bijdraagt aan de betrouwbaarheid en duurzaamheid in industriële toepassingen.

**Keuze voor de sick wl250-2p2431**

De keuze is uiteindelijk gevallen op de sick wl250-2p2431. Omdat deze sensor een zeer hoge schakelfrequentie van 1 kHz en een zeer korte aanspreektijd van 0,5 ms. Dit betekent dat de sensor in maximaal 1 ms kan reageren en een detectiefrequentie tot 1 kHz kan halen.

De sick wl250-2p2431 vereist een voedingsspanning tussen 10 en 30V DC, wat hoger is dan de 3,3V die de Arduino Nano ESP32-S3 kan leveren. Daarom wordt de sensor rechtstreeks gevoed met 12V vanuit de boordspanning van het voertuig.

Daarnaast moet het digitale uitgangssignaal (NPN) worden aangepast om compatibel te zijn met de 3,3V logica van de Arduino. De uitgangsmodus van de sensor (NPN) moet een geschikte spanningsomzetter of pull-down weerstand worden toegepast om correcte signaalverwerking te garanderen.

De datasheet van de sick wl250-2p2431 is als bijlage toegevoegd.



Afbeelding 4: De sick wl250-2p2431.

**Keuze van een externe RTC voor de Arduino Nano ESP32-S3**

De Arduino Nano ESP32-S3 heeft een ingebouwde RTC, maar deze is niet bijzonder nauwkeurig. Daarom is een externe RTC noodzakelijk voor toepassingen waar betrouwbare tijdregistratie vereist is.

Aangezien ik de opleiding Embedded Hardware niet heb gevolgd, is het voor mij geen optie om losse componenten te selecteren en op een PCB te solderen. Daarom heb ik gezocht naar kant-en-klare RTC-modules.

**Belangrijke aspecten van een betrouwbare RTC**

Uit verschillende wetenschappelijke papers blijkt dat de volgende elementen bijdragen aan een nauwkeurige en betrouwbare RTC:

* 32.768 kHz kristaloscillator met TCXO (Temperature Compensated Crystal Oscillator)

- Dit betekent dat de frequentie stabiel blijft, zelfs bij temperatuurschommelingen.

- RTC’s zonder TCXO kunnen enkele seconden tot minuten per maand afwijken.

- Een geïntegreerde oscillator vermindert afwijkingen en jitter ten opzichte van een extern kristal, dat gevoeliger is voor trillingen en temperatuurschommelingen.

* Back-up batterij (V\_BAT)
  + Hiermee blijft de tijd behouden bij stroomuitval, waardoor de RTC altijd correct blijft lopen.

Op basis van deze criteria heb ik een selectie gemaakt van geschikte RTC-modules.

**Gemaakte selectie**

Gezien het brede aanbod aan RTC-modules was een gerichte selectie noodzakelijk. Daarom heb ik vijf RTC’s beoordeeld op basis van de gestelde eisen:

* RV-3028-C7
* AB1805 (Abracon RTC)
* NXP PCF2129AT
* MAX31341
* DS3231 (Open Smart)

Zie hiervoor bijlage D, tabel 2 achterin dit document.

**Gekozen RTC: DS3231**

Voor een probleemloze integratie en betrouwbare prestaties op de Arduino Nano ESP32-S3 zijn de DS3231 en de AB1805 uitstekende keuzes. Uiteindelijk heb ik gekozen voor de DS3231 (zie afbeelding 5), omdat:

* De nauwkeurigheid ±2 PPM is, wat neerkomt op minder dan 1 minuut afwijking per jaar.
* De DS3231 volledig wordt ondersteund door de RTClib, die compatibel is met de ESP32-S3.
* Dit de implementatie eenvoudiger maakt en zorgt voor betrouwbare tijdregistratie.

De datasheet van de DS3231 is opgenomen achterin dit document.

Afbeelding met Stroomkringonderdeel, elektronica, Elektronisch onderdeel, Passief stroomkringonderdeel

Door AI gegenereerde inhoud is mogelijk onjuist.

Afbeelding 5: De RTC DS3231.

**Displaykeuze voor de datalogger**

Voor de datalogger wil ik graag een display waarmee een menustructuur kan worden weergegeven met enkele keuzemogelijkheden. Daarnaast moet de gebruiker via het menu kunnen communiceren met de hardware. Verder is het zinvol om op het display de status van het apparaat weer te geven, zoals of het in bedrijf is en of de meting loopt. Ook moet de gebruiker een indicatie krijgen van het aantal gedetecteerde reflectoren en of de metingen zijn geslaagd en opgeslagen.

