TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Processat de dades TLS (Terrestrial Laser Scanner)

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Elena Girones Llop

DIRECTOR: Michele Crosetto

SUPERVISOR: Jaume Piera Fernández

DATA: 28 de setembre de 2007

Títol del TFC: Processat de dades TLS (Terrestrial Laser Scanner)

Titulació: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

Autor: Elena Girones Llop

Director: Michele Crosetto

Supervisor: Jaume Piera Fernández

Data: 28 de setembre de 2007

Resum

En aquest treball de final de carrera, realitzat a l'Institut de Geomàtica, es fa una anàlisi dels núvols de punts obtinguts a partir d'un escàner làser terrestre (Terrestrial Laser Scanner – TLS). Aquests tipus d'aparells tenen la peculiaritat que per cadascuna de les captures que realitzen, emmagatzemen informació tant de distància com d'intensitat. El que es pretén és la realització de matching o aparellament d'objectes presents en núvols de punts diferents, realitzant primer una extracció dels mateixos i posteriorment efectuant el matching.

Pel tractament d'aquests núvols de punts s'han utilitzat diverses eines comercials com per exemple ENVI, així com eines desenvolupades al llarg del projecte amb codi C++.

Un dels objectius d'aquest projecte s'ha basat en l'extracció d'objectes tals com plans o esferes dels núvols de punts en qüestió. L'algoritme implementat per aquesta finalitat ha estat RANSAC, primer sobre núvols de punts simulats i després sobre imatges reals. Altres algoritmes, com CANNY per a la detecció de contorns, també han estats utilitzats.

Finalment es presenten diferents mètodes de matching existents per a identificar els mateixos objectes en núvols de punts diferents. D'entre tots els mètodes existents l'escollit ha estat el matching relacional, que utilitza a més de les pròpies característiques de les formes, relacions entre elles.

Title: TLS (Terrestrial Laser Scanner) data processing

Titulation: Technical Engineering of Telecomunication, speciality in Telematics

Author: Elena Girones Llop

Director: Michele Crosetto

Supervisor: Jaume Piera Fernández

Date: September, 28th 2007

Overview

In this work of final career, carried out in the Institute of Geomatics, is made an analysis of the point clouds obtained from a terrestrial laser scanner (Terrestrial Laser Scanner - TLS). These types of devices have the peculiarity that for each of the captures that they carry out, they store information of distance as well as of intensity. The goal is the realization of matching of objects present in different point clouds, carrying out first an extraction of the same ones and later making the matching.

For the treatment of these point clouds several commercial tools have been used like for example ENVI, and other tools developed along the project with C++ code.

One of the goals of this project has been based on the extraction of objects like plans or spheres of the point clouds. The algorithm implemented by this purpose has been RANSAC, first about simulated point clouds and afterwards about real ones. Other algorithms, like CANNY for the detection of contours, have also been used.

Finally different existing matching methods are shown to identify the same objects in different point clouds. Of among all the existing methods the select has been the relational matching, which uses, relations among them besides the characteristics themselves.

M’agradaria dedicar aquest TFC
als membres de l’Institut de Geomàtica
que m’han ajudat en la seva realització,
en especial al Michele Crosetto i
a l’Oriol Monserrat.

També a tots els companys i
amics amb els que m’he anat creuant
durant aquests 3 anys a l’EPSC;
especialment al Marc.

I sobretot a la meva mare.

A tots ells, gràcies.

ÍNDEX

INTRODUCCIÓ

La tècnica de l'escaneig làser utilitzant sensors terrestres permet adquirir quantitats massives de punts amb resolucions mil·limètriques, d'una forma més ràpida que les tècniques habituals, podent completar o substituir altres mètodes. Es basa en la utilització d'un dispositiu terrestre que fa servir un làser que mesura àngles, distàncies i intensitat dels punts il·luminats, de manera sistemàtica, a una tassa elevada i en temps reals. Com a resultat de l'adquisició de les dades, l'usuari obté un núvol de punts per cada exploració que representa l'àrea escanejada.

L'objectiu d'aquest treball es centra en primer lloc en l'anàlisi d'aquests núvols de punts amb la finalitat de reconèixer i extreure objectes, tals com plans, esferes, cons, etc. Posteriorment s'analitzaran diferents mètodes de matching que permetran relacionar cadascun d'aquests objectes amb els seus homòlegs a diferents núvols de punts. El document està estructurat en quatre capítols que es resumeixen breument a continuació.

En el primer capítol es dóna una visió general dels làser escàner terrestres (TLS) com a instruments de mesura. S'explica el seu principi bàsic de funcionament i es fa un resum dels diferents models que es poden trobar al mercat.

Al capítol dos, es fa referència al concepte de parametrització. L'objectiu és, donat un núvol de punts 3D, extreure objectes concrets com poden ser esferes o plans. A fi d'aconseguir aquest objectiu s'ha estudiat l'algoritme RANSAC. També es fa referència a un detector de contorns; el detector de contorns de Canny, com a pas previ a l'aplicació de RANSAC.

Posteriorment, al capítol 3 es mostren els resultats obtinguts en l'aplicació de RANSAC: primer en un núvol de punts simulat i després en un de real. S'ha aplicat aquest algoritme a l'extracció d'esferes i de plans.

Finalment al capítol 4 es parla de matching o aparellament d'imatges; es tracta de, donats dos núvols de punts, localitzar punts homòlegs entre aquests dos. De les diferents tècniques de matching s'ha escollit el matching relacional que utilitza a l'hora de fer l'aparellament, a més de les pròpies característiques de les formes, relacions entre elles.

Els annexos inclouen el codi utilitzat en la implementació dels algoritmes analitzats així com informació addicional.

CAPÍTOL 1. TERRESTRIAL LASER SCANNING (TLS)

L'*escaneig làser terrestre* conegut com a TLS és un mètode nou i eficient per a digitalitzar objectes grans o escenes senceres. Es basa en la utilització d'un dispositiu terrestre que fa servir un làser que mesura àngles, distàncies i intensitat dels punts il·luminats, de manera sistemàtica, a una tassa elevada i en temps reals. El resultat és un núvol de punts 3D que representa l'àrea escanejada. No és el substitut de tècniques existents, sinó una alternativa que gaudeix de certs avantatges com són velocitat i exactitud en les mesures. A dia d'avui hi ha un gran interès en el desenvolupament de les tècniques relacionades amb el TLS per a diferents tipus d'aplicacions com poden ser la reconstrucció i modelatge de dades 3D, l'anàlisi de deformacions, arquitectura, etc.

1.1 Instrumentació

Els TLS són aparells d'elevada precisió, capaços de treballar en diferents entorns sota condicions atmosfèriques adverses. Utilitzen mesures taquimètriques, que consisteixen en la combinació de la mesura de distàncies i angles. L'escàner escombra tot el seu camp visual (*Field of View, FoV*), variant la direcció del raig làser per tal de poder escanejar els diferents punts objecte de mesura. Això ho pot fer de dues maneres diferents: rotant el propi dispositiu o bé utilitzant un sistema de miralls rotatius. Aquest darrer mètode és el més utilitzat, ja que els miralls són més lleugers i poden girar més de pressa i amb una gran precisió. L'escaneig horitzontal s'anomena *frame scan* i pot tenir un FoV des de 40º fins a 360º. L'escaneig vertical és el *line scan* i pot assolir un FoV des de 40º fins a 310º. A la **Fig. 1.1** es pot veure la part fonamental d'un escàner làser terrestre.

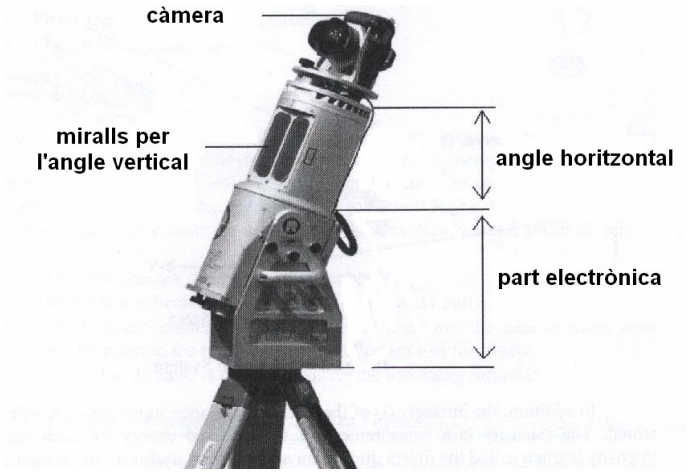


Fig. 1.1 TLS RIEGL LMS-Z420i

1.2 Principi de funcionament

El principi bàsic de funcionament del TLS [1] consisteix en la projecció d'un senyal òptic sobre un determinat objecte, i el corresponent processat de la senyal reflexada per a determinar-ne la distància a la que es troba; la precisió en les mesures de distància depèn de la intensitat d'aquesta última. Per cada senyal reflexada obtenim les següents mesures: dos angles corresponents a α i θ en la **Fig. 1.2**, la distància ρ i la intensitat.

Mitjançant els angles és capaç de definir la posició de cadascun dels punts de l'escena en un sistema de coordenades polars, que internament és transformat a un sistema cartesià utilitzant les equacions **(1.1)**, **(1.2)** i **(1.3)**.

$$x = \rho \cdot \cos\alpha \cdot \sin\theta \quad \textbf{(1.1)}$$

$$y = \rho \cdot \cos\alpha \cdot \cos\theta \quad \textbf{(1.2)}$$

$$z = \rho \cdot \sin\alpha \quad \textbf{(1.3)}$$

On:

- α, θ: angles en coordenades polars
- ρ: distància en coordenades polars
- X, Y, Z: coordenades cartesianes

A més a més, la intensitat de la senyal retornada és també emmagatzemada. La intensitat és una mesura de l'energia rebuda per cada punt. Amb tot, el TLS crea un núvol de punts on cada punt queda determinat per la seva posició (X, Y, Z) tal com es mostra a **Fig. 1.3, 1.4** i **1.5** i la intensitat (i) **Fig. 1.6**.

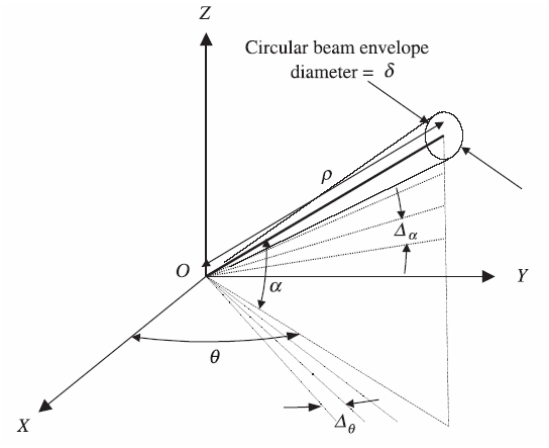


Fig. 1.2 Coordenades polars d'un punt qualsevol mesurat amb TLS

Fig. 1.3 Component X
núvol de punts

Fig. 1.4 Component Y
núvol de punts

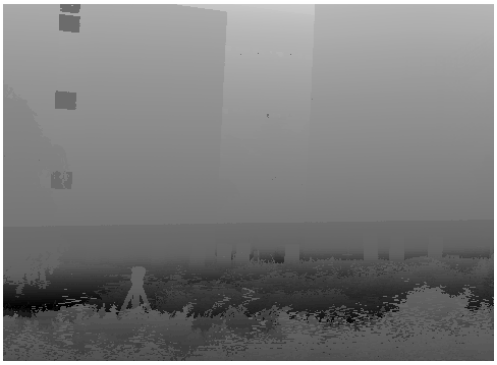


Fig. 1.5 Component Z
núvol de punts

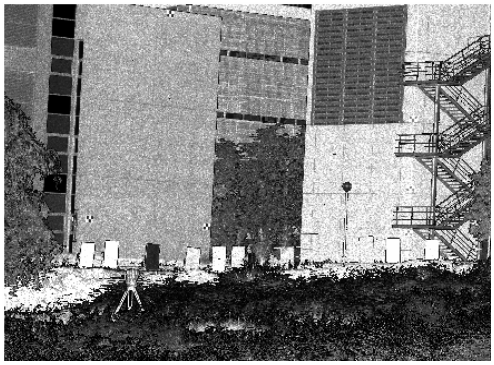


Fig. 1.6 Component *i*
núvol de punts

1.3 Classificació

És difícil fer una classificació de TLS. Hi ha moltes maneres de fer-ho, basades en el principi de mesura o en especificacions tècniques. Primer de tot val a dir que no hi ha un escàner làser terrestre universal per a totes les aplicacions. Alguns són específics per interiors i rangs mitjans, d'altres són útils per exteriors i rangs més amplis i també n'hi ha d'altres per aplicacions d'alguns pocs metres amb una precisió molt elevada. Depenent de l'aplicació doncs, n'escollirem un o bé un altre.

A continuació s'exposen 2 tipus de classificacions diferents: una primera basada en el principi de mesura de distància i una altra que té en compte la deflexió del raig làser.

1.3.1 Principi de mesura de distància

Té en compte tant el rang com la precisió del sistema. La precisió de les mesures de distància depèn de la intensitat de la llum reflectida i, per tant, directament de la reflectivitat de l'objecte. En podrem distingir tres tecnologies.

1.3.1.1 Temps de vol (Time of flight, TOF)

És el sistema de mesura més popular. Utilitza el temps que triga un pols d'energia en viatjar des del transmissor a l'objecte observat, i tornar. La llum és utilitzada com a font d'energia. La tècnica que utilitza és la següent:

- L'escàner genera el pols òptic i el llença cap a l'objecte que es vol mesurar
- Aquest pols es reflexat per l'objecte observat i retorna cap a ell
- Un comptador de gran velocitat situat a l'escàner mesura el temps de vol transcorregut des que s'ha llençat el pols inicial, fins que ha tornat
- Finalment, la distància es calcula aplicant **(1.4)**

$$d = \frac{c \cdot TOF}{2} \qquad \textbf{(1.4)}$$

On:

- d: la distància en metres entre l'emissor i l'objecte mesurat
- c: velocitat de la llum ($3 \cdot 10^8$ m/s)
- TOF: temps de vol. El temps TOF mesurat representa dos cops la distància; aquest és el motiu pel qual es divideix entre 2

L'avantatge d'aquest mètode és que és capaç d'operar sobre grans distàncies; de l'ordre de quilòmetres. Però com a inconvenient, té una precisió escassa per a determinades aplicacions.

1.3.1.2 Interferometria (diferència de fase)

Per a determinar la distància s'utilitza la diferència de fase entre la senyal emesa i la rebuda. Una ona sinusoïdal de freqüència *f* modula el corrent del díode làser emissor. Després de la reflexió, un fotodíode recull una part del raig làser. D'aquesta manera la distància és deduïda de la diferència de fase entre les dues senyals i es calcula aplicant l'equació **(1.5)**.

$$d = \frac{1}{2} c \frac{\Delta\varphi}{2\pi} f \quad \textbf{(1.5)}$$

$$\Delta\varphi = 2\pi f \Delta t \quad \textbf{(1.6)}$$

On:

- d: distància en metres entre l'emissor i l'objecte mesurat
- c: velocitat de la llum ($3 \cdot 10^8$ m/s)
- f: freqüència moduladora del corrent del díode làser emissor
- Δφ: diferència de fase entre la senyal emesa i la rebuda, calculada tal i com mostra l'equació **(1.6)**

Es tracta d'un bon mètode per a obtenir una precisió d'alguns mil·límetres, per a rangs per sobre del centenar de metres.

1.3.1.3 Triangulació òptica

El sistema de triangulació òptica utilitza un emissor de llum làser, una lent i un *fotodetector* per a calcular la distància que hi ha respecte un objecte determinat, situats tal i com es mostra a la **Fig. 1.7**.

On:

- x: distància entre el fotodetector i l'emissor làser
- α: angle de l'emissor làser
- θ: angle del fotodetector
- β: angle d'incidència calculat amb **(1.7)**
- d: distància des de l'objectiu al TLS

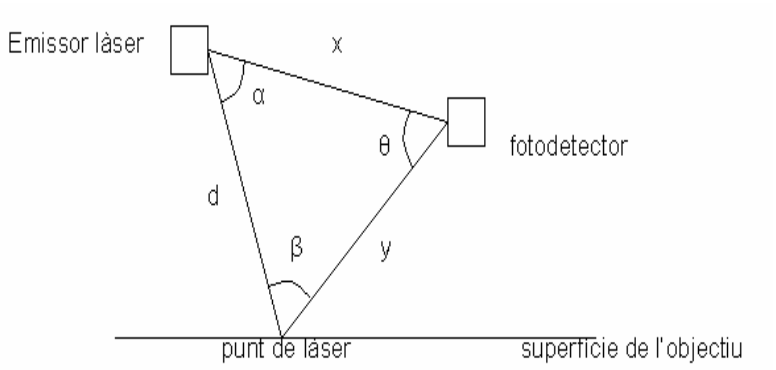


Fig. 1.7 Disposició dels elements que intervenen en un sistema de mesura basat en la triangulació òptica

Un làser és utilitzat per a generar un raig de llum enfocat a la superfície de l'objectiu. Una lent captura la llum reflexada i la focalitza en un fotodetector. La distància entre el làser emissor i l'objecte en qüestió es calcula utilitzant trigonometria:

$$\beta = 180° - \alpha - \theta \tag{1.7}$$

$$d = \frac{x \cdot \sin\theta}{\sin\beta} \tag{1.8}$$

Per tal de calcular la distància *d* apliquem el Teorema del Sinus **(1.8)**.

