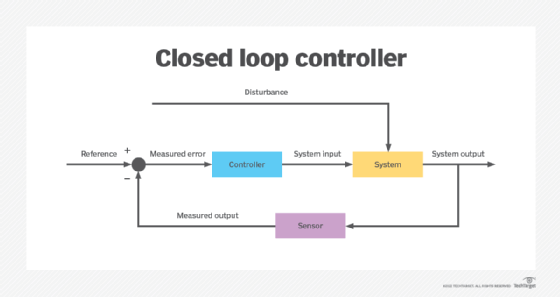
Closed Loop

Tiemon Steeghs



Figuur (Wright, 2022)

Inhoud

[1. Aanleiding 3](#_Toc137589958)

[2. Hoofdvraag: Hoe zet ik een closed loop multithreading systeem met abstractie laag op voor een autonome robotauto, waarbij gebruik wordt gemaakt van een ultrasonische sensor? 4](#_Toc137589959)

[2.1. Deelvraag 1: Hoe werkt de HC-SR04P? 4](#_Toc137589960)

[2.2. Deelvraag 2: Hoe pas ik de HC-SR04P sensor toe op het stm32 nucleo bord? 5](#_Toc137589961)

[2.2.1. Techniek 1 5](#_Toc137589962)

[2.2.2. Techniek 2 7](#_Toc137589963)

[2.3. Deelvraag 3 Wat is een closed loop systeem en hoe zet ik hem op met een PID controller? 8](#_Toc137589964)

[2.3.1. De PID 9](#_Toc137589965)

[2.3.2. PID keuze 11](#_Toc137589966)

[2.3.3. PID Berekenen 12](#_Toc137589967)

[2.3.4. PID metingen 14](#_Toc137589968)

[2.4. Deelvraag 4: Hoe zet ik een goede abstractie laag op voor mijn closed loop systeem, die veel functionaliteit bied en de inhoud van mijn programma overzichtelijker maakt? 15](#_Toc137589969)

[2.4.1. Klassendiagram 16](#_Toc137589970)

[2.5. Deelvraag 5: Hoe pas je multithreading toe met synchronisatie mechanismen aan de hand van de FreeRTOS library? 18](#_Toc137589971)

[2.5.1. Onderzoek synchronisatie mechanisme 18](#_Toc137589972)

[2.5.2. Keuze synchronisatie mechanisme 20](#_Toc137589973)

[3. Conclusie 21](#_Toc137589974)

[4. Bronvermelding 22](#_Toc137589975)

# Aanleiding

Ik maak deze challenge om te leren wat closed loops zijn en hoe ik de ultrasonische sensor moet toepassen. Dit ga ik doen aan de hand van verschillende requirements waaraan de opdracht moet voldoen. Deze challenge bestaat uit verschillende delen die er allemaal in verwerkt worden.

# Hoofdvraag: Hoe zet ik een closed loop multithreading systeem met abstractie laag op voor een autonome robotauto, waarbij gebruik wordt gemaakt van een ultrasonische sensor?

## Deelvraag 1: Hoe werkt de HC-SR04P?

De HC-SR04P is een ultrasone afstandssensor die vaak wordt gebruikt in toepassingen waarbij de afstand tussen objecten moet worden gemeten.

De sensor maakt gebruik van ultrasone geluidsgolven om de afstand tot een object te meten. Het werkt door het verzenden van een ultrasone geluidspuls vanuit de zender van de sensor naar het object dat gemeten moet worden. De puls kaatst terug van het object en wordt opgevangen door de ontvanger van de sensor.

De tijd tussen het verzenden van de puls en het ontvangen van de echo wordt gemeten en omgezet in een afstandsmeting. Dit gebeurt door het vermenigvuldigen van de tijd met de snelheid van het geluid.

De HC-SR04P is een verbeterde versie van de HC-SR04 sensor. Het verschil is dat hij ook op een voltage van 5 Volt werkt in tegenstelling tot het origineel die alleen werkt op 3 volt.

De sensor beschikt over vier pins.

