

---

## NXT - lijn detectie met behulp van een omni-directionele camera

---

*Auteurs: Tom Peerdeman &  
René Aparicio Saez*

*Datum: November 23, 2013*

## 1 Materiaal

Om de experimenten uit dit rapport te kunnen uitvoeren zijn de volgende materialen gebruikt:

- PC/Laptop met Matlab
- Boek: Autonomous Mobile Robots 2th Edition - Roland Siegwart et al.
- NXT-Robot
- Logitech Webcam
- Gloeilamp
- Zwarte tape

## 2 Introduction

Een autonome mobiele robot moet kunnen bepalen waar hij wel of niet mag rijden. Het is daarom nodig om de robot sensoren te geven die hem vertellen waar hij wel en niet kan rijden. Met behulp van een omni-directionele camera en een lijn-detectie algoritme kan de robot bepalen waar hij wel en niet naartoe kan rijden. Met behulp van een gewone webcam en een bolling met een weerkaatsend oppervlak kan eenvoudig een omni-directioneel beeld gemaakt worden. De code voor het rechttrekken van het beeld en het detecteren van lijnen is meegeleverd. Er hoeft slechts bepaald te worden wat de parameters voor de code zijn.

## 3 Methode

Enkele fotos waren meegeleverd waarmee getest kon worden. Echter is besloten nieuwe fotos te maken. De robot moest nog worden aangepast. Door op de webcam een plastic buis met daarop gemonteerd een gloeilamp aan de robot vast te tapen is de omni-directionele camera gemaakt. In figuur 1 is te zien hoe de gebruikte robot er uit zag. Aan de hand van de live beelden die de camera doorstuurde, konden enkel fotos gemaakt worden, waarmee verder gewerkt kon worden. De gemaakte fotos zijn van het formaat 'RGB24: 960 x 720'. De meegeleverde fotos waren van het kleinere formaat 'RGB24: 640 x 480'. Als de webcam met dit kleinere formaat opnames maakte, werd het beeld echter te wazig. Het grotere formaat behoudt de verhoudingen, en is scherper. Daarnaast zijn de opnames gemaakt met een verhoogd contrast. Dit zorgt ervoor dat er minder kleurverschil en minder ruis in het beeld aanwezig is.

## 4 Parameters bepalen

### 4.1 Calibreren van de camera

De eerste parameters die moeten worden gekozen zijn de coordinaten voor het middelpunt van de camera. Zodra deze coordinaten zijn gevonden, kunnen deze in de code worden vastgelegd, omdat het middelpunt van de camera nooit zal veranderen (hierbij wordt er van uitgegaan dat de camera goed op de robot is gemonteerd en niet beweegt). Als het middelpunt van de camera goed is gekozen zal het uitgeklapte beeld, de rand van de camera in een rechte lijn plaatsen (zie figuur 2 en figuur 3). Als dit niet het geval is zal de rand een slinger vertonen. Het middelpunt van de camera licht in het midden van de webcam, op de coordinaten  $x = 444.2568$  en  $y = 333.2305$ . Daarnaast moet de radius van de blinde vlek (de vlek waar de camera zichzelf ziet) worden bepaald. De radius van dit beeld is  $radius = 90.4013$ .

Bij het calibreren van de camera moet ook aangegeven worden naar welke afstanden gekeken moet worden, wordt er te dichtbij het midden van de webcam gezocht, zal de robot zichzelf zien en zal dit ruis gaan vormen. Kijkt de camera te ver door dan zal het beeld buiten de bolling vallen of te vere weg zijn om nog een goed resultaat op te leveren. De sensoren die aan de robot hangen steken helaas ver uit, dus moet er begonnen worden met kijken op een redelijk grote afstand bij de robot vandaan,  $R_{min} = 125$ . De afstand tot waar gekeken mag worden kan vrij gekozen worden, zolang deze maar niet te ver weg ligt,  $R_{max} = 215$ . De waarden van de coordinaten, de radius,  $R_{min}$  en  $R_{max}$  zijn in pixels.