El component més important en la triangulació òptica és el fotodetector. N'existeixen dos tipus diferents segons l'aplicació de la mesura:

- PSD (*Position Sensitive Detector*): només determina la posició del punt mesurat i la seva intensitat total
- CCD (*Charged Coupled Device*): poden ser utilitzats per a determinar la forma i la distribució de la intensitat del punt de llum al fotodetector

El rang de mesura que abasta aquest mètode queda limitat a alguns metres. Però per contra, la precisió és relativament elevada, de l'ordre de desenes de micròmetres.

Sistema de mesura	Rang [m]	Rang de precisió [mm]	Fabricants (exemples)
TOF	~ 1000	>10	Mensi, Riegl, Cyra, Callidus
Interferometria	< 100	<10	Zoller+Froehlich, IQSun
Triangulació òptica	< 10	< 1	Minolta, Leica

Taula 1.1 Classificació dels TLS segons el principi de mesura de distància [2]

1.3.2 Deflexió del raig làser

Si classifiquem els TLS depenent de la deflexió del raig làser obtenim els següents tipus d'aparells **Fig. 1.8**:

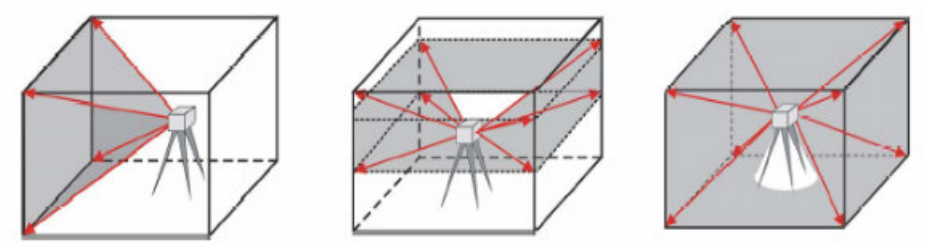


Fig. 1.8 Escàners làser terrestres de visió de càmera, visió híbrida i visió panoràmica, respectivament

- Visió de càmera: Utilitza dos miralls sincronitzats, un per a la deflexió del raig horitzontal i l'altre per a la del raig vertical. Els dos miralls es poden posicionar individualment i les mesures poden ser obtingudes amb una elevada precisió. El FoV està limitat a 60°x60°

- Visió panoràmica: Un únic mirall rotatiu que simultàniament fa girar el sistema tant horitzontal com verticalment. El FoV queda limitat pel suport que sosté el TLS

- Híbrida: Una de les dues rotacions no té cap tipus de limitació mentre que l'altra queda limitada pels miralls

També podem efectuar classificacions tenint en compte altres especificacions tècniques. De les més destacades podem citar la velocitat d'escaneig, la resolució espacial o la combinació amb d'altres dispositius muntats sobre el TLS.

1.4 Disponibilitat al mercat

El mercat dels escàners làser per aplicacions terrestres s'ha desenvolupat amb èxit al llarg dels darrers anys. A dia d'avui nombroses companyies ofereixen productes al mercat. Una comparació directa d'aquests productes és difícil a causa de les especificacions tècniques dels mateixos.

Els canvis més significatius que han sofert els TLS al llarg dels últims anys han estat enfocats cap a l'increment de la precisió i velocitat de les mesures, l'extensió del FoV i l'augment de la taxa de mostreig de les dades. Alguns d'aquests aparells que podem trobar a dia d'avui al mercat, junt amb algunes de les característiques comentades anteriorment, són els que es mostren a la **Taula 1.2**.

Els aparells escollits són els més avançats del mercat pel que fa a les característiques considerades:

- Respecte a la longitud d'ona, expressada en nanòmetres, veiem com varia d'uns fabricants als altres prenent valors que oscil·len des dels 532 nm. als infrarojos
- Les dues principals tècniques de mesura de distància utilitzades són la interferometria i la TOF, sent aquesta última la més comuna
- El rang de mesura oscil·la des dels 0.1m als 1500m
- El FoV vertical varia d'un fabricant a un altre dels 60˚ als 320˚ mentre que en l'horitzontal tots abracen els 360˚

Fabricant [model]	Imatge	Longitud d'ona del làser [nm]	Tècnica de mesura utilitzada	Rang mín./màx. [m]	FoV [Vertical x Horitzontal]
3rdTech *DeltaSphere-3000IR*		780	TOF	0.5/15.0	290˚x360˚
CALLIDUS *CP 3200*		905	TOF	1/80	140˚x360˚
I-SiTE *I-SiTE 4400*		905	TOF	2/400	80˚x360˚
Qsun *IQsun 880*		785	Interferometria	0.1/76	320˚x360˚
Leica Geosystems HDS *HDS 4500 (25m)*		Infrarojos	Interferometria	0.38/25.2	310˚x360˚
Optech *ILRIS-3D*		1550	TOF	3/1500	180˚x360˚
RIEGL *RIEGL LMS-Z420i*		Prop dels infrarojos	TOF	2/800	80˚x360˚
Trimble Navigation *Trimble GS*		532	TOF	1/100	60˚x360˚
Z+F *Imager 5003-53500*		780	Interferometria	1.0/53.5	320˚x360˚

Taula 1.2 TLS disponibles al mercat

A l'**Annex 1** es poden trobar referències dels fabricants anteriors.

CAPÍTOL 2. DETECCIÓ I EXTRACCIÓ D'OBJECTES EN NÚVOLS DE PUNTS TLS

L'objectiu perseguit en aquest capítol és la detecció d'objectes, tals com plans o esferes, en núvols de punts que ens proporciona l'escàner làser terrestre. Tal i com hem vist al capítol anterior, aquest aparell ens ofereix, per cada captura, informació 3D de la mateixa. L'esquema utilitzat serà el següent:

- Aplicació d'un detector de contorns per tal de poder extreure els contorns de les figures que composen el núvol de punts
- Aplicació d'un algoritme de parametrització que permeti obtenir els paràmetres que determinen els objectes

Per tal de ser útil a la pràctica, l'algoritme implementat ha de ser capaç de detectar petits detalls, ser robust enfront soroll i outliers, i ser lo suficientment eficient per a donar una resposta ràpida a l'usuari.

2.1 Detecció de contorns.

Un *contorn* és un canvi significatiu en el paràmetre estudiat: intensitat, distància, etc. La seva detecció és útil, per exemple, quan s'han de trobar característiques puntuals de determinats objectes, o quan fa falta l'establiment de límits entre aquests. Un detector de contorns serà doncs, un algoritme que obtingui un conjunt de contorns a partir d'una imatge tal com es mostra a la **Fig. 2.1**.

Els passos que segueix un algoritme d'aquest tipus són els següents:

- Filtrar per tal de ressaltar canvis en la intensitat: *suavitzat* i *realçat* **Fig. 2.2**.

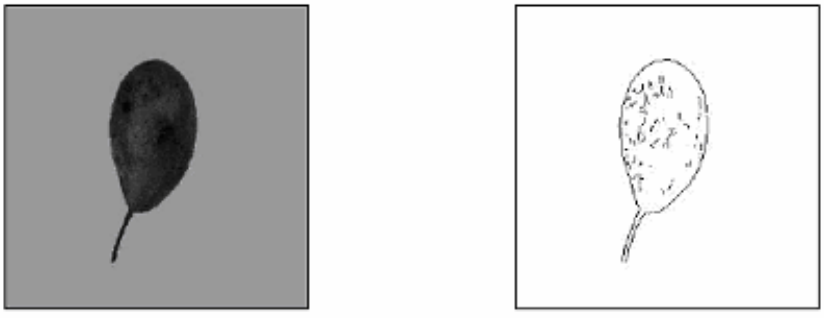


Fig. 2.1 Exemple d'aplicació d'un algoritme de detecció de contorns; a partir d'una imatge (imatge esquerra) obtenim els contorns de la mateixa (imatge dreta)

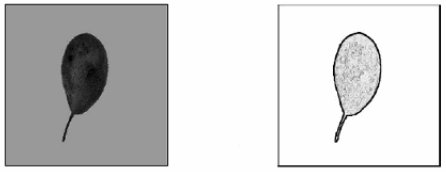


Fig. 2.2 Imatge original i imatge filtrada per tal de ressaltar canvis en la intensitat, respectivament

- Localitzar els contorns: *detecció* o *localització* (decidir si és contorn o errors: falsos contorns, contorns perduts) **Fig. 2.3**.

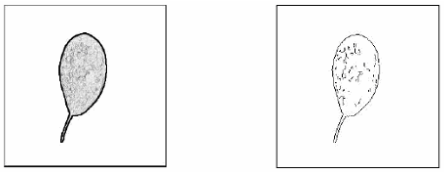


Fig. 2.3 Imatge suavitzada i imatge corresponent únicament als contorns, respectivament

2.1.1 Tipus de detectors

Podem trobar dos tipus de detectors de contorns: els que es basen en el *gradient* i els que es basen en la *laplaciana*. El gradient mesura un canvi local; una diferència de distància, d'intensitat, etc. És la primera derivada de f **(2.1)**:

$$\nabla f(x) = \frac{\partial f}{\partial x} \qquad \textbf{(2.1)}$$

La magnitud del gradient **(2.2)** ens proporciona la força del contorn i l'orientació **(2.3)** ens dóna informació sobre la direcció del mateix:

$$\left|\nabla f(x,y)\right| = m(x,y) = \sqrt{f_x^2 + f_y^2} \qquad \textbf{(2.2)}$$

$$\phi(x,y) = \arctan\frac{f_y}{f_x} \qquad \textbf{(2.3)}$$

Mitjançant l'aplicació del gradient s'identifiquen els contorns allà on hi ha un màxim local significatiu **Fig. 2.4**.

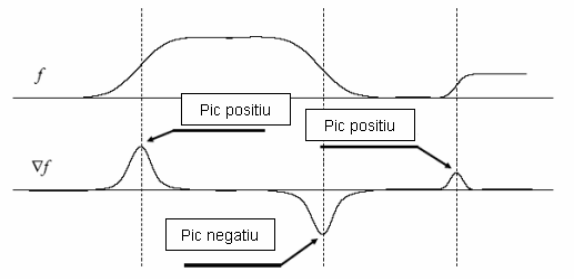


Fig. 2.4 Gràfic il·lustratiu de la funció gradient.
Allà on hi ha pics positius i negatius, s'identifiquen els contorns

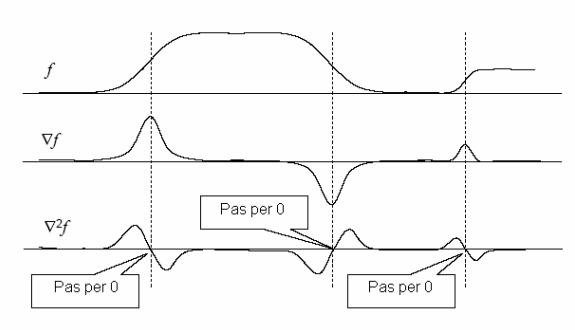


Fig. 2.5 Gràfic il·lustratiu de la funció Laplaciana.
Els passos per zero seran identificats com a contorns

La Laplaciana per la seva banda, mesura canvis al gradient **(2.4)**.

$$\nabla^2 f(x) = \frac{\partial^2 f}{\partial x^2} \qquad \textbf{(2.4)}$$

Mitjançant la seva aplicació s'identifiquen els contorns allà on hi ha un pas per zero **Fig. 2.5**.

Probablement el millor detector de contorns és el filtre de *Canny* [3], gràcies a la seva senzillesa i als bons resultats que s'obtenen amb ell. És tracta d'un detector basat en gradient. L'operador de Canny, després de suavitzar la imatge original amb un filtre gaussià, escull com a punts limítrofs aquells amb gradient màxim. Posteriorment es realitza un post processament aplicant un llindar amb un procés d'histèresis que elimina falsos límits. Els resultats obtinguts són gairebé òptims.

2.1.2 Canny

L'any 1986, John Canny va definir un conjunt d'objectius que calia que complís qualsevol detector de contorns, que són els següents:

- Taxa d'error: el detector de contorn ha de respondre només a contorns, i ha de trobar-los tots
- Ubicació: la distància entre els píxels del contorn trobats pel detector i el contorn real ha de ser la menor possible
- El detector de contorns no ha d'identificar múltiples píxels de contorns on només existeixi un únic contorn

I va descriure un mètode òptim per tal d'aconseguir-ho. El primer pas és el càlcul de la magnitud i direcció del gradient de la imatge suavitzada. El segon pas consisteix en l'aplicació d'un algoritme de supressió dels no-màxims (*non maximum supression*), que posa a 0 els valors de magnitud que no són màxim local a la direcció del gradient.

El tercer pas és l'execució d'un algoritme de rastreig de contorn: Canny va fer servir un llindar amb histèresis com s'explica a continuació. Es necessiten dos llindars, un baix T_1 i un alt T_2. Es defineix un punt de contorn com aquell punt que té un valor de magnitud més gran que el llindar alt. Després és mira si està connectat a un punt que tingui una magnitud per sobre del llindar baix, i si és així, aquest punt també es marca com un punt del contorn. Aquest procés continua fins que s'arriba al punt d'on es va sortir o bé si ja no hi ha més píxels al llarg del contorn que compleixin la condició d'estar per sobre del llindar baix. D'aquesta manera tots els píxels que tenen una magnitud de gradient per sobre del llindar baix, però que necessàriament estan connectats amb punts que tenen una alta confiança en el contorn, són marcats com a contorn, però punts candidats a contorn que no hi estan connectats són desestimats.

Un llindar T_2 escollit massa alt pot fer perdre informació important. Un llindar T_1 escollit massa baix pot fer identificar falsament informació irrellevant com si fos important. És difícil donar un llindar general que treballi bé per a totes les imatges.

L'aplicació del detector de contorns de Canny als núvols de punts obtinguts mitjançant el TLS, serà una de les entrades de l'algoritme de parametrització que presentarem a continuació. Partim de dos núvols de punts: distància i intensitat **Fig. 2.6**. La suma (OR) (**Fig. 2.8**) d'ambdós (**Fig. 2.7**) un cop aplicat Canny, serà l'entrada que farem servir.

Fig. 2.6 Núvol de punts de distància i d'intensitat, respectivament

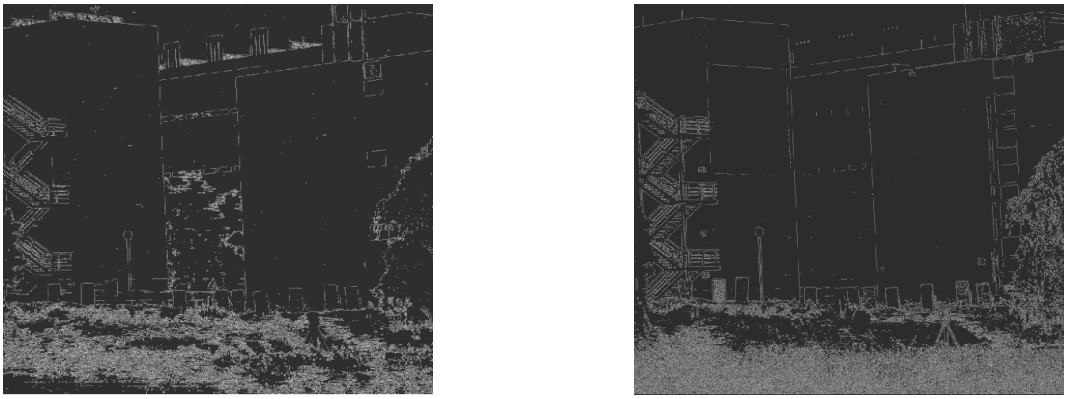


Fig. 2.7 Aplicació de CANNY sobre el núvol de punts de distància i sobre el d'intensitat, respectivament. Es tracta d'una imatge binària que pren valor 1 allà on hi ha contorn i valor 0 on no n'hi ha

Fig. 2.8 Canny distància (OR) intensitat

A l'**Annex 2** s'hi troba el codi utilitzat per a la implementació de l'algoritme de Canny.

2.2 Extracció d'objectes

Un cop tenim els contorns determinats ja podem aplicar un algoritme que ens permeti detectar i parametritzar determinats objectes, com esferes, repartits al llarg de l'escena. Molts són els algoritmes de parametrització que existeixen, però degut a les seves característiques, s'ha escollit l'algoritme de RANSAC [4] per a aconseguir aquest objectiu.