1. Vcc, voor de 5 volt input
2. Trig, de trigger pulse input pin. Op het moment dat deze wordt geactiveerd zal hij een ultrasone pulse uitzenden om de afstand tot een object te kunnen berkeken. Een puls duurt ongeveer 10 microseconden.
3. Echo, de echo puls output pin. Deze pin vangt de ultrasone pulsen op en krijgt ook de tijdsduur binnen die overeenkomt met de tijd die het koste om de puls te versturen en weer terug te laten keren.
4. GND, de grond pin

Afbeelding met diagram

Automatisch gegenereerde beschrijving

Figuur Bron: Datasheet

De afstand tot het object kan worden berekend door de tijd die de ultrasone puls nodig had om te reizen naar het object en terug te keren te delen door twee, omdat de ultrasone puls een afstand aflegt die gelijk is aan de afstand naar het object plus de afstand terug naar de sensor. Door deze berekening te maken kan de afstand tot het object worden bepaald met behulp van de formule:

Afstand = (Tijd x Snelheid van het geluid in de lucht) / 2

Snelheid van het geluid in de lucht = 343 m/s

## Deelvraag 2: Hoe pas ik de HC-SR04P sensor toe op het stm32 nucleo bord?

Om de HC-SR04P sensor juist toe te passen zijn er verschillende stappen die je moet ondergaan. Zoals eerder beschreven maakt de sensor gebruik van een trigger pin en een echo pin. Om de afstand tot een object te kunnen meten, heb je de tijd nodig die het duurt voor het versturen van het signaal, tot de binnenkomst bij de echo pin.

### Techniek 1

Een manier om dit te doen is door als eerste timer 3 in te stellen voor het genereren van een pwm signaal op de trigger pin. Deze stel ik dan in zodat de lengte van het signaal 10 uS is en de tijd tussen twee pulsen 25 ms.

Afbeelding met tekst, schermopname, software, Lettertype

Automatisch gegenereerde beschrijving

Vervolgens stel ik timer 4 in voor het berekenen van de pulsetijd (Echo time). Timer 4 laat ik continue doortellen zodat ik op elk moment de waarde van de timer kan opvragen. Dan vraag ik de waarde van de timer op, op het moment dat de triggerpin is geactiveerd. Als de echo pin hoog wordt en dus het signaal binnenkrijgt, zal er een interrupt getriggered worden. In de interrupt handle methode vraag ik de waarde van de timer weer op. Ik bereken dan het verschil tussen de twee waardes en zo heb ik de tijd dat het signaal erover gedaan heeft om heen en weer te gaan van de sensor, naar het obstakel en weer terug.

Hieronder zie je dat terug in de pulstraveltime die dus wordt berekend aan de hand van de interrupt.

Afbeelding met tekst, schermopname, software, scherm

Automatisch gegenereerde beschrijving

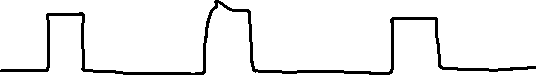
De uiteindelijke afstand bereken ik dat aan de hand van de formule. Omdat de pulstraveltime in microseconde is doe ik hem maal 0.0343 uS/cm (343 m/s is de snelheid van geluid door lucht) en vervolgens deel ik hem nog door twee, omdat anders de gehele tijd wordt meegerekend van het heen en weer gaan van de puls.



### Techniek 2

Er is ook nog een andere techniek die ik kan gebruiken om de afstand te bepalen. Hiervoor moet ik naast het PWM output signaal op de trigger pin ook nog een PWM input signaal hebben op de echo pin. Door deze twee timers juist te timen kan ik constant de afstand blijven uitlezen. Hierbij is het belangrijk dat de timer op de trigger pin een pulse stuurt niet sneller dan elke 20ms. Waardes sneller dan dit zou betekenen dat bij een afstand > 1 meter er al een nieuw trigger signaal komt als de vorige nog niet is opgevangen.

Om de afstand te bepalen moet ik steeds de echotime achterhalen, dit is de tijd dat de puls erover gedaan heeft om terug bij de sensor te komen. Dit kan ik doen door steeds de count te “capturen” op het moment dat de echo pin hoog wordt.



## Deelvraag 3 Wat is een closed loop systeem en hoe zet ik hem op met een PID controller?

Een closed loop systeem bestaat uit verschillende onderdelen en heeft duidelijk verschil met open loop systemen. Hieronder zie je een lijst van begrijppen die betrekking hebben tot closed loop systemen, de verschillende onderdelen komen hier aanbod.

**Open loop** = Een openloop systeem maakt niet gebruik van een feedback loop. De input wordt zelf gedefinieerd om de gewenste output te verkrijgen.