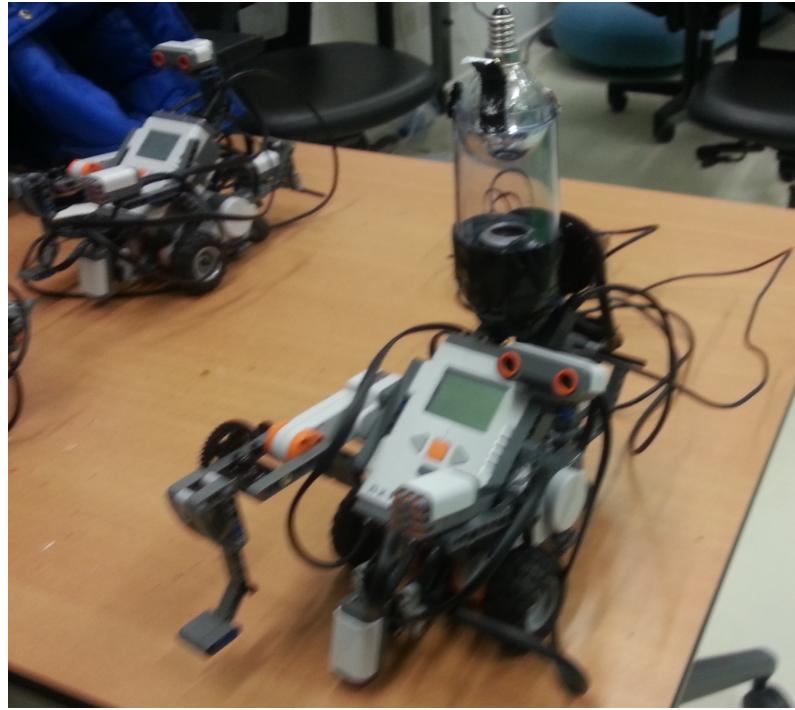


Figure 1: Originele robot links, aangepaste robot met camera rechts

## 4.2 Black-White Threshold

Als de camera is gecalibreerd kan het beeld dat van belang is (het beeld tussen  $R_{min}$  en  $R_{max}$ ) uit elkaar worden getrokken. Dit beeld wordt vervolgens omgezet in een zwart wit beeld aan de hand van een meegegeven threshold. Uit dit beeld kunnen vervolgens de lijnen worden gevonden die het dichtste bij de robot in de buurt liggen. Als de threshold niet goed wordt gekozen, zal het algoritme niet de volledige lijn vinden. Er moet dus een threshold worden gekozen die voldoende ruis wegfilterd, maar die wel de echte lijnen intact laat. In figuur zijn de resultaten voor verschillende thresholds te zien. Een threshold van 100 levert het beste resultaat op.