2.2.1 RANSAC

RANSAC o *RANdom Sampling Consensus* és un algoritme capaç de detectar plans, esferes, cilindres, cons ,etc., en un núvol de punts format per diferents objectes. És robust enfront el soroll, conceptualment simple i fàcil d'implementar, i permet separar el punts que poden ser explicats per un determinat model anomenats *inliers*, dels que no poden ser explicats pel mateix, els *outliers* **Fig. 2.9**.

L'algoritme va ser publicat per Fichler and Boller l'any 1981 [5]. Es basa en anar prenent petits conjunts de punts, en calcular el model a partir d'aquests punts, i en estimar quants punts del conjunt complet confirmen el model utilitzat. Aquest procés es repeteix fins a obtenir un conjunt de punts gran que compleixi amb el model **(Fig. 2.10)**.

En aquest algoritme tenim 3 llindars (T,k,t). El paràmetre t depèn del tipus de model que estem estimant, per exemple si el model és una recta, llavors el llindar es compara amb la distància d'un punt a la recta $d(p,l)<t$. El paràmetre T representarà el nombre de punts que nosaltres considerem el mínim d'inliers per a acceptar un model com a vàlid.

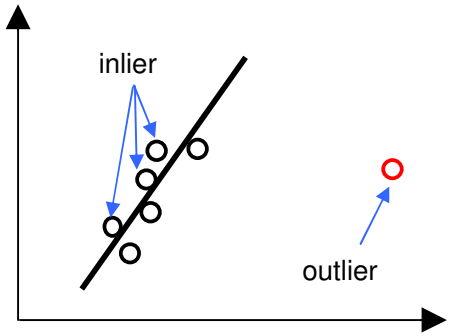


Fig. 2.9 Concepte inlier/outlier

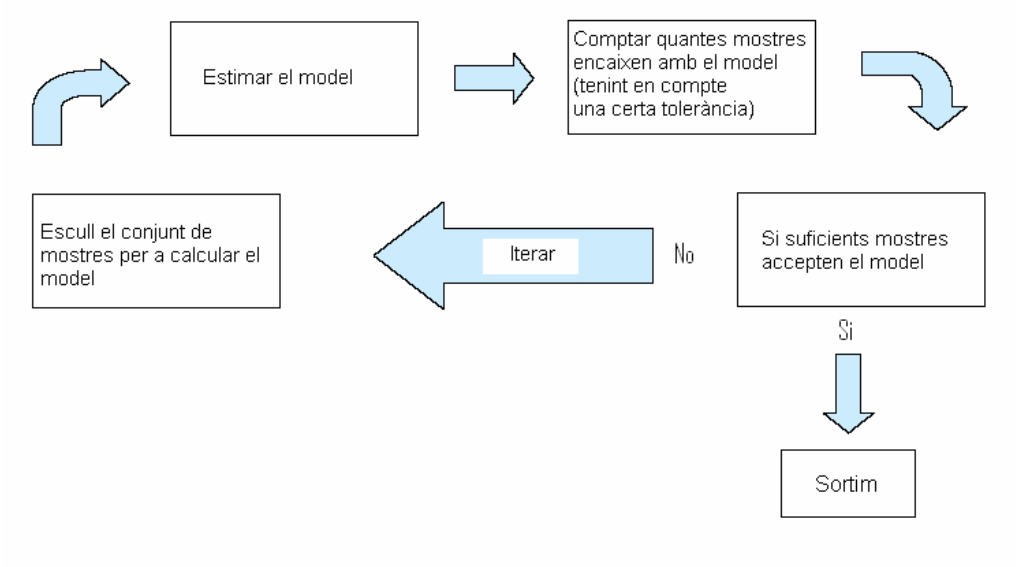


Fig. 2.10 Esquema de funcionament de l'algoritme RANSAC

El paràmetre *k* representa el nombre d'iteracions que l'algoritme durà a terme. És el més complex donat que pot incrementar considerablement el temps d'execució de l'algoritme. Per una banda no pot ser molt petit perquè la mostra final seleccionada podria no ser la més convenient, mentre que per altra banda, si és molt gran l'algoritme pot trigar massa temps en executar-se. A la **Fig. 2.13** es pot veure l'algoritme utilitzat a l'hora de seleccionar aquest paràmetre.

El funcionament d'aquest algoritme es pot mostrar utilitzant el problema d'ajust d'una recta a un núvol de punts. La idea és molt simple:

- Es seleccionen de forma aleatòria dos punts. Aquests dos punts defineixen una recta
- El suport per aquesta recta es mesura pel nombre de punts la distància normal dels quals a la recta cau dintre d'un llindar fixat prèviament
- Aquesta selecció aleatòria es repeteix un número determinat de cops i la recta amb major suport es considera la recta robusta

A l'esquema que hi ha a continuació, **Fig. 2.11**, queden definides les entrades i sortides de l'algoritme:

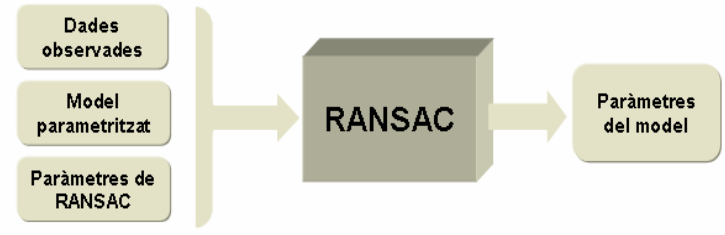


Fig. 2.11 Entrades / Sortides algoritme RANSAC

On:

- Dades observades: imatge sobre la que trebalalrem
- Model parametritzat: model que defineix la forma objecte d'estudi
- Paràmetres de RANSAC:
  - k: nombre d'iteracions fins a trobar un model òptim sense recórrer totes les possibles combinacions calculat aplicant **(2.5)**

$$k = \frac{\log(1-p)}{\log(1-w^n)} \qquad \textbf{(2.5)}$$

    - p: proporció d'inliers
    - w: proporció d'outliers
    - n: ajust del model; nombre de punts necessaris per a reconstruir el model

- Paràmetres del model: paràmetres que defineixen la forma objecte d'estudi

Els passos que segueix aquest algoritme són els descrits a continuació (**Fig. 2.12**):

```
data: conjunt de punts observats
model: model que defineix la figura que busquem
n: nombre mínim de punts necessaris per tal de determinar el model
k: nombre màxim d'iteracions
t: llindar per a determinar quan un punt es acceptat pel model
d: nombre d'inliers que ens permeten determinar si un model és bo

iteracions = 0
bestfit = null

while iteracions < k{
        maybeinliers = n valors aleatoris
        maybemodel = model extret a partir dels maybeinliers
        alsoinliers = conjunt buit
        per cada punt que no estigui en maybeinliers {
                si el punt s'aproxima a maybemodel amb un error menor que t
                afegim aquest punt a alsoinliers
        }

        si el nombre d'elements en alsoinliers > d {
                bettermodel = model de maybeinliers + alsoinliers
                bestfit = bettermodel
        }

        increment iteracions
}

return bestfit
```

Fig. 2.12 Esquema descriptiu algoritme RANSAC

```
k = ∞, número_mostres = 0

Mentre k > número_mostres repetir
       Escollir una mostra i comptar el número de inliers p
       w = 1 - (número de inliers)/(total de punts)
       Recalcular el valor de k a partir de w i p
       Incrementar en 1 el número_mostres
Acabar
```

Fig. 2.13 Algoritme que permet calcular el nombre d'iteracions del RANSAC

2.2.2 Implementació

Una primera implementació de RANSAC ha estat aplicant-lo a la detecció d'esferes, tal i com es mostra a la **Fig. 2.14.**

Com a paràmetres d'entrada tenim el Canny distància (OR) intensitat, l'equació que defineix una esfera, i els paràmetres propis de l'algoritme RANSAC. La limitació clara que suposa aquest esquema és que, pel fet d'introduir a l'entrada el Canny del núvol de punts, perdem informació referent al mateix: el Canny és una representació binària que pren valor 1 allà on hi ha contorn i valor 0 allà on no n'hi ha. S'ha d'aplicar addicionalment una funció que atorgui a cada valor 1 del Canny el valor corresponent tan de posició com d'intensitat del núvol de punts perquè RANSAC funcioni tal i com s'ha dissenyat.

El fet de partir d'una imatge 2D i passar-la a 3D implica ampliar el codi utilitzat, i no és òptim tant pel que fa al temps de processat de l'algoritme, com a la complexitat del mateix. A més ara ja no en tindrem prou amb els contorns dels objectes sinó que els necessitarem sencers.

A la **Fig. 2.15** es reflexa com ha quedat definitivament implementat RANSAC:

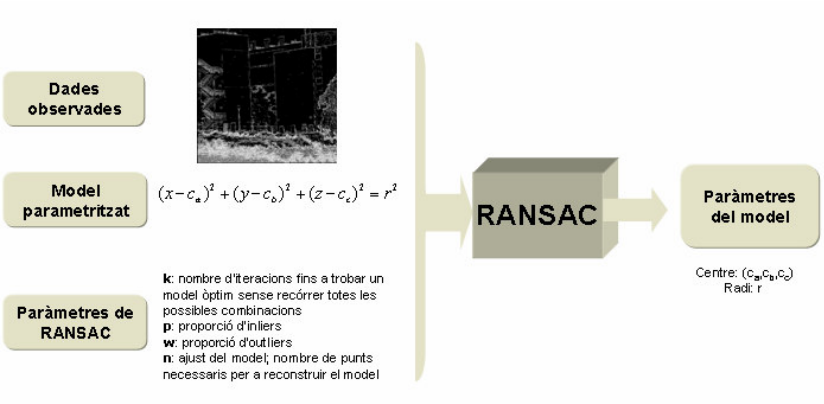


Fig. 2.14 Esquema primera implementació RANSAC

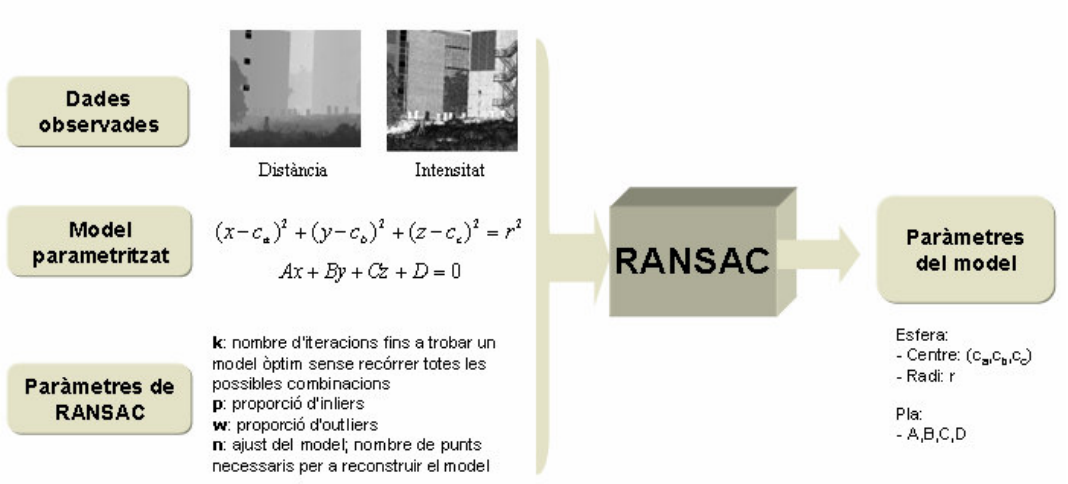


Fig. 2.15 Esquema RANSAC implementació definitiva

Aquí com a paràmetres d'entrada ja no tindrem el Canny sinó els núvols de punts de distància i intensitat que ens proporciona el TLS, els models parametritzats dels objectes que volem trobar, que seran esferes i plans, i els paràmetres propis de RANSAC ja comentats anteriorment. Pel que fa als paràmetres del model, aquests seran el centre i el radi en el cas de l'esfera, i les constants A, B, C i D en el cas del pla.

CAPÍTOL 3. IMPLEMENTACIÓ RANSAC

En aquests capítols es presenten els resultats de la implementació de l'algoritme RANSAC per a la detecció d'esferes i plans. S'han realitzat diferents estudis en núvols de punts simulats i en núvols de punts reals. El programa s'ha realitzat en codi C++ i les imatges sobre les que s'han treballat han estat cedides per l'IG.

3.1 Estructura del programa

El programa implementat consta de les diferents parts:

- Llibreria: *lidar.h*
- Fitxer input: *input.txt*
- Funció per a llegir núvols de punts TLS: *Read_data.cpp*
- Main: *RANSAC.cpp*
- Funcions per a calcular RANSAC:
  - *RANSAC_ESFERA.cpp*
  - *RANSAC_PLA.cpp*

La llibreria *lidar.h* conté la definició de les diferents variables que s'aniran utilitzant al llarg de tot el programa. Al fitxer d'input *input.txt* podrem ajustar els paràmetres corresponents a la imatge a considerar. Aquests paràmetre són els següents:

Number of files to be read
1
Name of the files
scanner2_task2_amb_esferes.xyz
Percentage
0.6
Tolerance
0.05
Paràmetre Esfera 1
0.06
Paràmetre Esfera 2
0.04
Paràmetre Pla 1
1
Paràmetre Pla 2
2
Number of lines of the Window
15 15
Number of lines of the scene:
1134 868

- Number of files to be read: nombre de núvols de punts que tractarem. Podríem aplicar RANSAC a més d'un núvol de punts de manera seqüencial si ens interessés
- Name of the files: nom del fitxer que emmagatzema el núvol de punts
- Percentage: nombre d'inliers que necessitem per a acceptar un model com a vàlid
- Tolerance: llindar per acceptar un punt com a inlier
- Paràmetre esfera 1 / Paràmetre esfera 2: paràmetres propis de les esferes, comentats posteriorment
- Paràmetre pla 1 / Paràmetre pla 2: paràmetres propis dels plans, comentats posteriorment
- Number of lines of the window: nombre de columnes i nombre de files de la finestra en que dividim els núvols de punts a tractar
- Number of lines of the scene: nombre de columnes i nombre de files dels núvols de punts a tractar

La funció *Read_data.cpp* ens permet llegir els núvols de punts construint una matriu on es guarden totes les dades, tan de posició com d'intensitat, de cadascun dels punts que els conformen.

La funció principal *RANSAC.cpp* funciona de la manera descrita a continuació. Anem aplicant l'algoritme per finestres de tamany prefixat al fitxer d'input; el motiu pel qual separem el núvol de punts en finestres i no l'ataquem directament, rau en el fet que aplicant-lo sobre la imatge sencera comporta molt temps de procés i és poc precís. Per cadascuna d'aquestes finestres escollirem aleatòriament el nombre de punts necessari per a determinar el model a considerar. Un cop tinguem el model hem de comprovar si la resta de punts de la finestra s'ajusten o no al model trobat, amb una certa tolerància que podem prefixar al fitxer d'input. El nombre d'iteracions que aplicarem aquest algoritme sobre cada finestra es calculat pel propi programa.

Tenim l'opció d'aplicar RANSAC per a determinar els paràmetres de diferents objectes com són esferes i plans. Si escollim l'opció *RANSAC_ESFERA.cpp*, el programa ens retornarà el centre i radi de l'esfera o esferes buscades. Si l'opció escollida és *RANSAC_PLA.cpp*, l'algoritme ens retornarà els paràmetres propis d'un pla: A, B, C i D.

Una de les limitacions en l'algoritme és que només podrem detectar objectes de forma eficient si coneixem aproximadament els seus paràmetres. Per aquest motiu al fitxer d'input apareixen els paràmetres propis de les figures a tractar. És tracta d'acotar els paràmetres buscats a fi de millorar el rendiment del programa [6]. En el cas de l'esfera els paràmetres propis 1 i 2 seran, respectivament, el radi màxim i mínim de les esferes buscades. Pel que fa al pla, el que acotarem serà la variable D (variable independent en l'equació del pla).