**Closed loop** = Een feedback systeem waarbij de output steeds wordt vergeleken met de gewenste ouptut, om zo afwijkingen en fouten te voorkomen. Closed loop systemen zijn goed voor het handelen van variaties in het systeem en voor het handelen van onverwachte data. In het geval van een broodrooster kijk je bijvoorbeeld steeds naar de status van het brood, inplaats de broodrooster voor een vaste tijd aan te zetten, iets wat je zou zien bij een open loop.

**Desired output (Reference, Setpoint, commanded variable)** = de output die je wilt hebben (bijv. dat de auto 50 km/h rijdt)

**Output** = de data die uit het systeem komt

**Controller** = het gene wat het systeem controlleert (bijv. de bestuurder van de auto)

**Actuator** = het gene wat de output beinvloed (bijv. het gaspadaal **in** een auto)

**Plant** = het object waarvan je de output wilt reguleren (bijv. de auto)  
**Sensor** = het gene wat de output van de plant bekijkt (bijv. speedometer)

**Disturbance** = kan invloed hebben op de plant (bijv. de weg wordt heel stijl bij een auto)

**Noise** = variatie in de sensor data die je niet wilt hebben. Een onderbreking van het signaal die afhankelijk is van de omgeving, implementatie en de staat van de sensor zelf.

**Error** = de waarde die de output afwijkt van de desired output

Afbeelding met tekst, schermopname, diagram

Automatisch gegenereerde beschrijving

Figuur Closed loop auto voorbeeld (MATLAB, 2016e)

### De PID

Voor het maken van een closed loop systeem is een PID controller een zeer belangrijke component. PID staat voor (Proportional-Integral-Derivative) en het draait om het reguleren van de input zodat je de gewenste output krijgt. De waarde die de ouptut afwijkt van de desired output wordt ook wel de error term of error genoemd.

* **Proportional** = Bij PID staat proportional voor de term die gaat over het direct reageren op de error. De proportional term kijkt naar de error op het huidige moment en is evenredig met de grootte van de error. Oftewel, als de error bijvoorbeeld kleiner wordt, zal de proportionale term ook kleiner worden.
* **Integral** = De integral term kijkt naar het verleden voor het behandelen van de error. De integral term telt eigenlijk de error op over een bepaalde tijd en kan hierdoor stationaire errors elimineren uit het systeem. Stationaire errors zijn errors die het systeem er niet echt uit krijgt, ze blijven constant aanwezig.
* **Derivative** = De derivative term meet de veranderingssnelheid van de error. Het voorspelt het toekomstige gedrag van de error door de snelheid van verandering te analyseren. De derivative term draagt bij aan de stabiliteit en responsiviteit van het regelsysteem.

De drie onderdelen hebben allemaal hun weighting factor. De factor van hoe zwaar ze meetellen. Deze factor wordt aangetoond met de letter ki, kp of kd afhankelijk van om welke onderdeel het gaat.

Afbeelding met tekst, schermopname, diagram, ontwerp

Automatisch gegenereerde beschrijving

Figuur PID Drone voorbeeld (MATLAB, 2018)

Bij gebruik van de **integral term** in een PID controller moet je uitkijken voor een fenomeen genaamd de **“integral windup”.** Dit is een fenomeen wat voorkomt als een actuator buiten zijn mogelijkheid gaat opereren en dat hierdoor de integral term te hoog gaat opbouwen. Al heb je bijvoorbeeld een motor van een drone die maximaal 1000 rpm kan draaien en je blijft hem vasthouden waardoor de error blijft, dan zal de integral term blijven optellen tot een onmogelijke rpm. Op het moment dat je dan vervolgens de drone weer loslaat zal je een grote vertraging hebben op de integral term, omdat deze veel tijd nodig heeft om weer te zakken. Hierdoor zal de drone ongewenst gedrag vertonen.

Er zijn verschillende manieren om integral windup te voorkomen. Een populaire manier is de clamping techniek. Deze techniek draait eigenlijk om het uitzetten van de integral term op het moment dat hij dus te ver doorschiet en dus “saturation” veroorzaakt. Voor dit uitzetten worden er naar twee dingen gekeken. Als eerste wordt er gekeken of er saturation aanwezig is. Dus dat bijvoorbeeld de motor de opdracht krijgt om een te grote snelheid te draaien. Daarna wordt er gekeken of dat de integral term dit probleem erger aan het maken is. Zo ja, dan wordt de integral term uitgeschakeld.

