2015•2016
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: energie

Masterproef

Laserscanmeting voor de herkenning van 3D-delen op een lopende band

Promotor :
dr. ir. Johan BAETEN

Promotor :
dr. ir. WIM PERSOONS

Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

Niels Vandekerkhof

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie

2015•2016

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: energie

Masterproef

Laserscanmeting voor de herkenning van 3D-delen op een lopende band

Promotor :
dr. ir. Johan BAETEN

Promotor :
dr. ir. WIM PERSOONS

Niels Vandekerkhof

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie

universiteit hasselt | KU LEUVEN

Woord vooraf

Deze masterproef is het sluitstuk van de opleiding master in de industriële wetenschappen met afstudeerrichting energie automatisering, gevolgd aan de universiteit Hasselt/KU Leuven.

In de eerste plaats zou ik mijn externe promotor Wim Persoons willen bedanken voor het mogelijk maken van deze masterproef en voor het geven van feedback.

Ten tweede zou ik graag mijn interne promotor Johan Baeten willen bedanken voor zowel de technische als taalkundige ondersteuning.

Als laatste zou ik graag Bert Van Den Hende bedanken voor de samenwerking bij deze masterproef en om mij de kans te geven om vroeger af te studeren.

Niels Vandekerkhof
januari 2016

Inhoudsopgave

Lijst van figuren

Abstract

KUKA Automatisering + Robots N.V. is een dochterfirma van de Duitse KUKA Systems GmbH en KUKA Roboter GmbH in Augsburg. KUKA is alom gekend als fabrikant van industriële robots en automatiseringssystemen.

Met deze masterproef wil KUKA kennis verwerven over 3D-metingen met een laserlijnscanner en point cloud matching. De hoofddoelstelling van de masterproef bestaat erin een laserlijnscanner in te zetten ter herkenning van 3-dimensionale objecten. Op deze manier legt dit eindwerk de basis voor het vervolgproject “Robotcel voor het grijpen van 3D-delen van een lopende band”, dat bin-picking als doel heeft.

De werking van de gewenste situatie is als volgt: de laserlijnscanner (LMS400) scant objecten op de transportband. Een Visual Studio programma verwerkt de afstandsmetingen tot een puntenwolk. Het “iterative closest point” algoritme uit de “Point Cloud Library” (PCL) vergelijkt de actuele puntenwolk met referentie-puntenwolken van eerder gescande objecten en bepaalt een mogelijke match. Ten slotte bepaalt het Visual Studio programma het zwaartepunt en de oriëntatie om zodoende manipulatie met een robot mogelijk te maken.

Met behulp van het ontwikkelde algoritme is het herkennen van 3D-objecten gerealiseerd. Ook de nauwkeurigheid van de positiebepaling blijft binnen de vooropgestelde grenzen. Hiermee is de basis voor het implementeren van de robot met succes gelegd.

Abstract in English

KUKA Automatisering + Robots N.V. is a subsidiary of the German KUKA Systems GmbH and KUKA Roboter GmbH in Augsburg. KUKA is widely known as a manufacturer of industrial robots and automation systems.

With this master thesis KUKA wants to acquire knowledge about 3D-measurements with a laser line scanner and about point cloud matching. The main objective of the thesis consists of deploying a laser line scanner for recognizing 3-dimensional objects. Thus this master thesis creates the foundation for the follow-up project “Robotcel voor het grijpen van 3D-delen van een lopende band”, which has bin-picking as objective.

The procedure of the desired situation is as follows: the laser line scanner (LMS400) scans objects on the conveyor belt. A Visual Studio program processes the distance measurements into a point cloud. The “iterative closest point” algorithm from the Point Cloud Library (PCL) compares the current point cloud with the reference point clouds from previously scanned objects and determines a possible match. Lastly the Visual Studio program will determine the centroid and the orientation, thus enabling manipulation with a robot.

With the aid of the developed matching algorithm recognizing 3D-objects has been successfully realized. The accuracy of the position measurements stay within the provided boundaries. Thus the foundation for the follow-up project (implementing a robot for bin-picking) has successfully created.

Inleiding

KUKA Automatisering + Robots N.V. is één van de bekendste fabrikanten van industriële robots en automatiseringssystemen. Het robotcenter in Houthalen is in 30 jaar uitgegroeid tot het grootste robotcenter van de Benelux. In de beginperiode lag de focus vooral bij de verkoop, service en opleidingen rond robots maar nu ontwerpt, bouwt en levert KUKA ook complete robotcellen en installaties [1].

De focus van de masterproef ligt bij visiesystemen. Er zijn verschillende 3D-visiesystemen op de markt. Performante, nauwkeurige 3D-camera's zijn over het algemeen heel duur. Goedkope 3D-camera's zijn vooralsnog behoorlijk onnauwkeurig. Bovendien is bij de meeste beschikbare visiesystemen de ontwikkelomgeving afgesloten voor de gebruiker of systeemintegrator. Hieruit volgt de vraag van KUKA om (in eigen beheer) een volwaardig en kostenefficiënt alternatief te ontwikkelen op basis van een laserlijnscanner in combinatie met een bewegend voorwerp.

De laserlijnscanner (LMS400), die zich boven de transportband bevindt, scant objecten in. Een C# programma verwerkt deze metingen en exporteert de metingen als een 3-dimensionele puntenwolk. Hierna zal een C++ programma zal deze puntenwolk gebruiken om het object te herkennen.

Hoofdstuk twee beschrijft de instelling van de LMS400. De nadruk ligt vooral op de parameters die de nauwkeurigheid bepalen. Hoofdstuk drie legt stap voor stap uit hoe het Visual Studio programma de afstandsmetingen van de LMS400 verwerkt tot een puntenwolk. De nadruk ligt hier op het wegwerken van onnauwkeurigheden en het conditioneren van de puntenwolk om het beste resultaat te verkrijgen. Hoofdstuk vier geeft de werking van het matching algoritme weer en beschrijft hoe de puntenwolken vergeleken worden en welke parameters belangrijk zijn. Hoofdstuk vijf somt de testresultaten op. Deze geven de nauwkeurigheid van het systeem weer. Als laatste volgt het besluit van deze masterproef in hoofdstuk zes.

1 Kader en onderzoeksopzet

De hoofddoelstelling van de masterproef bestaat erin een laserlijnscanner in te zetten ter herkenning van 3-dimensionale objecten en om al zo bin-picking mogelijk te maken. Bij bin-picking, wat het automatisch detecteren en grijpen van objecten is, tracht een robot het object in het zwaartepunt te grijpen om dit dan te manipuleren. De laserlijnscanner, die zich boven de transportband bevindt, scant de objecten op de transportband. De scanprocedure van een laserlijnscanner is te vergelijken met het scannen van een barcode: de scanner meet afzonderlijke punten in een rechte lijn, elk punt heeft zijn eigen coördinaat en waarde. Een laserscanner gebruikt dezelfde procedure maar geeft in plaats van een streepjescode een afstandsmetingenlijst.

Doelstelling

Omdat er geen programma's bestaan om met de LMS400 puntenwolken te creëren is het nodig om zelf een programma te ontwikkelen.

Doel van het programma:

- De verkregen meetwaardes bewerken om onnauwkeurigheden weg te werken.
- De bewerkte scan omzetten naar een puntenwolk om een 3D-beeld te verkrijgen.
- Deze puntenwolk vergelijken met eerder gescande objecten beschikbaar in een database en hiermee een match trachten te vinden.
- Uit deze match het zwaartepunt en oriëntatie bepalen om bin-picking mogelijk te maken.

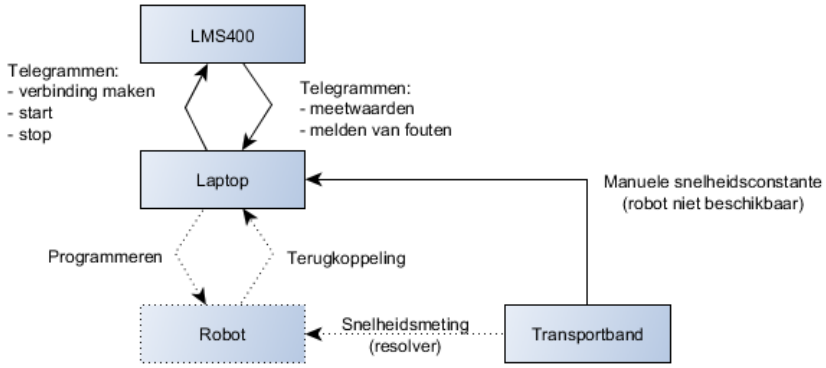
Ruis op de afstandsmeting van de LMS400 geeft een minimum onnauwkeurigheid van 3 mm. In het licht van deze randvoorwaarde zal getracht worden om de positie en grootte van het object tot op 5 mm nauwkeurig te bepalen, ongeacht de oriëntatie of ligging van het object op de transportband. Voor de nauwkeurigheid op de oriëntatie mag er maximaal een fout van 1 mm zijn bij een object met een lengte van 200 mm, dit komt neer op een maximale fout van 0,57° t.o.v. het middelpunt van het object.

Om de gewenste nauwkeurigheid te behalen is mogelijk een kalibratieprocedure vereist. Deze kalibratieprocedure geeft als voordeel dat bij een verplaatsing van de scanner de opstelling binnen enkele minuten opnieuw gebruiksklaar is.

Voor de objecten op de transportband gelden de volgende randvoorwaarden:

- de objecten mogen niet op elkaar liggen;
- de kortste zijde van het object mag niet meer dan 25 cm zijn;
- het materiaal mag geen hoge interferentie hebben (dit betekent dat een lichtstraal het object niet mag intreden om zodoende een foutieve meting te krijgen).

Materiaal en methode



Figuur 1: Opstelling van het gehele systeem

Voor het realiseren van dit project is door het stagebedrijf gekozen om een programma in .NET en in de programmeertaal C# te schrijven. De programmeerontwikkelomgeving Visual Studio. Visual Studio is hiervoor ideaal geschikt.

Het programma zal de scanwaarden in lezen en alle berekeningen uitvoeren. Deze berekeningen zijn:

- Het berekenen van de werkelijke afstanden.
- Het omvormen van de afstandsmetingen naar een puntenwolk.

De puntenwolken worden weergegeven in het .XYZ-formaat. Dit formaat is een opsomming van de punten met hun X-, Y- en Z-coördinaat.

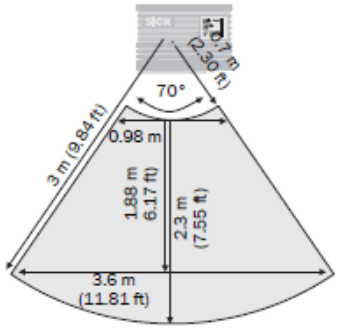
Voorbeeld:	-31	-62	11
	-34	-60	19
	-36	-59	18
	-31	-61	20
	-33	-60	32

Eens de puntenwolk gemaakt is kan een filter toegepast worden (bv: voor het verwijderen van de transportband en het verwijderen van eventuele foutieve metingen).