3.1.1 RANSAC cas esfera

Quatre punts determinen una única esfera, si i només si, no estan sobre el mateix pla. Aquesta esfera es pot trobar solucionant el següent determinant **(3.1)**:

$$\begin{vmatrix} x^2+y^2+z^2 & x & y & z & 1 \\ x_1^2+y_1^2+z_1^2 & x_1 & y_1 & z_1 & 1 \\ x_2^2+y_2^2+z_2^2 & x_2 & y_2 & z_2 & 1 \\ x_3^2+y_3^2+z_3^2 & x_3 & y_3 & z_3 & 1 \\ x_4^2+y_4^2+z_4^2 & x_4 & y_4 & z_4 & 1 \end{vmatrix} \quad \textbf{(3.1)}$$

De la resolució del determinant anterior en sorgeixen les següents equacions que ens determinaran el centre **(3.2) (3.3) (3.4)** i radi **(3.5)** de l'esfera:

$$x_0 = +0.5M_{12} / M_{11} \quad \textbf{(3.2)}$$

$$y_0 = -0.5M_{13} / M_{11} \quad \textbf{(3.3)}$$

$$z_0 = +0.5M_{14} / M_{11} \quad \textbf{(3.4)}$$

$$r_0^2 = x_0^2 + y_0^2 + z_0^2 - M_{15} / M_{11} \quad \textbf{(3.5)}$$

On:

- M_{ij}: matriu complementària de l'element a_{ij} que s'obté a l'eliminar la fila *i* i la columna *j* de la matriu quadrada A. Per exemple, la matriu complementària de l'element a_{11} és la matriu que resulta al suprimir de la matriu A la fila 1 i la columna 1.

3.1.1.1 Detecció d'esferes en un núvol de punts simulat

L'exemple que hem estudiat és el següent. S'ha generat un núvol de punts de dimensions 150x150, amb un pas d'1 cm entre punts consecutius. A cadascun dels punts que conformen el núvol se li ha afegit soroll {-2,2} cm. Dintre del núvol de punts s'han inserit 3 esferes de radi 5 cm. i amb centres (5,5,5), (9,9,9) i (12,12,12) **Fig. 3.1**.

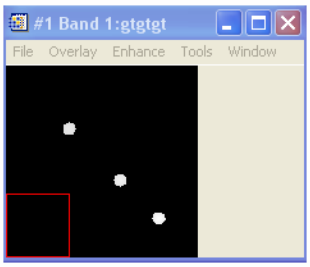


Fig. 3.1 Núvol de punts simulat amb esferes

Separarem el núvol de punts en finestres de tamany 15 x 15 punts (un total de 225 punts dels 22500 que el composen). Escollim 4 punts de la finestra aleatòriament per tal de definir el model. Pel que fa a la resta de punts, comprovarem quants compleixen l'equació de l'esfera **(3.6)** que hauran definit els punts anteriors. Els que ho compleixin seran considerats com a inliers. Aquelles finestres que tinguin un nombre més elevat d'inliers seran les finestres que contindran les esferes que busquem.

$$(x-x_o)^2+(y-y_0)^2+(z-z_0)^2=r^2 \qquad \textbf{(3.6)}$$

Aquesta és una mostra del fitxer de sortida:

Total_inliers: 80
Aquests son els paràmetres del model:
Centre: (5.001223,4.996793,5.002765) Radi: 0.0498730

Total_inliers: 81
Aquests son els paràmetres del model:
Centre: (8.997030,8.999171,8.999502) Radi: 0.0498585

Total_inliers: 81
Aquests son els paràmetres del model:
Centre: (11.999285,11.997172,11.994261) Radi: 0.0497878

Els paràmetres de RANSAC utilitzats en aquesta extracció són els següents:

- La tolerància per a acceptar un punt com a inlier és de l'ordre dels centímetres. Aquest llindar s'escull de manera empírica; si prenem una tolerància de l'ordre de decímetres o major, les esferes trobades no tenen res a veure amb les buscades

- Acceptem un model com a vàlid si conté més del 60 % d'inliers. Si el percentatge d'inliers és menor, les esferes detectades no són les buscades. A més necessitem que com a mínim al voltant del 65 % dels punts de les esferes estiguin dintre del núvol de punts per a que RANSAC pugui trobar-les

- El nombre d'iteracions que ha realitzat l'algoritme en cada finestra fins a obtenir el model, han estat 5000; un temps de procés de l'ordre de pocs minuts

3.1.1.2 Detecció d'esferes en un núvol de punts real

Ara veurem que passa quan apliquem RANSAC a un núvol de punts real, amb un soroll més elevat del que considerem com a òptim. Partirem del següent, captat mitjançant TLS i corresponent al poble de Castellfollit de la Roca **Fig. 3.2**. La resolució espacial del núvol de punts està al voltant dels 6 cm.

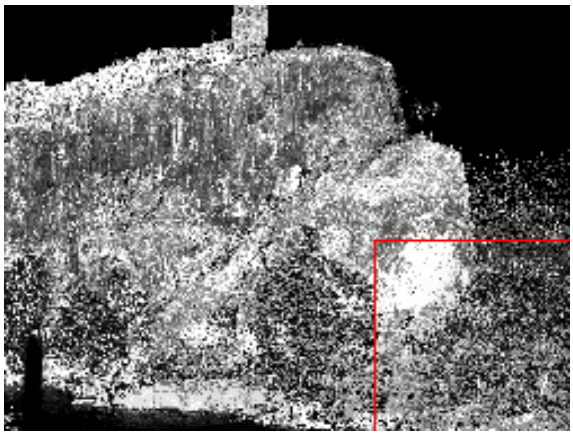


Fig. 3.2 Imatge d'intensitat corresponent a Castellfollit de la Roca

En aquest núvol de punts hi afegirem 5 esferes **Fig.3.3** totes elles de radi 5 cm. i amb centres:

---

centre1= (-14.446, 200, 1.002)

centre2= (9.164, 200, -11.470)

centre3= (-17.315, 200, 13.840)

centre4= (27.839, 200, 13.426)

centre5= (7.866, 200, 34.693)

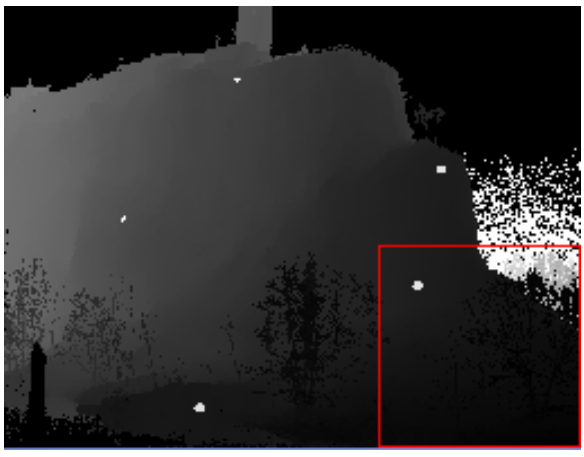


Fig. 3.3 Component Y del núvol de punts de Castellfollit de la Roca amb 5 esferes simulades

En aquest cas també separarem el núvol de punts en finestres de tamany 15 x 15 punts (un total de 225 punts dels 984312 que el composen). Aleatòriament escollim 4 punts de la finestra per tal de definir el model i comprovarem quants dels punts restants compleixen l'equació de l'esfera **(3.6)** que hauran definit els punts anteriors. Aquells que ho compleixin seran considerats com a inliers i les finestres que tinguin un nombre més elevat d'inliers seran les que contindran les esferes que busquem.

Aquesta és una mostra del fitxer de sortida:

Total_inliers: 76
Aquests son els paràmetres del model:
Centre: (-14.999663,200.999723,1.000992) Radi: 0.0498491

Total_inliers: 77
Aquests son els paràmetres del model:
Centre: (9.002434,200.004257,-11.002145) Radi: 0.0498274

Total_inliers: 73
Aquests son els paràmetres del model:
Centre: (-17.985343,200.989157,13.994951) Radi: 0.0499032

Total_inliers: 73
Aquests son els paràmetres del model:
Centre: (27.986282,200.017028,13.992673) Radi: 0.0501476

Total_inliers: 73
Aquests son els paràmetres del model:
Centre: (7.993426,200.988309,34.985061) Radi: 0.0495477

Els paràmetres de RANSAC utilitzats en aquesta extracció són els següents:

- La tolerància per a acceptar un punt com a inlier és de l'ordre dels centímetres pel mateix motiu que s'explica en el cas de la detecció d'esferes sobre núvols de punts simulats

- També acceptem un model com a vàlid si conté més del 60 % d'inliers i necessitem que com a mínim al voltant del 65 % dels punts de les esferes estiguin dintre del núvol de punts per a que RANSAC pugui trobar-les

- El nombre d'iteracions que ha realitzat l'algoritme en cada finestra fins a obtenir el model, han estat 30000; un temps de procés de l'ordre de 30-40 minuts. Com veiem, a mesura que augmenta el soroll, les iteracions augmenten considerablement i l'algoritme resulta més ineficient a causa del seu elevat temps de procés

3.1.2 RANSAC cas pla

Un punt i dos vectors linealment independents determinen un únic pla. L'equació implícita del pla queda definida per l'expressió **(3.7)**:

$$Ax + By + Cz + D = 0 \qquad \textbf{(3.7)}$$

Donats 3 punts de l'espai (x_1,y_1,z_1), (x_2,y_2,z_2), (x_3,y_3,z_3), l'equació del pla que els conté ve donada pels següents determinants **(3.8)**:

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix}, B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}, C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}, D = - \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} \qquad \textbf{(3.8)}$$

Escollim 3 punts de la finestra aleatòriament per tal de definir el model. Pel que fa a la resta de punts, comprovarem quants d'aquests tenen una distància **(3.9)** 0 al pla. Els que ho compleixin seran considerats com a inliers.

$$dist(P, p) = \frac{|Ax_0 + By_0 + Cz_0 + D|}{\sqrt{A^2 + B^2 + C^2}} \qquad \textbf{(3.9)}$$

3.1.2.1 Detecció de plans en un núvol de punts simulat

L'exemple que hem estudiat és el següent. S'ha generat un núvol de punts de dimensions 150x150, amb un pas d'1 cm entre punts consecutius. A cadascun dels punts que conformen el núvol se li ha afegit soroll {-2,2} cm. Dintre del núvol de punts s'han inserit 3 plans amb A=2, B=3, C=4 i D=5 **Fig. 3.4**.

Separarem el núvol de punts en finestres de tamany 15 x 15 punts (un total de 225 punts dels 22500 que el composen). Escollim 3 punts de la finestra aleatòriament per tal de definir el model. Pel que fa a la resta de punts, comprovarem quants compleixen que la seva distància al pla sigui 0 **(3.9)**. Els que ho compleixin seran considerats com a inliers.

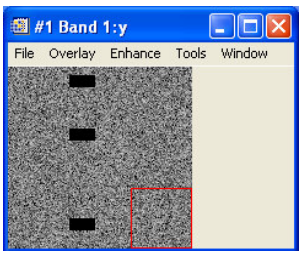


Fig. 3.4 Núvol de punts simulat amb plans

Aquesta és una mostra del fitxer de sortida:

Total_inliers Pla1: 196
Aquests son els paràmetres del model:
2.002671 x + 3.272069 y + 4.002507 z + 5.037128

Total_inliers Pla2: 196
Aquests son els paràmetres del model:
2.006005 x + 3.104073 y + 4.001336 z + 5.053740

Total_inliers Pla3: 196
Aquests son els paràmetres del model:
2.003065 x + 3.076163 y + 4.001102 z + 5.025323

Els paràmetres de RANSAC utilitzats en aquesta extracció són els següents:

- La tolerància per a acceptar un punt com a inlier és de l'ordre dels centímetres; paràmetre escollit de forma empírica
- Acceptem un model com a vàlid si conté més del 60 % d'inliers. Si el percentatge d'inliers és menor, els plans detectats no són els buscats. A més necessitem que com a mínim al voltant del 70 % dels punts dels plans estiguin dintre del núvol de punts per a que RANSAC pugui trobar-los
- El nombre d'iteracions que ha realitzat l'algoritme en cada finestra fins a obtenir el model, han estat 7000; un temps de procés de l'ordre de pocs minuts

3.1.1.2 Detecció de plans en un núvol de punts real

Ara veurem que passa quan apliquem RANSAC a un núvol de punts real, amb un soroll més elevat del que considerem com a òptim. Partirem del núvol de punts de Castellfollit de la Roca igual que ho fèiem amb les esferes. En aquest cas però hi afegirem 5 plans que compliran $2x + 3y + 4z + 5 = 0$.

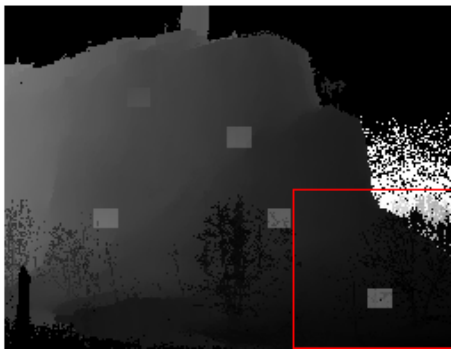


Fig. 3.5 Component Y del núvol de punts de Castellfollit de la Roca amb 5 plans simulats

Separarem el núvol de punts en finestres de tamany 15 x 15 punts (un total de 225 punts dels 984312 que el composen) i com en el cas anterior escollim 3 punts de la finestra aleatòriament per tal de definir el model. Comprovarem quants dels punts restants compleixen que la seva distància al pla sigui 0 **(3.9)** i els que ho compleixin seran considerats com a inliers.

Aquesta és una mostra del fitxer de sortida:

Total_inliers Pla1: 173
Aquests son els paràmetres del model:
2.003442 x + 3.013245 y + 4.009651 z + 5.012323

Total_inliersc Pla2: 192
Aquests son els paràmetres del model:
2.009263 x + 3.109234 y + 4.001356 z + 5.023740

Total_inliers Pla3: 187
Aquests son els paràmetres del model:
2.023423 x + 3.023673 y + 4.002302 z + 5.087452

Total_inliers Pla4: 191
Aquests son els paràmetres del model:
2.008265 x + 3.079823 y + 4.028782 z + 5.193009

Total_inliers Pla3: 199
Aquests son els paràmetres del model:
2.009971 x + 3.017283 y + 4.092802 z + 5.096252

Els paràmetres de RANSAC utilitzats en aquesta extracció són els següents:

- La tolerància per a acceptar un punt com a inlier és de l'ordre dels centímetres

- Acceptem un model com a vàlid si conté més del 60 % d'inliers i necessitem que com a mínim al voltant del 70 % dels punts dels plans estiguin dintre del núvol de punts per a que RANSAC pugui trobar-los

- El nombre d'iteracions que ha realitzat l'algoritme en cada finestra fins a obtenir el model, han estat 35000; un temps de procés de l'ordre de 30-40 minuts. Com veiem, a mesura que augmenta el soroll, les iteracions augmenten considerablement i l'algoritme resulta més ineficient a causa del seu elevat temps de procés

Als **Annexos 3** i **4** hi podem trobar el codi referent a les simulacions i a l'algoritme RANSAC, respectivament.

CAPÍTOL 4. CORRESPONDÈNCIA ENTRE PUNTS: MATCHING

El *matching* és un procediment que consisteix en la determinació de punts homòlegs. Mitjançant el matching és possible trobar la correspondència entre punts de diferentes adquisicions. Els punts corresponents són aquells que representen una projecció del mateix punt físic a l'espai 3D. A la figura que hi ha sota **Fig. 4.1** es poden apreciar 3 vistes d'un mateix objecte presses a partir de la rotació de l'eix central del mateix. Es pot observar que els punts m_1, m_2 i m_3 a les imatges 1, 2 i 3 respectivament, són corresponents entre sí perquè són les projeccions del mateix punt m de la tassa. En aquest capítol donarem resposta a qüestions com les següents: coneixent el punt m_1 de la imatge 1, on està el seu punt corresponent a les imatges 2 i 3? Coneixent els punts m_1 i m_2, i sabent que són corresponents, on es troba el punt corresponent en la tercera imatge?