Zoals eerder vermeld is er bij sensoren ook altijd **noise** aanwezig. De noise zorgt voor kleine afwijking in het signaal, maar kan voor grote invloed zijn op de **derivative term.** Om dit probleem op te lossen kan je een filter toevoegen aan de derivative. Om precies te zijn kan je hiervoor een zogeheten **low pass filter** toepassen. Deze filter laat alleen signalen van lagere frequenties door en blokkeert de signalen van hogere frequenties. Waar het punt precies ziet vanaf waar de frequenties worden geblokkeerd kan je zelf bepalen, maar is ook erg belangrijk om goed in te stellen. De reden waarom je hoge frequenties signalen wilt tegengaan is vanwege de hoge amplitude die ze hebben. Omdat de golf van het signaal steiler wordt zal de amplitude hoger worden en dus ook de derivative term.

Afbeelding met schermopname, tekst, lijn, Lettertype

Automatisch gegenereerde beschrijving

Figuur PID effect van noise op de derivate van MatLabs

Afbeelding met tekst, Lettertype, schermopname, diagram

Automatisch gegenereerde beschrijving

Om deze filter toe te passen zijn er vershillende formules die gebruikt kunnen worden. Een vrij simpele formule hiervoor is een exponentiële smoothing-formule. Deze ziet er bijvoorbeeld als volgt uit: *filteredValue = filterCoeff \* input + (1 - filterCoeff) \* lastFilteredValue*. In deze formule wordt er steeds gekeken naar de vorige waarde en afhankelijk van hoe groot de filterfcoeff is wordt de input hierop gefilterd. Het nadeel van het gebruiken van deze functie is wel dat je wat minder controlle hebt over de filter waardoor je hem dus minder goed kan afstellen op jouw implementatie.

## PID keuze

Bij het implementeren van een PID-controller voor mijn autonome robotauto, kan ik gebruik maken van een servo motor encoder en een ultrasonische sensor. De verschillen tussen deze twee zijn als volgt.

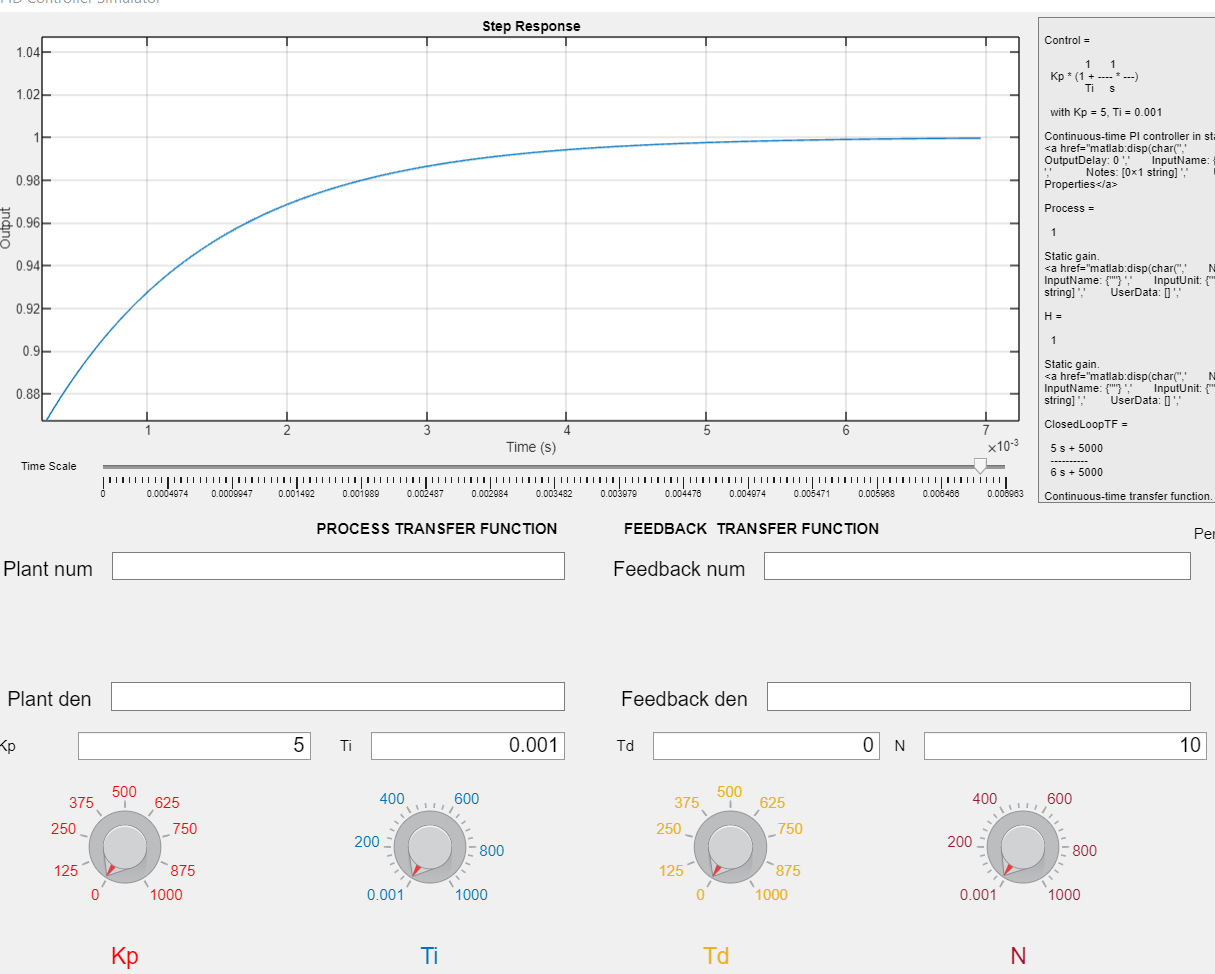
Een servo motor encoder biedt zeer nauwkeurige feedback over de positie en snelheid van de wielen. Dit kan waardevol zijn in situaties waarbij precisiebewegingen en regeling van de snelheid cruciaal zijn. Het gebruik van een encoder in combinatie met een PID-controller maakt het mogelijk om de robotauto met grote precisie te besturen.

