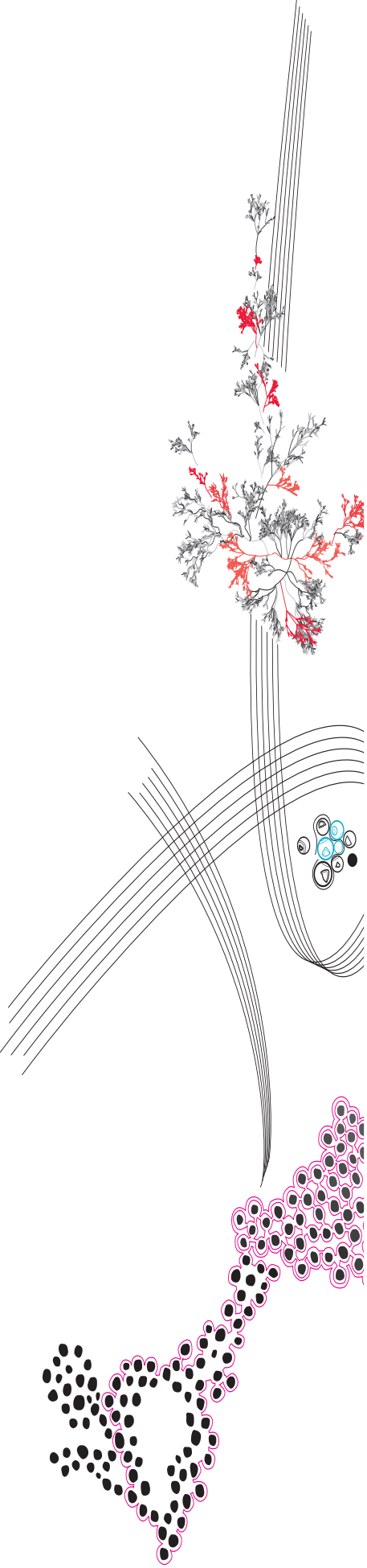


BACHELORSCRIPTIE

HAND-OOG COÖRDINATIE KALIBRATIE

Het bepalen van de rotatie en translatie componenten van de transformatie tussen de camera en de end effector

Puck Ooms

BIOMEDISCHE TECHNOLOGIE
DR.IR M. Vlutters

EXAMENCOMMISSIE
IR. A.H.G. Overbeek
DR.IR J.J. de Jong

DOCUMENT NUMMER
BE - 950

UNIVERSITY OF TWENTE.

Abstract

Nederlands

Dit onderzoek richt zich op het verkrijgen van een nauwkeurige hand-oog coördinatie kalibratie voor robots die worden ingezet in de gezondheidszorg. De kinematische keten, bestaande uit het robot base frame (ϕ_b), het nekframe (ϕ_n) en het cameraframe (ϕ_c), speelt hierbij een cruciale rol. Om de benodigde waarden binnen deze keten te verkrijgen, wordt een kalibratieproces uitgevoerd waarbij de positie en oriëntatie van de camera ten opzichte van de end effector (nekgewricht) berekend worden. Dit wordt bereikt door de robot twee bewegingen om niet-parallelle assen te laten uitvoeren, terwijl de camera gericht is op een visueel meetbaar punt in de ruimte. Hieruit ontstaan drie posities van zowel de camera en het nekgewricht ten opzichte van het robot basis frame. Het hand-oog coördinatie kalibratie probleem van de robot levert een homogene transformatievergelijking op van de vorm $AX = XB$. Hierin is 'A' de positie en oriëntatie van het nekgewricht ten opzichte van het robot basis frame, 'B' is de positie en oriëntatie van een ArUco marker ten opzichte van het camera frame en 'X' is de transformatie matrix tussen camera en de end effector. Het doel van het onderzoek is om uiteindelijk tot een homogene transformatie matrix 'X' te komen die de transformatie beschrijft tussen 'A' en 'B'.

De Daniilidis methode, zoals beschreven in de OpenCV 'Camera calibration and 3D reconstruction', wordt toegepast in dit onderzoek. Deze methode maakt gebruik van duale quaternionen om zowel rotatie als translatie simultaan te beschrijven. Singuliere waarden ontbinding wordt gebruikt om de best mogelijke schatting van transformatiematrix 'X' te verkrijgen. Uit het onderzoek blijkt dat een kleine afwijking in matrix 'X' tot grote doorreken fouten kan lijden. Dit kan verklaard worden door eventuele fouten bij het meten van de positie van camera en nekgewricht. De resultaten uit dit onderzoek kunnen niet gebruikt worden om de robot te implementeren binnen de gezondheidszorg. Dit komt door de afwijkingen die optreden bij het aansturen van de robotarm op basis van visuele input.

Engels

This research focuses on obtaining an accurate hand-eye coordination calibration for robots used in healthcare. The kinematic chain, consisting of the robot base frame (ϕ_b), the neck frame (ϕ_n) and the camera frame (ϕ_c), plays a crucial role in this. To obtain the necessary values within this chain, a calibration process is performed in which the position and orientation of the camera relative to the end effector (neck joint) are calculated. This is achieved by having the robot perform two movements around non-parallel axes, while the camera is pointed at a visually measurable point in space. This results in three positions of both the camera and the neck joint in relation to the robot base frame. The hand eye coordination calibration problem of the robot yields a homogeneous transformation equation of the form $AX = XB$. Here 'A' is the position and orientation of the neck joint relative to the robot base frame, 'B' is the position and orientation of an ArUco marker relative to the camera frame and 'X' is the transformation matrix between camera and the end effector. The aim of the research is to arrive at a homogeneous transformation matrix 'X' that describes the transformation between 'A' and 'B'.

The Daniilidis method, as described in the OpenCV 'Camera calibration and 3D reconstruction', is applied in this research. This method uses dual quaternions to describe both rotation and translation simultaneously. Singular value decomposition is used to obtain the best possible estimate of transformation matrix 'X'. The research shows that a small deviation in matrix 'X' can lead to major calculation errors. This can be explained by possible errors in measuring the position of the camera and neck joint. The results of this study cannot be used to implement the robot in healthcare. This is due to the deviations that occur when controlling the robot arm based on visual input.

Inhoudsopgave

1 Introductie

Mens-robot interactie is momenteel een snel groeiend vakgebied binnen de robotica [1]. Er zijn veel verschillende toepassingen die gebruik maken van mens-robot interactie. Denk hierbij aan toepassingen zoals schoonmaken, bewaking, zorg en in de industrie [2]. In deze scriptie wordt er gekeken vanuit het perspectief van de gezondheidzorg, waarbij robots ingezet kunnen worden om ondersteuning te bieden voor verplegend personeel en zorgontvangers. Hierbij moeten robots interactie kunnen hebben met hun omgeving en met andere mensen. Om te zorgen dat robots goed functioneren en kunnen communiceren met hun omgeving, is een vereiste dat hand-oog coördinatie nauwkeurig gebeurd. Dit is zowel belangrijk voor manipulatie van objecten en voor navigatie en het vermijden van obstakels. Hand-oog coördinatie is cruciaal voor robots, omdat het robots in staat stelt om visuele informatie van de omgeving te begrijpen en deze te vertalen naar doelgerichte acties.

Voor dit onderzoek wordt er gebruik gemaakt van een mensachtige robot (zie figuur 1). De robot heeft de grootte van een mens en bevat een camera ter hoogte van zijn ogen. Daarnaast heeft hij armen en vingers zoals een mens dat ook heeft. De robot heeft verschillende gewrichten die uiteindelijk ervoor zorgen dat de individuele onderdelen samenwerken om taken binnen de gezondheidszorg uit te voeren. Denk bijvoorbeeld aan patiënten van medicijnen voorzien, het assisteren bij fysiotherapie of het assisteren bij chirurgische ingrepen.

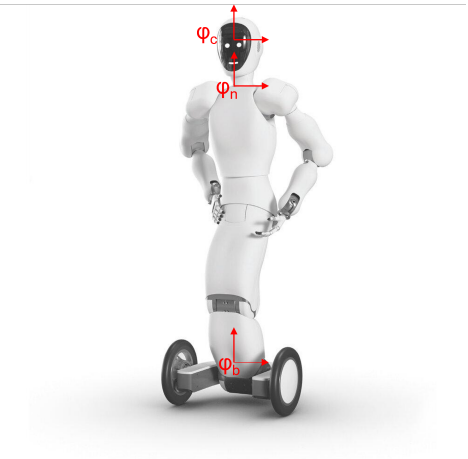
De robot kan door middel van de camera's beelden van zijn omgeving vastleggen. Door de beelden te analyseren, kan de robot objecten in zijn omgevingen detecteren en hun positie bepalen. Door de positie van objecten ten opzichte van de camera's te berekenen, kan de robot objecten lokaliseren en zijn arm nauwkeurig naar de gewenste positie bewegen. Ook kan de robot door middel van de bekende posities van de gewrichten ten opzichte van de robot basis de 3D-positie van de arm bepalen. Door de hoeken van de gewrichten en de lengtes van de segmenten te combineren met de kennis van de basispositie, kan de robot nauwkeurig de positie van zijn armen in de 3D-ruimte bepalen. Het is momenteel nog niet bekend of de translatievector en rotatiematrix van de berekende matrix gelijk is aan de gemeten translatievector en rotatiematrix. Dit probleem wordt ook wel het hand-oog kalibratie probleem genoemd en kan geschreven worden als een homogene matrixvergelijking van de vorm $AX = XB$. Hierin is 'X' de homogene transformatie matrix tussen de end effector en de robotcamera. 'A' is de transformatie van de end effector naar de robot basis en 'B' is de transformatie van target naar de camera. [3]. Om een nauwkeurige hand-oog coördinatie te verkrijgen moeten de positie en oriëntatie van het camera frame met betrekking tot het end effector frame bekend zijn. Dit is belangrijk om berekeningen en taken met de robot nauwkeurig uit te kunnen voeren. Hierin kan de end effector van alles zijn, in dit geval wordt er vanuit gegaan dat de end effector het nekgewricht is. Zo kan de positie en oriëntatie van de camera ten opzichte van het nekgewricht bepaald worden.

Het hand-oog coördinatie kalibratie experiment kan uitgevoerd worden door de robot naar een visueel meetbaar punt in de ruimte (target) te laten kijken. Vervolgens wordt het hoofd van de robot onder een andere hoek gezet waarna de romp van de robot gedraaid wordt. Dit resulteert in een andere oriëntatie en positie van zowel het nekgewricht als de camera. Hierdoor wordt er data verzameld over homogene matrix 'A' en de homogene matrix 'B' om zo uiteindelijk met een simultane oplossing tot de homogene transformatie matrix 'X' te komen.

Figuur 1: De Halodi EVEr3 [4].

2 Probleem-beschrijving

De waardes die niet bekend zijn bij de robot zijn de translaties $(\boldsymbol{t})$ en de rotaties (R) van de camera ten opzichte van het nekgewricht. Dit betekent dat de robot niet precies weet hoe de positie van de camera gerelateerd is aan de positie van het nekgewricht. Doordat deze waardes niet bekend zijn, is de kinematische keten van de robot niet compleet. De kinematische keten in dit onderzoek bestaat uit het robot base frame (ϕ_b), het nekframe (ϕ_n) en het cameraframe (ϕ_c) zoals te zien in figuur 2.