Om de puntenwolken te vergelijken is een fitting algoritme nodig, dit algoritme komt uit PCL (Point Cloud Library), wat een open-source algoritmebibliotheek is.
Momenteel wordt gebruik gemaakt van het “iterative closest point” algoritme. Dit algoritme legt twee puntenwolken op elkaar en berekent de afstand tussen samenliggende punten. De som van de afstanden is de matchscore. Wanneer een bestand met zichzelf vergeleken wordt is de score gelijk aan nul. Door ruis kunnen meerdere scans van hetzelfde object nooit 100% hetzelfde zijn en zal een score van 0 nooit bereikt worden.

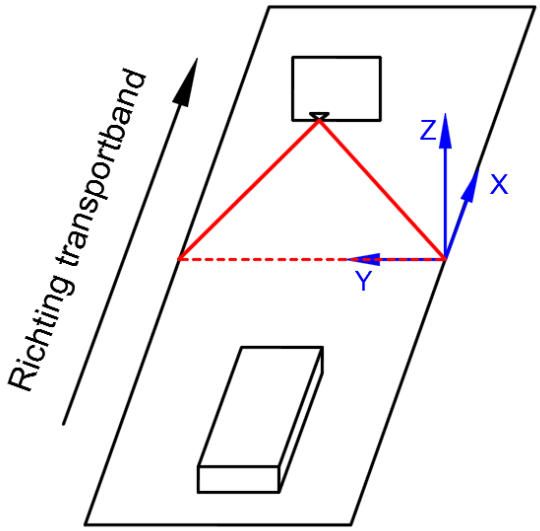
2 LMS400

De LMS400 is een lasermeetsysteem van SICK. Het meetbereik gaat van 0,7 tot 3 m zoals geschetst in figuur 2. Aangezien het om een lasersysteem gaat is het belangrijk dat de gebruiker nooit rechtstreeks in de laser kijkt.



Figuur 2: Meetbereik van de LMS400 [2]

Figuur 3 geeft een illustratie van de opstelling.



Figuur 3: Opstelling van de LMS400 en transportband

2.1 Verbinding met SOPAS

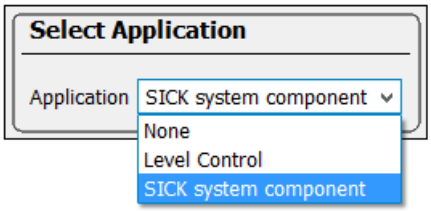
SOPAS is een programma van SICK dat toelaat de parameters van de LMS400 te configureren. Om parameters te configureren dient de gebruiker ingelogd te zijn. Dit kan onder de username “maintenance personnel” of onder “authorized client” waarvoor de wachtwoorden respectievelijk “main” en “client” zijn.

De eerste stap is het verbinden van SOPAS en het lasermeetsysteem. Dit kan over een seriële poort of over een ethernetinterface. In dit project is er gekozen voor een ethernetverbinding. Verbinding maken via SOPAS kan op verschillende manieren over een ethernet interface. Om verbinding te maken kan SOPAS rechtstreeks naar het IP-adres zoeken of tussen een bepaald bereik zoeken (bijvoorbeeld tussen 192.168.0.0 en 192.168.0.10). Indien er meerdere SICK apparaten op één netwerk aanwezig zijn kan SOPAS ook zoeken naar een specifieke familie van toestellen (bijvoorbeeld LMS4xx).

2.2 Configuratie met SOPAS

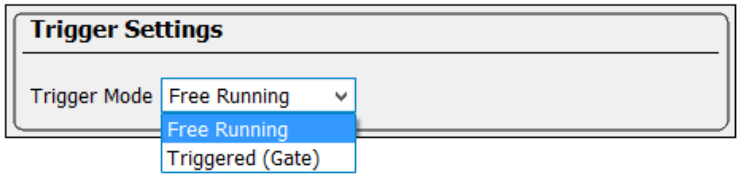
2.2.1 Applicatie

De LMS400 dient vooral voor “level control” waarbij de laserlijnscanner objecten kan herkennen aan de hand van de hoogte en/of breedte. In dit project vormt het Visual Studio programma een 3D-beeld door een object dat beweegt op een transportband continu te scannen. Om bij deze scans een hogere nauwkeurigheid te bekomen is het nodig om de applicatie in te stellen als “SICK system component” in plaats van “Level Control”. Figuur 4 geeft deze instelling weer.



Figuur 4: Keuze applicaties

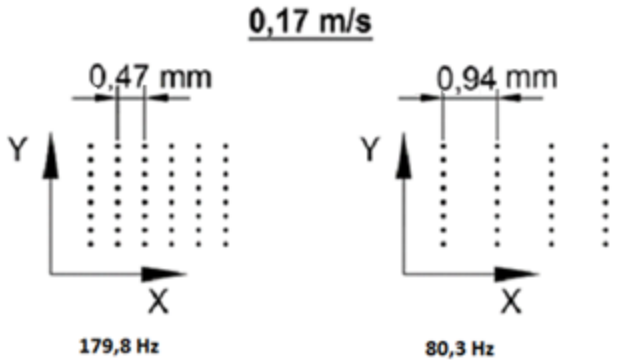
Indien de laser ingesteld stond op “Level control” dan kan het zijn dat de “Trigger settings” (figuur 5) nog als “Triggered” ingesteld is. Dit wil zeggen dat de scanner geen constante metingen maar wel één enkele pulsmeting zal uitvoeren. In de stand “Free Running” is het mogelijk om een scan continu uit te voeren. Een continue meting zal starten wanneer er een starttelegram ontvangen is en zal stoppen na ontvangst van een stoptelegram.



Figuur 5: "Trigger Settings" van SOPAS

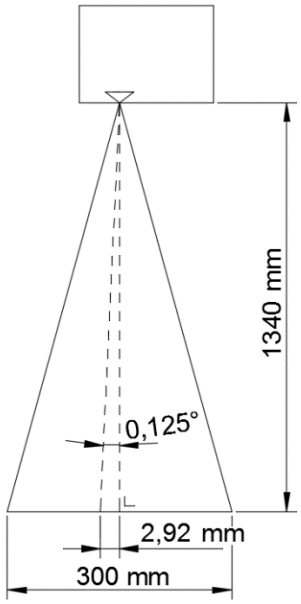
2.2.2 Parameters

Zodra de LMS400 verbonden is met SOPAS is het mogelijk om de parameters aan te passen. De eerste reeks parameters hebben betrekking op de positie en hoek van het meetsysteem ten opzichte van de transportband. Met behulp van de “scan frequency assistant” in SOPAS kunnen de scanfrequentie en de hoekresolutie ingesteld worden. De scanfrequentie bepaalt om de hoeveel tijd de laser een scan zal uitvoeren. Omdat de snelheid van de transportband constant is bepaalt enkel de scanfrequentie de nauwkeurigheid van de meting in de X-richting (dit is de richting van de transportband). Figuur 6 geeft een voorbeeld.



Figuur 6: Nauwkeurigheid in de X-richting bij twee verschillende scanfrequenties en gegeven transportbandsnelheid[A]

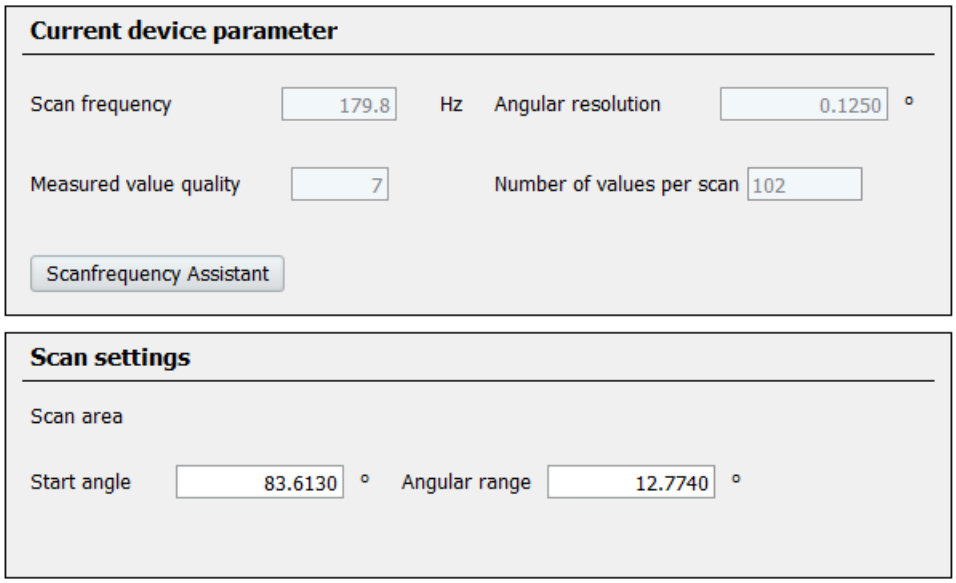
De gekozen scanfrequentie is 179,8 Hz. Dit wil zeggen dat de scanner om de 5,56 ms het meetgebied scant. De hoekresolutie geeft aan hoeveel graden er zich tussen ieder meetpunt bevindt, dit zal de nauwkeurigheid in de Y-richting bepalen. Hoe kleiner dit getal, hoe meer meetpunten het meetgebied zal bevatten. Er is gekozen voor een hoekresolutie van 0,125°. Figuur 7 schetst de actuele opstelling.



Figuur 7: Hoekresolutie en nauwkeurigheid Y-richting[A]

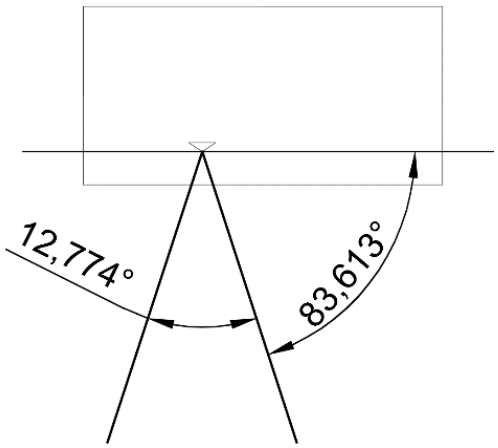
[A] Deze afbeelding is gemaakt door Bert Van Den Hende

De “measured value quality” is een waarde die aangeeft hoeveel tijd de scanner heeft om berekeningen uit te voeren. Bij een scanfrequentie van 179,8 Hz en een hoekresolutie van 0,125º bedraagt de “measured value quality” 7. Bij een waarde lager dan 7 kan verlies van nauwkeurigheid voorkomen. Figuur 8 geeft de scan settings weer.



Figuur 8: Scanfrequency assistant

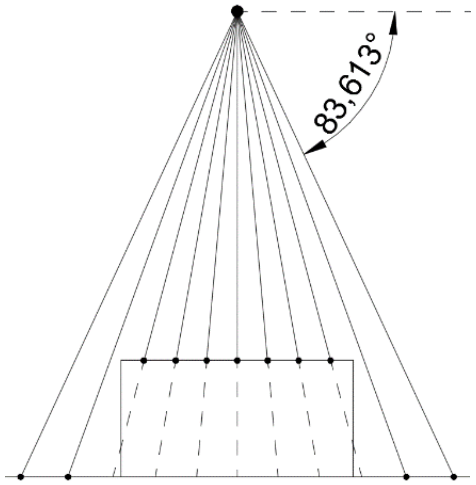
De starthoek is ingesteld op 83,61° waardoor het hoekbereik 12,77° is (figuur 9). Met een hoekresolutie van 0,125° wil dit zeggen dat er 102 (= 12,7740° / 0,125°) meetpunten zijn per scan.



