UVIC
UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL
DE CATALUNYA

eurecat
Centre Tecnològic de Catalunya

Treball Final de Màster

Implementació d'odometria visual per vehicles autònoms

David Ortiz Martínez

Màster en Robòtica

Tutor: Andreu Corominas Murtra

Barcelona, Novembre del 2017

RESUM TREBALL FINAL DE MÀSTER
MÀSTER EN ROBÒTICA

Títol: Implementació d'odometria visual per vehicles autònoms

Paraules clau: Odometria visual, ORB, KITTI *dataset*, Visió per ordinador, vehicles autònoms.

Autor: David Ortiz Martínez

Tutor: Andreu Corominas Murtra (Beta Robots)

Data: Barcelona, Novembre 2017

A la mateixa velocitat que la tecnologia continua avançant, els vehicles autònoms són cada vegada una realitat més palpable. De tots els reptes amb què aquest tipus de vehicles s'enfronten, un dels més importants es el de realitzar un sistema capaç de proporcionar una posició precisa del vehicle en el món, especialment en entorns tancats on no hi ha GPS. Un mètode per solucionar aquest problema és estimar incrementalment el moviment utilitzant imatges preses per una càmera digital, mètode conegut com Odometría Visual.

En aquesta tesi del màster s'expliquen els diferents dispositius dels que es pot obtenir l'odometria, els diferents tipus d'odometria visual, i la teoria matemàtica que hi ha darrera, incloent la detecció de punts característics, emparellament de punts característics, estimació de moviment, etc.

La tesi s'ha realitzat processant dades *offline*. Per això s'explica el conjunt de dades reals (*dataset*) que s'han utilitzat per poder provar els algoritmes d'odometria visual.

Per últim, s'introdueixen les diverses implementacions que s'han realitzat. Tot el codi s'ha desenvolupat en C++ i amb la llibreria OpenCV, funcionant sota ROS. Els resultats han donat una informació dels problemes que es poden trobar al implementar l'algoritme en un dispositiu real.

Barcelona, November 2017

MASTER'S THESIS ABSTRACT

MASTER IN ROBOTICS

Title: Visual odometry implementation for autonomous vehicles

Keywords: Visual odometry, ORB, KITTI *dataset*, computer vision, autonomous vehicles.

Autor: David Ortiz Martínez

Tutor: Andreu Corominas Murtra (Beta Robots)

Date: Barcelona, November 2017

At the same speed as the technology keeps advancing, autonomous vehicles are becoming a reality increasingly palpable. Among all the challenges that this type of vehicle faces, we must be able to make a system capable of providing a precise position of the vehicle in the world, especially in closed environments where there is no GPS. A method to deal with this problem is to estimate the movement incrementally using images taken by a digital camera, and this approach is known as Visual Odometry.

This master's thesis explains the different devices from which odometry can be obtained, the different types of visual odometry, and the mathematical theory behind these methods, including the detection of point features, comparison of point features, estimation of movement, etc.

The thesis has been done processing offline data. That is why we introduce a dataset of real data that has been used to test the algorithms of Visual Odometry.

Finally, the various implementations that have been developed are introduced. All the code has been made in C++ and with the OpenCV library, implementing all the algorithms under ROS. The results have given information about the problems that can be encountered when implementing the algorithm in a real device.

Index de continguts

1 Introducció....6
1.1 Objectiu....7
1.2 Estructura del document....7
2 Odometria Visual....8
2.1 Odometria....8
2.1.1 Odometria en roda:....8
2.1.2 Sistema de navegació inercial (INS)....9
2.1.3 GPS/GNSS....9
2.1.4 Sonars i sensors ultrasònics....10
2.1.5 Sensors Laser....11
2.1.6 Càmera monocular....11
2.2 Característiques de l'odometria visual....12
2.3 Tipus d'odometria visual....13
2.4 Formulació del problema d'OV....13
2.5 Vista general de l'algoritme....15
2.6 Caracterització d'una càmera *Pinhole*....15
2.7 Detecció de punts característics....16
2.7.1 Detecció de cantonades Harris....17
2.7.2 FAST....18
2.7.3 BRIEF....18
2.7.4 ORB....19
2.8 Correspondència de punts característics....20
2.8.1 Algoritme de força bruta....20
2.8.2 FLANN....20
2.9 Estimació del moviment....20
2.9.1 2D-a-2D....21
2.9.2 3D-a-2D....22