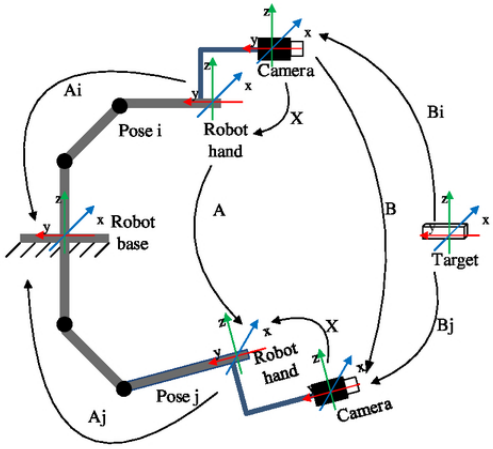


Figuur 2: Kinematische keten van de robot.

Om deze keten compleet te maken, moeten de homogene transformatie matrix tussen het nekgewricht en camera bepaald worden. Hierdoor kan de robot nauwkeuriger worden aangestuurd om bijvoorbeeld een taak van een verpleegkundige uit te voeren en kan de automatisering van de robot via de camerabeelden verbeterd worden. Het hand-oog coördinatie kalibratie experiment zoals in dit onderzoek kan toegepast worden op elke robot in een soort gelijke situatie. Het doel van dit onderzoek is dan ook om de transformatie tussen camera en end effector te bepalen, ook wel hand-oog coördinatie kalibratie van een robot genoemd.

Er zijn verschillende methodes om het robot-sensor kalibratie probleem op te lossen. Er zijn twee vormen van het hand-oog coördinatie kalibratie probleem, namelijk $AX = XB$ en $AX = YB$ [3, 5]. Het probleem wat hier besproken wordt, is een $AX = XB$ probleem. Het verschil tussen het $AX = XB$ en het $AX = YB$ probleem ligt in het feit dat in het eerste geval de camera en de end effector tegelijkertijd bewegen, terwijl ze in het tweede geval onafhankelijk van elkaar bewegen. De vergelijking $AX = XB$ impliceert dat er een consistente transformatie moet zijn tussen het coördinatensysteem van de robotarm en het coördinatensysteem van de camera, zodat waargenomen objecten in de omgeving door zowel de robotarm als de camera op dezelfde manier worden geïnterpreteerd. Hierdoor moet AX gelijk zijn aan XB.

In dit probleem gaat het om de nek als end effector die tegelijkertijd beweegt met de camera omdat het één geheel is. Het probleem in dit onderzoek kan gezien worden als figuur 3 waarbij de camera en end effector op twee verschillende posities staan. 'A_i' beschrijft de transformatie van de end effector naar de robot basis op positie 'i' en 'A_j' beschrijft de transformatie van de end effector naar de robot basis op positie 'j'. 'B_i' en 'B_j' beschrijven de transformatie van een visueel meetbaar punt in de ruimte naar de camera op positie 'i' en 'j' [3, 5, 6]. Hierin is 'A' gelijk aan $A_j^{-1}A_i$ en 'B' gelijk aan $B_jB_i^{-1}$. 'X' is onbekend en is de transformatie matrix tussen 'A' en 'B', ook wel de transformatie van camera naar end effector. De homogene matrix 'X' is de gewenste uitkomst van het onderzoek.



Figuur 3: Transformatie tussen verschillende frames bij positie 1 en positie 2 [6].

3 Theorie

Veel methodes in hand-oog kalibratie van een robot maken gebruik van quaternionen, daarom is er een basiskennis van quaternionen nodig. Er wordt dieper in gegaan op eerdere onderzoeken naar de hand-oog coördinatie kalibratie van een robot. Uiteindelijk wordt de theorie achter de methode van dit onderzoek uitgelegd. Verder wordt de ArUco marker geïntroduceerd, die gebruikt wordt in het onderzoek.

3.1 Quaternionen

Quaternionen [7] zijn een uitbreiding van complexe getallen naar vier dimensies en worden binnen de robotica voornamelijk gebruikt om rotaties in 3D-ruimte te representeren. In plaats van alleen lengte en richting, zoals bij vectoren, bevatten quaternionen ook informatie over rotatie-as en rotatiehoek. Ze bieden een compacte en efficiënte manier om de rotatie weer te geven. Een quaternion is gedefinieerd als: $\alpha = a_0 + a_1 \cdot \boldsymbol{i} + a_2 \cdot \boldsymbol{j} + a_3 \cdot \boldsymbol{k}$. Waarbij '$a_i$' met i=0, 1, 2, 3 de reële paramaters zijn en '**i**','**j**' en '**k**' zijn de drieruimtelijke orthogonale eenheidsvectoren. Quaternionen kunnen ook weergegeven worden als een lineaire combinatie van een scalaire en een ruimtevector: $\alpha = a_0 + \mathbf{a}$. Wanneer de geconjugeerde van een quaternion bepaald moet worden, dan moet het vector deel negatief gemaakt worden $\alpha^* = a_0 - \mathbf{a}$. Wanneer het eerste deel nul is dan wordt α een vector quaternion en als het tweede deel van de vergelijking nul is dan is het een scalair quaternion. Het is gebruikelijk om quaternionen in matrix vorm neer te zetten om zo vergelijkingen te versimpelen. Dit kan dan als volgt weergegeven worden: $\alpha = [a_0, a_1, a_2, a_3]^T = [a_0, \mathbf{a}^T]^T$.

Naast quaternionen worden ook duale quaternionen gebruikt. Duale quaternionen vertegenwoordigen zowel translatie als rotatie in tegenstelling tot quaternionen die alleen een rotatie representatie gebruiken [8]. Het verschil met normale quaternionen is dat een duale quaternion bestaat uit een duaal getal en een duale vector, terwijl een quaternion bestaat uit een reëel nummer en vector. Een duaal getal bestaat uit een reëel deel en een duaal deel dat geschreven kan worden als $a + b\epsilon$. Hierin is ϵ de duale eenheid en zijn a en b reële getallen. Een duale vector is een vector waarbij elk element een duaal getal is [9]. Het gebruik van quaternionen en duale quaternionen kan efficiënter zijn dan het gebruik van rotatiematrixen. Dit komt doordat met het gebruik van quaternionen en duale quaternionen de vereiste vermenigvuldigen en optellen/aftrekken wordt verminderd. Door de eigenschappen van quaternionen en duale quaternionen worden ze vaak gebruikt bij de hand-oog kalibratie om zo rotatie en translatie te beschrijven.

3.2 Eerdere onderzoeken

Het hand-oog kalibratieprobleem $AX = XB$ kan onderverdeeld worden in drie oplossingen, namelijk 'scheidbare oplossing', 'simultane oplossing' en 'iteratieve oplossing'. De drie oplossingen zullen kort besproken worden en aan de hand daarvan zal een beslissing gemaakt worden over de methode die voor dit experiment gebruikt gaat worden [3].

3.2.1 Scheidbare oplossing

Bij deze oplossing wordt het probleem verdeeld in een oriëntatie component $R_A R_X = R_X R_B$ en een positie component $R_A t_X + t_A = R_X t_B + t_X$ [3, 10]. Allereerst moet de rotatie, R_X opgelost worden en vervolgens kan hiermee de translate, t_X bepaald worden. De scheidbare oplossing kan weer onderverdeeld worden in verschillende methodes, afhankelijk van hoe de rotatie component R_X bepaald wordt [3]. Zo lossen Shiu en Ahmad [10] dit op door de rotatiematrix te berekenen met behulp van de hoek-as formulering van rotatie. Hierbij wordt vergelijking (1) gebruikt om R_X te bepalen [3]:

$$R_x = Rot(k_{A_i}, \beta_i) R_{X_{P_i}} \tag{1}$$

Hierin is:

Rot = Rotatiematrix
k_{A_i} = rotatie-as van 'A'
k_{B_i} = rotatie-as van 'B'
β_i = hoek
$R_{X_{P_i}}$ = Rot($\mathbf{v}, w$)
$\mathbf{v} = k_{B_i} \times k_{A_i}$
$w = atan2(|k_{B_i} \times k_{A_i}|, k_{B_i} \cdot k_{A_i})$

Tsai en Lenz [11] bepalen R_X door ook gebruik te maken van de hoek-as formulering. Echter maken Tsai en Lenz gebruik van een lineair systeem met vaste afmetingen, terwijl de grootte van het lineaire systeem van Shiu en Ahmad elke keer verdubbelt wanneer er een nieuw frame wordt toegevoegd aan het systeem. De methode van Tsai en Lenz is eenvoudiger en rekenkundig efficiënter [3]. Park en Martin [12] lossen het $AX = XB$ probleem op met behulp van de kleinstekwadratenmethode door gebruik te maken van de Lie-groep theorie. Hierbij worden 'A' en 'B' beschouwd als elementen van een Lie-groep, in dit geval de Euclidische groep die alle starre transformaties in de driedimensionale ruimte vertegenwoordigt. Hierdoor kan het probleem herschreven worden als een lineair systeem. Dit stelt hen in staat om de transformatiematrix 'X' te schatten met behulp van de kleinstekwadratenmethode. Chou en Kamel [13] hebben gebruik gemaakt van quaternionen om $R_A R_X = R_X R_B$ naar een lineair systeem te transformeren. Vervolgens losten ze de quaternion representatie van R_X op door middel van singuliere waarden ontbinding [14].

Het nadeel van deze scheidbare oplossing methode is dat de fout van de rotatie onvermijdelijk wordt overgedragen op het translatiegedeelte, omdat de rotatiewaarde gebruikt wordt om de translatie te berekenen [3, 15]. Om dit probleem op te lossen zijn onderzoekers tot de simultane en iteratieve oplossing gekomen [3].

3.2.2 Simultane oplossing

Deze methode geeft de oplossing voor rotatie en translatie componenten tegelijkertijd. Hierdoor wordt het doorreken probleem vermeden. Ook de simultane methode wordt door verschillende onderzoekers op verschillende manieren uitgevoerd. Lu en Chou [16, 17] lossen het hand-oog kalibratie probleem op door middel van de acht-dimensionale ruimte formulering die gebruikt maakt van quaternionen. De acht-dimensionale ruimte formulering is een wiskundige representatie om complexe bewegingen in de ruimte te beschrijven. Chen [18] lost het probleem op door gebruikt te maken van de schroeftheorie. Hij maakt gebruikt van de eigenschap dat de richting van de rotatie-as en de translatie van de schroefbeweging parallel zijn [14]. Dit is een belangrijke eigenschap, omdat het een verband legt tussen de rotatie- en translatiecomponenten van de beweging. Daniilidis [9, 17] maakt gebruik van dubbele quaternionen, om zo met een wiskundige benadering van de schroeftheorie de onbekende rotatie en translatie te schatten door middel van singuliere waarden ontbinding.