**Verschillen tussen TFT- en OLED-schermen**

* TFT (Thin-Film Transistor):
  + Gebruikt in LCD’s om individuele pixels aan te sturen.
  + TFT-LCD's vereisen een backlight, wat resulteert in lager contrast en beperktere kijkhoeken.
* OLED (Organic Light-Emitting Diode):
  + Elke pixel produceert zijn eigen licht, wat zorgt voor hoger contrast en bredere kijkhoeken.
  + OLED-schermen zijn dunner en flexibeler dan TFT-LCD’s.

Op basis van deze eigenschappen heb ik gekozen voor een OLED-scherm, omdat dit meer mogelijkheden biedt en gebruiksvriendelijker is.

**Belangrijke aspecten bij de aanschaf van een OLED-scherm**

**1. Schermgrootte en resolutie**

* Fysieke afmetingen (inch-diagonaal): Voor embedded systemen zijn 0,96” tot 2,42” gangbaar.
* Resolutie (pixels): Hoe hoger de resolutie, hoe scherper het beeld. Typische waarden zijn 128×32, 128×64 of 256×64 pixels.

**2. Interface en compatibiliteit**

* I²C vs. SPI:
  + I²C: Minder draden nodig (SDA, SCL), eenvoudig te integreren.
  + SPI: Sneller dan I²C, beter geschikt voor snelle updates.
* Voedingsspanning:
  + OLED-schermen werken vaak op 3,3V, maar sommige ondersteunen ook 5V.
  + Bij een 5V-microcontroller (zoals de Arduino Uno) is mogelijk een level shifter nodig.

**3. Besturingschip (Controller)**

OLED-schermen hebben een ingebouwde controllerchip die bepaalt hoe het scherm wordt aangestuurd. Veelgebruikte controllers zijn:

* SSD1306: Veel ondersteuning in Arduino en ESP32-projecten (I²C/SPI).
* SH1106: Vergelijkbaar met SSD1306, maar kan compatibiliteitsproblemen geven.
* SSD1322/SSD1327: Ondersteunt hogere resoluties en grijstinten.

**4. Beeldkwaliteit en kleur**

* Monochroom vs. kleuren-OLED:
  + Monochrome (wit, blauw, geel) zijn energiezuiniger.
  + Kleuren-OLED’s (RGB) bieden betere visuals, maar verbruiken meer stroom.
* Contrast en helderheid:
  + OLED heeft oneindig contrast (zwarte pixels zijn echt uit).
  + Sommige OLED’s ondersteunen grijstinten voor meer detail.

**5. Energieverbruik**

* OLED’s verbruiken minder stroom dan TFT-schermen, maar het verbruik hangt af van het beeld:
  + Veel witte pixels → hoger verbruik.
  + Meer zwart → energiezuiniger.
* Voor batterijgevoede projecten is laag stroomverbruik belangrijk.

**6. Software-ondersteuning**

* Arduino, ESP32 en Raspberry Pi hebben uitstekende bibliotheekondersteuning voor OLED’s.
* Controleer of drivers compatibel zijn met de gebruikte controllerchip.

**Vergelijking van OLED-schermen**

Ik heb drie OLED-displays met elkaar vergeleken:

* ZHONGJINGYUAN 2,42" OLED
* Waveshare 2,42" OLED
* Winstar 2,42" OLED

Zie hiervoor bijlage D, tabel 3 achterin dit document.

Omdat de schermen qua specificaties niet veel van elkaar verschillen en het display voornamelijk functioneel moet zijn, heb ik gekozen voor de ZHONGJINGYUAN 2,42" OLED. Dit scherm biedt voldoende ruimte en helderheid om alle benodigde informatie duidelijk weer te geven.

De datasheet van de DS3231 RTC is opgenomen achterin dit document.

**Afbeelding met tekst, nummer, Rechthoek, klok

Door AI gegenereerde inhoud is mogelijk onjuist.**

Afbeelding 6: De ZHONGJINGYUAN 2,42" OLED display White.

**geheugenmodule**

**​**Voor het opslaan van tijdstempels op de Arduino Nano ESP32-S3 is het selecteren van een geschikte (micro)SD-kaartlezer cruciaal. Hieronder is een tabel met enkele aanbevolen modules, gevolgd door de criteria..

Aanbevolen microSD-kaartlezer modules:

| Module | Interface | Spanningsniveau | Level Shifter | Prijs | Opmerking |
| --- | --- | --- | --- | --- | --- |
| DIYables Micro SD Card Adapter | SPI | 3.3V/5V | Ja | $6.99 voor 5 stuks | Geschikt voor Arduino, ESP32, Raspberry Pi |
| ANMBEST Micro SD TF Card Adapter | SPI | 3.3V/5V | Ja | $8.99 voor 10 stuks | Inclusief level conversie chip |
| Wishiot SD Card Module | SPI | 3.3V/5V | Ja | $7.98 voor 5 stuks | Inclusief Dupont kabels |

Belangrijke criteria bij het selecteren van een microSD-kaartlezer module:

1. **Interface (Communicatieprotocol):**
   * De meeste microSD-kaartmodules communiceren via de SPI-interface, wat compatibel is met de Arduino Nano ESP32-S3.​
2. **Spanningsniveau:**
   * Controleer of de module werkt op 3.3V, 5V of beide. Aangezien de ESP32-S3 op 3.3V werkt, is compatibiliteit hiermee essentieel.​
3. **Level Shifter (Spanningsomzetter):**
   * Een ingebouwde level shifter zorgt voor compatibiliteit tussen de spanningsniveaus van de microcontroller en de (micro)SD-kaart. Dit voorkomt mogelijke schade aan de kaart of de controller.​
4. **Kwaliteit en Betrouwbaarheid:**
   * Kies voor modules met positieve gebruikersrecensies en van gerenommeerde fabrikanten om betrouwbaarheid te waarborgen.​
5. **Beschikbaarheid van Documentatie en Ondersteuning:**
   * Modules met goede documentatie en community-ondersteuning kunnen het integratieproces aanzienlijk vereenvoudigen.​

**Conclusie:** Bij het selecteren van een (micro)SD-kaartlezer module voor je Arduino Nano ESP32-S3 is het essentieel om te letten op compatibiliteit met de SPI-interface, het juiste spanningsniveau en de aanwezigheid van een level shifter. De hierboven genoemde modules voldoen allemaal aan deze criteria en zijn allemaal geschikt.

Door mij werd een van deze modules gekozen, zie afbeelding 7.

**Afbeelding met elektronica, Elektronische engineering, Stroomkringonderdeel, Elektronisch onderdeel

Door AI gegenereerde inhoud is mogelijk onjuist.**

Afbeelding 7: SD Kaart Adapter Module.

**GPS-module.**

**Belangrijke criteria bij het selecteren van een GPS-module.**

Voor een zeer accurate tijdwaarneming en weergave van de actuele rijsnelheid, zijn de volgende eigenschappen van een GPS-module belangrijk:

**1. GNSS-ondersteuning (GPS, GLONASS, Galileo, BeiDou)**

* GPS (VS), GLONASS (Rusland), Galileo (EU), BeiDou (China) en QZSS (Japan) zijn de belangrijkste satellietsystemen.
* Multiconstellatie-ontvangers (GPS + Galileo + GLONASS + BeiDou) hebben een snellere fix en hogere nauwkeurigheid, vooral in stedelijke gebieden.

Aanbevolen: Een module die minimaal GPS en Galileo ondersteunt voor hoge nauwkeurigheid.

**2. Nauwkeurigheid van tijdwaarneming**

Voor zeer accurate tijdwaarneming:

* De PPS (Pulse Per Second) output geeft een zeer precieze tijdreferentie met een nauwkeurigheid tot nanoseconden.
* Sommige geavanceerde modules ondersteunen TCXO (Temperature Compensated Crystal Oscillator) voor betere stabiliteit.

Aanbevolen: Een GPS-module met PPS-output en eventueel TCXO voor minimale drift.

**3. Updatefrequentie (Refresh Rate)**

* De standaard GPS-updatefrequentie is 1 Hz (1 meting per seconde), wat geschikt is voor veel toepassingen.
* Hogere updatefrequenties (5 Hz, 10 Hz, 25 Hz) verbeteren de weergave van actuele rijsnelheid, vooral bij snelle bewegingen (bv. in een voertuig).

Aanbevolen: Minimaal 5 Hz updatefrequentie voor realtime snelheid.

**4. Gevoeligheid en Fix-tijd**

* Gevoeligheid bepaalt hoe goed de module satellietsignalen kan ontvangen, zelfs bij slecht weer of in stedelijke gebieden.
* Expressed in dBm, bijvoorbeeld:
  + -165 dBm (zeer goed, zelfs binnenshuis).
  + -145 dBm (basisniveau, goed voor buitengebruik).
* TTFF (Time To First Fix) bepaalt hoe snel de module een eerste fix verkrijgt bij inschakeling.

Aanbevolen: -160 dBm of beter + snelle TTFF (<10 sec cold start, <1 sec warm start).

**5. Interface (Communicatie met de Arduino Nano ESP32-S3)**

De Arduino Nano ESP32-S3 ondersteunt de volgende interfaces:

* UART (Seriële communicatie, TX/RX): Meest gebruikelijk en compatibel met de meeste GPS-modules.
* I²C: Minder gebruikelijk voor GPS, maar sommige modules ondersteunen het.
* SPI: Sneller, maar minder frequent gebruikt bij GPS.
* USB: Meestal voor PC-gebruik, niet relevant voor ESP32-S3.

Aanbevolen: UART (TX/RX) GPS-module voor eenvoudige integratie.