2.10 Eliminació de falses correspondències....23
3 KITTI Vision Benchmark Suite....25
3.1 Introducció....25
3.2 Plataforma de captura....25
3.3 Dataset per odometria visual....27
4 Implementació....28
4.1 Equip de proves....28
4.2 Nodes ROS....28
4.3 Disseny del programa en C++....29
4.4 Preparació del dataset KITTI....29
4.5 Algoritme OV....30
4.5.1 Algoritme d'odometria visual....31
4.5.2 Segona iteració....34
4.5.3 Tercera iteració....36
4.5.4 Quarta iteració....39
4.5.5 Cinquena iteració....42
5 Conclusions i treball futur....46
5.1 Conclusions....46
5.2 Treball futur....46
6 Bibliografia....48

1 Introducció

En robòtica mòbil (i en qualsevol dispositiu mòbil) una vegada hi ha un moviment, és especialment crític poder quantificar com s'ha mogut el vehicle. Saber-ho ens proporciona una informació vital per saber on està el robot en aquest moment i, per tant, poder afrontar el nou moviment amb seguretat i control. Si no tenim informació de com s'ha desplaçat el vehicle, no ens podem ubicar en el món i, per tant, podem dir que el vehicle està «perdut».

Aquest moviment es mesura utilitzant un sistema d'odometria, que ens proporciona informació per establir la seva posició. La informació obtinguda servirà per poder seguir la trajectòria planificada, poder plantejar com esquivar obstacles, etc.

Cada tipus de robot mòbil utilitza diferents tipus de sensors per mesurar l'odometria: En robots terrestres s'utilitza típicament la mesura de voltes de les rodes i en robots aeris s'utilitzen les IMUs (de les sigles en anglès *Inertial measurement* unit o Unitat de medició inercial). En entorns oberts, a més, es pot utilitzar el GPS (*Global Positioning System o sistema de posicionament global*). Per intentar millorar la mesura, normalment aquests tipus de sensors es fusionen, i així s'obté una mesura molt més precisa.

Com que molts dels robots mòbils actuals disposen de càmeres, es poden utilitzar com a una altra manera de mesurar l'odometria. L'odometria visual és un procés d'estimació del moviment d'un vehicle utilitzant les imatges obtingudes a través de la seva càmera o conjunt de càmeres. Un dels primers articles que es van publicar del tema es del 2004, i està signat per D. Nister, O. Naroditsky i J. Bergen [22].

L'odometria visual proporciona una mesura que no es veu afectada per problemes de derrapatge de les rodes, no té els problemes de cobertura que es poden donar amb els GPS, i a més, en dispositius on el pes és una variable crítica, no s'afegeix pes extra d'un sensor extra.

Com a contraprestació, per poder realitzar tots els càlculs, s'ha de tenir un bon processador que garanteixi que el sistema serà capaç de proporcionar una lectura correcta a totes les velocitats a les que es desplaci el robot. I a més, la mesura dependrà de la llum que tingui l'escena, és a dir, si no disposem d'una bona font de llum, el sistema no podrà navegar.

1.1 Objectiu

L'objectiu d'aquesta memòria és realitzar un sistema de odometria visual monocular que permeti estimar en tot moment la posició del robot. Els punts que es volen assolir són:

- Obtenir els punts característics de dues imatges consecutives i les seves correspondències.
- Obtenir el desplaçament a partir de les correspondències entre punts obtingudes en el punt anterior.
- Ajustar els valors del sistema complet.

1.2 Estructura del document

El diferents capítols tractaran de:

- **Capítol 1:** Introducció.
- **Capítol 2:** Explica la base teòrica que es necessita per poder realitzar l'odometria visual: Tipus d'odometries, datació de punts característics a una imatge, matemàtica d'anàlisis de moviment entre imatges, etc.
- **Capítol 3:** Introdueix els *datasets* que s'han utilitzat per provar l'algoritme.
- **Capítol 4:** S'explicarà l'aplicació que s'ha desenvolupat per implementar l'odometria visual.
- **Capítol 5:** S'exposen les conclusions a les que s'han arribat en aquest projecte, i s'explica el treball futur que es podria implementar per continuar-lo.
- **Capítol 6:** Bibliografia.

2 Odometria Visual

En aquest capítol s'explica la base teòrica que s'ha utilitzat per poder implementar l'odometria visual.

2.1 Odometria

Tenir una localització acurada del robot és fonamental a l'hora de poder enfrontar-nos a problemes de navegació autònoma, sent capaç d'esquivar obstacles, etc.

El terme odometria prové del grec, de la unió dels termes *«hodos»* (Que significa viatge o travessia) i *«metron»* (que significa mesurar). Al robots mòbils significa estimar el canvi de la pose del robot (La pose representa la localització i rotació d'un vehicle o robot en un sistema de coordenades) durant el temps, i per tant es tracta d'un posicionament del robot relatiu a l'inici de la trajectòria (inici del càlcul d'odometria).