De simultane oplossing kan echter wel zorgen voor verschillende resultaten afhankelijk van de schaling van de positionele component van de camera en het nekgewricht. Dit betekent dat kleinere veranderingen in de positiegegevens kunnen leiden tot onnauwkeurigheden in de kalibratieresultaten. Hierdoor is de iteratieve oplossing bedacht.

3.2.3 Iteratieve oplossing

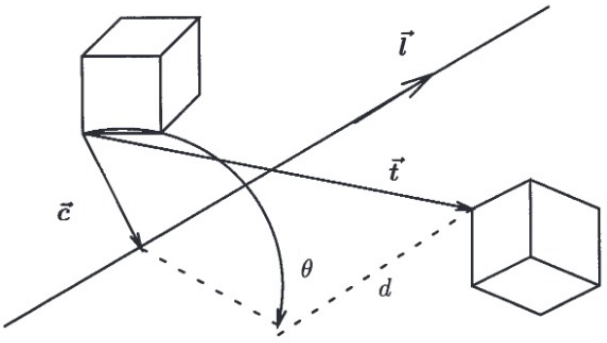
Naast de gescheiden en de simultane oplossing, is er ook nog een iteratieve oplossing. Deze methode lost de rotatie en translatie componenten op door middel van herhaling met behulp van optimalisatietechnieken zoals de Levenberg-Marquardt-algoritme en de gradiëntdaling [14]. Zhuang en Shiu maken gebruik van een één-stap iteratieve methode door middel van de levenberg-Marquardt algoritme. Deze methode lost de rotatie en translatie componenten tegelijkertijd op en is onafhankelijk van de robot oriëntatie [3].

De iteratieve methode vermijdt de error die ontstaat bij het apart berekenen van de rotatie en translatie componenten. Deze methode kan als complex ervaren worden vanwege verschillende stappen en vereiste wiskunde kennis. Wanneer het aantal vergelijkingen zal toenemen zal het verschil in uitkomst tussen de gescheiden, simultane en iteratieve oplossing steeds kleiner worden [3]. Door de meer complexe benadering van de iteratieve oplossing en het feit dat het verschil in uitkomst op gegeven moment verwaarloosbaar wordt, is er gekozen om voor de simultane oplossing te gaan.

3.3 Methode van Daniilidis

De methode die in deze scriptie wordt gebruikt om het oog-hand kalibratie probleem $AX = XB$ op te lossen is een algebraïsche uitdrukking die de eigenschappen van een schroef beschrijft, ook wel de eenheid duale quaternion. Het is een wiskundige representatie die wordt gebruikt om de beweging en rotatie van een schroef in de ruimte te beschrijven [9]. De methode van Daniilidis valt onder de simultane oplossingsmethode doordat de duale quaternionen zowel rotatie als translatie beschrijven [3]. Een tekortkoming van alleen gebruik te maken van quaternionen is dan ook dat een quaternion alleen de rotatiecomponent van een schroeftransformatie kan representeren en niet ook de translatiecomponent. Verder maakt de duale quaternionen methode van Daniilidis ook gebruik van singuliere waarden ontbinding om homogene transformatie matrix 'X' te schatten [9].

Zoals al eerder benoemd kan het hand-oog kalibratie probleem in de vorm van rotatie (**R**) en translatie ($\vec{\mathbf{t}}$) geschreven worden. Volgens Chasles' theorie kan een verplaatsing van een starlichaam ook gezien worden als een schroefbeweging, waarbij er rotatie om een unieke as plaatsvind en translatie om diezelfde as [19]. Omdat de schroefas een lijn in de ruimte is, is deze afhankelijk van vier parameters. De vier parameter, de rotatiehoek (θ) en de translatie langs de d-as (spoed-as) vormen de zes vrijheidsgraden van een starre transformatie [9, 19].



Figuur 4: De geometrie van een schroef [9].

Duale quaternionen kunnen gebruikt worden om een schroefbeweging te representeren, daarom wordt er ook dieper in gegaan op een schroefbeweging. De beweging ($\vec{\mathbf{l}}$) is parallel aan de rotatie-as zoals te zien in figuur 4. Verder is de spoed (d) de verplaatsing langs de rotatie-as van de schroefbeweging en kan berekend worden door $\vec{\mathbf{t}} \cdot \vec{\mathbf{l}}$. De hoek θ is voor zowel rotatie/translatie beschrijving als schroefbeschrijving hetzelfde. Het herstellen van het moment ($\vec{\mathbf{m}}$) gebeurd door middel van een punt op de schroefas ($\vec{\mathbf{c}}$), die de projectie van de oorsprong representeert. Dit punt en de schroefas zijn niet gedefinieerd als de hoek θ gelijk aan 0 of 180 graden is. De translatie die vervolgens ontstaat is d $\vec{\mathbf{l}}$ +(I-R) $\vec{\mathbf{t}}$. Vervolgens kan met behulp van de Rodrigues formule $\vec{\mathbf{c}}$ bepaald worden [9]:

$$\vec{c} = \frac{1}{2}(\vec{t} - (\vec{t} \cdot \vec{l})\vec{l}) + \cot\frac{\theta}{2}(\vec{l} \times \vec{t}) \tag{2}$$

De vector van het moment kan geschreven worden als [9]:

$$\vec{m} = \vec{c} \times \vec{l} = \frac{1}{2}(\vec{t} \times \vec{l} + \vec{l} \times (\vec{t} \times \vec{l}) cot\frac{\theta}{2} \tag{3}$$

Vervolgens kan er met de gegeven schroefparameters(θ, d, $\vec{\mathbf{l}}$, $\vec{\mathbf{m}}$) de duale quaternion $\breve{\mathbf{q}}$ berekend worden. Nu er laten zien is hoe de schroefbeweging in elkaar zit en wat de wiskundige representatie er van is, kan er worden overgaan naar de beschrijving van de duale quaternionen voor het hand-oog kalibratie probleem [9].

De samenvoeging van twee schroeven kan geschreven worden als een vermenigvuldiging van twee duale quaternionen. De schroef van een camerabeweging wordt aangegeven met $\breve{\mathbf{a}}$ en de schroef van de end effector beweging wordt aangegeven met $\breve{\mathbf{b}}$. De transformatie tussen deze twee is onbekend en is de uitkomst van deze scriptie en wordt genoteerd met $\breve{\mathbf{q}}$. De schroefsamenvoeging geeft een compacte formule voor de hand-oog relatie [9]:

$$\breve{\mathbf{a}} = \breve{\mathbf{q}}\breve{\mathbf{b}}\vec{\breve{\mathbf{q}}} \tag{4}$$

Het scalair deel (Sc) van de duale quaternion $\breve{\mathbf{a}}$ is ($\breve{\mathbf{a}}$ + $\vec{\breve{\mathbf{a}}}$) en daarom kunnen we vergelijking (4) omschrijven tot de volgende vergelijking [9]:

$$\begin{aligned} Sc(\breve{\mathbf{a}}) = \frac{1}{2}(\breve{\mathbf{a}} + \vec{\breve{\mathbf{a}}}) = \frac{1}{2}(\breve{\mathbf{q}}\breve{\mathbf{b}}\vec{\breve{\mathbf{q}}} + \breve{\mathbf{q}}\vec{\breve{\mathbf{b}}}\vec{\breve{\mathbf{q}}}) = \frac{1}{2}\breve{\mathbf{q}}(\breve{\mathbf{b}} + \vec{\breve{\mathbf{b}}})\vec{\breve{\mathbf{q}}} \\ = \breve{\mathbf{q}}Sc(\breve{\mathbf{b}})\vec{\breve{\mathbf{q}}} = Sc(\breve{\mathbf{b}})\breve{\mathbf{q}}\vec{\breve{\mathbf{q}}} = Sc(\breve{\mathbf{b}}) \end{aligned} \tag{5}$$

Dit kan zo gedefinieerd worden omdat $\breve{\mathbf{q}}\vec{\breve{\mathbf{q}}}$=1 de eenheid quaternion is. Verder zijn de scalaire delen gelijk aan de cosinus van duale hoeken [9]:

$$\begin{aligned} \cos\left(\frac{\theta_a}{2}\right) &= \cos\left(\frac{\theta_b}{2}\right) \\ d_a \sin\left(\frac{\theta_a}{2}\right) &= d_b \sin\left(\frac{\theta_b}{2}\right) \end{aligned} \tag{6}$$

Hierdoor zijn de hoek en de spoed van de end effector schroef gelijk aan de hoek en spoed van de camera-schroef, ook wanneer de coördinatentransformatie plaatsvindt [9]. De compacte vergelijking voor de schroef-samenvoeging bevat vier duale vergelijkingen. Omdat de scalaire delen gelijk aan elkaar zijn, maakt alleen het vector deel uit in de vergelijking (4) om aan bekende $\breve{\mathbf{q}}$ te komen [9]:

$$sin\frac{\breve{\theta}_a}{2}(0, \vec{\mathbf{a}}) = \breve{\mathbf{q}}(0, sin\frac{\breve{\theta}_b}{2}\vec{\mathbf{q}})\vec{\mathbf{q}} = sin\frac{\breve{\theta}_b}{2}\breve{\mathbf{q}}(0, \vec{\mathbf{q}})\vec{\mathbf{q}} \tag{7}$$

Wanneer de hoeken $\theta_{a,b}$ niet gelijk 0 of 360 graden zijn, dan kan vergelijking (7) versimpelt worden tot alleen de beweging van de lijnen van de schroefassen [9]:

$$(0, \vec{\mathbf{a}}) = \breve{\mathbf{q}}(0, \vec{\mathbf{q}})\vec{\mathbf{q}} \tag{8}$$

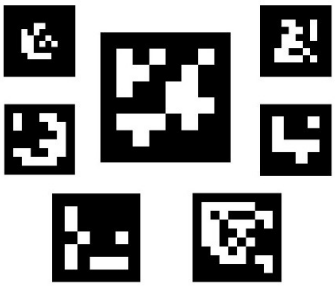
Hierdoor kan er geconcludeerd worden dat de hand-oog coördinatie kalibratie onafhankelijk is van de spoed en de hoek van zowel camerabeweging als end effector beweging. Verder kan het hand-oog kalibratie probleem ook gezien worden als het 3D bewegingsschattingsprobleem op basis van 3D-lijnovereenkomsten. Hierin stellen de lijnen de schroefassen van zowel de end effectoren en de camera's voor [9].

Voor de Daniilidis methode zijn twee strikte eisen. Er zijn minimaal drie verschillende end-effector posities nodig en er zijn dus minimaal twee bewegingen met niet-parallelle rotatie-assen nodig [9, 20]. Dit betekent dat de robot op minimaal drie posities moet worden gezet om aan de restricties van de methode te voldoen.

Verder maakt de Daniilidis methode gebruik van singuliere waarde ontbinding (SVD) om de beste schatting van homogene transformatie matrix 'X' tussen end effector en camera te verkrijgen. Door gebruik te maken van SVD kan de transformatie tussen robot end effector en de camera nauwkeurig worden geschat, terwijl er rekening wordt gehouden met de effecten van ruis en meetfouten op de schatting van de transformatie matrix 'X' [9, 21].

