De beweging van de robot wordt gemaakt met het unicycle model. Dit model beschrijft een robot die vooruit kan bewegen en kan draaien, maar niet zijwaarts kan glijden (zoals een auto).

De robot heeft twee besturingsinputs:

1. v = voorwaartse snelheid(cm/s)
2. w = hoekdraaisnelheid (rad/s)

De bewegingsvergelijkingen zijn:

$$\frac{dx}{dt} = v * \cos\theta$$

$$\frac{dy}{dt} = v * \sin\theta$$

$$\frac{d\theta}{dt} = w$$

Navigatie en richtingscorrectie

Om naar een doelpunt te bewegen, berekent de robot eerst in welke richting hij moet kijken.

De gewenste stuurhoek is:

$$\theta_{target} = \tan^{-1}(\frac{yt - y}{xt - x})$$

Het verschil tussen de gewenste richting en de huidige oriëntatie wordt de heading error genoemd.

$$e_\theta = \theta_{target} - \theta$$

Dit vertelt de robot:

Positief → Naar links draaien

Negatief → Naar rechts draaien

Nul → Niks te veranderen, al in de juiste richting kijkend

Proportioneel controller (P-Controller)

Om deze fout soepel te corrigeren wordt een proportionele regelaar gebruikt.

De hoeksnelheid is:

$$w = K_p * e_\theta$$

Waarbij:

1. w = draaisnelheid
2. Kp = proportioneel constante
3. e-theta = koersfout

Hiermee wordt bedoeld:

1. Als de koersfout groot is → sneller draaien
2. Als de koersfout klein is → Draai langzaam en voorzichtig

Deze regelaar was oorspronkelijk niet aanwezig in de robot code, maar werd toegevoegd nadat tijdens de simulatie bleek dat zonder deze correctie het draaigedrag instabiel werd.

Mapping en schoonmaakpad (spiraal)

Tijdens de simulatie gaat het systeem ervan uit dat de afmetingen van het zonnepaneel vooraf onbekend zijn.

Het bepaalt de grens door sonar data in vier richtingen te detecteren: omhoog, rechts, omlaag en links.

Op basis hiervan vindt de robot het volgende:

$$x_{min}, x_{max}, y_{min}, y_{max}$$

Deze waarden vormen een rechthoekig gebied:

$$A = (xmax - xmin) * (ymax - ymin)$$

Waarbij A = totale paneeloppervlakte

Voor het schoonmaken wordt een spiraalvormig pad gebruikt waarbij het werkgebied na elke iteratie kleiner wordt.

De krimpende afstand wordt als volgt gedefinieerd:

$$d = w * (1 - o)$$

Waarbij:

1. w = robot breedte
2. o = overlap verhouding (0.5 = 50%)

De grenzen bij lus k zijn:

$$x_{min}^{(k)} = x_{min} + k * d$$
$$x_{max}^{(k)} = x_{max} - k * d$$
$$y_{min}^{(k)} = y_{min} + k * d$$
$$y_{max}^{(k)} = y_{max} - k * d$$

De spiral stopt wanneer:

$$x_{min}^{(k)} \geq x_{max}^{(k)} \text{ or } y_{min}^{(k)} \geq y_{max}^{(k)}$$

Of te wel het is in het centerum van de paneel.

Vuildetectie

Elk camerabeeld wordt geanalyseerd door het neurale netwerk.

Het netwerk geeft een waarschijnlijkheid als uitvoer:

$$P(vuil \mid afbeelding)$$

Als P(vuil) > r, locatie als 'vuil' gemarkeerd

r = drempelwaarde

P = kans

Wanneer deze kans groter is dan een ingestelde drempelwaarde, wordt de locatie als vervuild opgeslagen.

Schoonmaak route-optimalisatie

“Nearest neighbor algorithm”

Na het scannen van het zonnepaneel kan de robot meerdere vuile plekken hebben gedetecteerd. Elke plek wordt beschreven door een coördinaat (x1, y1), (x2, y2), (x3, y3), …, (xn,yn). Waarbij ‘n’ het totale aantal gevonden vuile plekken is.

Stel dat de robot zich momenteel op positie (x_i, y_i)bevindt. Voor elke gevonden vuile plek wordt de Euclidische afstand tot de huidige positie van de robot berekend:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

De volgende locatie die de robot bezoekt wordt bepaald door de plek te kiezen waarvoor deze afstand minimaal is:

$$Next\ spot = \arg\min_{j} (d_{ij})$$

Min → geeft de kleinste waarde

Arg min → geeft aan welk punt de kleinste waarde oplevert

Nadat een volgorde is gekozen, dus 1->2->3 enzovoort, is de totale afgelegde afstand de som van alle individuele afstanden tussen opeenvolgende punten.

$$L = \sum_{k=1}^{n-1} d_{k,k+1}$$

Wat dus betekent:

$$L = d_{1,2} + d_{2,3} + \cdots + d_{n-1,n}$$

De robot verplaatst zich vervolgens naar deze locatie, waarna het proces wordt herhaald met de nieuwe positie als startpunt. Op deze manier ontstaat een efficiënte, lokaal optimale schoonmaakroute langs alle gedetecteerde vuile plekken.