Figuur 9: Hoekbereik en starthoek[A]

[A] Deze afbeelding is gemaakt door Bert Van Den Hende

Zoals figuur 10 aangeeft liggen de punten aan de rand van de transportband verder verwijderd van de LMS400, en zal de gemeten afstand aan de rand van de transportband dus groter zijn dan die in het midden.

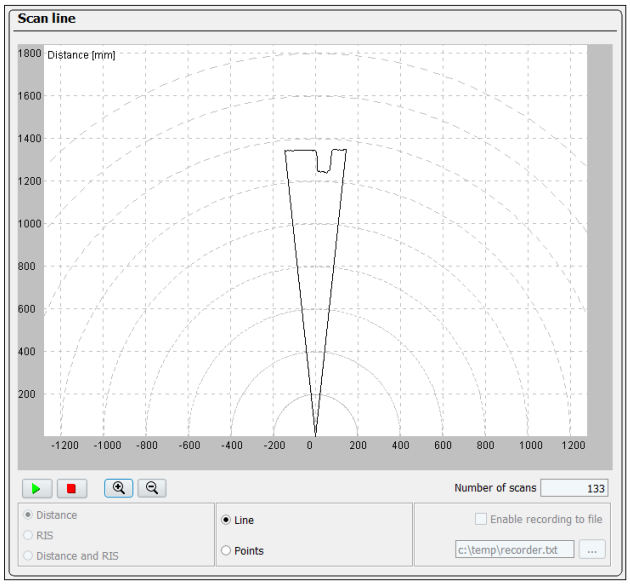


Figuur 10: Hoekbereik en starthoek

Dit wil ook zeggen dat de afstand tussen de meetpunten zal verschillen afhankelijk van hoe ver ze verwijderd zijn van het middelpunt van de transportband. Hoofdstuk 3.5.5 licht toe hoe het Visual Studio programma dit corrigeert.

2.2.3 Scan view

De scanviewfunctie (figuur 11) laat toe de correcte opstelling van de scanner te controleren.



Figuur 11: Scan view van een rechthoekig object

Daarnaast is er ook nog de optie om een datarecorder aan te zetten. Hierbij zal de LMS400 iedere 100 ms een scan uitvoeren en opslaan in een .txt bestand. Eventuele bewerkingen zijn mogelijk in Excel. Een gegeven object met een hoogte van 1250 mm resulteert in een visuele voorstelling van de scan zoals weergegeven in figuur 12.

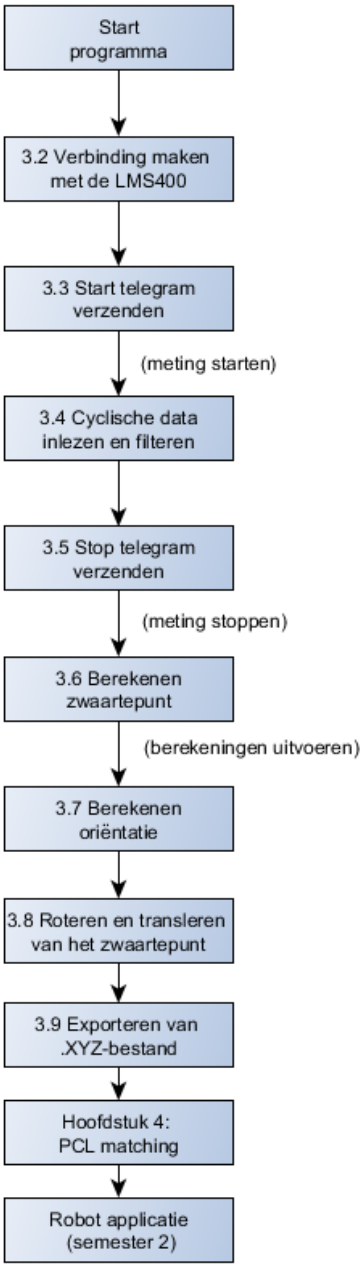
Figuur 12: Scan van een rechthoekige doos gevisualiseerd in Excel

3 Toelichting programma

3.1 Inleiding

Het in C# ontwikkelde softwareprogramma communiceert met de LMS400, verwerkt de meetgegevens en bepaalt type en positie van het 3D-object.

Figuur 13 geeft een beknopte samenvatting van dit programma. Dit hoofdstuk geeft bij iedere stap verdere toelichting.



Figuur 13: Samenvatting van het C# programma

Om functies in een programma tegelijkertijd uit te kunnen voeren moet gebruik gemaakt worden van Threading. Threading is het programmeren van de code zodat er verschillende functies parallel met elkaar kunnen werken. Vooraleer threading mogelijk is dient de functie toegevoegd te worden. Dit kan met de code “using System.Threading;”.

Eens het threaden is ingevoegd kan de functie geactiveerd worden met “invoke”.
In C# is dit “Invoke(Delegate)” of Invoke(Delegate,Object[]) maar omdat het programma in WPF (Windows Presentation Foundation) geschreven is kan deze functie niet gebruikt worden. Figuur 14 beschrijft het alternatief:

```
if (Dispatcher.Thread.Equals(Thread.CurrentThread))
{ uit te voeren programma }
else
{ this.Dispatcher.Invoke(new SocketActionHandler(NAME), new object[] { x }); }
```

Figuur 14: Invoke functie in WPF, C#

3.2 Verbinden met de LMS400

Om verbinding te maken met de LMS400 is gekozen voor asynchrone communicatie. Dit laat communicatie op een willekeurig tijdstip toe.

In C# is het mogelijk om gebruik te maken van de socket class (System.Net.Sockets.Socket). Voor asynchrone communicatie geeft dit de functies:

BeginConnect()	BeginReceive()
EndConnect()	EndReceive()

3.3 Starten van de meting

Bij het indrukken van de startknop zal het programma een starttelegram versturen. De exacte werking en opbouw van de starttelegram wordt uitgelegd in bijlage A. Figuur 15 geeft weer hoe de starttelegram verzonden wordt.

```
streamLMS = clientLMS.GetStream();

//Declaratie start telegram inclusief frames
byte[] sMN_mLRreqdata_21 = { 0x2, 0x2, 0x2, 0x2, 0x0, 0x0, 0x0, 0x11, 0x73, 0x4D,
                             0x4E, 0x20, 0x6D, 0x4C, 0x52, 0x72, 0x65, 0x71, 0x64,
                             0x61, 0x74, 0x61, 0x20, 0x32, 0x31, 0x76 };

//Versturen van de telegram inclusief frames
streamLMS.BeginWrite(sMN_mLRreqdata_21, 0, sMN_mLRreqdata_21.Length, written, null);
```

Figuur 15: Verzenden van een starttelegram

3.4 Inlezen en filteren van cyclische data

Als de LMS400 het starttelegram inleest dan zal de LMS400 vanaf dan continu metingen verzenden. In de methode “onDataReceive” worden telegrammen ingelezen en gefilterd om de meetwaardes in millimeters te verkrijgen. Dit gebeurt met volgende stappen:

1) Verplaats het telegram van LMSRecvBuffer naar RecvTelegram (byte array)

2) Controleer of deze array een grootte van meer dan 100 bytes heeft, om responsietelegrammen, die korter zijn, uit te sluiten.

3) Kopieer, indien het telegram meer dan 100 bytes bevat bevat, de array RecvTelegram (zonder het frame) naar RecvDistance (byte array).

4) Kopieer de waardes van RecvTelegram naar een integer array (intResponseArrObject). Deze array heeft dan als inhoud de meetwaardes van ieder meetpunt voor die scan.

5) Ga na of er minstens één meetwaarde in de array zit met een hoogte groter dan 10 mm (instelbare hoogte). Is dit niet het geval, dan is enkel de transportband opgemeten en gebeurt er verder niets met deze meetwaarde. Is dit wel het geval, dan is er een object gescand en kan het programma alle afstandsmetingen opslaan in de “afstandsMetingLijst”. Dit is een arraylijst waarin alle afstandsmetingen van de scan staan om later te verwerken.

Figuur 16 geeft weer hoe de cyclische data in het programma ingelezen en gefilterd wordt.

```
private void onDataReceive(IAsyncResult x)
{
    //callback procedure "OnDataReceive"

    if (Dispatcher.Thread.Equals(Thread.CurrentThread))
    {
        if (startButton.IsEnabled == true)
        {
            return;
        }

        //Aantal bytes in telegram(inclusief checksum exclusief startframe)
        int aantal = 0;
        aantal = streamLMS.EndRead(x);

        //Filteren
        byte[] RecvTelegram = new byte[(aantal - 1)];

        Array.Copy(LMSRecvBuffer, 0, RecvTelegram, 0, aantal - 1);


        if (aantal > 100)       //Dient om alleen afstandsmetingen op te slaan
        {
            bool booleanResponse;
            //Filteren
            byte[] RecvDistance = new byte[RecvTelegram.Length - 40];
            Array.Copy(RecvTelegram, 20, RecvDistance, 0, RecvDistance.Length);

            int AfstandRealtime = BitConverter.ToInt16(RecvDistance, 100);
            TextblockAfstandRT.Text = "Afstand in realtime: " + AfstandRealtime.ToString();


            bool[] bitsResponse = new bool[(RecvDistance.Length / 2)];
            int[] intResponseArrObject = new int[(RecvDistance.Length / 2)];
            int[] afstandsMetingArray = new int[(RecvDistance.Length / 2)];

            for (int i = 0; i <= RecvDistance.Length - 1; i += 2)
            {
                intResponseArrObject[(i / 2)] = BitConverter.ToUInt16(RecvDistance, i);
                if (intResponseArrObject[(i / 2)] <= intResponseArrConv102[(i / 2)] - 10 & intResponseArrObject[(i / 2)] > 0)
                {
                    booleanResponse = true;
                    eersteWaardeGevonden = true;
                }
                else
                {
                    booleanResponse = false;
                }
                bitsResponse[(i / 2)] = booleanResponse;
                afstandsMetingArray[i / 2] = intResponseArrObject[i / 2];
            }

            if (bitsResponse.Contains(true) || eersteWaardeGevonden==true)
            {
                sampleMatrix.Add(bitsResponse);
                afstandsMetingArray[0] = 1354;
                afstandsMetingLijst.Add(afstandsMetingArray);
            }
        }

        else if (aantal < 100)
        {
            //Confirmation and answer sensor weergeven
            //response = System.Text.Encoding.ASCII.GetString(Recvtelegram) & " "
        }
        //terug eerste 8 bytes inlezen om de lengte van de volgende telegram te vinden
        streamLMS.BeginRead(LMSRecvBuffer, 0, 8, received, null);
    }
    else
    {
        this.Dispatcher.Invoke(new SocketActionHandler(onDataReceive), new object[] { x });
    }
}
```