3.4 ArUco marker

In dit onderzoek wordt er gebruik gemaakt van een visueel meetbaar punt in de ruimte, namelijk een ArUco marker (zie figuur 5). Dit is een vierkante marker die bestaat uit een zwarte rand en in de binnenkant zit de binaire matrix die de identificatie bepaalt. De binaire codering maakt identificatie mogelijk en de zwarte rand zorgt ervoor dat de marker makkelijker te detecteren is. De ArUco marker die in dit onderzoek wordt gebruikt, heeft een binaire matrix van 5x5, dit komt overeen met 25 bits. De volgorde en combinatie van de bits zorgen voor een unieke identificatiecode van de marker. ArUco markers worden gebruikt voor positiebepaling in computer vision-toepassingen. De unieke identificatie code wordt gedetecteerd door de camera van de robot en wordt gebruikt om de positie en oriëntatie van de marker in de 3D-ruimte te bepalen [22]. Door in dit onderzoek de marker te gebruiken kan de robot zijn eigen positie en oriëntatie in de ruimte bepalen ten opzichte van de marker.



Figuur 5: ArUco markers [22].

4 Methode

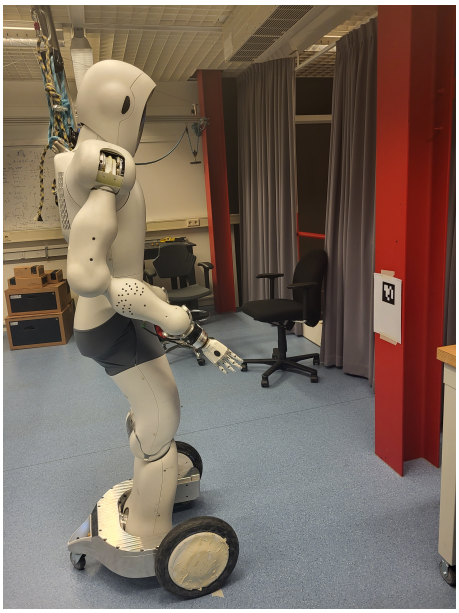
Om tot de uiteindelijke homogene transformatie matrix tussen de camera en de end effector te komen is er een experiment gedaan met behulp van een ArUco marker. Nadat het experiment is uitgevoerd, zijn de data verwerkt om vervolgens bij de resultaten tot een eindantwoord te komen. Om bij het eindantwoord te komen wordt er gebruik gemaakt van een script uit de OpenCV [20]. Deze code maakt gebruik van de simultane oplossingsmethode van Daniilidis [9].

4.1 Experimentele Opstelling

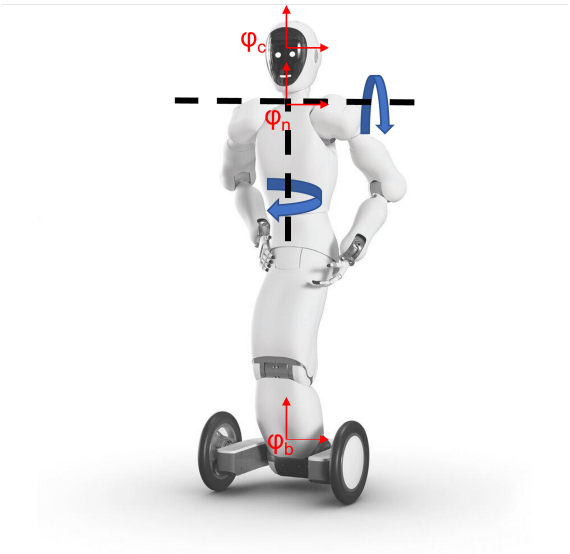
De experimentele opstelling bestond uit de Halodi EVEr3, een ArUco marker en een computer waar de robot en de code op werkt. Ook wordt er gebruik gemaakt van Python 3.10.11 en OpenCV 4.7.0.

De robot is zo geplaats dat de robot met verschillende nek en romp posities nog steeds data binnen kan halen van de ArUco marker (zie figuur 6). De robot staat recht tegenover de ArUco marker die geplaatst is op de rode balk. Het controleren of de camera data correct binnenkomt, wordt gedaan door controleren of bij alle hoofd en romp posities de camera waardes verschijnen op de computer display.

In figuur 7 is de kinematische keten weergegeven met de twee rotatie-assen. Het hoofd van de robot wordt op drie verschillende posities ten opzichte van het robot basis frame gezet. Dit wordt gedaan met een beweging om de horizontale rotatie-as in de nek en een beweging om de verticale rotatie-as in het lichaam. In het experiment wordt het hoofd eerst van beneden naar boven gezet. Om te voldoen aan de restricties voor het experiment is ook de romp bewogen. Dit is gedaan zodat er twee bewegingen om niet-parallelle assen plaatsvinden. De romp van de robot is per experiment één kant omgedraaid langs de verticale as. Om de beweging zo constant mogelijk uit te voeren, is het hoofd met behulp van de robot code op twee verschillende posities gezet. Het draaien van de romp is gedaan door de romp zelf te draaien en dit is een soepele beweging.



Figuur 6: Experimentele opstelling.



Figuur 7: Kinematische keten + rotatie-assen.

Bij de eerste twee experimenten is de romp rechtsom de verticale as gedraaid en het hoofd van beneden naar boven bewogen. Dit resulteert in de volgende poses voor meting 1 en 2:

	Nek hoek	**Romp hoek**
Pose 1 (rood in figuur 8)	Negatief	Centraal
Pose 2 (blauw in figuur 8)	Positief	Centraal
Pose 3 (groen in figuur 8)	Positief	Rechtsom

Tabel 1: De drie verschillende poses van meting 1 en 2

Bij meting 3 is de romp linksom de verticale as gedraaid en het hoofd van boven naar beneden bewogen, dit resulteert in de volgende poses:

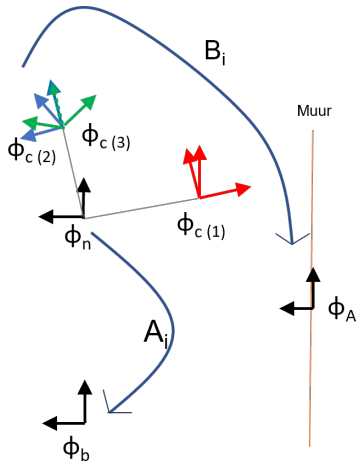
	Nek hoek	**Romp hoek**
Pose 1 (rood in figuur 8)	Positief	Centraal
Pose 2 (blauw in figuur 8)	Negatief	Centraal
Pose 3 (groen in figuur 8)	Negatief	Linksom

Tabel 2: De drie verschillende poses van meting 3

Om de opstelling nog duidelijker weer te geven is er een schematische weergave van het experiment in meting één en meting twee gemaakt (zie figuur 8). De schematische weergave bestaat uit drie verschillende frames, namelijk:

- ϕ_c: Het camera frame op drie verschillende posities
- ϕ_b: Het robot bais frame
- ϕ_A: Het ArUco marker frame

In pose 1 staat de romp van de robot recht tegenover de ArUco marker en het hoofd is naar beneden gekanteld, dus met de kin op de borst. Pose 2, is waarbij het hoofd van de robot omhoog gekanteld is om de horizontale as. Pose 3, is waarbij het hoofd van de robot omhoog gekanteld is en waarbij de romp van de robot rechtsom de verticale as geroteerd is. Zoals te zien is in figuur 8, resulteert de romp rotatie in rotatie van camera frame 3 ten opzichte van camera frame 2.



Figuur 8: Schematische weergave opstelling.

Er is te zien dat het frame van de end effector (in dit geval de nek) precies in nekgewricht gedefinieerd is. Verder is het robot base frame getekend onder het nek frame. Waar dit frame zich daadwerkelijk plaatsvindt in de robot is niet bekend en deze waardes zijn ook niet nodig voor de verdere berekening. Het ArUco frame is aan de rechterkant getekend en het midden van de ArUco marker bevindt zich in de oorsprong van het coördinaten stelsel. De transformatie 'A_i' is gedefinieerd als de transformatie van end effector (nekgewricht) naar de robot basis en de transformatie 'B_i' is gedefinieerd als de transformatie van de ArUco marker (target) naar het camera frame.

4.2 Data verwerken

Er wordt ingegaan op hoe de data verzamelt is en hoe vervolgens de verschillende posities van elkaar onderscheiden zijn. Uiteindelijk wordt de input van de Daniilidis code uit de OpenCV verkregen door de robot en camera data te verwerken. Vervolgens worden deze input elementen gebruikt om tot de uitkomst van dit onderzoek te komen.

4.2.1 Algemeen

Bij het experiment worden geen timestamps gegeven. De data van de robot en de data van de camera kan aan elkaar gekoppeld worden, door het hoofd en de romp op een bepaalde positie te zetten. Doordat alle waardes constant blijven gedurende een pose, kunnen de verschillende posities herkend worden bij zowel robot als camera data. Er zijn drie verschillende posities van de nek ten aanzien van de robot basis. Deze worden weergegeven met drie verschillende gebieden. De gebieden zijn opgedeeld aan de hand van tabel 1 en 2. Bij zowel robot data als camera data wordt er per gebied voor elke losse variabelen een gemiddelde genomen. Uiteindelijk ontstaan er dus drie elementen, die elk voor een andere nekpositie staan. Verder is het experiment drie keer uitgevoerd, die onderling met elkaar vergeleken kunnen worden.

4.2.2 Output robot

De output van de robot geeft het nekgewricht weer in de vorm van vier quaternionen en x,y en z. De informatie over de quaternionen en x,y en z wordt om de 1/500 seconden (500Hz) verkregen. Er kan daarom een plot gemaakt worden met alle losse variabelen uitgezet tegen de tijd. In deze plot kunnen de verschillende posities onderscheiden worden, aangezien er een verandering van de individuele waardes ontstaat.

De resulterende robot output bevat drie elementen, elk bestaande uit de gemiddelde waarde van de vier quaternionen (qx, qy, qz en qw) en x,y en z voor de verschillende hoofdposities. In bijlage A is het script te zien om uiteindelijk de drie elementen te verkrijgen. Dit moet vervolgens omgezet worden naar een rotatie en translatie beschrijving aangezien dat de input voor de Daniilidis methode is. De vier quaternionen kan je omzetten naar een rotatiematrix [23]. Dit wordt gedaan door middel van de volgende vergelijking:

$$R = \begin{bmatrix} 2(q_x^2 + q_y^2) - 1 & 2(q_y * q_z - q_x * q_w) & 2(q_y * q_w + q_x * q_z) \\ 2(q_y * q_z + q_x * q_w) & 2(q_x^2 + q_z^2) - 1 & 2(q_z * q_w - q_x * q_y) \\ 2(q_y * q_w - q_x * q_z) & 2(q_z * q_w + q_x * q_y) & 2(q_x^2 + q_w^2) - 1 \end{bmatrix}$$

De translatievector kan direct verkregen worden uit x,y en z. In bijlage B is het script te zien waarmee de quaternionen worden omgezet naar een 3x3 rotatiematrix en x,y en z naar een 3x1 translatievector.