**6. Extra functionaliteiten**

RTK (Real-Time Kinematics) voor cm-nauwkeurigheid?

* Standaard GPS heeft een nauwkeurigheid van 1-3 meter.
* RTK-GPS (zoals u-blox ZED-F9P) kan nauwkeurigheid van ~1 cm bereiken, maar vereist een extra referentiestation.

Datalogging

* Sommige modules hebben een interne opslag om GPS-data direct op te slaan zonder SD-kaart.

Aanbevolen: RTK alleen als extreme nauwkeurigheid vereist is.

Door mij werden vijf verschillende GPS-ontvangers geselecteerd en in een tabel gezet. Zie bijlage D tabel 4. Van deze werd de: ”Seeed Studio Grove GNSS-GPS Module - Air530”, gekozen voor de datalogger. Omdat deze sensor relatief gezien het goedkoopst is en de meeste nauwkeurigheid geeft.

Afbeelding met Elektronisch onderdeel, Stroomkringonderdeel, Passief stroomkringonderdeel, Elektronische engineering

Door AI gegenereerde inhoud is mogelijk onjuist.

Afbeelding 8:

**7.2 Softwarearchitectuur**

**Tweede subonderzoeksvraag:**

*Welke software-architectuur zorgt voor een efficiënte verwerking en opslag van tijdstempels?*

Om een efficiënte software-architectuur te ontwerpen voor het verwerken en opslaan van tijdstempels in een datalogger, moeten we rekening houden met:

* Real-time verwerking van binnenkomende data
* Betrouwbare opslag van tijdstempels
* Energiezuinigheid (indien op een embedded systeem)
* Data-integriteit bij herstarts of stroomuitval
* Efficiënte opslag en uitlezing

Hieronder beschrijf ik een optimale software-architectuur die voldoet aan deze eisen.

**1. Algemene Architectuurindeling**

De software-architectuur kan worden opgesplitst in vier lagen:

1. Sensorinterface – Verwerking van binnenkomende data van detectieapparatuur (bijv. afstandssensor, GPS, RTC).
2. Tijdstempelbeheer – Correcte tijdregistratie en berekening van tussenliggende tijden.
3. Buffer- en opslagbeheer – Tijdelijke en permanente opslag van data.
4. Uitvoer en uitlezing – Weergave op een scherm of overdracht naar een pc/server.

**2. Software-Architectuur in lagen**

Sensorinterface

Doel: Data betrouwbaar ontvangen van de sensoren en tijdig verwerken.

* Interrupt-driven verwerking: Detectiepaal-data en GPS-data komen binnen via interrupt-handlers.
* FIFO-buffering: Binnenkomende metingen worden eerst in een First In, First Out (FIFO) buffer geplaatst.
* Tijdstempel toewijzing: Zodra een voertuig wordt gedetecteerd, wordt een timestamp uit de RTC opgehaald.

Technieken:

* Interrupt Service Routines (ISR) voor snelle verwerking van detecties
* FIFO buffer om snelle pieken in metingen op te vangen zonder verlies

**Tijdstempelbeheer**

Doel: Correcte en nauwkeurige tijdstempels genereren en verwerken.

* Real-Time Clock (RTC) als tijdsbron (bijv. DS3231 of ingebouwde RTC in de ESP32).
* GPS als alternatieve tijdssynchronisatie, indien vereist.
* Berekening van tijd tussen detecties:
  + Zodra een voertuig een detectiepaal passeert, wordt de vorige timestamp opgezocht in de buffer.
  + De tijdsduur tussen detectiepunten wordt berekend en opgeslagen.

Technieken:

* RTC-bibliotheek voor nauwkeurige tijdstempels
* Tijdsdifferentiatie om de exacte reistijd te berekenen
* Failsafe-mechanisme: Als de RTC uitvalt, schakelt het systeem over naar GPS-tijd

**Buffer- en opslagbeheer**

Doel: Betrouwbare opslag van tijdstempels en tussenliggende tijdsintervallen, met bescherming tegen stroomuitval.

* Dubbele buffering:
  + Tijdstempels worden eerst tijdelijk in RAM opgeslagen (kleine buffer).
  + Na een bepaalde batchgrootte (bijv. 5 detecties) worden ze permanent naar de SD-kaart geschreven.
* Periodieke opslag per rit:
  + Elke rit wordt in een apart bestand opgeslagen (rit\_001.csv, rit\_002.csv).
* Fail-safe mechanisme:
  + Flush()-functie na elke batch om dataverlies te voorkomen bij stroomuitval.
  + FRAM als optionele extra buffer, zodat gegevens bewaard blijven bij een crash.

Technieken:

* Ringbuffer voor RAM-opslag (geheugenefficiënt en snel)
* SD-kaart opslag met CSV-formaat (compatibel en makkelijk uit te lezen)
* FRAM als failsafe buffer (optioneel)

**Uitvoer en uitlezing**

Doel: Realtime monitoring en gemakkelijke data-uitlezing.