Per mesurar l'odometria, existeixen diferents sensors i tècniques:

2.1.1 Odometria en roda:

Aquest tipus d'odometria és la més senzilla i utilitzada per estimar la posició de robots mòbils. Per calcular el desplaçament es compten el número de voltes que ha donat gràcies a un *encoder* col·locat a l'eix, i com que es coneix el perímetre de la roda, es pot calcular fàcilment la distància recorreguda per volta. Amb la distància recorreguda per cada roda i el model cinemàtic directe del vehicle, podem trobar el desplaçament en translació i rotació del punt central del vehicle.

Figura 1: *Encoder de roda pel càlcul de l'odometria*

El lliscament de la roda afecta molt a la mesura, i introdueix molta deriva en posició, provocant un error no acotat i sempre creixent en el temps. Per una altra banda, és un tipus de sistema molt econòmic i que té un bon comportament en distàncies curtes.

2.1.2 Sistema de navegació inercial (INS)

És una tècnica de posicionament relatiu que proporciona la posició i orientació d'un objecte respecte a un punt, orientació i velocitat inicial. Aquesta tècnica utilitza sensors acceleròmetres i giroscopis per calcular contínuament la posició, orientació i velocitat del vehicle, ja sigui terrestre, aeri o naval.

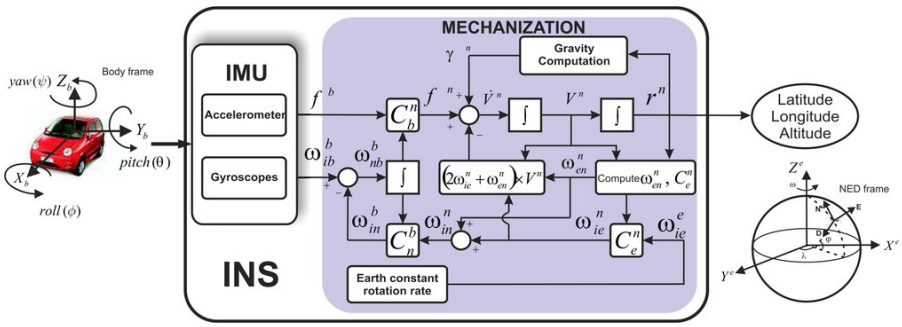

Figura 2: *Sistema de navegació inercial [1]*

Encara que és auto contingut, i no necessita de cap referència externa, és molt propens a acumular deriva en la mesura. Aquesta deriva apareix perquè per calcular la velocitat i la posició respecte el temps s'integra la acceleració respecte al temps, i el soroll i biaix en les mesures d'acceleració i velocitat de rotació comporten grans errors en l'estimació de velocitat i posició.

Per tot això, aquest tipus de sensors no s'utilitzen en períodes de temps molt llargs, i es recomana utilitzar-los acompanyats d'altres sensors que ajudin a corregir l'error de deriva.

2.1.3 GPS/GNSS

El terme GNSS (*Global Navigation Satellite System)* s'utilitza per especificar tots el tipus de ràdio navegació global, incloent l'americà GPS (*Global Positioning System*), el rus GLONASS (*Global Navigation Satellite System*) i l'europeu GALILEO (European Georgia Library Learning Online System).

El GPS és un sistema que engloba una xarxa de satèl·lits en òrbita geoestacionària que emeten uns senyals de ràdio codificats. Els receptors GPS detecten aquests senyals i obtenen la informació que emet. Amb 4 o més satèl·lits, es pot implementar un algoritme de trilateració que proporciona la posició 3D (latitud, longitud i altitud) i biaix de temps del receptor. Per tant,

aquesta tècnica proporciona un posicionament absolut. L'error depèn de la quantitat de satèl·lits que s'estiguin detectant i l'algoritme utilitzat.

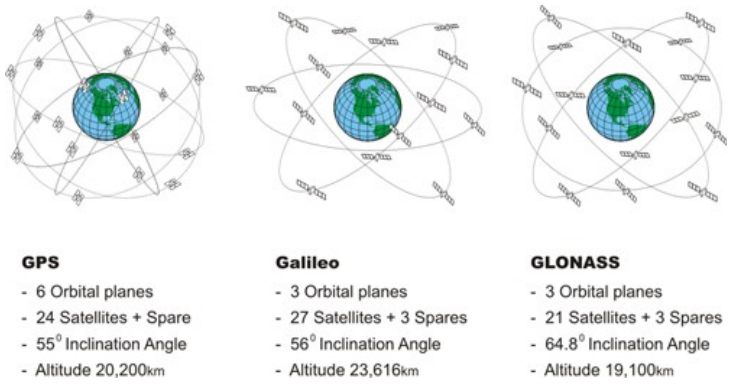

Figura 3: *Diferents sistemes de GPS [2]*

El problema que hi ha en aquest sistema és que el receptor només funciona en entorns oberts. En entorns tancats no funciona, ja que no es veuen els satèl·lits.