4.2.3 Output camera

De output van de camera is een kolom met verschillende waardes met onder andere de translatie en rotatie componenten. De Daniilidis methode maakt gebruik van een rotatiematrixen en translatievectoren. Dit betekent dat deze componenten uit de data gehaald moeten worden en opgeslagen moeten worden in een lijst. Dit wordt gedaan met behulp van het script in bijlage C. De informatie over de translatie- en rotatievector wordt om de 1/24 seconden (24 Hz) verzameld. Van de camera output moet eveneens ook een plot gemaakt worden met alle individuele variabelen tegen de tijd.

De drie elementen die ontstaan bevatten de gemiddelde translatie- en rotatievector per positie van het hoofd. Echter is de input van de Daniilidis methode een rotatiematrix en een translatievector. Dit betekent dat door middel van een Rodrigues functie [20] uit de OpenCV documentatie de rotatievector omgezet kan worden naar een rotatiematrix. Dit gebeurd met behulp van de volgende vergelijking:

$$R = \cos(\theta) I + (1 - \cos(\theta)) r r^T + \sin(\theta) \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}$$

Hierin is:

$R =$ De gewenste rotatiematrix
$\theta =$ rotatiehoek in radialen
$I =$ de identiteitsmatrix
$r =$ de genormaliseerde rotatievector
$r_x, r_y, r_z =$ De componenten van de rotatievector
$rr^T =$ Het uitproduct van 'r' en zijn getransponeerde

Wanneer dit gedaan is, zijn er drie elementen die gebruikt kunnen worden als input in de Daniilidis methode.

4.2.4 Controleren van robot en camera data

Om te controleren of de data van de robot output en de camera output correct is, is er gekeken naar de verschillende plots. Hierbij is er nagegaan of positie verandering van de nek de verwachte verandering in de individuele elementen geeft. Nu de robot en camera data verwerkt is en de input lijsten van de Daniilidis methode verkregen zijn, kunnen de rotatiematrixen en translatievectoren van zowel robot output als camera output gebruikt worden in de methode van Daniilidis. Hiervoor worden de functies uit de OpenCV "Camera Calibration and 3D Reconstruction"[20] gebruikt. De output van deze functie zijn de rotatiematrix en de translatievector die de transformatie van camera naar nekgewricht representeren. De rotatiematrix en de translatie matrix kunnen vervolgens samengevoegd worden tot homogene transformatie matrix 'X'. Het script met de functies uit de OpenCV en hoe het eindresultaat verkregen is, is te zien in bijlage D.

4.2.5 Uitkomst matrix 'X'

Om te controleren of 'X' correct en nauwkeurig is, kunnen de drie verschillende uitkomsten van de drie metingen met elkaar vergeleken worden. Het beste resultaat zou zijn, wanneer de drie matrixen dezelfde uitkomst geven. Om na te gaan of de uitkomst correct is, is het een vereiste dat de waardes van de verschillende schattingen van 'X' bij elkaar in de buurt moeten liggen. De drie verschillende experimenten zijn tevens met dezelfde robot en dezelfde camera gedaan, waar uiteindelijk één homogene transformatie matrix 'X' voor opgesteld moet worden. Echter hebben we te maken met zowel meetfouten als rekenfouten en is het logisch om te verwachten dat de drie verschillende uitkomsten van matrix 'X' niet exact aan elkaar gelijk zijn.

Er kan ook gekeken worden naar de standaarddeviatie (σ) van de verschillende translatiecomponenten tussen de drie metingen. De standaarddeviatie zegt iets over hoe constant en nauwkeurig een onderzoek is uitgevoerd. De standaarddeviatie is laag wanneer resultaten nagenoeg hetzelfde zijn en de standaarddeviatie is hoog wanneer de waardes veel verschillen. De standaarddeviatie kan met behulp van de volgende formule berekend worden:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{9}$$

waarbij σ de standaarddeviatie is, N het aantal waardes, x_i de individuele waardes en μ het gemiddelde van de waardes.

Verder kan er ook gecontroleerd worden of matrix 'X' een correcte transformatie matrix is door matrix 'A' of matrix 'B' te berekenen met de geschatte matrix 'X' en de gemeten matrix 'A' en 'B'. Het hand-oog kalibratie probleem $AX = XB$ kan ook geschreven worden als:

$$A^{(1)}_{calc} = XB^{(1)}X^{-1} \tag{10}$$

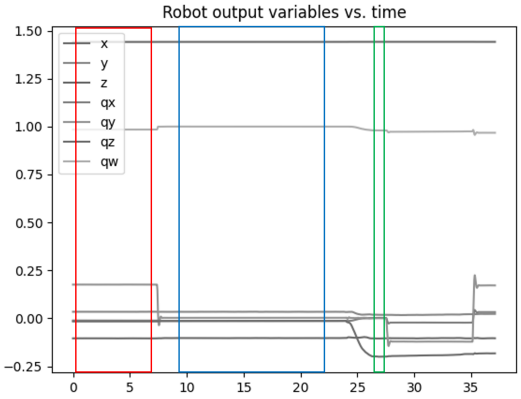
Hiermee wordt er gecontroleerd of matrix 'X' goed geschat is. Met de geschatte matrix 'X' en met gemeten transformatie matrix van ArUco marker naar camera op positie 1 ($B^{(1)}$) kan vervolgens de transformatie matrix van nekgewricht naar robot basis ($A^{(1)}$) berekend worden. Aangezien de methode van Daniilidis gebruikt maakt van een schatting is het aannemelijk dat er onnauwkeurigheden in het berekenen van transformatie matrix 'X' zitten. Dit zorgt ervoor dat de berekende waarde van de transformatiematrix van target naar camera niet exact gelijk is aan de gemeten matrix. Verder hebben onnauwkeurigheden in het meten van matrix 'A' en 'B' ook invloed op de nauwkeurigheid van de schatting van matrix 'X'. Er kan gekeken worden naar het verschil in de translatievector. Hieruit kan beredeneerd worden wat voor afwijking de robot zal ervaren bij het uitvoeren van verschillende taken. Per taak die de robot uit moet kunnen voeren is het verschillend wat voor gevolgen een afwijking heeft. Wanneer de robot een heel klein object moet oppakken of een handeling moet uitvoeren, dan zal een afwijking van 1 centimeter erg veel zijn, terwijl bij het oppakken van een beker is een afwijking van 1 centimeter te overzien. In dit onderzoek wordt er rekening mee gehouden dat grotere taken uitgevoerd moeten worden en de robot hoeft dus niet op de millimeter correct te navigeren. Alle afwijkingen onder de centimeter worden als correct gezien.

5 Resultaten

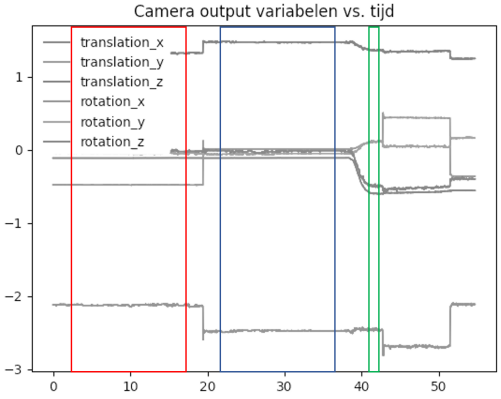
Het onderzoek is drie keer uitgevoerd. Allereerst wordt er ingegaan op de individuele waardes van de robot en camera output bij meting 1,2 en 3. Verder wordt er gekeken naar de homogene matrix 'X' om vervolgens berekeningen uit te voeren om te controleren hoe nauwkeurig de matrix 'X' is.

5.1 Algemeen

in figuur 9 en 10 zijn de gebieden zichtbaar die de drie verschillende posities voorstellen. De verdeling van de gebieden is gedaan volgens tabel 1 en 2. Na het laatste geselecteerde gebied is er te zien dat het hoofd weer twee keer wordt verplaatst. Deze gegevens worden niet gebruikt bij het berekenen van transformatie matrix 'X'. Hier is voor gekozen omdat er minimaal twee rotatie-assen niet parallel mogen zijn. Dit betekend dat wanneer de as net niet meer parallel staat, dat er wordt voldaan aan de restrictie. Echter is er niet bekend bij de methode of de methode minder nauwkeurig is als de rotatie-assen net niet parallel zijn. Om geen risico te nemen voor extra onnauwkeurigheid is er gekozen voor twee rotatie-assen die loodrecht op elkaar staan, zodat het verschil tussen de rotatie-assen groot is.



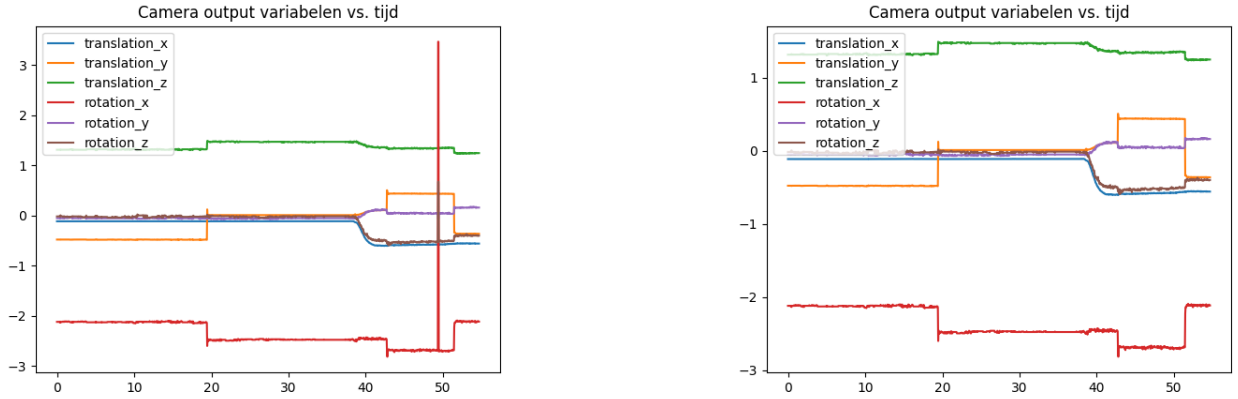
Figuur 9: Robot output variabelen tegen tijd met geselecteerde gebieden; rood:positie 1, blauw: positie 2, groen: positie 3



Figuur 10: Camera output variabelen tegen tijd met geselecteerde gebieden; rood:positie 1, blauw: positie 2, groen: positie 3

5.2 Meting 1 & 2

In figuur 11a zijn de camera output variabelen geplot tegen de tijd. Hier valt op dat er aan het eind van de meting een piek in de $rotation_x$ variabelen te zien is. Deze piek wordt verklaard doordat de camera de ArUco marker op sommige afstanden niet meer correct uit kan lezen. Deze waarden zijn niet nuttig voor het onderzoek en worden er uit gefilterd. Hierbij worden alle waardes van de individuele elementen overgeslagen en niet meegenomen in het plot. Het plot zonder de piek is te zien in figuur 11b.