* OLED-display voor live weergave van tijdstempels.
* Seriële uitvoer naar PC via USB voor live logging/debugging.
* Bestandsformaat keuze:
  + CSV-formaat voor compatibiliteit met Excel en Python-analyse.
  + JSON als alternatief voor draadloze transmissie naar een server.

Technieken:

* OLED-update via timers (geen blocking delay())
* CSV-bestanden schrijven via SD-kaart bibliotheek
* Optioneel: WiFi/Bluetooth-integratie voor live monitoring

**3. Software-Flowchart**

Hier is een overzicht van hoe de software werkt:

1. **Start**  
   ⮑ RTC controleren en synchroniseren  
   ⮑ SD-kaart checken en openen  
   ⮑ FIFO-buffer initialiseren
2. **Data-acquisitie**  
   ⮑ Wachten op interrupt van detectiesensor  
   ⮑ Tijdstempel ophalen van RTC  
   ⮑ Data opslaan in RAM-buffer
3. **Gegevensverwerking**  
   ⮑ Tijdsverschil berekenen tussen detecties  
   ⮑ Als buffer vol is → batch opslaan naar SD-kaart
4. **Data-opslag**  
   ⮑ Schrijven naar CSV-bestand (rit\_XXX.csv)  
   ⮑ Flush() oproepen om SD-kaart te updaten
5. **Live monitoring (optioneel)**  
   ⮑ OLED-display updaten met laatste detectie  
   ⮑ Seriële uitvoer naar PC
6. **Einde rit**  
   ⮑ Data opslaan en afsluiten  
   ⮑ Systeem resetten voor nieuwe rit

**4. Efficiëntie-overwegingen**

Om de architectuur zo efficiënt mogelijk te maken:

1. Gebruik DMA (Direct Memory Access) als de hardware dit ondersteunt → Vermindert CPU-belasting bij SD-kaart opslag.
2. Gebruik interrupt-gebaseerde detectie in plaats van polling → Bespaart CPU-tijd en energie.
3. Optimaliseer SD-kaartgebruik:
   * Gebruik grotere sector-schrijvingen om flash-slijtage te minimaliseren.
   * Schrijf data in batches i.p.v. elke meting afzonderlijk.

**5. Conclusie & Aanbevolen Software-Architectuur**

Voor een efficiënte verwerking en opslag van tijdstempels in een datalogger is de beste software-architectuur:

* Interrupt-gebaseerde sensorinterface voor snelle detectie
* RTC-gestuurde tijdstempels met (eventuele) backup via GPS
* FIFO-buffering in RAM voor tijdelijke opslag
* Batch-opslag naar SD-kaart om levensduur te verlengen
* CSV-formaat voor makkelijke uitlezing en analyse
* Failsafe-mechanismen zoals flush() en FRAM-buffering

Met deze architectuur krijg je:

* Nauwkeurige en efficiënte tijdstempellogging
* Minimale SD-kaartslijtage door batch-opslaan
* Data-integriteit bij stroomuitval
* Snelle verwerking en uitlezing via OLED of PC

Voor een embedded datalogger zijn deze de meest geschikte programmeertalen:

**1. Programmeertalen en Embedded Frameworks voor Microcontrollers**

| **Taal / Framework** | **Voordelen** | **Nadelen** | **Geschikt?** |
| --- | --- | --- | --- |
| **C++ (Arduino / ESP-IDF)** | Directe hardware-aansturing, efficiënt geheugenbeheer, brede ondersteuning voor microcontrollers | Minder abstract, vereist meer kennis | **Ja, beste keuze met name voor ESP32/Arduino** |
| **C (Bare-metal of HAL)** | Laagste overhead, volledige controle, breed ondersteund | Complex geheugenbeheer, handmatige optimalisatie nodig | **Ja, voor maximale efficiëntie** |
| **MicroPython** | Eenvoudig en snel te ontwikkelen, ideaal voor beginners | Langzamer dan C++, hogere geheugenbelasting | **Kan, maar minder efficiënt** |
| **CircuitPython (Adafruit)** | Gebruiksvriendelijker dan MicroPython, brede bibliotheekondersteuning | Minder geschikt voor real-time toepassingen | **Kan, maar niet optimaal voor snelle logging** |
| **Rust (voor ESP32, STM32, RISC-V, etc.)** | Zeer veilig, geheugenveiligheid zonder garbage collector, efficiënt | Minder ondersteuning, complexer dan C++ | **Te experimenteel** |
| **Go (TinyGo)** | Compact, efficiënt, goede ondersteuning voor RISC-V en ARM | Weinig ingebouwde bibliotheken voor microcontrollers | **Niet aanbevolen** |
| **Embedded Java (LeJOS, JVM op ARM/ESP32)** | Objectgeoriënteerd, Java-ecosysteem | Trager, hogere geheugenvereisten | **Te zwaar voor embedded toepassingen** |
| **Embedded C# (.NET nanoFramework)** | Gebruiksvriendelijk, moderne programmeerstijl | Beperkte ondersteuning, veel overhead | **Niet efficiënt voor microcontrollers** |
| **Forth (Mecrisp, eForth, etc.)** | Zeer compact, direct interactief testen mogelijk | Moeilijk te leren, beperkte community | **Te niche voor moderne projecten** |
| **Lua (eLua, NodeMCU)** | Compact, scripting mogelijk op ESP32 en STM32 | Langzamer dan C++, niet real-time geschikt | **Niet efficiënt genoeg** |
| **Assembly (ARM, AVR, RISC-V, etc.)** | Maximale snelheid en controle | Zeer complex, niet schaalbaar | **Niet praktisch voor grote projecten** |

