# Meetrapport titel

## Doel

Vast stellen of de Linux queue de beste optie is en zo niet wat beter kan werken

## Hypothese

We voeren dit onderzoek uit om vast te stellen wat de besten manier is om een queue te maken die gebruikt kan worden voor de communicatie tussen de verschillende onderdelen

## Werkwijze

Onderzoek doen in de Linux queue documentatie en op internet verder kijken naar mogelijk andere queue mogelijk zijn.

## Resultaten

Al na korten onderzoek doen bleek dat de Linux queue veel ingewikkelder zou gaan zijn dan dat nodig was voor de software. Zo zijn we bij de STD queue terecht gekomen deze had een eenvoudige werking en was makkelijker te implementeren

**STD::queue**

De werking van een std queue is ongeveer gelijk aan de werking van een vector. De std queue werkt via het FiFo principe. Dit betekent first in first out, dus het element dat er als eerste word ingestopt komt er ook weer als eerste uit. De std queue ondersteund de volgende commando’s:

|  |  |
| --- | --- |
| **Functie** | **Werking** |
| empty | Checkt of de queue leeg is of niet |
| size | Geeft de grootte van de queue |
| front | Geeft het eerste element uit de queue terug |
| back | Geeft het laatste element uit de queue terug |
| push(std::String) | Stopt een element in de queue |
| Pop | Verwijderd het eerste element uit de queue |
| Swap | Draait de inhoud van de queue om. |
|  |  |

.

## Conclusie

Uit uitvoerig onderzoek en meerdere experimenten met een linux queue vinden we dit geen goede oplossing. De oplossing is te complex om goed te implementeren naar onze mening. Als vervanging maken we de websocket nu onderdeel van het RTOS en gebruiken hier een STD::queue.

## Evaluatie

Leg een verband tussen de getrokken conclusie en het doel van het experiment (en de hypothese). Ga daarbij ook in op bijvoorbeeld de meetonzekerheid als gevolg van de gebruikte meetmethoden of eventuele meetfouten.