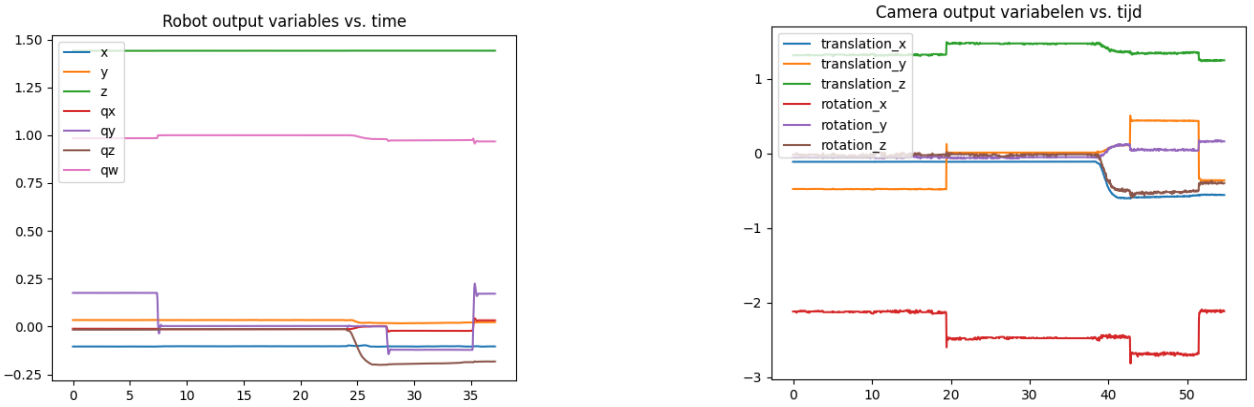


(a) Camera output geplot tegen tijd met piek

(b) Camera output geplot tegen tijd zonder piek

Figuur 11: Camera output voor en na filteren

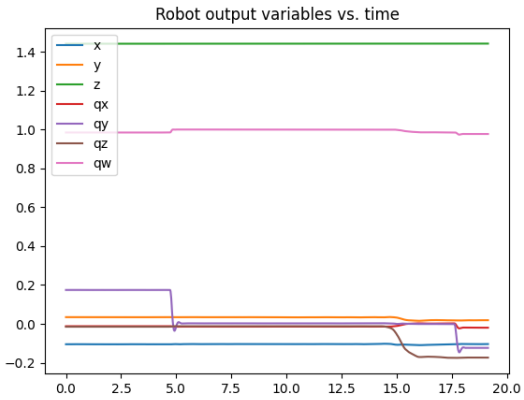
In figuur 12 en 13 zijn de robot en camera output variabelen van meting 1 en 2 geplot tegen de tijd. In alle plots is duidelijk de verplaatsing van zowel hoofd als romp te zien volgens tabel 1. Verder valt te zien dat de hoofdbeweging een stapsgewijze beweging is, aangezien het hoofd snel van positie één naar positie twee wordt gezet. De rompbeweging is als een soepele beweging te zien in de plots en dat komt overeen met hoe de romp in de realiteit op zijn positie is gezet. Meting 1 en meting 2 zijn op dezelfde manier uitgevoerd, zoals in tabel 1 is aangegeven. Het enige verschil is dat er bij meting één nog twee bewegingen van het hoofd zijn uitgevoerd na de romp beweging en bij meting twee is er nog één hoofdbeweging uitgevoerd. Zoals eerder benoemd worden deze waardes niet meegenomen om tot de homogene transformatie matrix 'X' te komen.



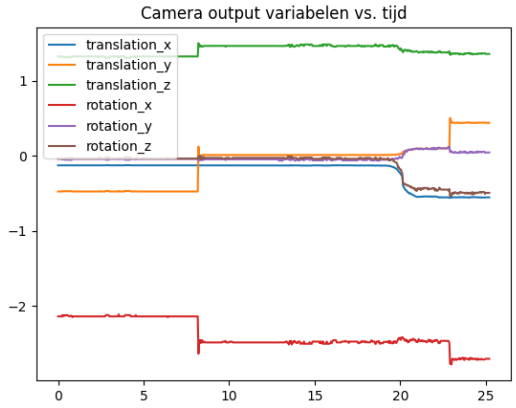
(a) Robot output geplot tegen tijd

(b) Camera output geplot tegen tijd

Figuur 12: Robot en camera output meting 1



(a) Robot output geplot tegen tijd

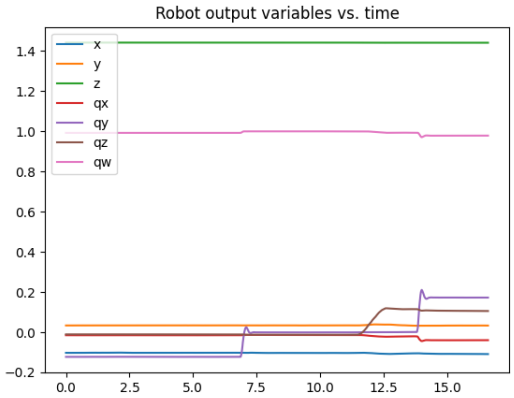


(b) Camera output geplot tegen tijd

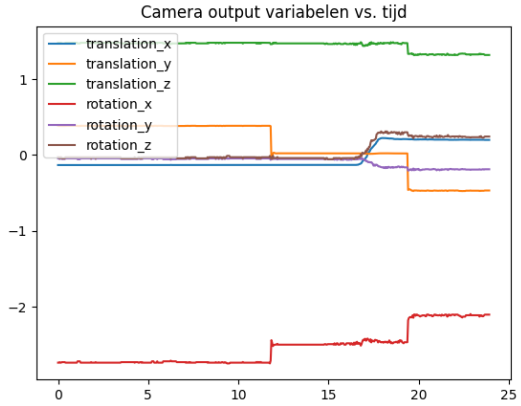
Figuur 13: Robot en camera output meting 2

5.3 Meting 3

In figuur 14 veranderen ook precies dezelfde variabelen als in figuur 12 en 13 echter zijn de veranderingen precies in tegenovergestelde richting, zoals ook te zien in tabel 2.



(a) Robot output geplot tegen tijd



(b) Camera output geplot tegen tijd

Figuur 14: Robot en camera output meting 3

5.4 Uitkomst 'X'

De data uit figuren 12,13 en 14 zijn gebruikt om uiteindelijke transformatie matrix 'X' van camera naar het nekgewricht te schatten voor meting 1, 2 en 3:

$$X_{Meting1} = \begin{bmatrix} 0.02051356 & -0.72912326 & -0.6840749 & 0.15251739 \\ 0.99949388 & 0.03159533 & -0.00370385 & 0.00605084 \\ 0.02431413 & -0.6836527 & 0.72940237 & 0.13110308 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$X_{Meting2} = \begin{bmatrix} 0.02917159 & -0.72192251 & -0.69135874 & 0.13469594 \\ 0.99934478 & 0.03588872 & 0.00469163 & 0.00287226 \\ 0.02142499 & -0.69104261 & 0.72249642 & 0.13311077 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$X_{Meting3} = \begin{bmatrix} 0.0284137 & -0.76921167 & -0.63836202 & 0.15170621 \\ 0.99956698 & 0.01697729 & 0.0240338 & -0.03480532 \\ -0.00764942 & -0.63876849 & 0.76936097 & 0.14764477 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De homogene matrixen geven vergelijkbare resultaten. Tijdens meting drie is er een andere beweging gemaakt en zelf met een andere beweging blijkt de transformatie matrix relatief hetzelfde te zijn. Om nog beter verschillen tussen de matrixen weer te geven, worden de translatievectoren met elkaar vergeleken. Verder wordt de standaarddeviatie met behulp van vergelijking (9) van de verschillende translatie componenten berekend om te bepalen of het resultaat van de transformatie matrixen 'X' constant en nauwkeurig zijn.

	Meting 1	Meting 2	Meting 3	Standaarddeviatie
translatie x	0.15251739	0.13469594	0.15170621	0.00821659
translatie y	0.00605084	0.00287226	-0.03480532	0.01436293
translatie z	0.13110308	0.13311077	0.14764477	0.00737033

Tabel 3: Translatievectoren van de drie verschillende matrixen + de standaarddeviatie per translatie component.

De translatievectoren van meting één en twee lijken erg op elkaar. De y component van translatievector uit meting drie lijkt wat af te wijken van de waarde in meting één en meting twee. Dit is ook terug te zien in de standaarddeviatie van component y. De standaarddeviaties van alle drie de componenten blijken relatief klein te zijn. Dit duidt er dus op dat de consistentie van de methode om transformatie matrix 'X' te schatten groot is.

5.5 Controleren van translatievector

Het controleren van het verschil in translatievector tussen de gemeten transformatie matrix van nekgewricht naar robot basis ($A^{(1)}$) en de berekende transformatie matrix van nekgewricht naar robot basis ($A^{(1)}_{calc}$) met behulp van vergelijking (10) wordt gedaan voor meting 1, 2 en 3.

Meting één geeft de volgende resultaten:

$$A^{(1)}_{calc} = \begin{bmatrix} -0.526391848 & 0.00520577621 & -0.85022616 & -0.209113145 \\ -0.0213666921 & 0.999584458 & 0.0193488113 & -0.133426384 \\ 0.84997358 & 0.028351577 & -0.526061879 & 1.35711902 \\ 4.11074262 \times 10^{-16} & -1.38066063 \times 10^{-16} & -1.14568057 \times 10^{-16} & 1 \end{bmatrix}$$

$$A^{(1)} = \begin{bmatrix} -0.93787564 & 0.01680452 & 0.34656441 & -0.10441562 \\ -0.02834941 & -0.99919825 & -0.02826942 & 0.03409447 \\ 0.3458115 & -0.0363381 & 0.93760009 & 1.44120337 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Meting twee geeft de volgende resultaten:

$$A_{calc}^{(1)} = \begin{bmatrix} -0.535430692 & 0.0217238282 & -0.844299739 & -0.255415180 \\ -0.0115502714 & 0.999387317 & 0.0330390819 & -0.140059280 \\ 0.844500187 & 0.0274420296 & -0.534851727 & 1.36906043 \\ 3.22802163 \times 10^{-16} & -4.06789018 \times 10^{-17} & 4.27884464 \times 10^{-16} & 1 \end{bmatrix}$$

$$A^{(1)} = \begin{bmatrix} -0.93884786 & 0.01824383 & 0.3438486 & -0.10478804 \\ -0.02838428 & -0.99929728 & -0.02448029 & 0.03444857 \\ 0.34316035 & -0.03274316 & 0.93870594 & 1.44126002 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Meting drie geeft de volgende resultaten:

$$A_{calc}^{(1)} = \begin{bmatrix} -0.917779835 & 0.010649636 & -0.396946796 & -0.893061660 \\ 0.018465526 & 0.999703489 & -0.0158731825 & -0.0898660631 \\ 0.396660054 & -0.0218979181 & -0.917704355 & 1.11393192 \\ -1.10743606 \times 10^{-16} & 2.78385199 \times 10^{-16} & 6.78656917 \times 10^{-16} & 1 \end{bmatrix}$$

$$A^{(1)} = \begin{bmatrix} -0.96944253 & 0.03257311 & -0.24314639 & -0.10241667 \\ -0.02735468 & -0.9993179 & -0.02480854 & 0.03386853 \\ -0.24378863 & -0.01739926 & 0.96967226 & 1.44157245 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Om te controleren hoe nauwkeurig het berekenen van matrix 'A' is gegaan, wordt er gekeken naar het verschil in de translatievector. Het absolute verschil in de componenten van de translatievectoren van de verschillende metingen wordt weergegeven in de volgende tabel:

	Meting 1	**Meting 2**	**Meting 3**
Translatie x (m)	0.104697525	0.15062714	0.79064599
Translatie y (m)	0.167520854	0.17450773	0.1237345931
Translatie z (m)	0.08408435	0.07219959	0.32764053

Tabel 4: Verschil in translatie componenten

Er is te zien dat het verschil bij de meeste componenten groter is dan 10 centimeter. Het grootste verschil is te zien in de translatie x component van meting drie, waarbij het verschil tussen de gemeten en de berekende translatie bijna 80 centimeter is.