El GPS, doncs, no seria una tècnica purament d'odometria, però al tenir un posicionament absolut sempre podem calcular-ne un de relatiu.

2.1.4 Sonars i sensors ultrasònics

Aquest tipus de sensors utilitzen l'energia acústica per detectar objectes i mesurar distàncies al sensor. Els sensors, instal·lats a la base mòbil, emeten un senyal controlat que és reflectit pels obstacles, i retornada al sensor. Com que se sap quina és la velocitat a la que es mou el senyal ultrasònic, es pot calcular fàcilment la distància a la que es troba l'obstacle, i utilitzant dues mesures consecutives es pot calcular la velocitat a la que es desplaça l'objecte. També es pot utilitzar una base fixa que estigui transmetent una senyal controlada, i amb els receptors a la base mòbil calcular la distància i la velocitat a la que es desplaça el robot.

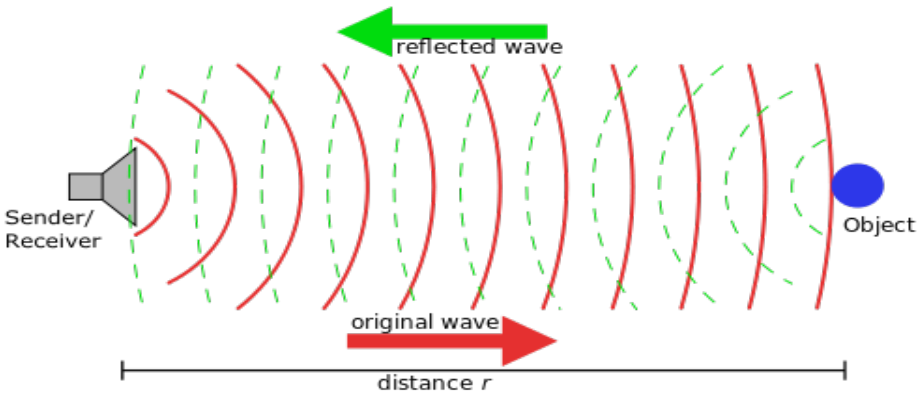

Figura 4: *Funcionament sonar (Wikipedia)*

Aquesta tècnica té molts problemes, ja que la reflexió dependrà del tipus de material, i si hi ha més robots movent-se per la mateixa àrea, pot haver-hi interferències.

2.1.5 Sensors Laser

Els sensors laser es poden utilitzar per diverses aplicacions relacionades amb el posicionament. Normalment s'anomena LIDAR (*Light Detection and Ranging sensor*). Funciona emetent un raig laser contra un obstacle, i es mesura el temps que triga en tornar la llum reflectida. Mesurant aquest temps, el sistema és capaç de mesurar la distància a la que es troba un obstacle, i per tant, es pot obtenir una mesura de la velocitat a la que es desplaça el robot en vers l'obstacle.

EL LIDAR, de totes maneres, s'utilitza normalment per detectar obstacles, fer mapes i capturar moviment.

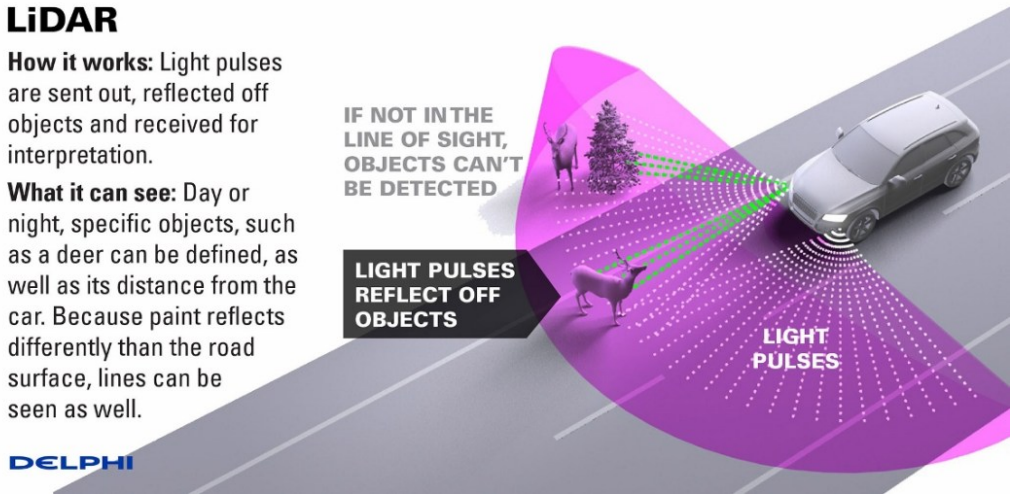

Figura 5: *Descripció del funcionament del LIDAR (Font: Delphi)*

Encara que la mesura és molt precisa, el cost dels dispositius LIDAR és en general força elevat.