Figuur 16: Filteren van Meetwaardes

3.5 Stoppen van de meting

3.5.1 Stoptelegram

Het lasermeetsysteem zal meetdata blijven versturen totdat deze een stoptelegram ontvangt. Het versturen van de stoptelegram (figuur 17) kan men automatisch laten gebeuren of manueel (door het indrukken van de stopknop). De exacte werking en opbouw van de stoptelegram wordt uitgelegd in bijlage A.

```
//Stop telegram definiëren
byte[] sMN_mLRstopdata = { 0x2, 0x2, 0x2, 0x2, 0x0, 0x0, 0x0, 0xF, 0x73, 0x4D,
                           0x4E, 0x20, 0x6D, 0x4C, 0x52, 0x73, 0x74, 0x6F,
                           0x70, 0x64, 0x61, 0x74, 0x61, 0x2B };

//Stop telegram naar het lasermeetsysteem te schrijven
streamLMS.BeginWrite(sMN_mLRstopdata, 0, sMN_mLRstopdata.Length, written, null);
```

Figuur 17: Verzenden van een stoptelegram

Figuur 18 geeft weer hoe het programma de stoptelegram verstuurd wanneer er na het ontvangen van de eerste meetwaarde over een afstand van 10 cm geen object meer gedetecteerd wordt.

```
if (!bitsResponse.Contains(true) & eersteWaardeGevonden == true)
{
    afstandZonderMeetp += (1 / scanFrequency) * velConv * 1000;
    if(afstandZonderMeetp > 100)
    {
        StopProcedure();
    }
}
```

Figuur 18: Automatisch activeren van de stop procedure

3.5.2 Niets gemeten

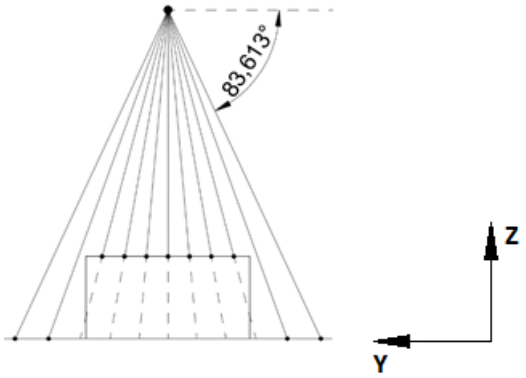
Wanneer de laserlijnscanner niets heeft gemeten dan is het nuttig om de berekeningen over te slaan. Hiervoor dient de “goto” functie (figuur 19).

```
if(eersteWaardeGevonden == false)
{
    goto NietsGemeten; //Mark1
}
```

Figuur 19: Overslaan programma indien niets gemeten

3.5.3 Werking van de LMS400 als afstandsmeter

De LMS400 is in wezen een afstandsmeter die de afstanden meet tussen de 102 meetpunten en het nulpunt van het lasermeetsysteem. De onderlinge afstanden tussen de meetpunten in de X-richting zijn vrij gemakkelijk te berekenen aangezien deze enkel afhankelijk zijn van de snelheid van de transportband en de scanfrequentie. De onderlinge afstanden in de Y- en Z-richting zijn afhankelijk van de hoogte van het object en de positie op de transportband.



Figuur 20: Principe meetsysteem Y- en Z-richting

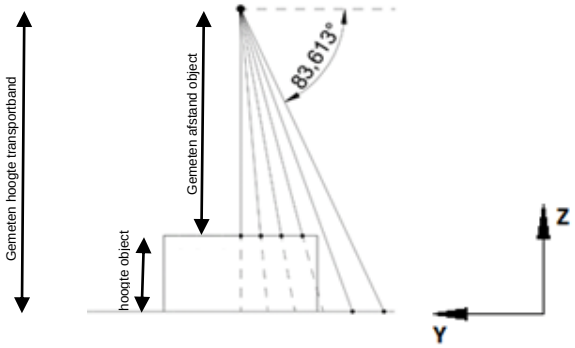
3.5.4 Bepaling van de hoogte volgens de Z-richting

Enkel bij het punt dat zich loodrecht onder de laserlijnscanner bevindt is de gemeten afstand gelijk aan de hoogte tussen het object en de laserscanner. De andere meetpunten zijn niet de hoogtemetingen maar wel enkel de afstand van het middelpunt tot de transportband. Om de werkelijke hoogte te weten moeten de schuin gemeten afstanden herrekend worden tot de hoogte. Daarvoor wordt de volgende formule gebruikt:

$$Z = \cos(0{,}125° \cdot i - (51 \cdot 0{,}125°))$$

Eens de verschillende hoogtes berekend zijn kan hier de werkelijke hoogte van het object uit berekend worden. De werkelijke hoogte van een object is dus volgens figuur 21:

$$Hoogte\ object = \text{Gemeten hoogte transportband} - \text{gemeten afstand object}$$



Figuur 21: Berekenen van de werkelijke hoogte

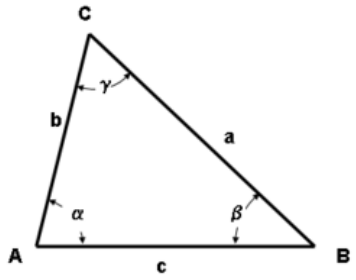
In de code gebeurt dit bij het exporteren van de puntenwolk als .XYZ-bestand

3.5.5 Corrigeren in de Y-richting

Wanneer de twee meetpunten zich op dezelfde hoogte bevinden kan de onderlinge afstand tussen de twee punten berekend worden met de cosinusregel aangezien de meting a en b gekend zijn en de hoekresolutie (op de tekening γ) 0,125° bedraagt.

Figuur 22 geeft de relatie tussen de drie zijden van een driehoek en de cosinus van een hoek weer.

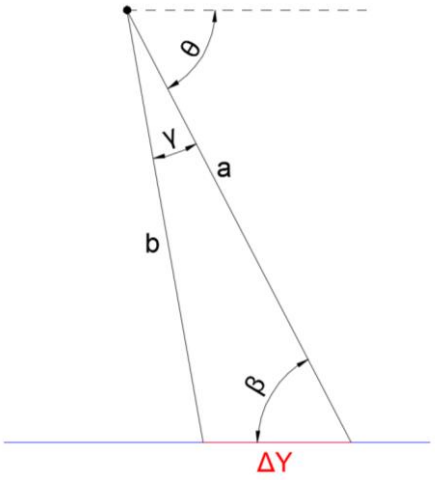
$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$
$$b^2 = c^2 + a^2 - 2ca\cos\beta$$
$$c^2 = a^2 + b^2 - 2ab\cos\gamma$$



Figuur 22: De cosinus regel en bijhorende formules [3]

Het toepassen van de cosinus regel bij de opstelling volgens figuur 23 geeft de volgende formule:

$$\Delta Y = \sqrt{a^2 + b^2 - 2.a.b.\cos(\gamma)}$$



Figuur 23: Berekening van ΔY indien er geen hoogteverschil is[A]

Ook de hoek β kan met de cosinusregel berekend worden:

$$\beta = cos^{-1}\left(\frac{\Delta Y^2 + a^2 - b^2}{2 \cdot \Delta Y \cdot a}\right)$$

[A] Deze afbeelding is gemaakt door Bert Van Den Hende

Indien de meetpunten zich niet op dezelfde hoogte bevinden (bijvoorbeeld één meetpunt op de transportband, het ander meetpunt op het object) zou dit een grote fout geven. Daarom zijn er extra berekeningen nodig om de werkelijke afstand tussen de twee meetpunten te weten.

De onderstaande situaties geven weer hoe de afstand ΔY berekend kan worden.
De hoek θ is afhankelijk van de rangorde van de meting en de beginhoek (83,613°).
$\theta = 83{,}613° + n \cdot 0{,}125°$

Wanneer c, β, θ en γ gekend zijn is het steeds mogelijk om ΔY in drie stappen te berekenen. Er zijn echter vier mogelijke situaties (figuur 24) waardoor de berekening telkens iets ander gebeurt.

Situatie 1 en 2: De fout bevindt zich aan de rechterkant van de transportband.

Situatie 1:
a < b θ < 90°

Situatie 2:
a > b θ < 90°

Situatie 3 en 4: De fout bevindt zich aan de linkerkant van de transportband.

Situatie 3:
a < b θ > 90°

Situatie 4:
a > b θ > 90°

Figuur 24: De vier situaties voor Y-afstanden[A]

Situatie 1 en 3 geeft de volgende formule: $\Delta Y = \big(sin(90° + \theta - \beta)\big) \cdot c$

Situatie 2 en 4 geeft de volgende formule: $\Delta Y = \big(cos(\theta - \beta)\big) \cdot c$

Dit kan beschreven worden als één algemene formule: $\Delta Y = abs\big(cos(\theta - \beta)\big) \cdot c$

[A] Deze afbeelding is gemaakt door Bert Van Den Hende

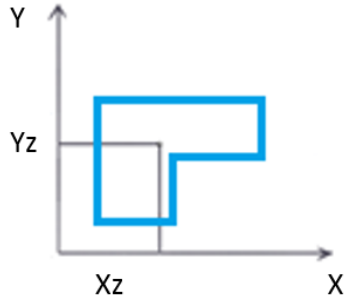
Figuur 25 geeft weer hoe het programma de correctie in de Y-richting berekent.

```
//Onderlinge Y-afstanden tussen de meetwaardes berekenen
double[] DeltaYKolom = new double[102];
DeltaYKolom[0] = 0;
for(int j = 0; j < afstandsMetingLijst.Count(); j++)
{
    for (int i = 1; i < 101; i++ )
    {
        int a = afstandsMetingLijst[j][i-1];
        int b = afstandsMetingLijst[j][i];
        Single gamma = 0.125F;                          //Hoekresolutie
        double gammaRad = gamma * Math.PI / 180;        //Hoekresolutie in Radialen
        Single theta = 83.613F + gamma * (i-1);
        double thetaRad = theta * Math.PI / 180;

        double c = Math.Sqrt(Math.Pow(a,2) + Math.Pow(b,2) - 2*a*b*Math.Cos(gammaRad));
        double betaRad = Math.Acos((Math.Pow(b,2) - Math.Pow(c,2) - Math.Pow(a,2))/(-2*c*a));

        DeltaYKolom[i] = Math.Abs(Math.Cos(thetaRad-betaRad)) * c;
    }
    afstandenTussenMeetp.Add(DeltaYKolom);
}
```

Figuur 25: Berekening van de correctie in de Y-richting

3.6 Bepalen van het zwaartepunt

Het zwaartepunt bepalen gebeurt ten opzichte van het nulpunt van de transportband.
Elk punt krijgt een massa m toegekend. De waarde x is de afstand van het nulpunt tot aan het punt zelf in de X-richting.