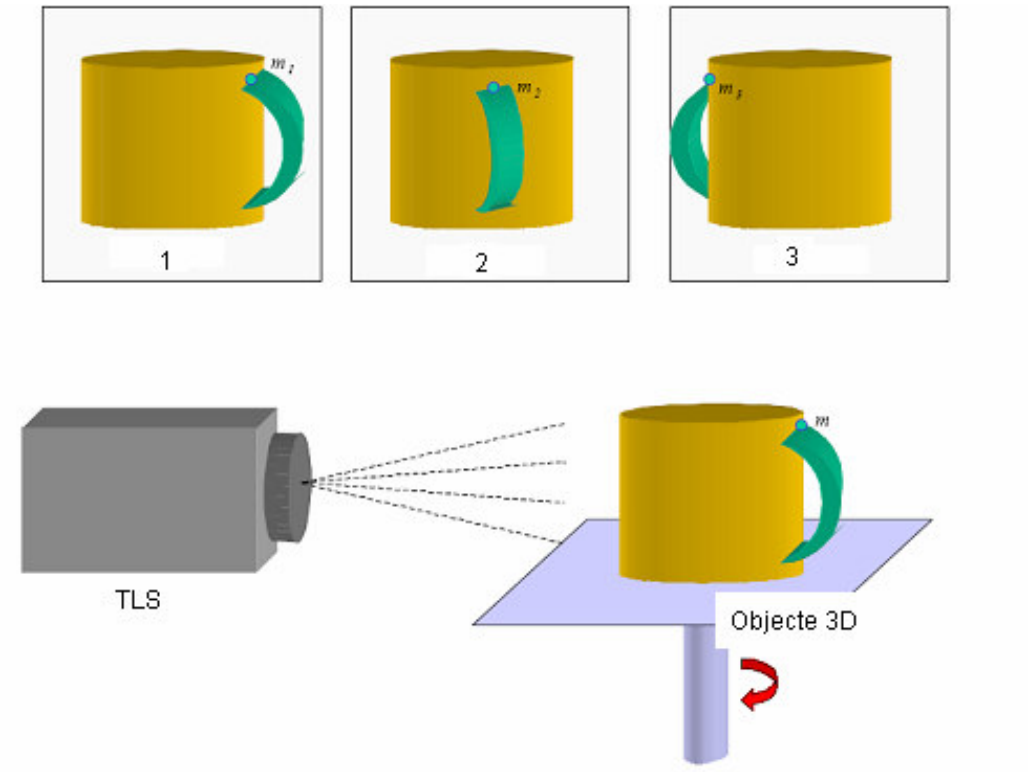


Fig. 4.1 Concepte de matching

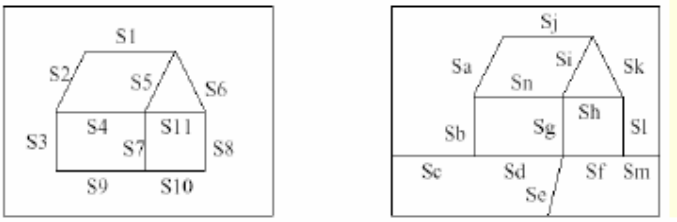


Fig. 4.2 Interpretació del concepte de Matching

4.1 Classificació dels mètodes de matching

El terme matching s'ha imposat en la literatura com l'expressió més apropiada per a expressar la *correlació* entre imatges digitals en *fotogrametria*.

En les dues imatges de la **Fig. 4.2**, el resultat del matching seria el següent: f(S1)=Sj, f(S2)=Sa, f(S3)=Sb, f(S4)=Sn, f(S5)=Si, f(S6)=Sk, f(S7)=Sg, f(S8)=Sl, f(S9)=Sd, f(S10)=Sf, f(S11)=Sh.

Existeixen diferents mètodes de matching que podem agrupar de la manera següent:

- Matching basat en àrea (*Area-Based Matching – ABM*): també conegut com matching a nivell de gris. Es compara la distribució dels nivells de gris en àrees petites de dues imatges, i la similitud és mesura mitjançant *correlació creuada* [7] o tècniques de *mínims quadrats* [7]

- Matching basat en característiques (*Feature-Based Matching – FBM*): el que es compara són grups de píxels que representen punts, eixos o àrees. La similitud és mesurada per una funció de cost. Aquests mètodes són en general més robustos

- Matching Relacional (*Relational Matching-RM*): utilitza relacions entre objectes com poden ser veïnatge, connexió, etc. També mesura la similitud per una funció de cost. Contràriament als altres dos mètodes mencionats, el matching relacional en comptes d'utilitzar la forma o localització com a criteri de similitud, compara propietats topològiques

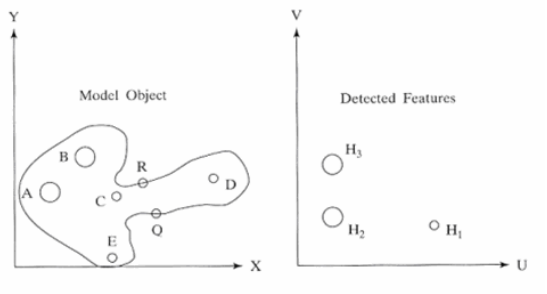
En aquest capítol ens centrarem amb el darrer mètode mencionat.

4.2 Relational matching

El matching relacional [8] fa ús de *descriptors relacionals*. Un descriptor relacional pot ser tan una primitiva (elements bàsics com punts, contorns, corbes), com una relació (a la dreta de , veí de, per sobre de). La mesura de similitud entre dos descriptors relacionals es determina afegint una "funció de cost", la qual expressa el grau de semblança entre ells. Molta similitud significa

un cost baix. Cinc són les tècniques o mètodes que podem fer servir a l'hora d'aplicar matching relacional.

Existeix l'*arbre d'interpretació*; un arbre que representa tots els possibles aparellaments entre dues imatges. Quan es troben relacions inconsistents al llarg d'una branca, es retalla el camí de recerca. Aquesta tècnica pot ser utilitzada per a trobar totes les possibles parelles, o la millor parella. Les comparacions individuals de consistència són locals, però un camí des de l'origen de l'arbre de recerca fins a cada fulla produeix un gran horitzó de comparació. A la **Fig. 4.3**, es pot apreciar com funciona un arbre d'aquest tipus.



Point	Coordinates	to A	to B	to C	to D	to E
A	(8,17)	0	12	15	37	21
B	(16,26)	12	0	12	30	26
C	(23,16)	15	12	0	22	15
D	(45,20)	37	30	22	0	30
E	(22,1)	21	26	15	30	0

Point	Coordinates	to A	to B	to C
H_1	(31,9)	0	21	26
H_2	(10,12)	21	0	12
H_3	(10,24)	26	12	0

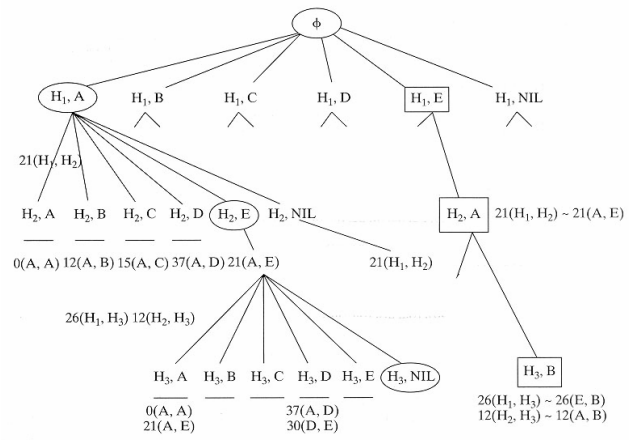


Fig. 4.3 Arbre d'interpretació

Es tracta d'aparellar A, B, C, D i E amb H_1, H_2 i H_3. Comencem buscant la parella de H_1 que pot ser qualsevol de les lletres (A,B,C,D,E). Es calcula la distància de H_1 respecte cadascuna d'elles, i la menor serà l'escollida, en aquest cas la A. Aquest serà el primer nivell de l'arbre. El segon nivell serà H_2 respecte qualsevol de les lletres i el tercer i últim nivell serà H_3. Així queden representats tots els possibles aparellaments.

Uns altres mètodes són la *relaxació discreta* i la *relaxació continuada*. En la relaxació discreta s'utilitzen només característiques locals a l'hora de realitzar el matching enlloc d'utilitzar totes les característiques disponibles. La relaxació continuada és una tècnica similar en concepte a la relaxació discreta, però prioritza la qualitat per sobre de la quantitat de parelles possibles. Cap de les dues tècniques garanteix trobar una solució òptima.

També hi ha el *matching relacional de distància*. És el mètode més adequat quan el que es necessita és trobar la solució més bona. El mètode millora la cerca eficient, utilitzant una mesura de similitud estructural.

Per últim tenim la *indexació relacional* que de manera ràpida troba tots els models que tenen algunes estructures específiques semblants i després els aparella amb més detall.

4.3 Procés de matching

El procediment que seguirem serà el descrit a continuació. És una variant d'arbre d'interpretació, extret de [9].

4.3.1 Extracció d'objectes mitjançant el RANSAC explicat al capítol 2

4.3.2 Definició del descriptor relacional: triangulació de Delaunay

4.3.3 Procediment de matching

4.3.1 Extracció de formes

El primer pas consistirà en l'extracció dels objectes que desitgem aparellar. RANSAC tal i com s'ha explicat al capítol 2, ha estat l'algoritme escollit per a dur a terme aquest objectiu. El proper pas serà l'elecció del descriptor relacional.

4.3.2 Definició del descriptor relacional

La idea perseguida a l'hora d'escollir el descriptor relacional rau en associar en una única estructura topològica tots els objectes del núvol de punts que desitgem aparellar. Per aconseguir aquest objectiu s'utilitzarà la *triangulació de Delaunay*. Els avantatges que això comporta són els següents: primer de tot, com que només considerarem triangles necessitarem menors requeriments de

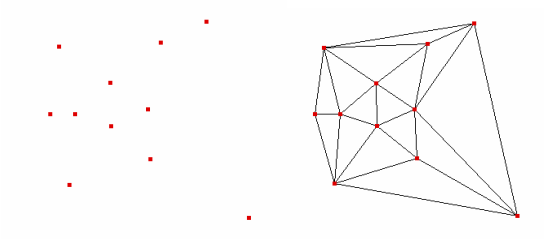


Fig. 4.4 Núvol de punts i triangulació de Delaunay del mateix

memòria i menor redundància. Segon, els triangles de la triangulació de Delanunay tenen un bon poder de discriminació ja que, d'entre tots els possibles triangles són els únics que satisfan certes propietats.

Una triangulació d'un núvol de punts és una partició del domini que defineixen, el tancament convex, en triangles. En la **Fig. 4.4** es pot veure un núvol de punts i una triangulació del seu tancament convex.

En general, per un núvol de *n* punts existeix una quantitat de triangulacions que depèn exponencialment de *n*. Això significa que, quan *n* sigui mitjanament gran, el nombre de triangulacions és suficientment elevat com per a que cap ordenador, per sofisticat que sigui, pugui calcular-les totes. Llavors convé preguntar-se quina d'elles és la millor per cada cas. En aquest estudi s'ha escollit la triangulació de Delaunay **Fig. 4.5**; una xarxa de triangles que compleix la condició de Delaunay, que diu que la circumferència circumscrita de cadascun dels triangles de la xarxa no ha de contenir cap vèrtex de cap altre triangle.

4.3.2.1 Triangulació de Delaunay

La circumferència circumscrita d'un triangle és la circumferència que conté els tres vèrtexs del mateix. Segons la definició de Delaunay la circumferència circumscrita és buida si no conté altres vèrtexs a part dels 3 que la defineixen. La condició de Delaunay diu que una xarxa de triangles és una triangulació de Delaunay si totes les circumferències circumscrites de tots els triangles de la xarxa són buides. Aquesta condició assegura que els angles de l'interior dels triangles són el més grans possible, és a dir, maximitza l'extensió de l'angle més petit de la xarxa.

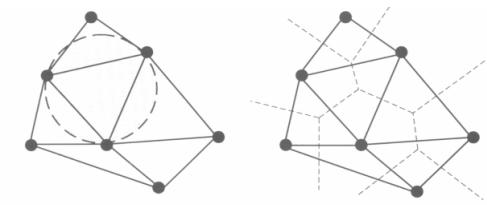


Fig. 4.5 Triangulació de Delaunay d'un núvol de punts

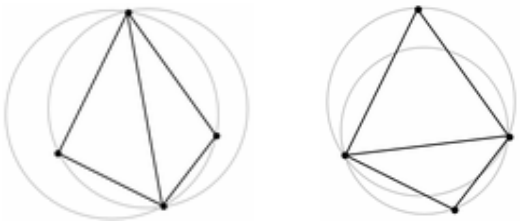


Fig. 4.6 A l'esquerra observem una triangulació que no compleix la condició de Delaunay. A la dreta s'observa el flip de l'aresta comú, el qual produeix una triangulació que si que compleix la condició de Delaunay

Una aresta d'un triangle d'una triangulació és incorrecta, si al canviar-la, augmenta l'angle mínim dels triangles adjacents. A aquesta aresta se l'anomena *aresta il·legal*. La circumferència definida pels vèrtexs d'un triangle conté un altre punt de la triangulació, si i només si, el triangle té una aresta il·legal o no vàlida. Les arestes vàlides són aquelles per a les que és possible fer un *flip*. El flip diagonal **Fig. 4.6** consisteix en, donats dos triangles que comparteixen un aresta, si el quadrilàter que defineixen és convex, canviar l'aresta comuna per l'altra diagonal del quadrilàter.

L'algoritme en pseudocodi de la triangulació de Delaunay seria el següent:

Entrada: Un conjunt finit de punts al pla

Pas 1: Siguin p1, p2 i p3 tres punts tals que P està contingut en el triangle que formen

Pas 2: Inicialitzar T com una triangulació d'un únic triangle (p1-p2-p3)

Pas 3: Realitzar una permutació qualsevol p1, p2, ..., pn de P

Pas 4: Per r=1 to n

Pas 5: Fer (* Afegir pr a T *)

Pas 6: Trobar un triangle pi-pj-pk de T que contingui pr

Pas 7: Si pr cau dintre del triangle pi-pj-pk

Pas 8: Llavors afegir arestes des de pr als tres vèrtexs de pi-pj-pk, dividint aquest triangle en tres

Pas 9: legalitza_costat (pr, pi-pj,T)

Pas 10: legalitza_costat (pr, pj-pk,T)

Pas 11: legalitza_costat (pr, pk-pi,T)

Pas 12: en cas contrari (* pr cau sobre uns dels costats del triangle pi-pj-pk, per exemple el costat pi-pj *)

Pas 13: Afegir arestes des de pr a pk i al tercer vèrtex de l'altre triangle que comparteix les arestes pi-pj, d'aquesta manera dividim els dos triangles que comparteixen l'aresta pi-pj en quatre triangles

Pas 14: legalitza_costat (pr, pi-pl,T)

Pas 15: legalitza_costat (pr, pl-pj,T)

Pas 16: legalitza_costat (pr, pj-pk,T)

Pas 17: legalitza_costat (pr, pk-pi,T)

Pas 18: descarta p1, p2 i p3 i totes les arestes que parteixen d'ells de T

Pas 19: retorna T

Sortida: La triangulació de Delaunay del conjunt de punts

L'algoritme legalitza_costat, comprova la validesa de l'aresta, si l'aresta és il·legal fa el flip corresponent i comprova de nou la validesa dels nous costats.

Pas 1: el punt que s'està inserint és pr, i pi-pj és l'aresta de T a la que pot ser necessari fer un intercanvi d'arestes

Pas 2: si pi-pj és no vàlida

Pas 3: llavors, sigui pi-pj-pk el triangle adjacent a pr-pi-pj compartint l'aresta pi-pj

Pas 4: (* Flip pi-pj *) reemplaçar pi-pj per pr-pk

Pas 5: legalitza_costat (pr, pi-pk, T)

Pas 6: legalitza_costat (pr, pk-pj, T)

Un cercle a través de 3 punts d'un triangle de Delaunay no conté cap altre punt. Això implica que la inserció d'un nou punt a la triangulació de Delaunay afecta només els triangles els circumcercles dels quals contenen aquell punt. Com a resultat, el soroll només afecta la triangulació de Delaunay localment.

A més a més, l'angle mínim al llarg de tots els angles en tots els triangles en una triangulació de Delaunay és més gran que el mínim en qualsevol altra triangulació dels mateixos punts. Aleshores, els triangles obtinguts no són desestructurats. Això és molt útil per la nostra aplicació ja que la computació de les transformacions geomètriques entre imatges es basa en la correspondència de triangles. Utilitzar triangles dèbils ens pot induir cap a inestabilitats i errors. En diferents estudis s'ha comprovat que la triangulació de Delaunay (**Fig. 4.7**) té la millor estabilitat estructural sota pertorbacions posicionals aleatòries.

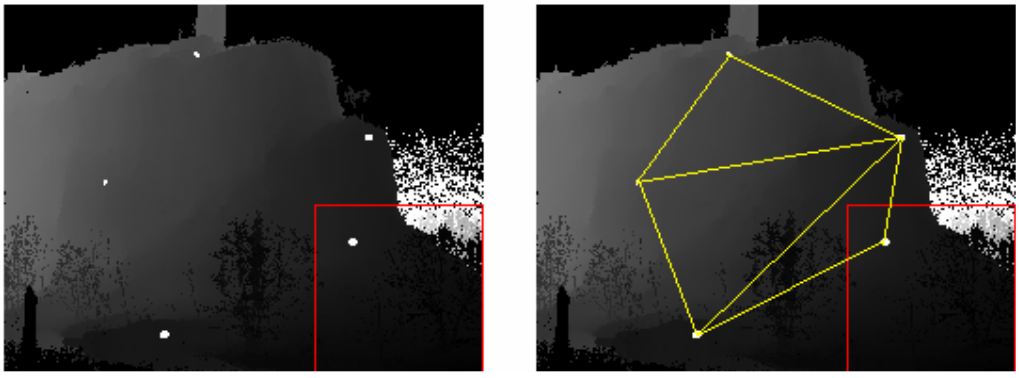


Fig. 4.7 Triangulació de Delaunay de les esferes extretes mitjançant RANSAC

4.3.3 Procediment de matching

Un cop tenim la triangulació de Delaunay construirem una taula d'indexació formada gràcies als triangles obtinguts. Abans de decidir quins paràmetres seran computats de cada triangle, s'ha de definir la transformació geomètrica que s'ha dut a terme entre les imatges objecte de matching. Normalment s'assumeix que sigui una transformació rígida o similar; en el nostre cas assumirem transformacions de similitud tals com translacions, rotacions o escalat. Donat un triangle, computem 3 paràmetres que seran utilitzats per a formar un índex. Els paràmetres seran els costats dels angles del triangle **(4.1)**.

$$l_1 \le l_2 \le l_3 \qquad \textbf{(4.1)}$$

Després es calculen les següents relacions **(4.2)**, **(4.3)** i **(4.4)**, en les dues imatges a aparellar.

$$0 \le \frac{l_1}{l_3} \le 1$$
$$0 \le \frac{l_2}{l_3} \le 1 \qquad \textbf{(4.2) (4.3) (4.4)}$$
$$-1 \le \cos(A) \le 1$$

On A és l'angle més petit entre 2 costats **Fig. 4.8**.