2.1.6 Càmera monocular

A les aplicacions de robòtica mòbil, normalment s'utilitzen càmeres i sistemes de visió per realitzar diverses tasques, i entre elles està la localització i odometria. Una càmera és un sensor molt versàtil, perquè el podem utilitzar per detectar obstacles, i per calcular l'odometria visual.

En comparació amb els altres tipus de sensors, utilitzar una càmera és, en general, més econòmic i elimina alguns dels problemes que hem vist en els altres tipus de sensors. En contraprestació, es necessita una potència de càlcul molt elevada, i les imatges obtingudes

amb càmeres són molt sensibles a les condicions ambientals, és a dir, un canvi en les condicions lumíniques pot canviar el resultat de l'algoritme.

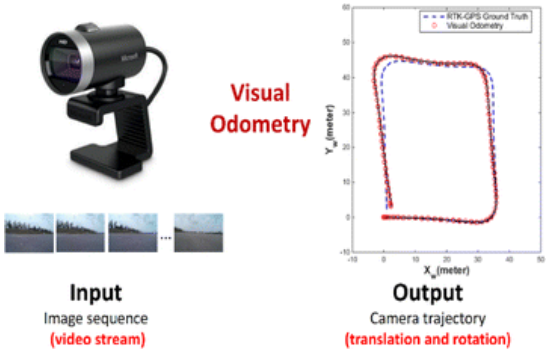

Figura 6: *Càmera per odometria visual [3]*

2.2 Característiques de l'odometria visual

L'odometria visual és el procés d'estimar el moviment d'un robot utilitzant únicament càmeres, ja sigui una de sola o vàries. El nucli de l'odometria visual (a partir d'aquí OV) és proporcionar una estimació incremental de la posició del robot analitzant les seqüències d'imatges capturades per una càmera.

Aquest mètode té un bon balanç entre costos, fiabilitat i complexitat d'implementació: És econòmic perquè es pot utilitzar una càmera d'ús general. I no és molt complexe d'implementar perquè hi ha llibreries especialment dissenyades per poder tractar imatges d'una manera còmoda.

Les avantatges en comparació als altres tipus d'odometria, es poden enumerar com:

- No té problemes per errors causats pel derrapatge de les rodes .
- Pot funcionar en entorns oberts i tancats, no com el GPS, que només pot treballar en entorns on es rebi senyal de satèl·lit.
- És un sensor passiu que no emet energia, a diferència dels sonars o lidars.
- Una càmera es pot utilitzar en qualsevol tipus de vehicle, ja sigui aeri, terrestre, marítim o submarí.

El desavantatges son:

- Es necessita una font de llum suficient per poder distingir correctament l'entorn.
- Les imatges han de tenir la suficient textura per extreure punts d'interès i així establir una correspondència entre imatges.
- Les imatges s'han d'obtenir a suficient taxa de captura per assegurar una mínima superposició en l'escena entre dues imatges consecutives.

2.3 Tipus d'odometria visual

L'OV es classifica segons el tipus de càmera que s'utilitza per estimar la trajectòria del robot. Hi ha molts tipus de càmeres que es poden utilitzar per aquesta aplicació : Monocular, estèreo, omnidireccional, RGB-D, etc, ... tot i que la major part de les aplicacions es realitzen amb les càmeres estèreo o monoculars:

- **Càmeres estèreo:** És aquella que disposa de dues lents, amb un sensor d'imatge per a cada un. Com que la distància entre els centres òptics és coneguda, es pot obtenir, a partir d'un procés de triangulació, la profunditat de la imatge i la seva escala, i per tant, obtenir la profunditat dels píxels en metres.

  Com a part negativa, aquest tipus de càmeres son més cares que les monoculars, i necessiten un calibratge més acurat, ja que un error en el calibratge afecta directament a la estimació del moviment.

  Una altre consideració és a l'hora de captar les imatges, ja que s'han d'obtenir al mateix temps, un petit desfasament entre captures afecta al valor obtingut, per tant, s'ha d'aplicar un esforç considerable per garantir aquesta sincronia.

  La distància entre òptiques també és important, perquè si és molt petita, una càmera estèreo tendirà a comportar-se com una de monocular.

- **Càmeres monoculars:** Una càmera monocular és molt més econòmica que una càmera estèreo, més lleugera i més fàcil d'instal·lar.

  El problema que tenen és la incertesa per establir l'escala mètrica. Les superfícies que es capturen són desiguals i l'escala va fluctuant, per tant, assumir una escala pot afegir molt error a la mesura.

Aquest projecte està realitzat utilitzant una càmera monocular per l'estimació de l'odometria visual.