Figuur 26: Zwaartepunt van een willekeurige figuur

Aangezien er met puntenwolken gewerkt wordt is er geen fysieke massa. Daarom wordt voor de massa “1” gekozen. Dit geeft:

$$Xz = \frac{\sum m_i \cdot x_i}{\sum m_i} = \frac{\sum x_i}{\sum m_i}$$

Figuur 27 geeft weer hoe het programma dit berekent.

```
double huidigeAfstand2 = 0;
for (Xarr = 0; Xarr < afstandsMetingLijst.Count - 1; Xarr++)
{
    for (Yarr = 0; Yarr < 101; Yarr++)
    {
        if (intResponseArrConv102[Yarr] - afstandsMetingLijst[Xarr][Yarr] > filterBand)
        {
            sumX += huidigeAfstand2;
        }
    }
    huidigeAfstand2 += (1 / scanFrequency) * velConv * 1000;
}
double Xzmm = sumX / sumTot;
```

Figuur 27: Zwaartepunt in de X-richting

In de Y-richting volgt hetzelfde principe.

$$Yz = \frac{\sum m_i \cdot y_i}{\sum m_i} = \frac{\sum y_i}{\sum m_i}$$

```
int Xarr;
int Yarr;
double sumTot = 0;
double sumY = 0;
double sumX = 0;
int ArrLength = (int)sampleMatrix.Count;        //aantal scans
//Y-coordinaat van het zwaartepunt.
for (Xarr = 0; Xarr < afstandsMetingLijst.Count - 1; Xarr++) // 101 = ArrLength -1            //Yarr = 0; Yarr < 101; Yarr++
{
    double huidigeAfstand1 = 0;
    for (Yarr = 0; Yarr < 101; Yarr++)    //Xarr = 0; Xarr < afstandsMetingLijst.Count - 1; Xarr++
    {
        huidigeAfstand1 += afstandenTussenMeetp[Xarr][Yarr];
        if (intResponseArrConv102[Yarr] - afstandsMetingLijst[Xarr][Yarr] > filterBand)
        {
            sumY += huidigeAfstand1;
            sumTot += 1;
        }
    }
}
double Yzmm = (sumY / sumTot);
```

Figuur 28: Zwaartepunt in de Y-richting

3.7 Bepalen van de oriëntatie

Na het berekenen van de coördinaten van het zwaartepunt kan de oriëntatie berekend worden. Op basis van deze oriëntatie kan de puntenwolk gedraaid worden om sneller een match te vinden.

[4] geeft aan dat oriëntatie als volgt berekend kan worden:

$$\tan(2 \cdot \alpha) = \frac{2 \cdot Ixy}{Iyy - Ixx} \quad \rightarrow \quad \alpha = \frac{1}{2} \cdot \tan^{-1}\left(\frac{2 \cdot Ixy}{Iyy - Ixx}\right)$$

Figuur 29 geeft weer hoe het programma de oriëntatie bepaalt.

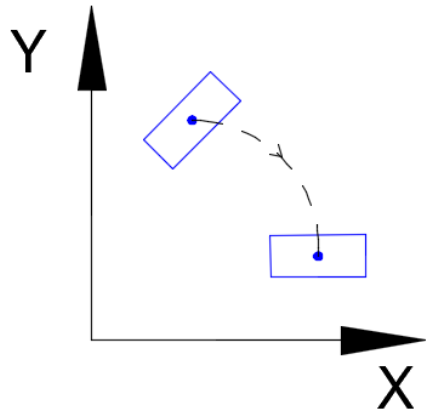
```
//Oriëntatie berekenen
double Xdistance = 0;
double Ydistance = 0;
double Ixx = 0;
double Iyy = 0;
double Ixy = 0;
double angle = 0;
for (int Xori = 0; Xori < afstandsMetingLijst.Count - 1; Xori++)
{
    for (int Yori = 0; Yori < 101; Yori++)
    {
        if (intResponseArrConv102[Yori] - afstandsMetingLijst[Xori][Yori] > filterBand)
        {
            Xdistance = (Xori * (1 / scanFrequency) * velConv * 1000) - Xzmm;
            Ydistance = 0;
            for (int i = 0; i < Yori; i++)
            {
                Ydistance += afstandenTussenMeetp[Xori][Yori];
            }
            Ydistance = Ydistance - Yzmm;

            //Massatraagheidsmomenten/producten
            Ixx += (Ydistance * Ydistance);
            Iyy += (Xdistance * Xdistance);
            Ixy += (Xdistance * Ydistance);

            //Oriëntatie van het object

            float hoekTangens = Convert.ToSingle(Math.Atan2(2 * Ixy, Iyy - Ixx));
            angle = -1 * Convert.ToSingle(0.5 * hoekTangens * (180F / Math.PI) + (Math.PI / 2));
        }
    }
}
```

Figuur 29: Berekenen van de oriëntatie

3.8 Roteren en transleren rond het zwaartepunt

Om bij het matchen een beter resultaat te krijgen wordt de scan geroteerd en getransleerd op basis van de eerder berekende oriëntatie en zwaartepunt. Uit tests blijkt dat het niet uit maakt welke actie eerst gebeurt, de verstreken tijd is nagenoeg hetzelfde. Het roteren en transleren geeft als voordeel dat elke scan van hetzelfde object zich op exact dezelfde positie bevindt, waardoor fouten direct zichtbaar zijn en het matchen sneller kan gebeuren.

Figuur 30 geeft het roteren van een scan weer:

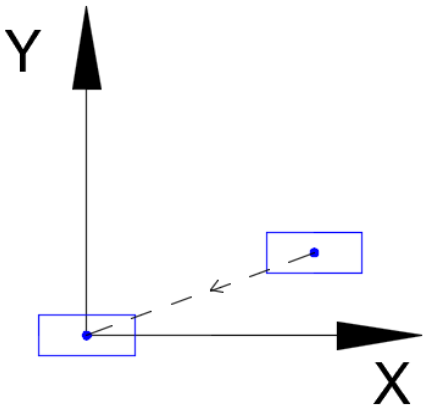


Figuur 30: Roteren van de scan[A]

```
//Zwaartepuntcoördinaten roteren (om later mee te transleren)
double XzwRot = Math.Cos((angle * Math.PI) / 180) * Xzmm - Math.Sin((angle * Math.PI) / 180) * Yzmm;
double YzwRot = Math.Sin((angle * Math.PI) / 180) * Xzmm + Math.Cos((angle * Math.PI) / 180) * Yzmm;
```

Figuur 31: Roteren van een scan (code)

Figuur 32 geeft het transleren van een scan weer:



Figuur 32: Transleren van de scan[A]

```
double xn = (Math.Cos((angle * Math.PI) / 180) * x - Math.Sin((angle * Math.PI) / 180) * y) - XzwRot;
double yn = (Math.Sin((angle * Math.PI) / 180) * x + Math.Cos((angle * Math.PI) / 180) * y) - YzwRot;
```

Figuur 33: Transleren van een scan (code)

[A] Deze afbeelding is gemaakt door Bert Van Den Hende

3.9 Exporteren puntenwolk naar XYZ-bestand

Meteen nadat het stoptelegram verstuurd is zal de opgeslagen data verwerkt worden. Eerst moeten de afstandsmetingen omgevormd worden naar een puntenwolk (figuur 34). Hierbij wordt voor ieder meetpunt een X-, Y- en Z-waarde berekend ten opzichte van het transportband-assenstelsel.

```
FileStream file = File.Create(filename);
StreamWriter writer = new StreamWriter(file);

for (int i = 0; i < afstandsMetingLijst.Count(); i++)
{
    double x = i * (1 / scanFrequency) * velConv * 1000;
    double y = 0;
    double z = 0;


    for (int j = 0; j < 101; j++)
    {
        y = y + afstandenTussenMeetp[i][j];

        double xn = (Math.Cos((angle * Math.PI) / 180) * x - Math.Sin((angle * Math.PI) / 180) * y) - XzwRot;
        double yn = (Math.Sin((angle * Math.PI) / 180) * x + Math.Cos((angle * Math.PI) / 180) * y) - YzwRot;

        z = intResponseArrConv102[j] - afstandsMetingLijst[i][j];
        if (z<1000 & z > filterBand)
        {       writer.WriteLine("{0} {1} {2}",
                xn.ToString(CultureInfo.InvariantCulture), //={0}
                yn.ToString(CultureInfo.InvariantCulture), //={1}
                z.ToString(CultureInfo.InvariantCulture)); //={2}
        }
    }
    // close file
}
writer.Close();
```

Figuur 34: Exporteren van XYZ-coördinaten als .XYZ-bestand

4 Matchen via PCL

Voor dit project is “PCL 1.7.2 All-in-one Installer MSVC2013 Win32” gebruikt.

Te downloaden op de volgende link:

http://unanancyowen.com/?p=1255&lang=en (laatst geraadpleegd op 19/12/15)

Na de installatie zijn eventueel ontbrekende .debug bestanden te vinden in:

C:\Program Files (x86)\PCL 1.7.2\bin

Indien er .debug bestanden ontbreken dan zal het programma dit aangeven door een foutmelding.

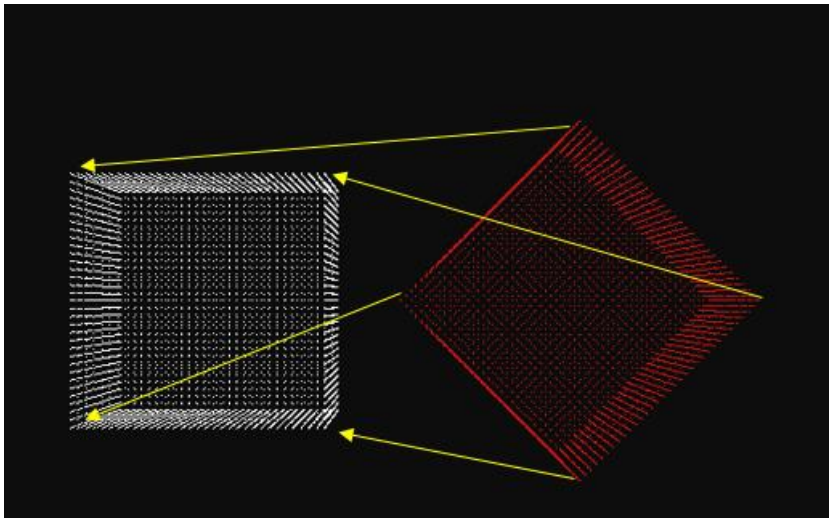
Alle .debug bestanden moeten zich bevinden in dezelfde folder als het matching algoritme.

Het matching algoritme kan gebruikt worden zonder de installatie van PCL maar indien de code gewijzigd dient te worden dan is het installeren van PCL 1.7.2 vereist.