Aanbevolen keuze: C/C++ met Arduino IDE, PlatformIO of ESP-IDF

* Arduino IDE is makkelijk voor snelle prototyping.
* PlatformIO is krachtiger en ondersteunt meer features.
* ESP-IDF is specifiek voor de ESP32 en biedt maximale controle en efficiëntie.

**Conclusie:**

Een efficiënte software-architectuur voor de verwerking en opslag van tijdstempels in een embedded datalogger bestaat uit vier lagen: sensorinterface, tijdstempelbeheer, opslagbeheer en data-uitlezing. De sensorinterface maakt gebruik van interrupt-gebaseerde detectie (UART/I²C) voor snelle en energiezuinige verwerking van binnenkomende data. In de tijdstempellaag worden RTC (bijv. DS3231) of GPS-data gebruikt om nauwkeurige tijdsregistraties te genereren en berekeningen uit te voeren met millis() of difftime(). De opslaglaag combineert een ringbuffer in RAM voor tijdelijke opslag en een SD-kaart met CSV-bestanden (via SdFat.h) voor permanente opslag, waarbij optioneel FRAM als failsafe buffer voorkomt dat gegevens verloren gaan bij stroomuitval. Tot slot verzorgt de uitleeslaag een OLED-scherm (Adafruit\_SSD1306) voor real-time monitoring, seriële uitvoer via UART en optioneel WiFi/Bluetooth voor draadloze data-overdracht. Door gebruik te maken van C++ met Arduino IDE of PlatformIO, en indien nodig FreeRTOS voor multitasking, biedt deze architectuur een schaalbare, robuuste en energie-efficiënte oplossing voor betrouwbare tijdstempellogging.

**7.3 Prestatie-eisen**

**Derde subonderzoeksvraag:**

*Wat zijn de prestatie-eisen voor een nauwkeurige tijdsvergelijking?*

Voor een nauwkeurige tijdsvergelijking in een embedded datalogger zijn verschillende prestatie-eisen van belang, afhankelijk van de vereiste precisie en de gebruikte hardware. Hieronder bespreek ik de belangrijkste factoren en eisen.

**1. Resolutie en Nauwkeurigheid van de Tijdbron**

Om betrouwbare tijdstempels te genereren en te vergelijken, moet de tijdsbron nauwkeurig en stabiel zijn.

| **Tijdsbron** | **Resolutie** | **Nauwkeurigheid** | **Geschikt voor?** |
| --- | --- | --- | --- |
| **Interne millis() / micros() (ESP32, Arduino)** | 1 ms / 1 µs | ± 0.1% (afhankelijk van kristal) | Redelijke precisie voor korte metingen |
| **RTC-module (DS3231, PCF8563)** | 1 s / 1 ms | ± 2 ppm (± 0.17 s per dag) | Goede stabiliteit voor lange termijn |
| **GPS PPS (Pulse Per Second)** | 1 µs | ± 30 ns (zeer nauwkeurig) | Hoge precisie en synchronisatie |

Aanbevolen keuze:  
Voor milliseconde-nauwkeurigheid is een RTC zoals de DS3231 geschikt, maar als tijdsvergelijkingen met hoge precisie nodig is, dan moet er gebruik gemaakt worden van GPS PPS.

**2. Tijdregistratie en Vergelijkingsalgoritme**

* De microcontroller moet tijdstempels zonder jitter of vertraging kunnen vastleggen.
* Gebruik interrupts in plaats van polling om vertragingen te minimaliseren.
* Tijdstempelopslag in 64-bit integer (uint64\_t) voorkomt overflow bij langdurige metingen.
* Referentie-tijdstempels synchroniseren (bijv. RTC bijstellen met GPS).