Aan de andere kant biedt een ultrasone sensor de mogelijkheid om obstakels te detecteren en de afstand tot deze obstakels te meten. Hoewel de nauwkeurigheid van een ultrasone sensor mogelijk niet zo hoog is als die van een encoder, is de exacte positionering op basis van de sensorgegevens vaak belangrijker bij het autonoom navigeren en vermijden van obstakels.

Ik heb er daarom voor gekozen om mijn PID toe te passen op mijn ultrasonische sensor. De snelheid regulering die ik hieruit krijg is goed genoeg voor mijn implementatie. Mijn autonome auto moet sowieso instaat zijn om objecten te vermijden, dus ik kom er toch niet onderuit om een sensor te gebruiken.

### PID Berekenen

In het geval van mijn robotauto zijn er meerdere factoren waar je naar moet kijken om je PID juist in te stellen. Als je alleen een proportional term instelt op bijvoorbeeld 5 zal de auto richting een object rijden en steeds meer snelheid verminderen naarmate hij dichterbij komt.



Echter zal het door verschillende factoren vanuit de omgeving moeilijk zijn om precies op de gewenste afstand te blijven. Daarom implementeren we dus ook een integral term. Het effect daarvan is dat de auto geen steadystate error meer heeft en dat betekent dus dat hij veel beter op de gewenste afstand kan blijven. Een grote integral kan er echter wel weer voor zorgen dat de auto gaat overshooten. Dit zie je ook hieronder terug, het kost wat tijd voordat de integral is gestabiliseerd.

Afbeelding met tekst, schermopname, diagram, Perceel

Automatisch gegenereerde beschrijving

Als je tenslotte ook nog een derivative toevoegd zul je zien dat de overshoot veel minder wordt. Hoe beter je dit afstelt hoe minder de overshoot. Echter moet je wel uitkijken met een te grootte derivative omdat je niet wilt dat hij te heftig gaat reageren op een plotselinge verandering van de error.

Afbeelding met tekst, schermopname, diagram, Perceel

Automatisch gegenereerde beschrijving

### PID metingen

Om het effect van de termen beter in kaart te brengen heb ik verschillende metingen gedaan met mijn robotAuto. Ik heb mijn code laten uitprinten wat de afstand en de tijd is op het moment van de meting. Deze metingen heb ik vervolgens in een tabel gestopt om daarvan weer een grafiek te maken.

De eerste grafiek hieronder laat alleen de proportional term ziet. Hier zijn de integral en de derivative nog uitgeschakeld. Wat hier opvalt is de aanwezigheid van een steady state error. De auto bereikt nooit volledig de setpoint omdat als hij te dichtbij zijn snelheid al te laag is.