4.1 Inleiding

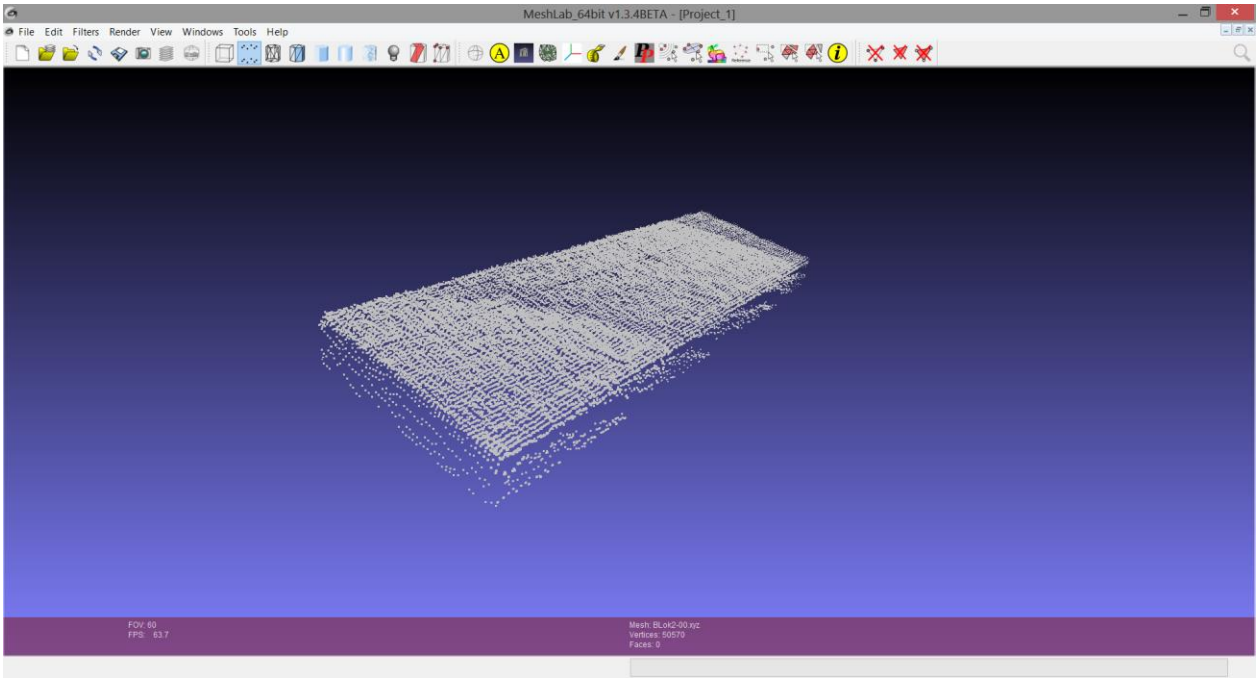
Het matchen gebeurt met het iterative closest point algoritme (= ICP). ICP wordt vooral gebruikt om een 3D-vlak te creëren uit meerdere scans maar in dit geval wordt het gebruikt om scans te vergelijken. Deze scans zijn de referentie- en bron-puntenwolken. Hierbij wordt de bron-puntenwolk gealigneerd met de referentie-puntenwolk. Het aligneren gebeurt met matrices en vectoren. De inschatting van rotatie en translatie kan met de gemiddelde kwadratische fout. Eens het aligneren gedaan is berekent het programma een score om aan te geven hoe goed te uitlijning verlopen is. Deze score is de som van de afstand tussen samenliggende punten. Een perfecte uitlijning zou een score van 0 geven.

Figuur 35 geeft weer hoe het algoritme twee identieke scans op elkaar legt om de tussenliggende afstanden te berekenen.



Figuur 35: Het matchen van kubussen [5]

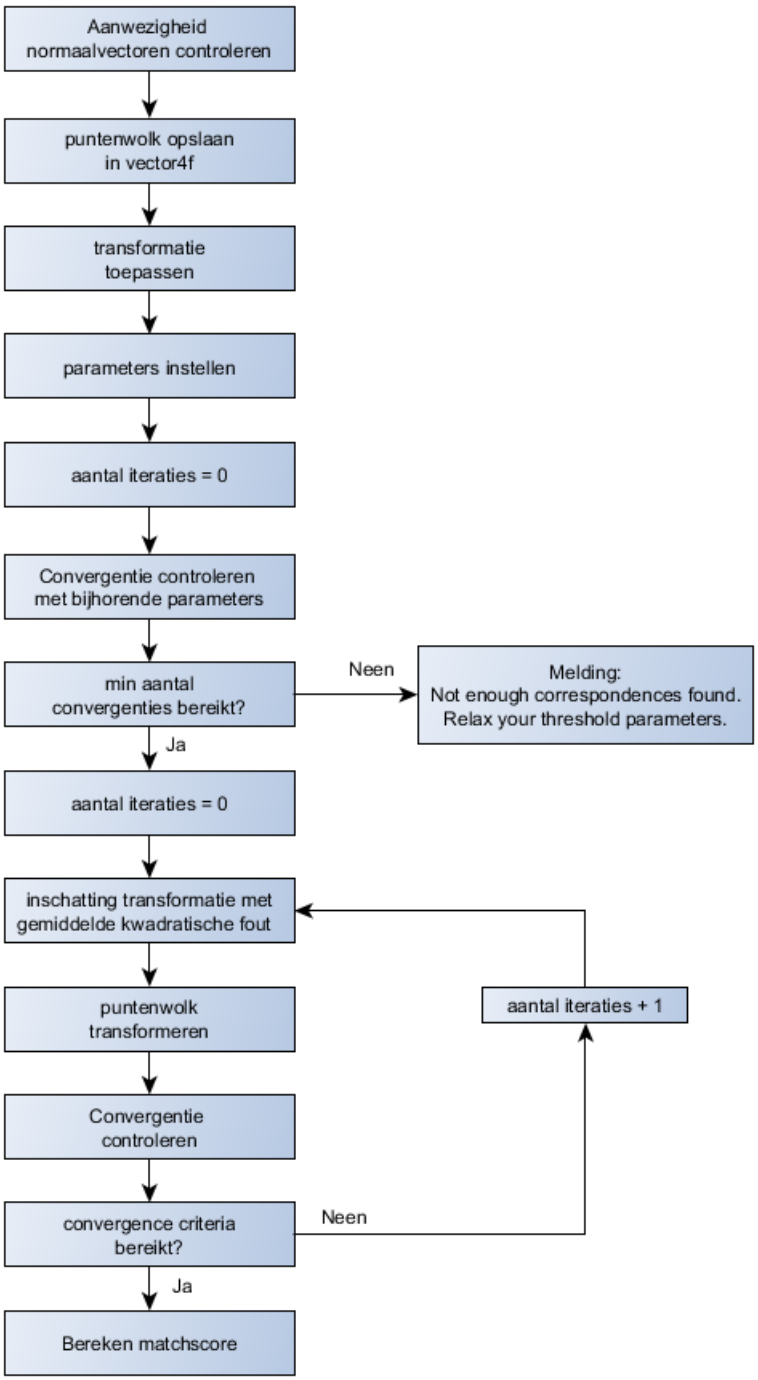
Het visualiseren van puntenwolken kan met het gratis verkrijgbaar programma MeshLab (figuur 36). Tevens kunnen hier bewerkingen in uitgevoerd worden om bijvoorbeeld enkel de zijdes te tonen.



Figuur 36: MeshLab visualisatie van puntenwolken

Eventuele bewerkingen zijn mogelijk in Excel en/of kladblok.

4.2 Werking van het ICP algoritme



Figuur 37: Flowchart over de werking van het ICP algoritme

Het programma gebruikt de term “Vector4f pt”, wat een vector van vier floating points is. Deze vier floating points zijn respectievelijk de vier dimensies: X, Y, Z en W. De vierde dimensie, W, is de schaalfactor. Door gebruik te maken van een schaalfactor kan een translatie uitgedrukt worden als een vermenigvuldiging in plaats van som. Het gebruik van deze schaalfactor zorgt ervoor dat matrices makkelijker te berekenen zijn.

Figuur 38 geeft weer hoe de vector en matrix geparametriseerd worden.

```
Eigen::Vector4f pt (0.0f, 0.0f, 0.0f, 1.0f), pt_t;
Eigen::Matrix4f tr = transform.template cast<float> ();
```

Figuur 38: Instellen van vector en matrix in het ICP algoritme

De eerste stap is het controleren of er zich normaalvectoren in het .XYZ-bestand bevinden. Bij het creëren van het .XYZ-bestand worden geen normaalvectoren toegevoegd dus zullen deze niet gevonden worden.

Na het controleren van de normaalvectoren wordt Vector4f ingevuld met elk punt in de puntenwolk door de functie memcpy (= memory copy).

```
memcpy (destination, source, size_t num );
```

Na het invullen van elk punt kan Vector4f getransformeerd worden door een vermenigvuldiging met Matrix4f (figuur 39).

```
for (size_t i = 0; i < input.size (); ++i)
{
  const uint8_t* data_in = reinterpret_cast<const uint8_t*> (&input[i]);
  uint8_t* data_out = reinterpret_cast<uint8_t*> (&output[i]);
  memcpy (&pt[0], data_in + x_idx_offset_, sizeof (float));
  memcpy (&pt[1], data_in + y_idx_offset_, sizeof (float));
  memcpy (&pt[2], data_in + z_idx_offset_, sizeof (float));

  if (!pcl_isfinite (pt[0]) || !pcl_isfinite (pt[1]) || !pcl_isfinite (pt[2]))
    continue;

  pt_t = tr * pt;

  memcpy (data_out + x_idx_offset_, &pt_t[0], sizeof (float));
  memcpy (data_out + y_idx_offset_, &pt_t[1], sizeof (float));
  memcpy (data_out + z_idx_offset_, &pt_t[2], sizeof (float));
}
```

Figuur 39: Het transformeren van de puntenwolk in het ICP algoritme

Deze eerste transformatie is de “gok transformatie”. Dit resultaat zal later gebruikt worden om te vergelijken of de volgende transformatie een beter resultaat geeft. Om volgende transformaties correct in te schatten maakt het algoritme gebruik van de gemiddelde kwadratische fout (= mean squared error). Het transformeren zal zich voortdoen tot het beëindigingscriteria is bereikt. Nadat één van de drie beëindigingscriteria bereikt zijn volgt de berekening van de matchscore.

De matchscore wordt opgeslagen in het bestand “RetVal.txt”, dit maakt het uitlezen van de matchscore in het C# programma mogelijk.

4.3 In te stellen parameters

De drie beëindigingscriteria van het ICP algoritme zijn de volgende:

1) Het maximum aantal iteraties bereiken
   In te stellen via setMaximumIterations.

2) De som van alle euclidische afstandsfuncties is kleiner dan de gedefinieerde waarde.
   Dit houdt in dat er een match gevonden is waarbij de som van alle afstanden lager is dan de ingestelde waarde. In te stellen via setEuclideanFitnessEpsilon.

3) De gemiddelde kwadratische fout is kleiner dan de gedefinieerde waarde.
   Dit houdt in dat er een match gevonden is wanneer de gemiddelde kwadratische fout (= “mean squared error”), tussen twee opvolgende transformaties, kleiner is dan de ingestelde waarde. In te stellen via setRelativeMSE.

Ook tijdens het transformeren zijn er drie criteria:

1) Het maximale epsilon verschil (= “the epsilon difference”)
   Dit is het maximale verschil dat zich tussen twee opeenvolgende transformaties kunnen voordoen. In te stellen via setTransformationEpsilon.

2) De translatie threshold (= “translation threshold”).
   Dit is het maximale verschil dat er zich tussen twee opeenvolgende transformaties kunnen voordoen in de translatie parameters.
   In te stellen via setTranslationThreshold.

3) De rotatie treshhold (= “rotation threshold cosine angle”)
   Dit is de maximale hoek dat er zich tussen een rotatie van twee opeenvolgende transformaties kan voordoen. In te stellen via setRotationThreshold.

Tijdens het berekenen van de matchscore zijn er twee parameters:

1) max correspondence distance
   Deze stelt de maximale afstand tussen twee punten in. Indien de afstand te groot is dan wordt het punt verwaarloosd om de score te berekenen. Dit geeft als voordeel dat foutieve punten niet meegerekend worden in de matchscore.