Encara que la triangulació de Delaunay evita la formació de triangles desestructurats, no sempre serà així; llavors aquests triangles no són desitjables i la computació de la transformació geomètrica esdevé inestable. Per aquest motiu es descarten aquells triangles l'angle més gran dels quals és major que un cert llindar (168 graus).

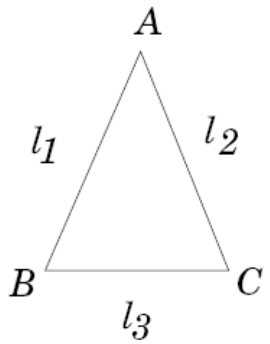


Fig. 4.8 Triangle extret a partir de la triangulació de Delaunay

Quan ja tenim construida la taula d'ambdues imatges ja podem realitzar el matching comparant els resultats obtinguts. Trobariem aquells triangles que són semblants i, de retruc, els objectes que formen els vèrtexs d'aquests triangles serien aparellats.

CONCLUSIONS

La detecció d'objectes i el matching de núvols de punts obtinguts mitjançant un escàner làser terrestre ha estat l'eix central d'aquest treball.

L'algoritme seleccionat per la detecció d'objectes ha estat RANSAC. Després de la seva implementació en podem extreure diferents conclusions. És un algoritme robust, fàcil d'entendre però que requereix una gran cura a l'hora de la implementació ja que en ell hi intervenen molts paràmetres a tenir en compte.

Els paràmetres d'entrada són claus per a la seva eficiència, com també ho és el soroll. En núvols de punts amb poc soroll l'eficiència de RANSAC és bona. Aquesta però, disminueix exponencialment en núvols de punts reals on el soroll és més elevat. En situacions amb molt soroll és doncs un algoritme ineficient.

Una de les limitacions més importants a l'hora d'implementar-lo ha estat el fet d'haver d'introduir, a més de la gran quantitat de paràmetres propis del mateix, paràmetres aproximats propis dels objectes que es volen detectar. Això significa que només serà capaç de detectar aquells objectes dels quals coneixem alguns dels seus paràmetres prèviament; sinó la complexitat i el temps de procés del mateix incrementa considerablement obtenint resultats no sempre satisfactoris.

Una primera implementació de RANSAC va ser amb el Canny distància (OR) intensitat del núvol de punts com un dels paràmetres d'entrada de l'algoritme. Finalment és va desestimar aquesta opció substituint aquesta entrada pels núvols de punts distància i intensitat. Pel fet d'introduir a l'entrada el Canny del núvol de punts, que recordem que és 2D, es perdia informació referent al mateix perquè RANSAC funcionés tal i com s'ha dissenyat (per a treballar en 3D).

La idea del matching que s'ha estudiat, tot i que no ha estat implementada, segueix un esquema rígid i concís. Aprofitant l'algoritme RANSAC i utilitzant la triangulació de Delaunay, es construeix una taula per cadascuna de les imatges a aparellar amb els resultats de la triangulació. Després es procedeix a realitzar el matching. Aquest mètode és utilitzat per al reconeixement d'empremtes dactilars amb uns resultats força bons.

Amb tot, aquest ha estat el producte de l'esforç realitzat durant 6 mesos, i espero que sigui d'utilitat per qui el consulti.

BIBLIOGRAFIA

[1] Rudolf Staiger, “Terrestrial Laser Scanning. Technology, Systems and Applications”, 2nd FIG Regional Conference Marrakech, 2003

[2] Markus-Christian Amann, Thierry Bosch, Marc Lescure, Risto Myllylä Marc Rioux, “Laser ranging: a critical review of usual techniques for distance measurement”, Optical Engineering, Vol. 40 No. 1, January 2001

[3] J.F. Canny, “A computational approach to edge detection”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6): 679-698, 1986

[4] R. Schnabel, R. Wahl, R. Klein, “Shape Detection in point clouds”, Universität Bonn, 2006

[5] Martin A. Fischler, Robert C. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”, Communications of the ACM, 24(6):381-395, 1981

[6] Schnable R., Wahl R., Wessel R. Klein R., “Shape Recognition in 3D Point Clouds”, Computer Graphics Technical Reports, Universität Bonn, 2007

[7] www.infra.kth.se/courses/1E1440/F3_05.ppt

[8] G. Vosselman, “Relational Matching”, Springer-Verlag New York, 1992

[9] G. Bebist, T. Deaconut, M. Georgiopoulos, “Fingerprint Identification Using Delaunay Triangulation”, Department of Computer Science, University of Nevada, 1999

Escola Politècnica Superior
de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTOL DEL TFC: Processat de dades TLS (Terrestrial Laser Scanner)

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Elena Girones Llop

DIRECTOR: Michele Crosetto

SUPERVISOR: Jaume Piera Fernández

DATA: 28 de setembre de 2007

ÍNDEX ANNEXOS

ANNEX 1. FABRICANTS TLS

En la taula que hi ha continuació es poden trobar les pàgines web dels Terrestrial Laser Scanner objecte d'estudi en aquest treball.

FABRICANT	PÀGINA WEB
3rdTech	http://www.3rdtech.com
CALLIDUS	http://www.callidus.de
i-SiTE Pty. Ltd	http://www.isite3d.com
Qsun	http://www.iqvolution.eu
Leica	http://www.leica-geosystems.com
Optech	http://www.optech.ca
Riegl	http://www.riegl.com
Trimble	http://www.trimble.com
Zoller + Fröhlich	http://www.zf-laser.com

ANNEX 2. DETECTOR DE CONTORNS CANNY MITJANÇANT IDL

El detector de contorns de Canny ha estat implementat mitjançant IDL. Aquests són els passos que segueix aquesta aplicació a l'hora d'aplicar-lo:

- **Suavitzar** la imatge mitjançant un filtre Gaussià

- Calcular la **orientació i magnitud del gradient**. Per a fer-ho s'utilitzen un parell de màscares (G_X i G_Y)

- La **direcció del contorn** es calculada fent servir ATAN (G_X,G_Y)
  - Posteriorment és vinculada amb una direcció que pot ser traçada en una imatge

- Un punt del contorn és definit com un punt, la magnitud del gradient del qual, és localment màxima en la direcció del gradient: '**nonmaxima supression'**
  - Després de la 'nonmaxima supression', s'obté una imatge amb valor 0 excepte als punts màxims locals, on el valor de la magnitud del gradient és reservat

- S'aplica **histèresis** per a eliminar els forats
  - Cada píxel en la imatge 'nonmaxima supression' que té un valor més elevat que T_HIGH se suposa que és un píxel de contorn
  - Després cadascun dels píxels que estan connectats a aquest píxel trobat i que tenen un valor més gran que T_LOW, també queda seleccionat per a ser un píxel de contorn

La sintaxi de la funció a utilitzar és la següent:

CANNY=(Imatge [,HIGH=valor][,LOW=valor][,SIGMA=valor]

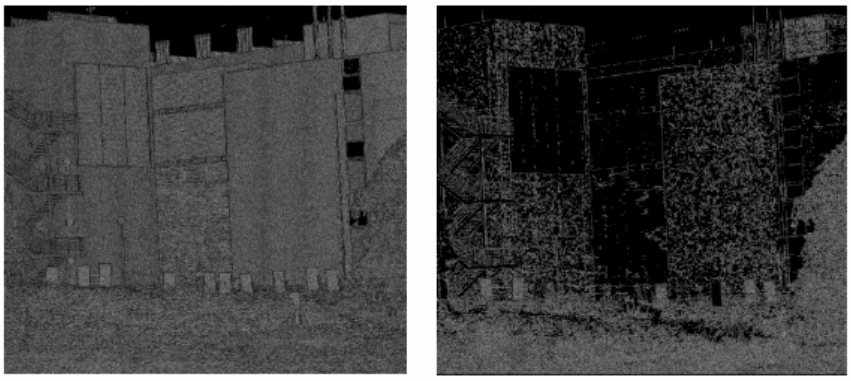
On:
- Imatge: una imatge 2D
- HIGH: utilitzat per a calcular el llindar més alt durant la detecció de contorns
  - El rang d'entrada és [0-1]
  - El valor per defecte és 0,8
- LOW: Utilitzat per a calcular el llindar més baix durant la detecció de contorns, donat com un factor del valor HIGH
  - El rang d'entrada és [0-1]
  - El valor per defecte és 0,4
- SIGMA: tamany del filtre Gaussià de suavitzat
  - El valor per defecte és 0.6

El codi utilitzat és el que es mostra a continuació:

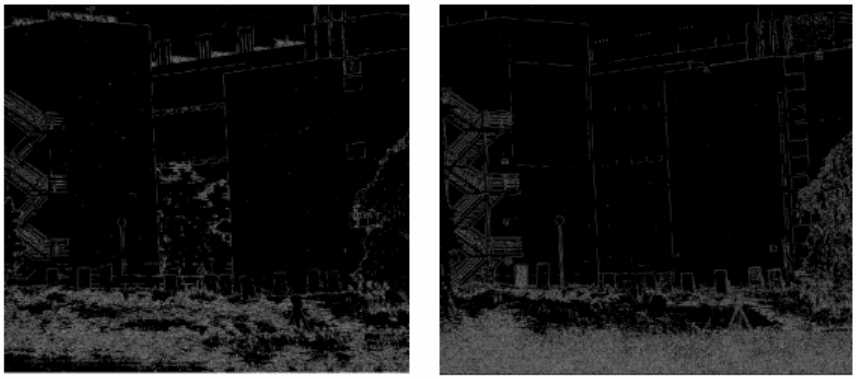
```
DEVICE, DECOMPOSED = 0, RETAIN = 2
LOADCT, 0
image = READ_BINARY(FILEPATH('Nom_imatge.img',
SUBDIRECTORY=['nom_subdirectori_I', 'nom_subdirectori_II']),
DATA_DIMS=[nombre_columnes,nombre_files],DATA_TYPE=4)

dims = SIZE(image, /DIMENSIONS)
WINDOW, 0, XSIZE = nombre_columnes, YSIZE = nombre_files, TITLE =
'CANNY'

cannyImg= CANNY(image,HIGH=0.8,LOW=0.7,SIGMA=5) //valors òptims

TVSCL, cannyImgw
```

Alguns exemples d'aplicació de l'algoritme utilitzant diferents llindars són els que es mostren a continuació:

Imatges distància i intensitat originals



cannyImg= CANNY(image,HIGH=0.8,LOW=0.4,SIGMA=0.6)



cannyImg= CANNY(image,HIGH=0.8,LOW=0.7,SIGMA=5)

ANNEX 3. CODI C ALGORITMES SIMULACIONS

Simulació núvol de punts

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string>
#include <math.h>
#include <conio.h>
#include <search.h>
#include <time.h>

void main()
{
	float x,y,z,xsoroll,ysoroll,zsoroll;
	int i,j;

	FILE *fp;

	fp=fopen("nuvol_punts_soroll.txt","w");

	srand((unsigned)time(NULL));

	z=0;

	//Construcció del núvol de punts
	for(i=0;i<150;i++)
	{
		x=0;
		y=0;
		z=z+0.1;

		for(j=0;j<150;j++)
		{
			x=x+0.1;

			//Definició del nivell de soroll a aplicar al núvol de punts; en aquest cas un nivell de soroll
			//de 2 cm.
			xsoroll=x-0.01+(float)rand()/RAND_MAX*0.02;
			ysoroll=y-0.01+(float)rand()/RAND_MAX*0.02;
			zsoroll=z-0.01+(float)rand()/RAND_MAX*0.02;

			fprintf(fp,"%f\t%f\t%f\t0\n",xsoroll,ysoroll,zsoroll);
		}
	}

	fclose(fp);
}
```

Simulació esferes sobre núvol de punts simulat

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string>
#include <math.h>
#include <conio.h>
#include <search.h>
#include <time.h>
#define MAXELE 22500
```

```
void main()
{
	float centrex1,centrey1,centrez1;
	float centrex2,centrey2,centrez2;
	float centrex3,centrey3,centrez3;
	float radi;
	float y,y0;
	float **bufferaux;
	int i,j,k,cont1,cont2,cont3;

	FILE *fp;
	FILE *fp2;

	fp=fopen("nuvol_punts_soroll.txt","r");
	fp2=fopen("nuvol_punts_soroll_amb_esferes.txt","w");

	bufferaux=(float**)malloc(sizeof(float*)*MAXELE);

	cont1=0;
	cont2=0;
	cont3=0;

	for(i=0;i<MAXELE;i++)
	{
		bufferaux[i]=(float*)malloc(sizeof(float)*4);
	}

	i=0;

	for(j=0;j<MAXELE;j++)
	{
		fscanf(fp,"%f %f %f %f\n",&bufferaux[i][0],&bufferaux[i][1],&bufferaux[i][2],&bufferaux[i][3]);
		i=i+1;
	}

	//Definició de radi i centre de les esferes que s'afegiran al núvol de punts; en aquest cas tres seran les
	//esferes inserides
	radi=(float)0.5;

	centrex1=(float)5;
	centrey1=(float)5;
	centrez1=(float)5;

	centrex2=(float)9;
	centrey2=(float)9;
	centrez2=(float)9;

	centrex3=(float)12;
	centrey3=(float)12;
	centrez3=(float)12;

	//Col·locació de les esferes definides anteriorment al núvol de punts
	for(k=0;k<MAXELE;k++)
	{
		y0=radi*radi-(bufferaux[k][0]-centrex1)*(bufferaux[k][0]-centrex1)-(bufferaux[k][2]-
		centrez1)*(bufferaux[k][2]-centrez1);

		if(y0>=0)
		{
			y=centrey1+sqrt(y0);
			bufferaux[k][1]=bufferaux[k][1]+y;
			cont1 ++;
		}
	}

	for(k=0;k<MAXELE;k++)
	{
		y0=radi*radi-(bufferaux[k][0]-centrex2)*(bufferaux[k][0]-centrex2)-(bufferaux[k][2]-
		centrez2)*(bufferaux[k][2]-centrez2);

		if(y0>=0)
		{
			y=centrey2+sqrt(y0);
			bufferaux[k][1]=bufferaux[k][1]+y;
```

```
                        cont2 ++;
                }
        }

        for(k=0;k<MAXELE;k++)
        {
                y0=radi*radi-(bufferaux[k][0]-centrex3)*(bufferaux[k][0]-centrex3)-(bufferaux[k][2]-
                centrez3)*(bufferaux[k][2]-centrez3);

                if(y0>=0)
                {
                        y=centrey3+sqrt(y0);
                        bufferaux[k][1]=bufferaux[k][1]+y;
                        cont3 ++;
                }
        }

        i=0;

        for(i=0;i<MAXELE;i++)
        {
                fprintf(fp2,"%.3f %.3f %.3f %.0f\n",bufferaux[i][0],bufferaux[i][1],bufferaux[i][2],bufferaux[i][3]);
        }

        printf("Cont1: %d\n",cont1);
        printf("Cont2: %d\n",cont2);
        printf("Cont3: %d\n",cont3);

        fclose(fp);
        fclose(fp2);
}
```

Simulació plans sobre núvol de punts simulat

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string>
#include <math.h>
#include <conio.h>
#include <search.h>
#include <time.h>
#define MAXELE 22500

void main()
{
        float **bufferaux;
        int i,j,k,cont1,cont2,cont3;
        float a,b,c,d,x,y,z,aux1,aux2;