**Aanbevolen aanpak:**

1. Bij elke detectie een interrupt genereren.
2. Timestamp opslaan met micros() of een RTC.
3. Vergelijking uitvoeren met een 64-bit integer om overflows te voorkomen.
4. Gebruik een filter (bijv. moving average) als er variatie in meetwaarden is.

**3. Opslageisen en Gegevensverwerking**

* Opslag van tijdstempels in een SD-kaart of RAM-buffer moet efficiënt gebeuren.
* Tijdsvergelijking moet met lage CPU-belasting worden uitgevoerd.
* Voor lange termijn logging (RTC): registreer tijd als YYYY-MM-DD HH:MM:SS.mmm.
* Voor korte meetintervallen (micros() of GPS PPS): registreer tijd als uint64\_t microseconden sinds start.

**Optimalisatie-tips:**

* Ringbuffer in RAM gebruiken om recente tijdstempels snel te vergelijken.
* Batch-schrijfoperaties naar de SD-kaart om de prestaties te verbeteren.
* Bijlage van CRC-checksums voor data-integriteit.

1. **Maximale Verwerkingsvertraging**

| **Stap** | **Max. Toegestane Vertraging** | **Optimalisatie** |
| --- | --- | --- |
| **Interrupt detectie** | < 10 µs | Gebruik hardware interrupts |
| **Timestamp vastleggen** | < 50 µs | Gebruik micros() of hardware timers |
| **Tijdsverschil berekenen** | < 500 µs | Gebruik integer wiskunde (geen floating point) |
| **Schrijven naar SD-kaart** | < 10 ms | Gebruik een buffer en batch-schrijven |

Voor kritische toepassingen:

* Directe SD-kaart schrijfacties vermijden (gebruik een buffer).
* Hardware-timers in de ESP32 of STM32 gebruiken voor µs-nauwkeurigheid.

**Conclusie: Eisen voor een nauwkeurige tijdsvergelijking**

1. Gebruik een RTC (DS3231) voor milliseconde-logging en GPS PPS voor microseconde-nauwkeurigheid.
2. Gebruik hardware interrupts voor directe tijdstempelregistratie zonder vertraging.
3. Sla tijdstempels op als 64-bit integer (microseconden) of in CSV-formaat (milliseconden).
4. Minimaliseer opslagvertragingen door een buffer en batch-schrijfoperaties te gebruiken.
5. Beperk verwerkingstijd door integer wiskunde en hardware timers te gebruiken.

**7.4 Opslag en verwerking**

**Vierde subonderzoeksvraag:**

*Welke opslag- en verwerkingsmogelijkheden zijn nodig voor logging?*

Om deze vraag te beantwoorden moet ik eerst bepalen met welke factoren er rekening gehouden dient te worden:

**Dataformaat: Hoeveel bytes per logregel?**

Er worden logregels geschreven die de volgende parameters bevatten.

* Detectienummer (bijv. 4 cijfers) → 4 bytes
* Tijdstempel (dd/mm/yyyy hh:mm:ss) → 19 bytes
* Tijd tussen detecties (ss/ssss, bijv. 12/3456) → 7 bytes
* Scheidingssymbolen (komma's, spaties, newline, etc.) → 5 à 6 bytes
* Totaal per logregel: ~35 tot 40 bytes

Ik verwacht tussen de 60 en 100 logregels per validatierit.  
Totaal opslaggebruik: 4 kB maximaal per validatierit.

Conclusie: Niet extreem veel dataopslag nodig.

**Schrijfsnelheid: Hoe vaak per seconde log je?**

Hooguit 2 keer per seconde, maar alleen bij detectie van een detectiepunt.

Schrijfsnelheid: Dit is laag genoeg voor een SD-kaart of FRAM zonder prestatieproblemen.  
Conclusie: iedere SD-kaart of FRAM kan dit aan.

**Gegevensopslagduur: Hoe lang moet de data bewaard blijven?**

Tot na de validatie, dus een paar weken tot maanden.

Het totaal blijft onder de 4 kB per project, wat makkelijk op een SD-kaart, EEPROM of FRAM past.  
Conclusie: Opslagcapaciteit is geen beperking.

**Betrouwbaarheid: Moet het bestand bestand zijn tegen herstarts of stroomuitval?**

Ik wilt voorkomen dat alle data verloren gaat als er tijdens een rit iets misgaat.

Opties:

1. Na elke detectie de logregel direct naar de SD-kaart schrijven.
   * Dit voorkomt dat data in RAM verloren gaat bij een crash.
2. Gebruik een tijdelijke buffer (FRAM of EEPROM)
   * Eerst naar FRAM (bijv. 8 kB I²C of SPI FRAM-chip) loggen en aan het einde van een rit overzetten naar de SD-kaart.
   * Voorkomt dat de SD-kaart telkens wordt beschreven, dit verlengt de levensduur.
3. Elke rit in een apart bestand opslaan
   * Mocht er iets misgaan, dan verlies je hooguit de data van één rit.