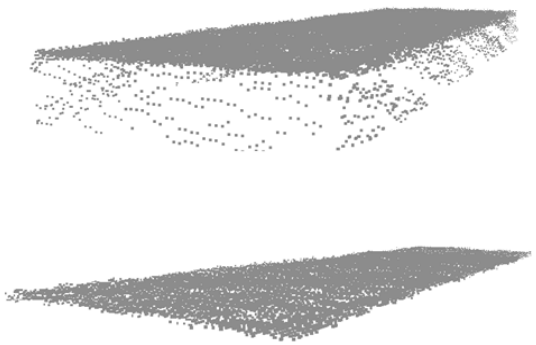
2) min_number_correspondences
   Indien er te weinig punten overeenkomen verschijnt de error:
   “Not enough correspondences found. Relax your threshold parameters."

5 Testresultaten

Om de nauwkeurigheid van het systeem te bepalen werden er tests uitgevoerd op houten blokken. Het gebruik van houten objecten vermijdt onnauwkeurigheden ten gevolge van interne reflectie.

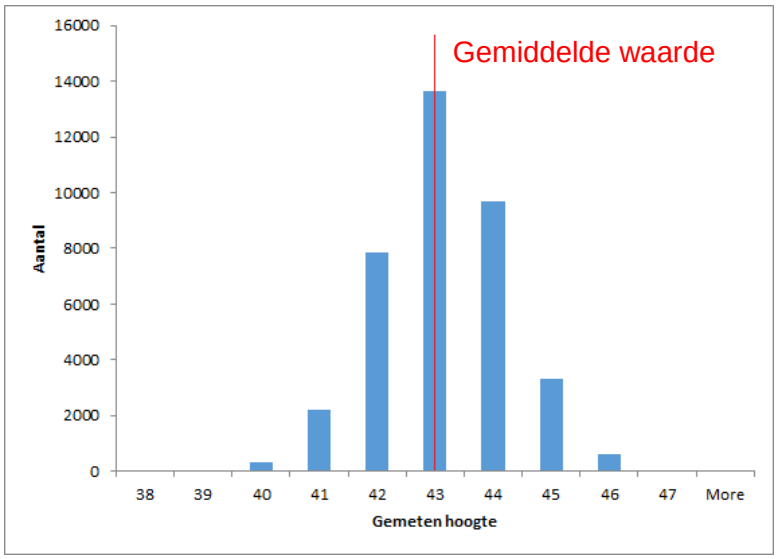
5.1 Nauwkeurigheid in de Z-richting (hoogte)

Figuur 40 toont de puntenwolk van de ingescande blok hout. Met behulp van het programma Meshlab worden de punten aan de zijkanten verwijderd, zodat zij geen invloed hebben op de berekening van de hoogte.



Figuur 40: Puntenwolken van een houten blok voor en na de bewerking

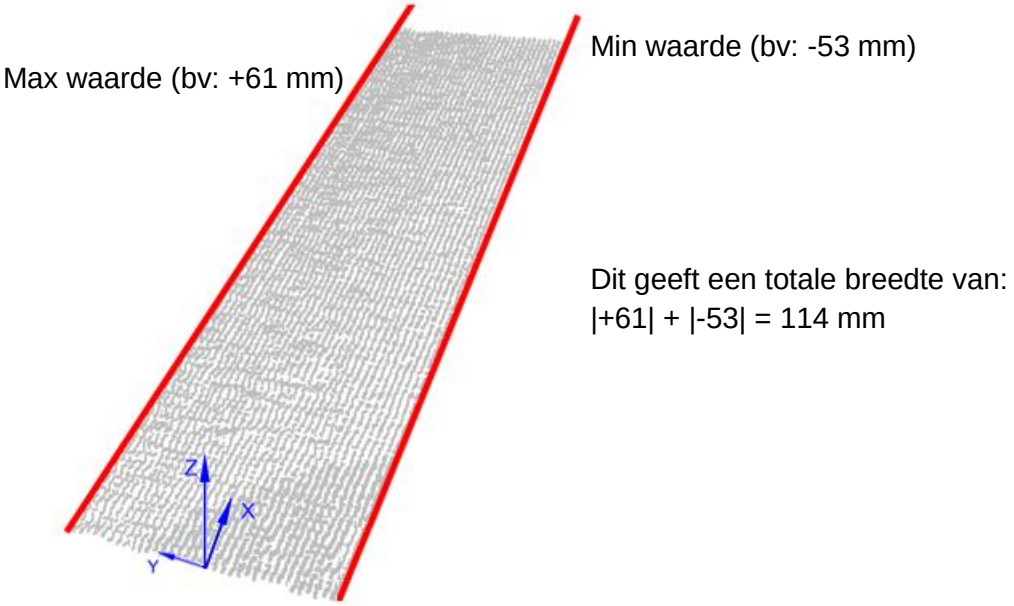
Figuur 41 geeft het histogram van de gemeten hoogte voor elk punt in de resterende puntenwolk. De werkelijke hoogte is 45 mm. De meest voorkomende en ook gemiddelde waarde is 43 mm. Deze scan heeft dus een onnauwkeurigheid van 2 mm.



Figuur 41: Histogram, meting blok van 45 *mm dikte*

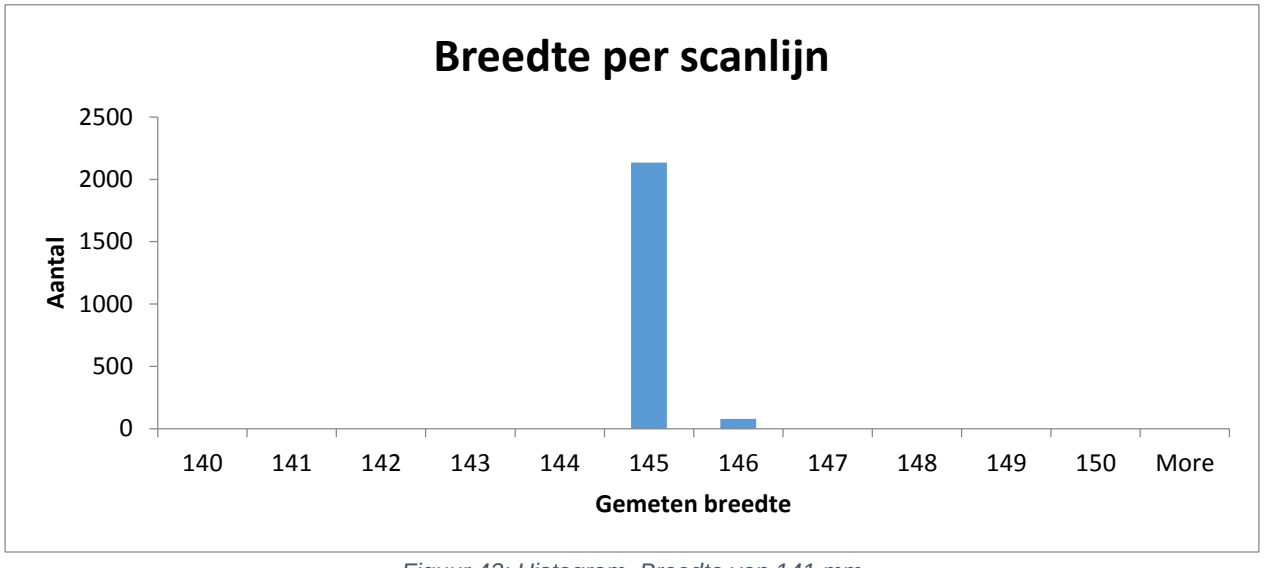
5.2 Nauwkeurigheid in de Y-richting (breedte)

De breedte van een object is gelijk aan het verschil van maximale en minimale Y-waarde. Figuur 42 geeft een illustratie hiervan.



Figuur 42: Minimum en maximum Y-waardes in een puntenwolk

De werkelijke breedte van blok hout bij deze test is 141 mm. De meest voorkomende gemeten breedte is 146 mm, zoals weergegeven in figuur 43. Dit geeft een fout van 4 mm op de breedte.

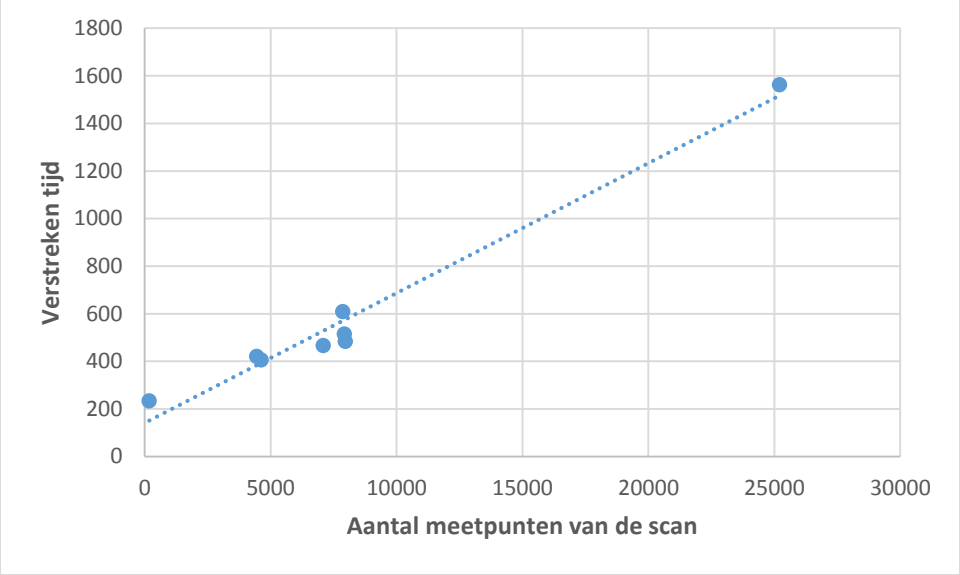


Figuur 43: Histogram, Breedte van 141 mm

Uit deze tests blijkt dat het programma consequent is in zijn berekeningen. De mogelijke systematische fout in de Y-richting, te bepalen met een extern meetsysteem, kan in een later stadium gecorrigeerd worden.

5.3 Duur van het matchen

De tijd die nodig is voor het matchen van een puntenwolk is afhankelijk van het aantal meetpunten. De meeste puntenwolken zijn niet veel groter dan 10 000 meetpunten, in dit geval is de scan zeker voltooid binnen één seconde. Figuur 44 geeft de rekentijd in functie van het aantal meetpunten weer. Bijvoorbeeld, voor het Ford-onderdeel waren dit er 7929.



Figuur 44: Meetpunten en de tijd nodig om te matchen

6 Besluit

Deze masterthesis test de geschiktheid van de LMS400, een time of flight laserlijnscanner, voor het herkennen van 3D-objecten.

De doelstelling, de mogelijkheid tot het herkennen van objecten ongeacht hun positie en oriëntatie, is aangetoond.
Uit de meetresultaten blijkt dat de nauwkeurigheid in de hoogte zeer acceptabel is, namelijk tot op 5 mm nauwkeurig. Het bepalen van de breedte gebeurt tot op 4 mm nauwkeurig. De vereiste absolute nauwkeurigheid van ± 5 mm is hiermee behaald.

De variatie op de breedtemeting van een rechthoekig blok is zeer klein (1 mm). Deze hoge relatieve nauwkeurigheid onderlijnt de validiteit van de gekozen werkingsmethode.