Simulatie versies

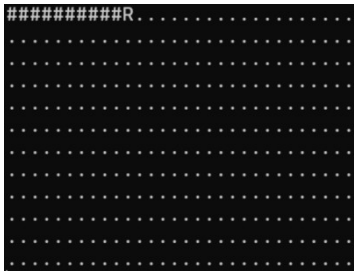
Versie.1 C++ navigatie test

In het eerste versie van het simulatie werd vooral het testen van het spiraal navigatie en robot mapping uitgetest. Daarvoor werd een simpel C++ programma geschreven waarbij het formules voor coordinaten en spiraal navigatie met grenzen getest.

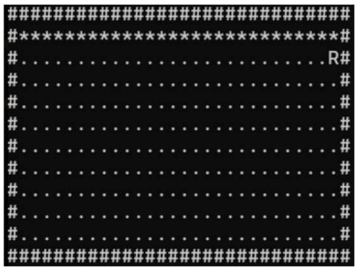
R = Robot

= Mapping

* = Navigatie



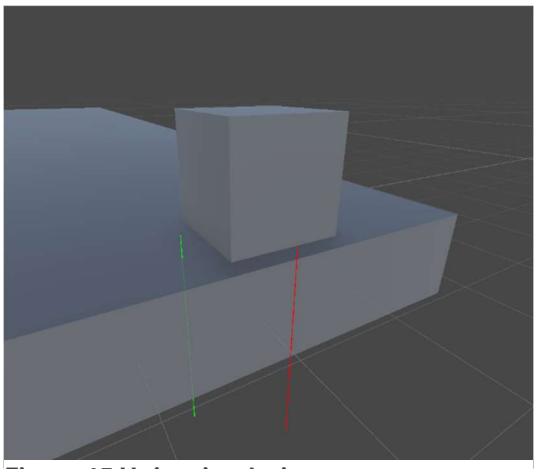
Figuur 13 C++ simulatie omgeving



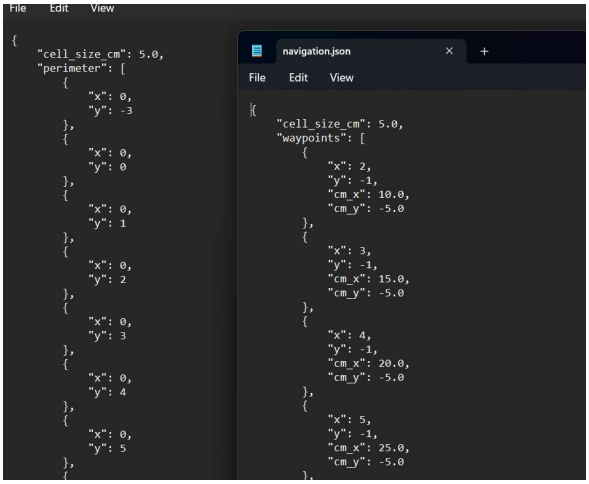
Figuur 14 C++ simulatie omgeving

Versie.2 Unity robot bewegingsmodel en mapping test

In het tweede versie werd het bewegingsmodel en mapping concept in Unity uitgewerkt tot een werkende simulatie. Hierbij werd het formules boven gebruikt om de robots bewegingsmodel realistish te maken, en daarbij ook een sensoor simulatie. In figuur 15 is de kubus boven de simulatie robot, en daaronder de zonnepaneel. Er is een object voor en links van de robot geplaats, wat constant naar beneden kijkt of er grond is of niet. Als er geen grond is, word de sensoor rood, en als er wel grond is word de sensoor groen.



Figuur 15 Unity simulatie



Figuur 16 JSON bestand

Als de voorkant grond is, terwijl de linkerkant niet grond is, betekent dat de robot nu op een rand zit. Dan volgt de robot dit rand tot dat het aan het voorkant ook geen grond meer te detecteren is, dan draait het 90 graden naar rechts, noteer de coordinaten van het huidige positie, en herhaalt wat het hier voor heeft gedaan, tot dat het alle vier hoeken heeft gevonden. Dit word dan in een JSON bestand opgeslagen, zodat het volgende code, de navigatie, hier mee een pad kan maken.