2.4 Formulació del problema d'OV

Si tenim un robot movent-se amb una càmera fixada al seu cos prenent fotos en un temps discret *k*, les imatges es poden denominar com la seqüència $I_{0:n}=(I_0,\dots,I_n)$. Dues posicions de càmera en instants de temps adjacents *k-1* i *k* estan relacionades per una transformació rígida de cos $T_{k,k-1}\in\mathbb{R}^{4x4}$ de la següent forma:

$$T_{k,k-1}=\begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}$$

on $R_{k,k-1} \in \mathbb{R}^{3x3}$ és la matriu de rotació, i $t_{k,k-1} \in \mathbb{R}^{3x1}$ és el vector de translació. Per facilitar la notació, a partir d'ara $T_{k,k-1}$ s'escriurà com T_k .

Una seqüència de $T_{1:n}=(T_1,\dots,T_n)$ contindrà tots els moviments consecutius que s'han obtingut amb la càmera. Si disposem d'aquesta seqüència T_k, podem obtenir la posició a l'instant actual coneixent la posició anterior de la càmera amb la fórmula $C_n=C_{n-1}\cdot T_n^{-1}$. Aquesta equació ens dóna una seqüència de les poses de la càmera $C_{0:n}=(C_0,\dots,C_n)$, amb l'única incògnita de la posició de la càmera al instant *k* = 0 (C_0), que sol ser establerta per l'usuari, per exemple utilitzant la matriu identitat $(C_0=I_{4x4})$.

En resum, la principal funció de l'OV és computar la transformacions relatives T_k a partir de les imatges I_k i I_{k-1}, i amb això concatenar les transformacions per recuperar tota la trajectòria $C_{0:n}$ de les poses de la càmera. Per tant, es pot veure que la recuperació de la trajectòria es realitza incrementalment.

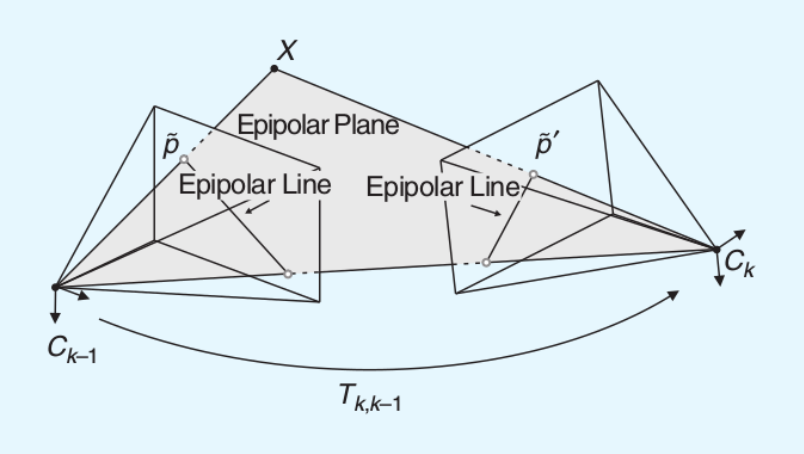

Figura 7: *Il·lustració de la constant epipolar, de Davide Scaramuzza [4]*

Hi ha dues maneres de computar aquest moviment relatiu T_k:

- Mètode d'aproximació per aparença, en el que s'utilitza la informació d'intensitat de tots els píxels de les dues imatges.
- Mètode de cerca de característiques en el que només s'utilitzen les característiques extretes i coincidents en les dues imatges.

El primer mètode, que també es pot anomenar dens, és menys precís i més complexe computacionalment. El segon mètode requereix l'habilitat de poder establir coincidències robustes entre les dues imatges, però és més ràpid i precís que els mètodes densos. És per això que normalment s'implementen solucions basades en el segon mètode.

2.5 Vista general de l'algoritme

L'algoritme per implementar l'OV es el següent:

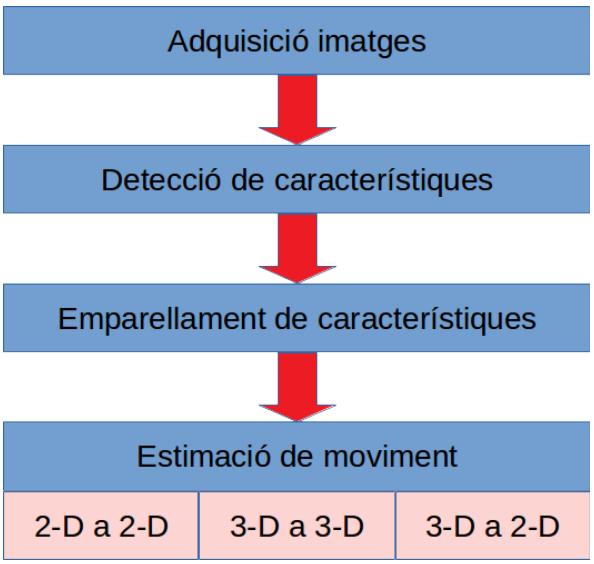

Figura 8: *Diagrama algoritme OV [4]*

La primera part és la dedicada a capturar les imatges des de la càmera. La segona part analitza aquesta imatge i extreu les característiques més destacades. La tercera part compara aquestes característiques extretes amb les obtingudes a l'anterior imatge per buscar els punts característics que són semblants en les dues imatges i així establir emparellaments entre punts de dues imatges consecutives. Per últim, la quarta etapa realitza el càlcul del moviment utilitzant les parelles de punts trobades en el pas anterior.