Afbeelding met tekst, lijn, diagram, Perceel

Automatisch gegenereerde beschrijving

Figuur De grafiek van proportional term

De volgende grafiek geeft het effect van de proportional samen met de integral weer. Hier zie je dus dat hij in het begin de setpoint overshoot, maar dat hij uiteindelijk stabiliseert en ook de aanwezigheid van een steady state error heeft beperkt.

Afbeelding met tekst, lijn, Perceel, diagram

Automatisch gegenereerde beschrijving

Figuur De grafiek van de integral term

De laatste grafiek hieronder laat het effect van alle drie de termen tegelijk zien. Je ziet nu dat de robotauto vrij soepel richting de setpoint gaat en dat hij ook niet overshoot. Voor mij implementatie is dit het gewenste resultaat.

Afbeelding met tekst, lijn, Perceel, diagram

Automatisch gegenereerde beschrijving

Figuur De grafiek van de Derivative term

## Deelvraag 4: Hoe zet ik een goede abstractie laag op voor mijn closed loop systeem, die veel functionaliteit bied en de inhoud van mijn programma overzichtelijker maakt?

Om een goeie abstractielaag op te zetten is het belangrijk dat je gaat kijken naar welke objecten er zich allemaal bevinden in jouw systeem. Door te kijken naar welke objecten er allemaal zijn en door vervolgens de connecties tussen deze objecten te bestuderen, zal je erachter komen welke klassen er allemaal nodig zijn in jouw programma.

Voor het geval van mijn robotauto ben ik ook op basis van dit stappenplan tewerk gegaan. De beschrijving van de klassen is hieronder weergegeven in de tabel.

### Klassendiagram

|  |  |
| --- | --- |
| Klasnaam | Beschrijving |
| RobotAuto | De hoofdklasse vanaf waar alles wordt geregeld. De robotauto beschikt over de andere klasse zodat vanuit deze klasse alles bestuurd kan worden. |
| DistanceSensor | De distancesensor klasse is de klasse van de ultrasonische afstand sensor. Deze sensor berekent de afstand tussen de auto en het object voor hem. |
| ServoMotor | De servomotor klasse voor de twee servomotoren die zich op de robotauto bevinden. |
| PIDController | De PIDController klasse die de daadwerkelijke PID regelt. De controller gebruikt de afstand van de distanceSensor om zo de waarde te berekenen voor de servomotoren. |
| UIManager | De UIManager klasse regelt alle UI gebeurtenissen. Met het UI kan de gebruiker verschillende dingen van de Robotauto instellen. |
| CarTimer | De CarTimer klasse is een klasse waarin alle timers staan die de Servomotoren en distance sensor gebruiken. |
| Pins | De Pins klasse is een klein bestand waarin de verschillende pins staan omschreven. |

De klassendiagram waarin alle verbindingen tussen de klassen te zien zijn. Hier zie je dus ook dat de vier kleinere klassen ook beschikken over een interface zodat er meerdere implementaties gebruikt kunnen worden door de “hoofd” klasse, Robotauto.

Afbeelding met tekst, diagram, Parallel, nummer

Automatisch gegenereerde beschrijving

## Deelvraag 5: Hoe pas je multithreading toe met synchronisatie mechanismen aan de hand van de FreeRTOS library?

### Onderzoek synchronisatie mechanisme

Multithreading kan je op verschillende manieren toepassen. Om de juiste werking te krijgen tussen threads kan je synchronisatiemechanismen toepassen. Queues, semaforen, event groups en mutexes zijn allemaal synchronisatiemechanismen die kunnen worden gebruikt bij FreeRTOS. De synchronisatiemechanismen luiden als volgt:

**Semaforen:**

* Binaire semafoor: Een binair semafoor kan worden gebruikt om de toegang tot een gedeelde bron te beperken tot één taak tegelijk. Het fungeert als een eenvoudige vergrendeling. Bijvoorbeeld, een gedeelde variabele kan worden beschermd door een binair semafoor, waarbij een taak het semafoor vastlegt voordat het de variabele wijzigt en het semafoor vrijgeeft wanneer het klaar is.
* Counting semafoor: Een tellend semafoor houdt het aantal beschikbare bronnen bij. Het kan bijvoorbeeld worden gebruikt om een beperkt aantal gelijktijdige toegangen tot een specifieke bron te regelen. Een taak kan het semafoor vastleggen om toegang te krijgen tot de bron en het semafoor vrijgeven wanneer het klaar is, waardoor het aantal beschikbare bronnen wordt verhoogd. Bijvoorbeeld, als er maximaal drie printers zijn en vier taken willen afdrukken, kunnen ze een tellend semafoor gebruiken om te wachten tot een printer beschikbaar is.