6 Discussie & conclusie

Een beperking in dit onderzoek is dat de camera na een kleine afstand vanaf de ArUco marker al geen correcte waardes meer geeft. Dit zorgt ervoor dat de robot dicht bij de ArUco marker moest staan, Hierdoor zijn er minder metingen van posities van het hoofd mogelijk. Als de robot dichterbij de ArUco marker staat en het hoofd wordt gekanteld of de romp wordt gedraaid, dan zal de ArUco marker sneller uit het zicht van de robot camera verdwijnen. Om er voor te zorgen dat dit geen beperking in het onderzoek is, zal er naar het script van de camera in de robot gekeken moeten worden om te bepalen waar het probleem ligt dat de ArUco marker niet correct gelezen kan worden op grotere afstanden.

De simultane oplossing kan zorgen voor verschillende resultaten afhankelijk van de schaling van de positionele component. Dit betekend dat kleinere veranderingen in de positiegegevens kunnen leiden tot onnauwkeurigheden in de kalibratie resultaten. Er kan niet gezegd worden hoe accuraat de positiegegevens van de camera en het nekgewricht zijn bepaald en dat kan er toe leiden dat de oplossing minder geschikt is voor een soort gelijk experiment.

De methode die gebruikt is voor dit onderzoek is alleen toepasbaar wanneer de end effector en de robot aan elkaar bevestigd zijn en dus gelijk bewegen. Dit betekent dus dat deze toepassing niet mogelijk is voor elk hand-oog kalibratie probleem. Er bestaan tevens ook oog-hand kalibratie problemen waar end effector en camera onafhankelijk van elkaar bewegen, dit probleem wordt beschreven met AX=YB.

Er zijn relatief weinig posities gebruikt, waardoor de foutmarge relatief groot kan zijn. De transformatie matrix 'X' is nu berekend aan de hand van drie verschillende posities, terwijl de waarde van Matrix 'X' nog veranderd wanneer er meer posities worden gebruikt. Om daadwerkelijk een correct en nauwkeurig antwoord van de transformatie matrix te verkrijgen zouden dus veel meer verschillende hoofdposities gebruikt moeten worden. Dit zou gedaan kunnen worden door het hoofd en de romp soepel te laten bewegen terwijl de robot naar de marker kijkt. Vervolgens zouden alle punten als aparte positie genomen kunnen worden mits de beweging om minstens twee niet-parallelle rotatie-assen is. Echter was dit voor dit onderzoek niet mogelijk omdat er geen timestamps bekend waren. Dit betekend dat de data van de camera en de data van de robot niet aan elkaar gekoppeld konden worden. Dit is dan ook meteen de reden waarom er voor gekozen is om het experiment op een statische manier uit te voeren en de gemiddelde waardes de nemen.

Het valt op dat de homogene transformatie matrixen 'X' vergelijkbare resultaten geeft, maar wanneer er een berekening mee wordt gedaan de verschillen aanzienlijk groter zijn. Dit zou betekenen dat een klein verschil in transformatie matrix 'X' grote gevolgen heeft voor positionering van de end effector. Zo is er in tabel 4 te zien dat de verschillen in de translatie componenten over het algemeen groter is dan 10 centimeter, wat er voor zorgt dat de positionering van de robot arm 10 centimeter naast een object kan zitten. Daarom is het van belang om nauwkeurige metingen uit te voeren om zo een best mogelijke schatting van de transformatie matrix 'X' te verkrijgen. De grote afwijkingen die te zien zijn in tabel 4 kunnen verklaard worden door meetfouten die ontstaan zijn bij het bepalen van de positiegegevens van de camera en het nekgewricht. '$A_{calc}^{(1)}$' wordt berekend met behulp van gemeten B en geschatte matrix 'X'. Dit betekend wanneer er een fout zit in gemeten matrix 'B' dat er een nog grotere doorreken fout zichtbaar is bij het berekenen van '$A_{calc}^{(1)}$'. Deze fout zit tevens al verwerkt in matrix 'X', maar natuurlijk ook in matrix 'B'. Om de fout zo klein mogelijk te maken, moeten er veel metingen uitgevoerd worden. Ook is het belangrijk dat er nagegaan wordt of het bepalen van matrix 'A' en matrix 'B' op een nauwkeurige manier gedaan wordt.

De resultaten uit dit onderzoek tonen aan dat de verkregen transformatie matrix 'X' niet geschikt is voor het gebruik van de robot in de gezondheidszorg, voornamelijk vanwege de afwijking die optreden bij het aansturen van de robotarm op basis van visuele input.

Literatuurlijst

[1] Sheridan TB. Human-robot interaction: status and challenges. Human Factors. 2016;58(4):525 532. doi:10.1177/0018720816644364.

[2] Garcia E, Jimenez MA, Santos PGD, Armada M. The evolution of robotics research. IEEE Robotics Automation Magazine. 2007;14(1):90-103. doi:10.1109/MRA.2007.339608.

[3] Shah M, Eastman RD, Hong T. An Overview of Robot-SEnsor Calibration Methods for Evaluation of Perception Systems. Association for Computing Machinery. 2012:15-20. doi:10.1145/2393091.2393095.

[4] Sanfilippo F, Smith J, Bertrand S, Svendsen THS. Mixed reality (MR) Enabled Proprio and Teleoperation of a Humanoid Robot for paraplegic Patients. International Conference on Information and Computer Technologies. 2022:153-8. doi:10.1109/ICICT55905.2022.00034.

[5] Wu J, Liu M, Zhu Y, Zou Z, Dai MZ, Zhang C, et al. Globally Optimal Symbolic Hand-Eye Calibration. IEEE/ASME Transactions on Mechatronics. 2021;26(3):1369-79. doi:10.1109/TMECH.2020.3019306.

[6] Zhang Y, Liu X, Chen H, Cai Z, Tang Q, Peng X, et al. An accurate hand-eye calibration algorithm with global optimization. International Society for Optics and Photonics. 2018;10827. doi:10.1117/12.2500972.

[7] Chou JCK. Quaternion Kinematic and Dynamic Differential Equations. IEEE Transactions on Robotics and Automation. 1992;8(1):53-64. doi:10.1109/70.127239.

[8] Gouasmi M, Ouali M, Brahim F. Robot Kinematics Using Dual Quaternions. International Journal of Robotics and Automation. 2012;1(1):13-30. doi:10.11591/ijra.v1i1.275.

[9] Daniilidis K. Hand-Eye Calibration Using Dual Quaternions. International Journal of Robotics Research. 1999;18(3):286-98. doi:10.1177/02783649922066213.

[10] Shiu YC, Ahmad S. Calibration of Wrist-Mounted Robotic Sensors by Solving Homogeneous Transform Equations of the Form AX=XB. IEEE Transactions on Robotics and Automation. 1989;5(1):16-29. doi:10.1109/70.88014.

[11] Tsai RY, Lenz RK. A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. IEEE Transactions on Robotics and Automation. 1989;5(3):345-58. doi:10.1109/70.34770.

[12] Park FC, Martin BJ. Robot sensor calibration: solving AX=XB on the Euclidean group. Vision Research. 1994;10(5):717-21. doi:10.1109/70.326576.

[13] Chou JCK, Kamel M. Finding the Position and Orientation of a Sensor on a Robot Manipulator Using Quaternions. The International Journal of Robotics Research. 1991;10(3):240-54. doi:10.1177/027836499101000305.

[14] Qiu S, Wang M, Kermani MR. A Modern Solution for an Old Calibration Problem. IEEE Instrumentation & Measurement Magazine. 2021;24(3):28-35. doi:10.1109/MIM.2021.9436097.

[15] Wang X, Song H. Optimal robot-world and hand-eye calibration with rotation and translation coupling. Robotica. 2022;40(9):2953-68. doi:10.1017/S0263574721002034.

[16] Lu YC, Chou JCK. Eight-space quaternion approach for robotic hand-eye calibration. IEEE International Conference on Systems, Man and Cybernetics. 1995;4:3316-21. doi:10.1109/ICSMC.1995.538297.

[17] Wu J, Sun Y, Wang M, Liu M. Hand-Eye Calibration: 4-D Procrustes Analysis Approach. IEEE Transactions on Instrumentation and Measurement. 2020;69(6):2966-81. doi:10.1109/TIM.2019.2930710.

[18] Chen HH. A screw motion approach to uniqueness analysis of head-eye geometry. IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 1991:145-51. doi:10.1109/CVPR.1991.139677.

[19] Zhao Z, Liu Y. Hand-Eye Calibration Based on Screw Motions. 18th International Conference on Pattern Recognition. 2006;3:1022-6. doi:10.1109/ICPR.2006.620.

[20] Camera Calibration and 3D Reconstruction. OpenCV: Open Source Computer Vision Library. Available from: `https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html`.

[21] Seymour S. A Method for Computing the Partial Singular Value Decomposition. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1982;PAMI-4(6):671-6. doi:10.1109/TPAMI.1982.4767324.

[22] Detection of ArUco Markers. OpenCV: Open Source Computer Vision Library. Available from: `https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html`.

[23] Kuipers JB. Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality. Princeton University Press. 2002:125.