**Conclusie beste aanpak:**

* Optie 1: Direct wegschrijven naar SD-kaart met flush().
* Optie 3: Elke rit in een apart bestand.

**Levensduur opslagmedium: Hoe vaak kan het medium overschreven worden?**

Dit hangt af van het medium:

| **Opslagmethode** | **Schrijflimiet (per sector)** | **Geschikte optie?** |
| --- | --- | --- |
| **SD-kaart (FAT32)** | ~100.000 schrijfbewerkingen per sector | Ja, als je slim schrijft (buffers & niet constant overschrijven) |
| **FRAM (I²C/SPI)** | Onbeperkte schrijfcycli | Ideaal voor bufferdata |
| **EEPROM (intern of extern)** | ~1.000.000 schrijfbewerkingen | Kan, maar onnodig |
| **Flash (ESP32-S3 NVS of LittleFS)** | 10.000 - 100.000 schrijfbewerkingen | Niet aanbevolen, slijtagegevoelig |

Ongeveer 60-100 regels per project, dus zelfs een SD-kaart zou pas na miljoenen ritten slijten.

Conclusie:

* Een SD-kaart is prima, zolang je niet constant herschrijft.
* FRAM als tussenbuffer is een mooie extra, maar niet per se nodig als je een SD-kaart gebruikt met flush().

**7.5 Validatie en testplan**

**Vijfde subonderzoeksvraag:**

*Hoe kan de datalogger worden getest en gevalideerd op nauwkeurigheid?*

Om dit proces eenvoudig, maar effectief te houden, kan de volgende aanpak worden gehanteerd:

**1. Testopstelling en Referentiepunten**

Om een betrouwbare vergelijking te maken, heb je een duidelijk referentiepunt nodig. Hier zijn enkele opties:

**Optie 1:** GPS als referentiepunt (eenvoudig en praktisch)

* Gebruik een GPS-datalogger als externe referentie.
* GPS geeft een nauwkeurig UTC-tijdsignaal dat je kunt vergelijken met je datalogger.
* Vergelijk de tijdstempels van detectiemomenten tussen je logger en de GPS-datalogger.

**Optie 2:** Extra detectiesensor op de auto

* Monteer een industriële lichtsensor (Korrevit) of radar die een exacte detectie van objecten registreert.
* Dit kan worden gebruikt als referentie om te bepalen wanneer een voertuig een detectiepaal passeert.

**Optie 3:** Externe meetapparatuur (hogere precisie, maar complexer)

* Gebruik een hogesnelheidscamera met een tijdstempel-overlay.
* Hiermee kun je de exacte passage van een voertuig bij een detectiepunt handmatig valideren.

**Aanbevolen keuze:** **Optie 1 (GPS-logger) + eigen datalogger** voor een eenvoudige, robuuste test.

**Testprocedure voor Validatie**

Om consistentie en betrouwbaarheid te waarborgen, deze stappen:

1. **Setup van de testauto**
   * Monteer twee bestaande dataloggers + eign datalogger op een stabiele positie.
   * Zorg ervoor dat alle apparaten op hetzelfde voedingspunt zitten om timingafwijkingen te minimaliseren.
   * Synchroniseer vooraf de klokken (RTC of GPS-tijd).
2. **Uitvoeren van referentieritten**
   * Rij een vaste route met meerdere detectiepunten.
   * Log de exacte tijdstempels van elke detectie in CSV-formaat.
   * Herhaal de ritten minimaal 10 keer om meetfouten te detecteren.
3. **Data-analyse en vergelijking**
   * Exporteer de logbestanden van alle dataloggers.
   * Gebruik Excel of Python (Pandas, Matplotlib) om verschillen in tijdstempels te visualiseren.
   * Bereken de gemiddelde afwijking (in ms of µs) en bepaal de maximale foutmarge.

**Validatiecriteria**

De tijdwaarneming van de datalogger wordt als betrouwbaar beschouwd als deze binnen een acceptabele foutmarge blijft ten opzichte van de referentiedataloggers. De afwijking wordt vastgesteld door een statistische vergelijking van de tijdstempels, waarbij consistentie tussen meerdere meetmomenten en ritten wordt geanalyseerd. Op basis van de technische specificaties van de tijdbron (RTC of GPS) wordt de maximale toelaatbare afwijking bepaald. Indien de afwijking binnen deze marges valt, wordt de datalogger als nauwkeurig gevalideerd.

# Verwijzingen

*https://www.logmore.com/post/what-is-data-logging*. (sd). Opgehaald van https://www.logmore.com/: https://www.logmore.com/post/what-is-data-logging#:~:text=Data%20logging%20is%20the%20process,regardless%20of%20the%20method%20used.