**Queues**:

* Een wachtrij maakt het mogelijk om gegevens tussen taken te verzenden. Een taak kan gegevens in de wachtrij plaatsen met behulp van de functie xQueueSend() en andere taken kunnen die gegevens lezen en verwerken met behulp van de functie xQueueReceive(). Bijvoorbeeld, een taak kan sensorgegevens in een wachtrij plaatsen en een andere taak kan die gegevens vervolgens lezen en verwerken.

**Mutexen**:

* Een mutex wordt gebruikt om de toegang tot gedeelde bronnen te beheren. Een taak kan een mutex vastleggen met behulp van de functie xSemaphoreTake() voordat het een gedeelde bron benadert en de mutex vrijgeven met behulp van xSemaphoreGive() wanneer het klaar is. Als een taak een mutex vasthoudt, kan geen andere taak die mutex vastleggen. Bijvoorbeeld, als meerdere taken een gedeelde variabele willen wijzigen, kunnen ze een mutex gebruiken om ervoor te zorgen dat slechts één taak tegelijkertijd toegang heeft tot die variabele.

**Softwaretimers**:

* Softwaretimers kunnen worden gebruikt om taken periodiek te activeren op basis van een tijdsinterval. Je kunt een softwaretimer maken met behulp van de functie xTimerCreate() en deze starten met behulp van xTimerStart(). Bijvoorbeeld, je kunt een softwaretimer instellen om elke 1 seconde een taak te activeren, waardoor periodieke acties kunnen worden uitgevoerd, zoals het verzenden van een statusbericht.

**Eventgroups**:

* Eventgroepen bieden een mechanisme om taken te laten wachten op specifieke gebeurtenissen. Je kunt een eventgroep maken met behulp van de functie xEventGroupCreate() en taken kunnen wachten op bepaalde vlaggen met behulp van xEventGroupWaitBits(). Bijvoorbeeld, je kunt een eventgroep gebruiken om taken te laten wachten op verschillende gebeurtenissen, zoals het ontvangen van gegevens over een communicatie-interface of het voltooien van een bepaalde bewerking.

Dus in het kort om de verschillen duidelijk te hebben zijn binaire semaforen voor situaties waarbij meerdere gebruikers toegang willen tot één gedeelde bron, maar dat er maar één tegelijk toegang mag hebben.

Mutexes worden gebruikt in een vergelijkbare context maar hebben als verschil dat alleen de gebruiker die de bron heeft hem kan vrijgeven en dat de prioriteit van de gebruiker ook relevant is. “Mutexes and [binary semaphores](https://www.freertos.org/a00121.html) are very similar but have some subtle differences: Mutexes include a priority inheritance mechanism, binary semaphores do not. This makes binary semaphores the better choice for implementing synchronisation (between tasks or between tasks and an interrupt), and mutexes the better choice for implementing simple mutual exclusion.” (FreeRTOS, 2022) Dus bij een duidelijk eigenaarschap zou je mutex beter kunnen gebruiken dan een binaire semafore.

Counting semaforen onderscheiden zich van binaire semaforen in dat ze meerdere gebruikers toegang kunnen geven tot een bron. Dus als je bijvoorbeeld wilt dat een bron maximaal door die gebruikers tegelijk kan worden gebruikt.

Queues kan je het best gebruiken als je een buffer van data nodig hebt bijvoorbeeld een sensor die veel data geeft en meerdere actuatoren die op deze data reageren.

Softwaretimers kan je gebruiken voor het plannen en uitvoeren van periodieke taken op basis van een tijdsinterval.

En tenslotte Eventgroups kan je gebruiken om taken te laten wachten op bepaalde gebeurtenissen.

### Keuze synchronisatie mechanisme

Voor mijn robotauto gebruik ik twee threads. Één thread voor de user interface en de andere voor de functionaliteit van de robotauto en het verwerken van de input van de gebruiker. Deze twee threads hebben een gedeelde “choice” variabel waarin de input van de gebruiker wordt opgeslagen en gelezen. Omdat in mijn context het eigenaarschap van de gedeelde bron duidelijk is en omdat ik wil dat de thread die de bron in bezit heeft deze ook weer vrijgeeft is een mutex de beste keuze. De thread die de bron gebruikt is de eigenaar van deze bron en geeft hem weer vrij als hij klaar is. Ik wil niet dat de implementatie thread de bron gaat proberen vrij te geven als de user interface nog aan het wachten is op input van de gebruiker.