2.6 Caracterització d'una càmera *Pinhole*

Hi ha diversos tipus de models de càmera per escollir quan es treballa amb OV, per exemple: projecció catadiòptrica, model esfèric per perspectiva, omnidireccional i projecció de perspectiva. Malgrat això, el més utilitzat és la projecció de perspectiva. Aquest model assumeix una sistema de projecció *pinhole* on la intersecció de tots els raigs de llum que passen per la càmera formen la imatge.

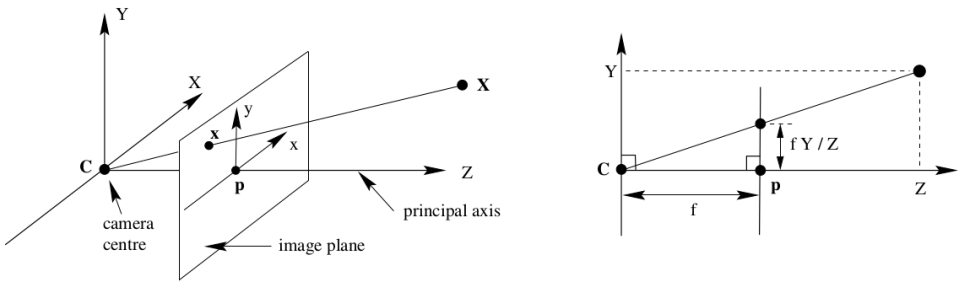

Figura 9: *Geometria càmera pinhole. [6]*

La projecció d'un punt 3D és $X=(x,y,z)^T$ en el pla de la imatge (xy). Establint que les coordenades en imatge del punt són $p=(u,v)^T$, les coordenades de la imatge respecte les coordenades del món segueixen la següent relació:

$$\lambda\begin{pmatrix}u\\v\\1\end{pmatrix}=\underbrace{\begin{pmatrix}f & 0 & u_0\\0 & f & v_0\\0 & 0 & 1\end{pmatrix}}_{K}\begin{pmatrix}x\\y\\z\end{pmatrix}$$

On λ es el factor d'escala, *f* es la longitud focal i u_0, v_0 son las coordenades del centre de projecció. Aquests paràmetres són coneguts com la *matriu de calibratge* o els *paràmetres intrínsecs*, i formen la matriu *K*. Aquesta matriu de calibratge es pot trobar seguint un procés de calibració com el descrit a [7].

Cal destacar que aquest tipus de matrius estan pensades per una càmera ideal en la que els seus píxels son quadrats perfectes. Això no és sempre veritat, i per compensar aquest error s'afegeixen uns termes a la matriu:

$$K=\begin{pmatrix}f\cdot s_x & f\cdot s_\varphi & u_0\\0 & f\cdot s_y & v_0\\0 & 0 & 1\end{pmatrix}$$

on s_x i s_y són els factors d'escala que defineixen la mida del píxel, i s_φ és el factor de *skew*, que és proprorcional a φ, que és l'angle que formen els eixos x i y.

2.7 Detecció de punts característics

La detecció de punts característics consisteix en reconèixer certs punts d'interès en una imatge, anomenats punts clau (*keypoints* en anglès). Aquest punts d'interès poden ser de diferents tipus: vores, cantonades, taques, etc..

Les qualitats que té un bon detector de punts característics són:

- És eficient computacionalment
- És geomètricament invariant (tant en rotació com en escala)
- És precís en la localització
- És invariant a la il·luminació.
- És repetible (és a dir, detecta les mateixes característiques en imatges consecutives)
- És robust (tant a desenfocament com a soroll en la imatge)

2.7.1 Detecció de cantonades Harris

Un dels primers intents de detecció de característiques utilitzats va ser el detector de cantonades Harris (*Harris Corner Detector*), al 1988. Es basa en la idea de detectar les cantonades presents a la imatge, on existeix una gran variació de la intensitat en totes les direccions [8].