Bijlagen

A Filteren van robot output

```
import ast
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import SpanSelector

# Read in the data from the Txt-file
x = []
y = []
z = []
qx = []
qy = []
qz = []
qw = []

filename = 'C:/Users/pckoo/Documents/Python/Bachelor opdracht/Redo/output3.txt'
with open(filename, 'r') as file:
    first_line = next(file)  # Read first row and skip 'start recording'
    for line in file:
        line = line.strip()
        if line.startswith('[') and line.endswith(']'):
            values = ast.literal_eval(line)
            x.append(float(values[0]))
            y.append(float(values[1]))
            z.append(float(values[2]))
            qx.append(float(values[3]))
            qy.append(float(values[4]))
            qz.append(float(values[5]))
            qw.append(float(values[6]))

# Define sample time (500Hz) and create time array
t = 1 / 500
time = np.arange(0, len(x)) * t

# Variables for selected region
selected_values = []
x_start = None
x_end = None

# Function to process the selected region
def process_region(xmin, xmax):
    global x_start, x_end, selected_values
    x_start, x_end = xmin, xmax

    selected_values = []
    for t, val_x, val_y, val_z, val_qx, val_qy, val_qz, val_qw in
    zip(time, x, y, z, qx, qy, qz, qw):
        if x_start <= t <= x_end:
            selected_values.append((val_x, val_y, val_z, val_qx,
            val_qy, val_qz, val_qw))

# Create the plot
fig, ax = plt.subplots()
ax.plot(time, x, label='x')
```

```
ax.plot(time, y, label='y')
ax.plot(time, z, label='z')
ax.plot(time, qx, label='qx')
ax.plot(time, qy, label='qy')
ax.plot(time, qz, label='qz')
ax.plot(time, qw, label='qw')
ax.legend(loc='upper left')
plt.title('Robot output variables vs. time')

# Create the SpanSelector
span_selector = SpanSelector(ax, process_region, 'horizontal',
            useblit=True, props=dict(alpha=0.5, facecolor='blue'))

# Show the plot
plt.show()

# Calculate mean of the individual variables from the selected region
x_selected = [values[0] for values in selected_values]
y_selected = [values[1] for values in selected_values]
z_selected = [values[2] for values in selected_values]
qx_selected = [values[3] for values in selected_values]
qy_selected = [values[4] for values in selected_values]
qz_selected = [values[5] for values in selected_values]
qw_selected = [values[6] for values in selected_values]

# Calculate mean values
x_mean = np.mean(x_selected)
y_mean = np.mean(y_selected)
z_mean = np.mean(z_selected)
qx_mean = np.mean(qx_selected)
qy_mean = np.mean(qy_selected)
qz_mean = np.mean(qz_selected)
qw_mean = np.mean(qw_selected)

# Print mean values
print("Mean values:")
print("x:", x_mean)
print("y:", y_mean)
print("z:", z_mean)
print("qx:", qx_mean)
print("qy:", qy_mean)
print("qz:", qz_mean)
print("qw:", qw_mean)
```

B Transformatie van robot output

```
import numpy as np
def transformRobotOutput(qx, qy ,qz, qw, x, y, z):
    """
    Convert 4 quaternions into 3x3 rotation matrix
    Convert x,y and z into 3x1 translation vector
    """
    r00 = 2 * (qx * qx + qy * qy) - 1
    r01 = 2 * (qy * qz - qx * qw)
    r02 = 2 * (qy * qw + qx * qz)

    r10 = 2 * (qy * qz + qx * qw)
    r11 = 2 * (qx * qx + qz * qz) - 1
    r12 = 2 * (qz * qw - qx * qy)

    r20 = 2 * (qy * qw - qx * qz)
    r21 = 2 * (qz * qw + qx * qy)
    r22 = 2 * (qx * qx + qw * qw) - 1

    a_mat_r = np.array([[r00, r01, r02],
                        [r10, r11, r12],
                        [r20, r21, r22]])

    a_mat_t = np.array([[x],[y],[z]])

    return a_mat_r, a_mat_t
```

C Filteren van camera output

```
import re
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import SpanSelector


def importCsvCamera(filePath):
    with open(filePath, 'r') as file:
        data = file.read()

    elements = re.split('---\n', data)

    data_objects = []

    for element in elements:
        if element.strip():
            i = re.split("\n", element)
            if len(i) > 10:
                obj = {
                    'translation_vectors': [float(i[12][2:]),float(i[13][2:]),
                    float(i[14][2:])],
                    'rotation_vectors': [float(i[16][2:]),float(i[17][2:]),
                    float(i[18][2:])]
                }
                data_objects.append(obj)

    return data_objects
```

```
data_objects = importCsvCamera("C:/Users/pckoo/Documents/Python/Bachelor
opdracht/Redo/out3.txt")

time_step = 1 / 24
translation_x = []
translation_y = []
translation_z = []
rotation_x = []
rotation_y = []
rotation_z = []
timestamps = []

for i, point in enumerate(data_objects):
    rotation_x.append(point['rotation_vectors'][0])
    if len(rotation_x) > 1:
        previous_rotation = rotation_x[-2]  # Get the previous value added to rotation_x
        current_rotation = rotation_x[-1]  # Get the current value being added
        if current_rotation - previous_rotation >= 2:
            rotation_x.pop(-1)  # Remove the current value from
            rotation_x
            continue

    translation_x.append(point['translation_vectors'][0])
    translation_y.append(point['translation_vectors'][1])
    translation_z.append(point['translation_vectors'][2])

    rotation_y.append(point['rotation_vectors'][1])
    rotation_z.append(point['rotation_vectors'][2])
    timestamps.append(i * time_step)


# Variables for selected region
selected_values = []


# Function to process the selected region
def process_region(xmin, xmax):
    global selected_values
    selected_values = []
    for t, val_translation_x, val_translation_y, val_translation_z,
    val_rotation_x, val_rotation_y, val_rotation_z
    in zip(timestamps, translation_x, translation_y, translation_z,
    rotation_x, rotation_y, rotation_z):
        if xmin <= t <= xmax:
            selected_values.append(
                (val_translation_x, val_translation_y, val_translation_z,
                val_rotation_x, val_rotation_y, val_rotation_z))

    # Calculate mean of individual variables
    translation_x_mean = np.mean([val[0] for val in selected_values])
    translation_y_mean = np.mean([val[1] for val in selected_values])
    translation_z_mean = np.mean([val[2] for val in selected_values])
    rotation_x_mean = np.mean([val[3] for val in selected_values])
    rotation_y_mean = np.mean([val[4] for val in selected_values])
```

```
    rotation_z_mean = np.mean([val[5] for val in selected_values])

    # Print mean of individual variables
    print("Gemiddelde waarden:")
    print("translation_x_mean:", translation_x_mean)
    print("translation_y_mean:", translation_y_mean)
    print("translation_z_mean:", translation_z_mean)
    print("rotation_x_mean:", rotation_x_mean)
    print("rotation_y_mean:", rotation_y_mean)
    print("rotation_z_mean:", rotation_z_mean)


fig, ax = plt.subplots()
ax.plot(timestamps, translation_x, label='translation_x')
ax.plot(timestamps, translation_y, label='translation_y')
ax.plot(timestamps, translation_z, label='translation_z')
ax.plot(timestamps, rotation_x, label='rotation_x')
ax.plot(timestamps, rotation_y, label='rotation_y')
ax.plot(timestamps, rotation_z, label='rotation_z')
plt.legend(loc='upper left')
plt.title("Camera output variabelen vs. tijd")

# Create the SpanSelector
span_selector = SpanSelector(ax, process_region, 'horizontal', useblit=True,
                             props=dict(alpha=0.5, facecolor='blue'))

# Show plot
plt.show()
```

D Daniilidis methode

```
import cv2
import numpy as np
import Robot_Output as ro
import csv
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.widgets import SpanSelector

def import_csv(file_path, startColumn):
    data_list = []
    with open(file_path, 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader) # Skip first row
        for row in csv_reader:
            data = [float(item) for item in row[startColumn:]]
            # Exclude the first item and add the remaining items to
            # the list and convert string to float
            data_list.append(data)
    return data_list

# Import robot data
lijstRobotData = import_csv('C:/Users/pckoo/Documents/Python/Bachelor
opdracht/Redo/Robot1.csv', 0)

lijstRotationGripper2Base = []
lijstTranslationGripper2Base = []
```

```
# For each element in list loop
for i in lijstRobotData:
    # Calculate aMatR and aMatT with qx, qy, qz, qw, x, y, z
    aMatR, aMatT = ro.transformRobotOutput(i[3], i[4], i[5], i[6], i[0], i[1], i[2])
    # Add to list
    lijstRotationGripper2Base.append(aMatR)
    lijstTranslationGripper2Base.append(aMatT)

# Import camera data
lijstCameraData = import_csv('C:/Users/pckoo/Documents/Python/Bachelor opdracht
/Redo/Camera1.csv', 0)

lijstRotationTarget2Cam = []
lijstTranslationTarget2Cam = []

for i in lijstCameraData:
    # Transform rvec to normal rotation matrix
    rvec = [float(i[3]), float(i[4]), float(i[5])]
    tvec = np.array([float(i[0]), float(i[1]), float(i[2])])
    BMatR, _ = cv2.Rodrigues(np.array(rvec))

    # Add to list
    lijstRotationTarget2Cam.append(BMatR)
    lijstTranslationTarget2Cam.append(tvec)

R_cam2gripper, t_cam2gripper = cv2.calibrateHandEye(lijstRotationGripper2Base,
lijstTranslationGripper2Base, lijstRotationTarget2Cam, lijstTranslationTarget2Cam,
cv2.CALIB_HAND_EYE_DANIILIDIS)

# Check if the answer is right:
# https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#gaebfc1c9f7434196a374c382abf43439b
H_cam2gripper = np.eye(4, dtype=np.float64)
H_cam2gripper[:3, :3] = R_cam2gripper
H_cam2gripper[:3, 3] = t_cam2gripper.flatten()

# Define H_gripper2base positie 1
H_gripper2base_1 = np.eye(4, dtype=np.float64)
H_gripper2base_1[:3, :3] = lijstRotationGripper2Base[0]
H_gripper2base_1[:3, 3] = lijstTranslationGripper2Base[0].flatten()

# Define H_gripper2base positie 2
H_gripper2base_2 = np.eye(4, dtype=np.float64)
H_gripper2base_2[:3, :3] = lijstRotationGripper2Base[1]
H_gripper2base_2[:3, 3] = lijstTranslationGripper2Base[1].flatten()

# Define H_gripper2base positie 3
H_gripper2base_3 = np.eye(4, dtype=np.float64)
H_gripper2base_3[:3, :3] = lijstRotationGripper2Base[2]
H_gripper2base_3[:3, 3] = lijstTranslationGripper2Base[2].flatten()

# Define H_target2cam positie 1
H_target2cam_1 = np.eye(4, dtype=np.float64)
H_target2cam_1[:3, :3] = lijstRotationTarget2Cam[0]
H_target2cam_1[:3, 3] = lijstTranslationTarget2Cam[0].flatten()
```

```
# Define H_target2cam positie 2
H_target2cam_2 = np.eye(4, dtype=np.float64)
H_target2cam_2[:3, :3] = lijstRotationTarget2Cam[1]
H_target2cam_2[:3, 3] = lijstTranslationTarget2Cam[1].flatten()

# Define H_target2cam positie 3
H_target2cam_3 = np.eye(4, dtype=np.float64)
H_target2cam_3[:3, :3] = lijstRotationTarget2Cam[2]
H_target2cam_3[:3, 3] = lijstTranslationTarget2Cam[2].flatten()

# calculate H_gripper2base with known H_target2came en H_cam2gripper
H_gripper2base_1calc = H_cam2gripper @ H_target2cam_1 @ np.linalg.pinv(H_cam2gripper)
```