        FILE *fp;
        FILE *fp2;

        fp=fopen("nuvol_punts_soroll.txt","r");
        fp2=fopen("nuvol_punts_soroll_plans.txt","w");

        bufferaux=(float**)malloc(sizeof(float*)*MAXELE);

        a=2;
        b=3;
        c=4;
        d=5;

        cont1=0;
        cont2=0;
        cont3=0;
```

```
for(i=0;i<MAXELE;i++)
{
	bufferaux[i]=(float*)malloc(sizeof(float)*4);
}

i=0;

for(j=0;j<MAXELE;j++)
{
	fscanf(fp,"%f %f %f %f\n",&bufferaux[i][0],&bufferaux[i][1],&bufferaux[i][2],&bufferaux[i][3]);
	i=i+1;
}

i=0;
k=0;
aux1=1100;
aux2=1120;

while(k!=10)
{
	for(i=aux1;i<=aux2;i++)
	{
		bufferaux[i][1]=bufferaux[i][1]+(-a*bufferaux[i][0]-c*bufferaux[i][2]-d)/b-
		0.01+(float)rand()/RAND_MAX*0.02;
		cont1++;
	}
	k++;
	aux1=1100+150*k;
	aux2=1120+150*k;
}

i=0;
k=0;
aux1=7700;
aux2=7720;

while(k!=10)
{
	for(i=aux1;i<=aux2;i++)
	{
		bufferaux[i][1]=bufferaux[i][1]+(-a*bufferaux[i][0]-c*bufferaux[i][2]-d)/b-
		0.01+(float)rand()/RAND_MAX*0.02;
		cont2++;
	}
	k++;
	aux1=7700+150*k;
	aux2=7720+150*k;
}

i=0;
k=0;
aux1=18800;
aux2=18820;

while(k!=10)
{
	for(i=aux1;i<=aux2;i++)
	{
		bufferaux[i][1]=bufferaux[i][1]+(-a*bufferaux[i][0]-c*bufferaux[i][2]-d)/b-
		0.01+(float)rand()/RAND_MAX*0.02;
		cont3++;
	}
	k++;
	aux1=18800+150*k;
	aux2=18820+150*k;
}

i=0;

for(i=0;i<MAXELE;i++)
{
	fprintf(fp2,"%.3f %.3f %.3f %.0f\n",bufferaux[i][0],bufferaux[i][1],bufferaux[i][2],bufferaux[i][3]);
}

printf("Cont1: %d\n",cont1);
printf("Cont2: %d\n",cont2);
```

```
	printf("Cont3: %d\n",cont3);

	fclose(fp);
	fclose(fp2);
}
```

Simulació esferes sobre núvol de punts real

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string>
#include <math.h>
#include <conio.h>
#include <search.h>
#include <time.h>
#define MAXELE 984312


void main()
{
	float centrex1,centrey1,centrez1;
	float centrex2,centrey2,centrez2;
	float centrex3,centrey3,centrez3;
	float centrex4,centrey4,centrez4;
	float centrex5,centrey5,centrez5;
	float centrex6,centrey6,centrez6;
	float radi;
	float y,y0;
	float **bufferaux;
	int i,j,k,cont1,cont2,cont3,cont4,cont5,cont6;

	FILE *fp;
	FILE *fp2;

	fp=fopen("scanner2_task2.xyz","r");
	fp2=fopen("scanner2_task2_amb_esferes.xyz","w");

	bufferaux=(float**)malloc(sizeof(float*)*MAXELE);

	cont1=0;
	cont2=0;
	cont3=0;
	cont4=0;
	cont5=0;
	cont6=0;

	for(i=0;i<MAXELE;i++)
	{
		bufferaux[i]=(float*)malloc(sizeof(float)*4);
	}

	i=0;

	for(j=0;j<MAXELE;j++)
	{
		fscanf(fp,"%f %f %f %f\n",&bufferaux[i][0],&bufferaux[i][1],&bufferaux[i][2],&bufferaux[i][3]);
		i=i+1;
	}

	//Definició de radi i centre de les esferes que s'afegiran al núvol de punts; en aquest cas sis seran
	les esferes inserides
	radi=(float)0.5;

	centrex1=(float)-14.446;
	centrey1=(float)200;
	centrez1=(float)1.002;
	centrex2=(float)9.164;
	centrey2=(float)200;
```

```
centrez2=(float)-11.47;
centrex3=(float)-17.315001;
centrey3=(float)200;
centrez3=(float)13.84;
centrex4=(float)27.839001;
centrey4=(float)200;
centrez4=(float)13.426;
centrex5=(float)7.866;
centrey5=(float)200;
centrez5=(float)34.693001;

//Col·locació de les esferes definides anteriorment al núvol de punts
for(k=0;k<MAXELE;k++)
{
	y0=radi*radi-(bufferaux[k][0]-centrex1)*(bufferaux[k][0]-centrex1)-(bufferaux[k][2]-
	centrez1)*(bufferaux[k][2]-centrez1);

	if(y0>=0)
	{
		y=centrey1+sqrt(y0);
		bufferaux[k][1]=bufferaux[k][1]+y;
		cont1 ++;
	}
}

for(k=0;k<MAXELE;k++)
{
	y0=radi*radi-(bufferaux[k][0]-centrex2)*(bufferaux[k][0]-centrex2)-(bufferaux[k][2]-
	centrez2)*(bufferaux[k][2]-centrez2);

	if(y0>=0)
	{
		y=centrey2+sqrt(y0);
		bufferaux[k][1]=bufferaux[k][1]+y;
		cont2 ++;
	}
}

for(k=0;k<MAXELE;k++)
{
	y0=radi*radi-(bufferaux[k][0]-centrex3)*(bufferaux[k][0]-centrex3)-(bufferaux[k][2]-
	centrez3)*(bufferaux[k][2]-centrez3);

	if(y0>=0)
	{
		y=centrey3+sqrt(y0);
		bufferaux[k][1]=bufferaux[k][1]+y;
		cont3 ++;
	}
}

for(k=0;k<MAXELE;k++)
{
	y0=radi*radi-(bufferaux[k][0]-centrex4)*(bufferaux[k][0]-centrex4)-(bufferaux[k][2]-
	centrez4)*(bufferaux[k][2]-centrez4);

	if(y0>=0)
	{
		y=centrey4+sqrt(y0);
		bufferaux[k][1]=bufferaux[k][1]+y;
		cont4 ++;
	}
}

for(k=0;k<MAXELE;k++)
{
	y0=radi*radi-(bufferaux[k][0]-centrex5)*(bufferaux[k][0]-centrex5)-(bufferaux[k][2]-
	centrez5)*(bufferaux[k][2]-centrez5);

	if(y0>=0)
	{
		y=centrey5+sqrt(y0);
		bufferaux[k][1]=bufferaux[k][1]+y;
		cont5 ++;
	}
}
```

```
	for(k=0;k<MAXELE;k++)
	{
		y0=radi*radi-(bufferaux[k][0]-centrex6)*(bufferaux[k][0]-centrex6)-(bufferaux[k][2]-
		centrez6)*(bufferaux[k][2]-centrez6);

		if(y0>=0)
		{
			y=centrey6+sqrt(y0);
			bufferaux[k][1]=bufferaux[k][1]+y;
			cont6 ++;
		}
	}


	i=0;

	for(i=0;i<MAXELE;i++)
	{
		fprintf(fp2,"%.3f %.3f %.3f %.0f\n",bufferaux[i][0],bufferaux[i][1],bufferaux[i][2],bufferaux[i][3]);
	}

	printf("Cont1: %d\n",cont1);
	printf("Cont2: %d\n",cont2);
	printf("Cont3: %d\n",cont3);
	printf("Cont4: %d\n",cont4);
	printf("Cont5: %d\n",cont5);
	printf("Cont6: %d\n",cont6);

	fclose(fp);
	fclose(fp2);
}
```

Simulació plans sobre núvol de punts real

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string>
#include <math.h>
#include <conio.h>
#include <search.h>
#include <time.h>
#define MAXELE 984312

void main()
{
	float **bufferaux;
	int i,j,k,cont1,cont2,cont3,cont4,cont5;
	float a,b,c,d,aux1,aux2;

	FILE *fp;
	FILE *fp2;

	fp=fopen("scanner2_task2.xyz","r");
	fp2=fopen("scanner2_task2_amb_plans.xyz","w");

	bufferaux=(float**)malloc(sizeof(float*)*MAXELE);

	a=2;
	b=3;
	c=4;
	d=5;

	cont1=0;
	cont2=0;
	cont3=0;
	cont4=0;
	cont5=0;
```

```
for(i=0;i<MAXELE;i++)
{
	bufferaux[i]=(float*)malloc(sizeof(float)*4);
}

i=0;

for(j=0;j<MAXELE;j++)
{
	fscanf(fp,"%f %f %f %f\n",&bufferaux[i][0],&bufferaux[i][1],&bufferaux[i][2],&bufferaux[i][3]);
	i=i+1;
}

 i=0;
k=0;
aux1=113551;
aux2=113611;

while(k!=50)
{
	for(i=aux1;i<=aux2;i++)
	{
		bufferaux[i][1]=bufferaux[i][1]+(-a*bufferaux[i][0]-c*bufferaux[i][2]-d)/b-
		0.01+(float)rand()/RAND_MAX*0.02;
		cont1++;
	}
	k++;
	aux1=113551+1134*k;
	aux2=113611+1134*k;
}

i=0;
k=0;
aux1=341033;
aux2=341093;

while(k!=50)
{
	for(i=aux1;i<=aux2;i++)
	{
		bufferaux[i][1]=bufferaux[i][1]+(-a*bufferaux[i][0]-c*bufferaux[i][2]-d)/b-
		0.01+(float)rand()/RAND_MAX*0.02;
		cont2++;
	}
	k++;
	aux1=341033+1134*k;
	aux2=341093+1134*k;
}

i=0;
k=0;
aux1=567500;
aux2=567560;

while(k!=50)
{
	for(i=aux1;i<=aux2;i++)
	{
		bufferaux[i][1]=bufferaux[i][1]+(-a*bufferaux[i][0]-c*bufferaux[i][2]-d)/b-
		0.01+(float)rand()/RAND_MAX*0.02;
		cont3++;
	}
	k++;
	aux1=567500+1134*k;
	aux2=567560+1134*k;
}

i=0;
k=0;
aux1=681150;
aux2=681210;

while(k!=50)
{
	for(i=aux1;i<=aux2;i++)
```

```
		{
			bufferaux[i][1]=bufferaux[i][1]+(-a*bufferaux[i][0]-c*bufferaux[i][2]-d)/b-
			0.01+(float)rand()/RAND_MAX*0.02;
			cont4++;
		}
		k++;
		aux1=681150+1134*k;
		aux2=681210+1134*k;
	}

	i=0;
	k=0;
	aux1=340600;
	aux2=340660;

	while(k!=50)
	{
		for(i=aux1;i<=aux2;i++)
		{
			bufferaux[i][1]=bufferaux[i][1]+(-a*bufferaux[i][0]-c*bufferaux[i][2]-d)/b-
			0.01+(float)rand()/RAND_MAX*0.02;
			cont5++;
		}
		k++;
		aux1=340600+1134*k;
		aux2=340660+1134*k;
	}

	i=0;

	for(i=0;i<MAXELE;i++)
	{
		fprintf(fp2,"%.3f %.3f %.3f %.0f\n",bufferaux[i][0],bufferaux[i][1],bufferaux[i][2],bufferaux[i][3]);
	}

	printf("Cont1: %d\n",cont1);
	printf("Cont2: %d\n",cont2);
	printf("Cont3: %d\n",cont3);
	printf("Cont4: %d\n",cont4);
	printf("Cont5: %d\n",cont5);

	fclose(fp);
	fclose(fp2);
}
```

ANNEX 4. CODI C ALGORITME RANSAC

Main: **RANSAC.cpp**

```
#include "lidar.h"

void ransac_pla(long kkk,pointf *points_window, long finestra);
void ransac_esfera(long kkk,pointf *points_window, long finestra);

pointf *points_window;

void main()
{
	long var=0;
	long aux1,aux2;
	long iii,jjj,multiplicacio;
	long k,w,kk,ww,kk2,ww2,cont;
	long ransac;
	long finestra=0;

	srand( (unsigned)time( NULL ) );

	read_data("input.txt",var);

	points_window=new pointf[datasets[0].windowlin*datasets[0].windowcol];

	aux1=nlin[0]/datasets[0].windowlin;
	aux2=ncol[0]/datasets[0].windowcol;

	k=0;
	w=0;
	kk2=k+datasets[0].windowlin-1;
	ww2=w+datasets[0].windowcol-1;

	printf("******* RANSAC *********\n\n");
	printf("Escull la deteccio de figures que vulguis dur a terme:\n\n");
	printf("\tEsfera --> 1\n");
	printf("\tPla --> 2\n");
	scanf("%d",&ransac);

	//Recorre cadascuna de les finestres en que s'ha dividit el fitxer d'entrada
	for(iii=0;iii<aux1;iii++)
	{
		for(jjj=0;jjj<aux2;jjj++)
		{
			cont=0;

			//Guarda en un vector els punts de cada finestra; RANSAC s'aplicarà sobre aquest vector
			if(kk2-k==datasets[0].windowlin-1)
			{
				for(kk=k;kk<kk2;kk++)
				{
					if(ww2-w==datasets[0].windowcol-1)
					{
						for(ww=w;ww<ww2;ww++)
						{
							if(point_grid[0][kk][ww] != -1)
							{

	points_window[cont].X=points[point_grid[0][kk][ww]].X;
	points_window[cont].Y=points[point_grid[0][kk][ww]].Y;



	points_window[cont].Z=points[point_grid[0][kk][ww]].Z;

								cont++;
							}
```

```
                                                }
                                        }
                                }
                        }

                        if(cont!=0 && cont!=1)
                        {
                                switch(ransac){
                                case 1:
                                        ransac_esfera(cont,points_window,finestra);
                                        finestra++;
                                        break;
                                case 2:
                                        ransac_pla(cont,points_window,finestra);
                                        finestra++;
                                        break;
                                }

                        }
                        w=w++;
                        ww2=w+datasets[0].windowcol-1;
                }
                w=0;
                ww2=w+datasets[0].windowcol-1;
                k=k++;
                kk2=k+datasets[0].windowlin-1;
        }

        multiplicacio=(iii)*(jjj);

        printf("Nombre total de finestres recorregudes: %d \n\n\n\n",multiplicacio);
}
```

Funcions per a calcular RANSAC: **RANSAC_ESFERA.cpp**

```
#include "lidar.h"

void ransac_esfera(long kkk, pointf *points_window, long finestra)
{
        long k1,k2,k3,k4;
        float x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,m11,x0,y0,z0,r0,xaux,yaux,zaux,r00;
        float t;
        long i;
        long tambe_inliers,total_inliers,possibles_inliers,model_definitiu;
        double iteracions1,iteracions2,iteracions;
        float prob_error,numero_mostres;
        float d;
        int aux;

        FILE *fp;

        fp=fopen("OUT.txt","a");

        for(aux=0;aux<filesnum;aux++)
        {
                d=datasets[aux].percentage*(datasets[aux].windowlin*datasets[aux].windowcol);
                iteracions=100000;
                numero_mostres=0;
                model_definitiu=0;

                while(iteracions>numero_mostres)
                {
                        possibles_inliers=4;
                        //Elecció de 4 punts aleatoris;nombre mínim de punts necessaris per a definir una esfera

                        k1=rand() % (kkk-1);
                        k2=rand() % (kkk-1);
                        k3=rand() % (kkk-1);
                        k4=rand() % (kkk-1);
```

```
x1=points_window[k1].X;
y1=points_window[k1].Y;
z1=points_window[k1].Z;
x2=points_window[k2].X;
y2=points_window[k2].Y;
z2=points_window[k2].Z;
x3=points_window[k3].X;
y3=points_window[k3].Y;
z3=points_window[k3].Z;
x4=points_window[k4].X;
y4=points_window[k4].Y;
z4=points_window[k4].Z;

//Càlcul de M11
m11=(x1*y2*z3-x1*y2*z4-x1*y3*z2+x1*y3*z4+x1*y4*z2-x1*y4*z3-
x2*y1*z3+x2*y1*z4+x2*y3*z1-x2*y3*z4-x2*y4*z1+x2*y4*z3+x3*y1*z2-x3*y1*z4-x3*y2*z1+x3*y2*z4+x3*y4*z1-x3*y4*z2-
x4*y1*z2+x4*y1*z3+x4*y2*z1-x4*y2*z3-x4*y3*z1+x4*y3*z2);