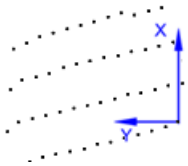
Voor het manipuleren van het object kan een gepast grijpmechanisme, bijvoorbeeld met zelfcentrerende grijppunten of vacuümdetectie, de resterende fouten omzeilen.

De absolute oriëntatie van het object en de absolute positie van het zwaartepunt van het object werden nog niet nauwkeurig bepaald. Hiervoor is immers een extern meetsysteem noodzakelijk.

Na de implementatie van een robot, kan het intern meetsysteem van de robot hiervoor gebruikt worden. Uit manuele controle van de meting van het zwaartepunt blijkt voorlopig een onnauwkeurigheid van ongeveer 8 mm t.o.v. het absolute nulpunt. Bij gebrek aan terugkoppeling van de positiemeting van de transportband, is ook de nauwkeurigheid in de lengterichting nog niet getest. In de voorliggende experimenten was de gebruikte verplaatsing van de transportband louter een schatting.

Ondanks de bovenstaande onzekerheden is het hoofddoel van deze masterproef, het implementeren van de LMS400 om 3D-objecten te herkennen, met succes behaald. Toch zijn er nog optimalisaties mogelijk en noodzakelijk.

- Zoals figuur 45 aangeeft, al de laserlijnscanner de bewegende objecten schuin inscannen. Dit moet nog in rekening gebracht worden met behulp van de snelheidsmeting van de transportband.



Figuur 45: Rechthoekig object ingescand onder hoge snelheid

- Als bij kalibratie van de opstelling en bij het gebruik van een extern meetsysteem blijkt dat er bij bepaalde objecten een systematische meetfout ontstaat, kan hiervoor alsnog gecompenseerd worden.

Ondanks de noodzakelijke optimalisaties zijn er al duidelijke voordelen:

- Omdat de bestanden in de database zelf ingescand worden bevinden de scans zich in ideale omstandigheden. Tevens wordt alleen de bovenkant van het object gescand waardoor de scans veel kleiner zijn. Wanneer deze zou vergeleken worden met een .stl-bestand dan zou er ook een vergelijking gebeuren met de zijdes die de laserlijnscanner niet kan zien. Dit zorgt voor een langere matchingsprocedure en kan zelfs voor fouten tijdens het matchen zorgen.
- Omdat het gehele programma in eigen beheer is gemaakt zijn aanpassingen altijd mogelijk.

Literatuurlijst

[1] KUKA. Bedrijfsprofiel. Geraadpleegd op 1 november 2015, http://www.kuka.be/main/company/profile/n_profile.htm

[2] SICK. (2013-04-09). Betriebsanleitung LMS400. Geraadpleegd op 28 augustus 2015, https://www.mysick.com/saqqara/im0010698.pdf#page=19

[3] Svdmolen (5 september 2004). driehoek-cosinusregel.png, Geraadpleegd op 29 december 2015, https://nl.wikipedia.org/wiki/Bestand:Driehoek-cosinusregel.png

[4] Meriam, J L. Kraige, L G. Wiley, J. (2006). Engineering Mechanics, Volume 1 – Statics. Geraadpleegd op 10 oktober 2015, http://www.eng.auburn.edu/~marghitu/MECH2110/C_4.pdf#page14

[5] VictorLamoine (4 februari 2014). "Matrix transformation example", Geraadpleegd op 8 december 2015. http://www.pcl-users.org/PCL-Visualiser-deformation-td4032137.html

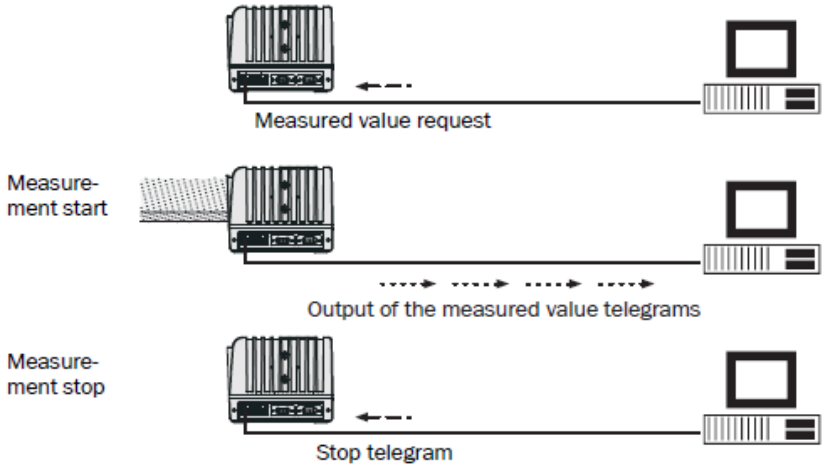
[6] SICK. (2013-04-09). Betriebsanleitung LMS400. Geraadpleegd op 28 augustus 2015, https://www.mysick.com/saqqara/im0010698.pdf#page=22

[7] SICK. (2013-04-09). Betriebsanleitung LMS400. Geraadpleegd op 3 september 2015, https://www.mysick.com/saqqara/im0010698.pdf#page=50

Bijlagen

BIJLAGE A: Communicatie tussen LMS400 en computer.

De LMS400 communiceert met behulp van ethernet telegrammen om metingen op te vragen of om parameters te wijzigen. De handleiding van de LMS400 licht alle mogelijke telegrammen toe. De parameters in dit project zijn éénmalig ingesteld met behulp van SOPAS, dit wil zeggen dat het instellen niet meer dient te gebeuren via het Visual Studio programma. Wat wel nog vereist is zijn het start- en stoptelegram.
Na het ontvangen van het starttelegram zal de LMS400 een bevestigingstelegram terugsturen die aangeeft dat deze bezig is met het starttelegram te verwerken. Zodra de LMS400 klaar is met het verwerken van het starttelegram stuurt deze opnieuw het telegram om aan te geven of er al dan niet een fout is voorgekomen (00000000 = no error).
Als er geen fout is dan zal de LMS400 om de 5,56 ms (1 / scanfrequentie) data verzenden met daarin de afstandsmetingen van ieder meetpunt. Dit zal aanhouden totdat de LMS400 een stoptelegram ontvangt.



Figuur 46: Opvragen van cyclische data [6]

Structuur van het telegram

In dit project worden de telegrammen en frames omgezet naar hexadecimale vorm zodat deze waardes rechtstreeks gebruikt kunnen worden. Het telegram is de boodschap die verstuurd moet worden, deze is steeds in een frame geplaatst. Het frame begint altijd met een startframe en eindigt met een stopframe. De structuur van deze twee zijn afhankelijk van het type interface, in dit geval ethernet.
De vier bytes na het startframe noemen STX, wat staat voor Start of Text.
“STX“ in ASCII is 02 (hexadecimaal). Deze vier bytes zien er als volgt uit: 02 02 02 02.
De vier volgende bytes geven de lengte van het telegram weer.
De laatste byte van het frame noemt de checksum. Dit is een controlecijfer om eventuele fouten te detecteren. De LMS400 berekent het controlecijfer door een XOR operator toe te passen op iedere byte van de data (zonder het frame).

	Frame						Telegram	Frame	
Code	TCP/IP Start Frame	STX	STX	STX	STX	Telegram length	Data (see *11.2 on page 80*)	Check-sum	TCP/IP Stop Frame
Length (byte)	Defined by the trans-mission	1	1	1	1	4	≤2495	1	Defined by the trans-mission
Description		Start of text character				Data length without CS, Motorola format	Binary encoded. The length is dependent on the previous send telegram.	See *"Calcu-lation of the check-sum"* further below	

Figuur 47: Telegram in een ethernet interface [7]

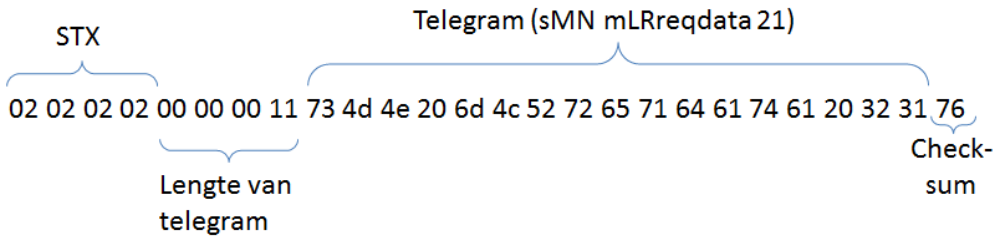
De handleiding geeft de opbouw van het telegram weer. Deze bestaat uit vier blokken:

- Definitie blok (20 bytes): geeft aan of er afstanden en/of remissie-waarden (= kwaliteit van de reflectie in een object) gemeten zijn, toont de schaal, starthoek, hoekresolutie, aantal meetpunten, scanfrequentie, remissie verschaling, remissie start- en eindwaarde.
- Meetwaardes (204 bytes oftewel 2 bytes per meetpunt): bevat de afstandsmetingen van ieder meetpunt (ook remissiemetingen als dit ingesteld is).
- I/O status (10 bytes): bevat informatie over eventuele digitale inputs.
- Sensor status (8 bytes): teller van het aantal keer dat de sensor gescand heeft, het aantal verstuurde telegrammen en de verstreken tijd bij tussen twee telegrammen.

Omdat in dit project enkel de afstandsmetingen van belang zijn filtert het Visual Studio programma de eerste 20 en de laatste 18 bytes weg. Hierna vormt het programma de byte-waardes om naar integers om de afstanden in millimeters te verkrijgen.

Start-telegram

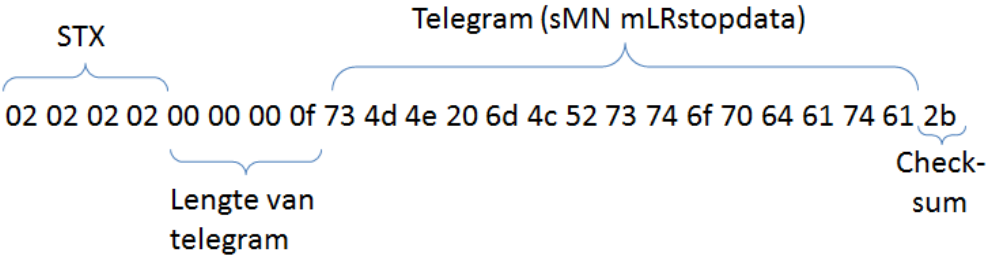
Het starttelegram heeft volgens de LMS400 handleiding als inhoud “sMN mLRreqdata 21”. De waarde 21 geeft aan dat het enkel om afstandsmetingen gaat. Na het omvormen van de ASCII-tekens naar hexadecimale getallen ziet dit er als volgt uit:



Figuur 48: Starttelegram met frame in hexadecimale vorm

Stop-telegram

Het stoptelegram (sMN mLRstopdata) ziet er als volgt uit:



Figuur 49: Stoptelegram met frame in hexadecimale vorm

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Laserscanmeting voor de herkenning van 3D-delen op een lopende band

Richting: **master in de industriële wetenschappen: energie-automatisering**
Jaar: **2016**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Vandekerkhof, Niels

Datum: **14/01/2016**