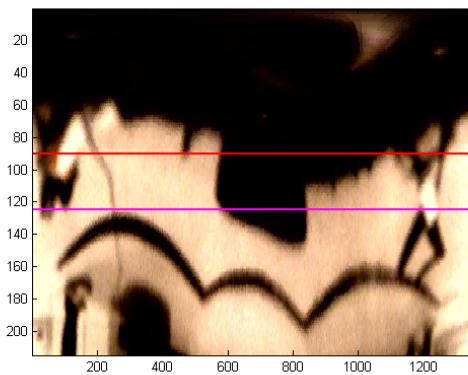


Figure 2: Incorrect middelpunt

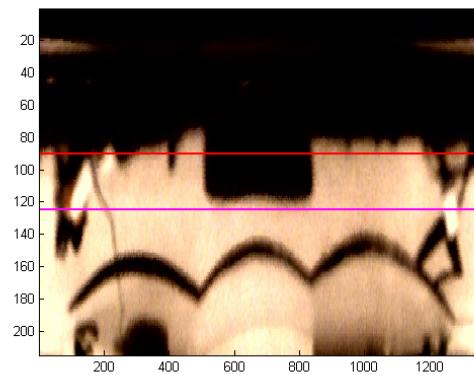


Figure 3: Correct middelpunt

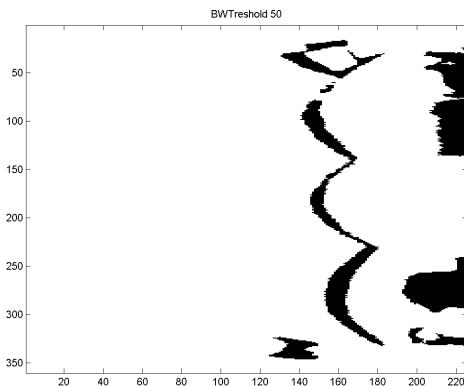


Figure 4: Treshold = 50

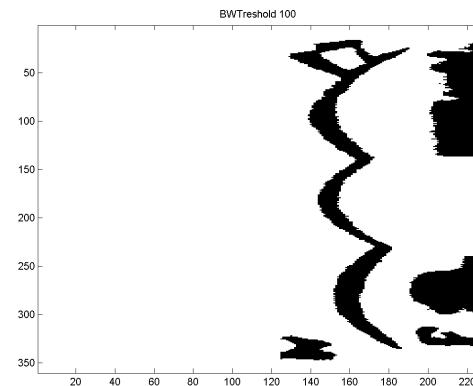


Figure 5: Treshold = 100

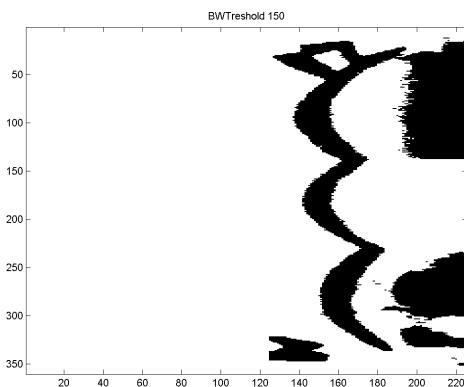


Figure 6: Treshold = 150

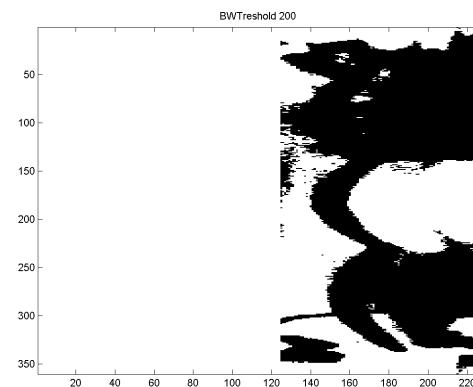


Figure 7: Treshold = 200

## 5 Werkelijke afstand

Als alle parameters zijn gevonden, kan uit het zwart wit beeld een lijst met pixelafstanden waarden opgemaakt tot de dichtsbijzijnde lijn. Echter Is het voor de gebruiker veel fijner om de afstanden in meters terug te krijgen. Aan de hand van de bolling, de hoogte van deze bolling ten opzichte van de camera en een gemeten werkelijke afstand kan een formule worden opgesteld die de pixelafstanden om kan zetten in werklijke afstanden. In figuur **insert figuur nodig die er niet is** is te zien hoe dit kan worden bepaald.  $d$  is de afstand in meters op de grond,  $\rho$  is de afstand in pixels op het camera vlak. De afstand voor een bepaalde hoek  $\theta$  is dan  $d(\theta) = h \cdot \tan(\theta)$ . De relatie tussen  $\theta$  en  $\rho$  kan worden verkregen aan de hand van een eerste orde Taylor reeks:  $\theta = \frac{1}{\alpha} \rho$ . Hierin moet  $\alpha$  worden bepaald. Dit is afhankelijk van de vorm van de bolling. Uiteindelijk verkrijgen we de formule  $d(\rho) = h \cdot \tan\left|\frac{\rho}{\alpha}\right|$ . Deze formule zet de pixel afstanden om in afstanden in meters.

Om  $\alpha$  te bepalen wordt een algoritme gebruikt. Alle alphas tussen 1 en 360 worden getest, en de alpha die een werkelijke afstand opleverd die daadwerkelijk opgemeten is, is de alpha die gebruikt kan worden. In figuur 8 is een geanimeerd plaatje te zien voor vele alphas. De gekozen waarde voor alpha is  $\alpha = 160$ , dit is te zien in figuur 9.

## 6 Error correctie

Er zal altijd ruis in de meting zitten. Om aan te geven hoe groot de afwijking waarschijnlijk per punt is moet de gemiddelde error bepaald worden per punt. Deze error kan bepaald worden. Hiervoor is om te beginnen de standaard deviatie  $\sigma_\rho$  van  $\rho$  (de pixelafstanden) voor nodig. Omdat het beeld niet altijd een gesloten omgeving ziet, moet de bepaling van  $\rho$  worden aangepast. De

Figure 8: Geanimeerd figuur met de werkelijke afstanden voor verschillende alpha's

pixelafstanden die niet worden gemeten worden standaard op oneindig gezet. Dit gooit de bepaling in de war, daarom worden deze waarden simpelweg uit  $\rho$  verwijderd. Daarnaast worden de punten die dichtbij gemeten worden niet standaard geset naar 0 maar naar de waarde van  $R_{min}$ .

Als  $\sigma_\rho$  bepaald is, kan de error worden bepaald per punt. Dit wordt gedaan met de volgende formule:  $\sigma_{d(i)} = \frac{h}{\alpha} * (1 + \tan(\frac{\rho_i}{alpha}))^2 * \sigma_\rho$ . In figuur is het resultaat te zien.

## 7 Verbeteringen

Zoals te zien is in figuur 9, is er ruis te zien in het beeld linksonder de middelpunt van de robot. Deze ruis wordt veroorzaakt door een sensor die aan de robot hangt, deze sensor zou korter gemaakt, of zelfs verwijderd kunnen worden voor betere resultaten.

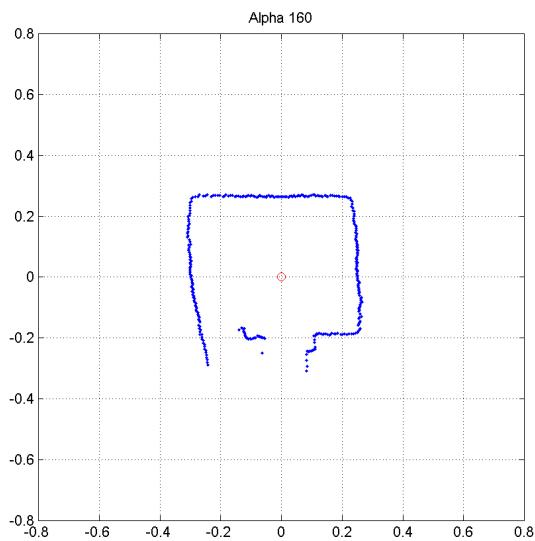


Figure 9: werkelijke afstand met alpha=160