//Si M11=0 --> Coplanaris
if(m11=!0)
{
	//Càlcul component x centre esfera
	x0=(0.5*(x4*x4*y1*z3-z2*z2*y1*z3+y2*y2*y3*z1-x4*x4*y3*z1-x1*x1*y3*z2-
z4*z4*y1*z2-y1*y1*y3*z2-y4*y4*y3*z1-x4*x4*y1*z2+z2*z2*y3*z1+y4*y4*y3*z2-y3*y3*y1*z4+y2*y2*y1*z4-x1*x1*y2*z4-
y2*y2*y1*z3-y2*y2*y4*z1-x4*x4*y2*z3-x3*x3*y4*z2-
y3*y3*y2*z1+y4*y4*y2*z1+z3*z3*y4*z1+y1*y1*y4*z2+x2*x2*y1*z4+x1*x1*y4*z2+z4*z4*y1*z3-
z3*z3*y1*z4+z4*z4*y2*z1-x3*x3*y1*z4+z1*z1*y4*z2-
z4*z4*y2*z3+y1*y1*y3*z4+x1*x1*y2*z3+y3*y3*y2*z4+x3*x3*y1*z2+z1*z1*y3*z4+z4*z4*y3*z2-
z3*z3*y4*z2+x2*x2*y4*z3+x3*x3*y2*z4-z2*z2*y3*z4-z1*z1*y2*z4-y4*y4*y1*z2-y3*y3*y4*z2-
z1*z1*y3*z2+x4*x4*y2*z1+y3*y3*y4*z1-y1*y1*y2*z4+x1*x1*y3*z4-z3*z3*y2*z1+z3*z3*y1*z2+z1*z1*y2*z3+y2*y2*y4*z3-
y2*y2*y3*z4+x3*x3*y4*z1+x4*x4*y3*z2+z3*z3*y2*z4-x1*x1*y4*z3-x3*x3*y2*z1-y1*y1*y4*z3-z4*z4*y3*z1+x2*x2*y3*z1-
y4*y4*y2*z3+y1*y1*y2*z3+z2*z2*y1*z4+y3*y3*y1*z2-z1*z1*y4*z3-x2*x2*y1*z3-x2*x2*y4*z1+y4*y4*y1*z3-x2*x2*y3*z4-
z2*z2*y4*z1+z2*z2*y4*z3)/(x1*y2*z3-x1*y2*z4-x1*y3*z2+x1*y3*z4+x1*y4*z2-x1*y4*z3-x2*y1*z3+x2*y1*z4+x2*y3*z1-
x2*y3*z4-x2*y4*z1+x2*y4*z3+x3*y1*z2-x3*y1*z4-x3*y2*z1+x3*y2*z4+x3*y4*z1-x3*y4*z2-x4*y1*z2+x4*y1*z3+x4*y2*z1-
x4*y2*z3-x4*y3*z1+x4*y3*z2));
	//Càlcul component y centre esfera
	y0=		-(0.5*(z3*z3*x1*z2+z3*z3*x2*z4+z4*z4*x3*z2-z4*z4*x3*z1-
z4*z4*x2*z3+z4*z4*x2*z1-y2*y2*x4*z1-x4*x4*x3*z1-x2*x2*x1*z3+z3*z3*x4*z1-x3*x3*x2*z1+x4*x4*x2*z1-
y3*y3*x2*z1+y1*y1*x3*z4-z1*z1*x4*z3+x1*x1*x2*z3-z3*z3*x1*z4+y2*y2*x4*z3+x4*x4*x3*z2-
y4*y4*x1*z2+y1*y1*x2*z3+y3*y3*x1*z2+x3*x3*x2*z4+x1*x1*x4*z2+z2*z2*x1*z4-x3*x3*x1*z4-y1*y1*x2*z4+y4*y4*x2*z1-
x2*x2*x4*z1+z2*z2*x3*z1+y4*y4*x3*z2+z1*z1*x2*z3+x3*x3*x1*z2-z4*z4*x1*z2-z3*z3*x2*z1+x3*x3*x4*z1+y1*y1*x4*z2-
y2*y2*x3*z4+y3*y3*x2*z4+y3*y3*x4*z1-y4*y4*x3*z1-y1*y1*x4*z3-
z2*z2*x4*z1+z1*z1*x4*z2+z4*z4*x1*z3+z1*z1*x3*z4+x2*x2*x4*z3+x1*x1*x3*z4-z3*z3*x4*z2+y4*y4*x1*z3-x1*x1*x4*z3-
x2*x2*x3*z4-x4*x4*x1*z2-x1*x1*x3*z2-x3*x3*x4*z2-z1*z1*x2*z4+x4*x4*x1*z3-y2*y2*x1*z3-z1*z1*x3*z2-
y1*y1*x3*z2+z2*z2*x4*z3-z2*z2*x1*z3-z2*z2*x3*z4+x2*x2*x3*z1+y2*y2*x1*z4-y3*y3*x4*z2+x2*x2*x1*z4+y2*y2*x3*z1-
x4*x4*x2*z3-y3*y3*x1*z4-x1*x1*x2*z4-y4*y4*x2*z3)/(x1*y2*z3-x1*y2*z4-x1*y3*z2+x1*y3*z4+x1*y4*z2-x1*y4*z3-
x2*y1*z3+x2*y1*z4+x2*y3*z1-x2*y3*z4-x2*y4*z1+x2*y4*z3+x3*y1*z2-x3*y1*z4-x3*y2*z1+x3*y2*z4+x3*y4*z1-x3*y4*z2-
x4*y1*z2+x4*y1*z3+x4*y2*z1-x4*y2*z3-x4*y3*z1+x4*y3*z2));
	//Càlcul component z centre esfera

	z0=(0.5*(x3*x3*x1*y2+y2*y2*x3*y1+z3*z3*x2*y4+y3*y3*x4*y1+z1*z1*x2*y3+z2*z2*x4*y3-y2*y2*x1*y3-
z1*z1*x3*y2-y2*y2*x4*y1-y1*y1*x4*y3+y2*y2*x1*y4+x2*x2*x3*y1+y1*y1*x2*y3-x2*x2*x1*y3-z3*z3*x4*y2-x4*x4*x3*y1-
x1*x1*x2*y4+y2*y2*x4*y3-z2*z2*x4*y1-x2*x2*x4*y1-z4*z4*x1*y2+y1*y1*x3*y4-
y1*y1*x3*y2+x3*x3*x2*y4+x1*x1*x3*y4+x2*x2*x4*y3-x4*x4*x2*y3-
z4*z4*x3*y1+z2*z2*x1*y4+z2*z2*x3*y1+x4*x4*x2*y1+z1*z1*x3*y4-x3*x3*x1*y4+x1*x1*x4*y2-y2*y2*x3*y4+z3*z3*x4*y1-
z1*z1*x4*y3-z2*z2*x3*y4+x2*x2*x1*y4-x1*x1*x4*y3+y4*y4*x2*y1-x3*x3*x4*y2-y4*y4*x2*y3+z1*z1*x4*y2+x4*x4*x1*y3-
y4*y4*x1*y2+z4*z4*x2*y1+z3*z3*x1*y2-x1*x1*x3*y2-x2*x2*x3*y4-y3*y3*x2*y1+y3*y3*x2*y4+z4*z4*x1*y3-z3*z3*x1*y4-
y1*y1*x2*y4-y3*y3*x1*y4-x4*x4*x1*y2+x3*x3*x4*y1-y3*y3*x4*y2-
z3*z3*x2*y1+x1*x1*x2*y3+y1*y1*x4*y2+y4*y4*x1*y3+z4*z4*x3*y2-z1*z1*x2*y4-z4*z4*x2*y3+y3*y3*x1*y2-x3*x3*x2*y1-
y4*y4*x3*y1+y4*y4*x3*y2-z2*z2*x1*y3+x4*x4*x3*y2)/(x1*y2*z3-x1*y2*z4-x1*y3*z2+x1*y3*z4+x1*y4*z2-x1*y4*z3-
x2*y1*z3+x2*y1*z4+x2*y3*z1-x2*y3*z4-x2*y4*z1+x2*y4*z3+x3*y1*z2-x3*y1*z4-x3*y2*z1+x3*y2*z4+x3*y4*z1-x3*y4*z2-
x4*y1*z2+x4*y1*z3+x4*y2*z1-x4*y2*z3-x4*y3*z1+x4*y3*z2));
	//Càlcul del radi de l'esfera
	r0=(sqrt(x0*x0+y0*y0+z0*z0-(y1*y1*x3*y4*z2-y1*y1*x3*y2*z4+y1*y1*x4*y2*z3-
z1*z1*x2*y4*z3+z1*z1*x2*y3*z4-y1*y1*x4*y3*z2+z1*z1*x4*y2*z3+z1*z1*x3*y4*z2-
z1*z1*x3*y2*z4+z2*z2*x3*y1*z4+z2*z2*x1*y4*z3-z2*z2*x1*y3*z4+x2*x2*x4*y3*z1-x2*x2*x4*y1*z3-
y2*y2*x3*y4*z1+y2*y2*x3*y1*z4+y2*y2*x1*y4*z3-y2*y2*x1*y3*z4+x2*x2*x4*y3*z1-x2*x2*x4*y1*z3-
x2*x2*x3*y4*z1+x2*x2*x3*y1*z4+x2*x2*x1*y4*z3-x2*x2*x1*y3*z4-z1*z1*x4*y3*z2+y3*y3*x4*y1*z2+y3*y3*x2*y4*z1-
y3*y3*x2*y1*z4-y3*y3*x1*y4*z2+y3*y3*x1*y2*z4-x3*x3*x4*y2*z1+x3*x3*x4*y1*z2+x3*x3*x2*y4*z1-x3*x3*x2*y1*z4-
x3*x3*x1*y4*z2+x3*x3*x1*y2*z4+z2*z2*x4*y3*z1-z2*z2*x4*y1*z3-z2*z2*x3*y4*z1-
y4*y4*x2*y3*z1+y4*y4*x2*y1*z3+y4*y4*x1*y3*z2-y4*y4*x1*y2*z3+x4*x4*x3*y2*z1-x4*x4*x3*y1*z2-
x4*x4*x2*y3*z1+x4*x4*x2*y1*z3+x4*x4*x1*y3*z2-x4*x4*x1*y2*z3-z3*z3*x4*y2*z1+z3*z3*x4*y1*z2+z3*z3*x2*y4*z1-
z3*z3*x2*y1*z4-z3*z3*x1*y4*z2+z3*z3*x1*y2*z4-y3*y3*x4*y2*z1+z4*z4*x3*y2*z1-z4*z4*x3*y1*z2-
z4*z4*x2*y3*z1+z4*z4*x2*y1*z3+z4*z4*x1*y3*z2-z4*z4*x1*y2*z3+y4*y4*x3*y2*z1-y4*y4*x3*y1*z2-
x1*x1*x2*y4*z3+x1*x1*x4*y2*z3-x1*x1*x4*y3*z2+x1*x1*x3*y4*z2-y1*y1*x2*y4*z3-
x1*x1*x3*y2*z4+x1*x1*x2*y3*z4+y1*y1*x2*y3*z4)/(x1*y2*z3-x1*y2*z4-x1*y3*z2+x1*y3*z4+x1*y4*z2-x1*y4*z3-
```

```
x2*y1*z3+x2*y1*z4+x2*y3*z1-x2*y3*z4-x2*y4*z1+x2*y4*z3+x3*y1*z2-x3*y1*z4-x3*y2*z1+x3*y2*z4+x3*y4*z1-x3*y4*z2-
x4*y1*z2+x4*y1*z3+x4*y2*z1-x4*y2*z3-x4*y3*z1+x4*y3*z2)));

                                        //Es tracta de veure si els punts que no han estat escollits aleatoriament
                                        //s'ajusten al model anterior
                                        tambe_inliers=0;

                                        t=datasets[aux].tolerance;
                                        i=0;

                                        if(r0<datasets[aux].parametre_esfera1 && r0>datasets[aux].parametre_esfera2)
                                        {
                                                for(i=0;i<kkk;i++)
                                                {
                                                        xaux=points_window[i].X-x0;
                                                        yaux=points_window[i].Y-y0;
                                                        zaux=points_window[i].Z-z0;

                                                        r00=(sqrt(xaux*xaux+yaux*yaux+zaux*zaux));
                                                        if(fabs(r00-r0)<t)
                                                        {
                                                                tambe_inliers++;
                                                        }
                                                }
                                                total_inliers=possibles_inliers+tambe_inliers;

                                                if(total_inliers>d)
                                                {
                                                        if(total_inliers>model_definitiu)
                                                        {
                                                                model_definitiu=total_inliers;
                                                                fprintf(fp,"Numero de punts bons dintre de la
                                                                finestra %d: %d\n\n",finestra, kkk);
                                                                fprintf(fp,"\tTotal_inliers: %d\n", model_definitiu);
                                                                fprintf(fp,"\tAquests son els parametres del model:
                                                                \n");
                                                                fprintf(fp,"\tCentre: (%f,%f,%f)",x0,y0,z0);
                                                                fprintf(fp,"\t\tRadi: %f \n",r0);

        fprintf(fp,"**********************************************************************\n\n");
                                                        }
                                                        else
                                                        {
                                                                model_definitiu=model_definitiu;
                                                        }
                                                }
                                                prob_error=1-(float)model_definitiu/(float)kkk;
                                                iteracions1=log10(1-0.99);
                                                iteracions2=log10(1-(1-prob_error)*(1-prob_error)*(1-prob_error)*(1-
                                                prob_error));
                                                iteracions=iteracions1/iteracions2;
                                        }
                                }
                                numero_mostres++;
                        }
        }
        finestra++;
        fclose(fp);
}
```

Funcions per a calcular RANSAC: **RANSAC_PLA.cpp**

```
#include "lidar.h"

void ransac_pla(long kkk, pointf *points_window,long finestra)

{

        long A,B,C;
        float a,b,c,D,distancia_punt_pla,AB,AC;
        long i;
```

```
	long tambe_inliers,total_inliers,possibles_inliers,model_definitiu;
	double iteracions1,iteracions2,iteracions;
	float prob_error,numero_mostres;
	float d;

	FILE *fp;

	fp=fopen("OUT.txt","a");

	d=datasets[0].percentage*(datasets[0].windowlin*datasets[0].windowcol);
	iteracions=100000;
	numero_mostres=0;
	model_definitiu=0;

	while(iteracions>numero_mostres)
	{
		possibles_inliers=3;

		//Elecció de 3 punts aleatoris;nombre mínim de punts necessaris per a definir un pla
		A=rand() % (kkk-1);
		B=rand() % (kkk-1);
		C=rand() % (kkk-1);

		AB=sqrt((points_window[A].X-points_window[B].X)*(points_window[A].X-
points_window[B].X)+(points_window[A].Y-points_window[B].Y)*(points_window[A].Y-
points_window[B].Y)+(points_window[A].Z-points_window[B].Z)*(points_window[A].Z-points_window[B].Z));
		AC=sqrt((points_window[A].X-points_window[C].X)*(points_window[A].X-
points_window[C].X)+(points_window[A].Y-points_window[C].Y)*(points_window[A].Y-
points_window[C].Y)+(points_window[A].Z-points_window[C].Z)*(points_window[A].Z-points_window[C].Z));

		a=(points_window[A].Y-points_window[B].Y)*(points_window[A].Z-points_window[C].Z)-
(points_window[A].Z-points_window[B].Z)*(points_window[A].Y-points_window[C].Y);
		b=-(points_window[A].X-points_window[B].X)*(points_window[A].Z-points_window[C].Z)-
(points_window[A].Z-points_window[B].Z)*(points_window[A].X-points_window[C].X);
		c=(points_window[A].X-points_window[B].X)*(points_window[A].Y-points_window[C].Y)-
(points_window[A].Y-points_window[B].Y)*(points_window[A].X-points_window[C].X);

		D=a*points_window[C].X-b*points_window[C].Y+c*points_window[C].Z;

		//Es tracta de veure si els punts que no han estat escollits aleatoriament s'ajusten al model anterior
		tambe_inliers=0;

		i=0;


		for(i=0;i<kkk;i++)
		{
			//Càlcul de la distància entre cadascun dels punts i el pla
			distancia_punt_pla=
abs((a*points_window[i].X+b*points_window[i].Y+c*points_window[i].Z+D)/(sqrt(a*a+b*b+c*c)));

			if(distancia_punt_pla==0)
			{
				tambe_inliers++;
			}
		}
		total_inliers=possibles_inliers+tambe_inliers;

		//Es tracta de veure si els punts que no han estat escollits aleatoriament s'ajusten al model anterior
		if(D<datasets[0].parametre_pla1 && D>datasets[0].parametre_pla2)
		{
			if(total_inliers>d)
			{
				if(total_inliers>model_definitiu)
				{
					model_definitiu = total_inliers;

					fprintf(fp,"Numero de punts bons dintre de la finestra %d:
%d\n\n",finestra, kkk);
					fprintf(fp,"\tTotal_inliers: %d\n", model_definitiu);


					fprintf(fp,"\tAquests son els parametres del model: \n");
					fprintf(fp,"\t\t %f x + %f y + %f z + %f\n",a,b,c,D);
```

```
		fprintf(fp,"*****************************************************************\n\n");
                                        }
                                        else
                                        {
                                                model_definitiu=model_definitiu;
                                        }
                                }
                                prob_error=1-(float)model_definitiu/(float)kkk;
                                iteracions1=log10(1-0.99);
                                iteracions2=log10(1-(1-prob_error)*(1-prob_error)*(1-prob_error));
                                iteracions=iteracions1/iteracions2;
                        }
                        numero_mostres++;
                }
                fclose(fp);
}
```