# Conclusie

De onderzoeksvraag aan het begin van dit onderzoek was: **Hoe zet ik een closed loop multithreading systeem met abstractie laag op voor een autonome robotauto, waarbij gebruik wordt gemaakt van een ultrasonische sensor?** Om deze vraag te beantwoorden heb ik verschillende deelvragen opgesteld die inzoomen op de losse onderdelen van dit onderzoek.

Allereerst werd de werking van de HC-SR04P-sensor onderzocht. Hierbij werden de specificaties en functionaliteiten van de sensor geanalyseerd om een goed begrip van de mogelijkheden en beperkingen te krijgen.

Vervolgens werd de integratie van de HC-SR04P-sensor met het STM32 Nucleo-bord bekeken. Technieken werden onderzocht en toegepast om de sensor effectief te kunnen gebruiken binnen het systeem.

De derde vraag ging over het maken van een closed loop systeem met een PID controller. De theorie achter PID heb ik me eigen gemaakt om zo de juiste PID-parameters te bepalen voor mijn context. Ik heb ook nog metingen uitgevoerd om de prestaties van het PID-systeem te ondervinden en te evalueren.

Daarna ben ik me gaan focussen op het maken van een goede abstractielaag voor het closed loop systeem. Ik heb een klassendiagram opgesteld om de structuur en relaties tussen de verschillende systeemcomponenten overzichtelijk weer te geven.

Tot slot werd multithreading toegepast met behulp van synchronisatiemechanismen uit de FreeRTOS-library. Ik heb verschillende synchronisatiemechanismen onderzocht en ik heb een weloverwogen keuze gemaakt op basis van de specifieke behoeften van het systeem.

Ik heb in dit onderzoek veel geleerd over het opzetten van een closed loop multithreading systeem met een abstractielaag voor een autonome robotauto, waarbij gebruik wordt gemaakt van een ultrasonische sensor. De behandelde onderdelen hebben bijgedragen aan de benodigde kennis en technieken om het systeem succesvol te implementeren.

# Bronvermelding

Datasheet van de HC-SR04P:

<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>

*Getting Started with the HC-SR04 Ultrasonic sensor*. (z.d.). projecthub.arduino.cc. <https://projecthub.arduino.cc/Isaac100/7cabe1ec-70a7-4bf8-a239-325b49b53cd4>

Pounder, L. (2018). Tooling Tuesday - HCSR04P Ultrasonic Sensor. *bigl.es*. <https://bigl.es/tooling-tuesday-hcsr04p-ultrasonic-sensor/>

Presentatie over timers van de school git

BestandsLocatie: \t-sem3-db\embedded-systems\Presentations

Presentatie over closed loop van de school git  
BestandsLocatie: \t-sem3-db\embedded-systems\Presentations

MATLAB. (2016, 25 oktober). *Understanding Control Systems: Introduction* [Video]. YouTube. <https://www.youtube.com/watch?v=VZIsHshZnFU>

MATLAB. (2016b, oktober 25). *Understanding Control Systems, Part 1: Open-Loop Control Systems* [Video]. YouTube. <https://www.youtube.com/watch?v=FurC2unHeXI>

MATLAB. (2016c, november 1). *Understanding Control Systems, Part 2: Feedback Control Systems* [Video]. YouTube. <https://www.youtube.com/watch?v=5NVjIIi9fkY>

MATLAB. (2016d, november 16). *Understanding Control Systems, Part 3: Components of a Feedback Control System* [Video]. YouTube. <https://www.youtube.com/watcdh?v=u1pgaJHiiew>

MATLAB. (2018, 22 mei). *What Is PID Control? | Understanding PID Control, Part 1* [Video]. YouTube. <https://www.youtube.com/watch?v=wkfEZmsQqiA>

FreeRTOS. (2022, 14 juni). *FreeRTOS - A FREE Open Source RTOS for small real time embedded systems*. <https://www.freertos.org>

*Main Page*. (z.d.). <https://arm-software.github.io/CMSIS_5/RTOS2/html/index.html>