Per una coordenada de desplaçament (*u*, v), la fórmula per detectar les cantonades és:

$$E(u,v)=\sum_{x,y} w(x,y)\cdot\left[I(x+u,y+v)-I(x,y)\right]^2$$

on *w(x,y)* és la funció de finestra que s'aplica, ja sigui rectangular o quadrada, per donar pes als píxels que siguin dintre de la finestra. La funció *I(x,y)* és la intensitat del píxel a les coordenades *x*, *y*.

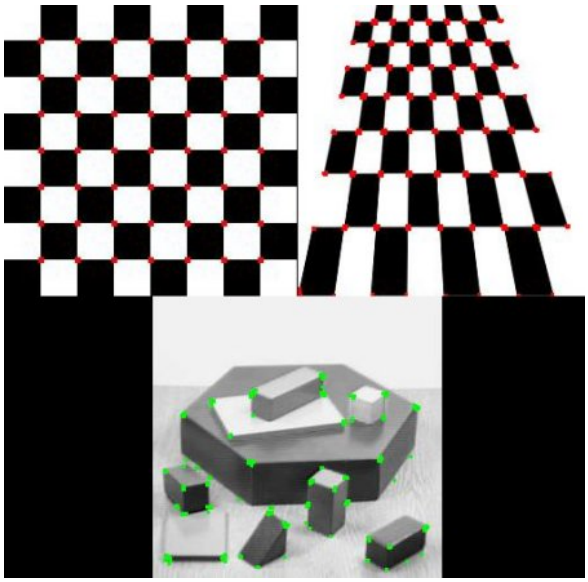

Figura 10: *Exemple de deteccions de cantonades [8]*

Els punts detectats provenen de màxims locals detectats amb la fórmula. Aquest tipus de detecció és invariant a rotacions de la imatge, però no és immune a canvis d'escala, és a dir, un objecte que es detecti a una certa distància, pot no ser detectat a una distància superior.

2.7.2 FAST

L'algoritme *Feature from Accelerated Segment Test (FAST) é*s un detector de cantonades que, tal i com l'acrònim indica, té un temps de resposta ràpid. En comparació amb altres algoritmes, com per exemple el SIFT o SURF, ha demostrat que és molt més ràpid computacionalment, però té una menor eficàcia [9]. És menys eficaç perquè no és massa robust a elevats nivells de soroll la imatge, el resultat depèn molt del nivell de llindar que s'esculli.

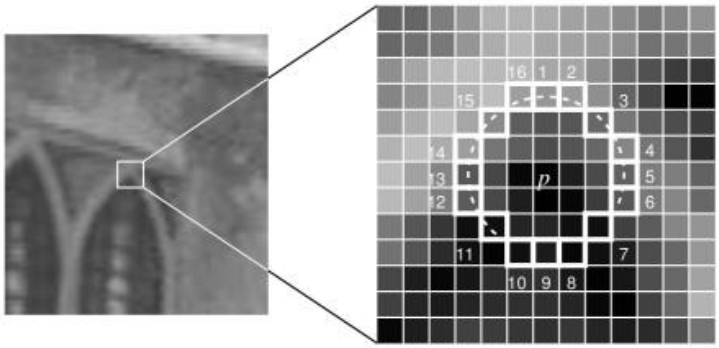

Figura 11: *Detecció de cantonades FAST[9]*

El funcionament és el següent:

- Donat un punt, es seleccionen uns punts veïns, en el cas de la figura 11, són 16 punts.
- Es fixa un llindar de detecció *k*.
- Si tenim al menys *N* veïns que tenen una intensitat major a $I_p + t$ o uns altres *N* menors a $I_p - t$, llavors *p* es un punt d'interès. Normalment *N* s'ajusta a 12 per tenir una detecció de característiques de gran qualitat.
- Per accelerar la detecció, es comproven només els punts 1 i 9, i es verifica si compleixen la condició ($I_p \pm t$). Si la compleixen, es comprova 5 i 13. Si 3 dels 4 punts compleixen la condició, és un punt característic.

Si s'accelera la detecció, poden aparèixer molts falsos positius.

2.7.3 BRIEF

L'algoritme *Binary Robust Independent Elementary Features* (BRIEF) és un descriptor de característiques que es pot combinar amb qualsevol detector [10].

Un descriptor és un vector amb tota la informació dels punts detectats. Els altres tipus de descriptors, com SIFT i SURF, tenien una mida de 128 o 64 dimensions, ocupant uns 512 o 256 bytes, requerint molta quantitat de memòria. A més es necessitava algun tipus de comparador de descriptors molt ràpid. En canvi, el descriptor del tipus BRIEF és una cadena binària de mida 128, 256 (per defecte) o 512 bits (16, 32 o 64 bytes), el que estalvia memòria. I la comparació entre descriptors es pot fer amb una operació lògica XOR.

El problema d'aquest tipus de descriptor és que no és invariant a la rotació, i per tant no detectarà els mateixos punts en imatges que estiguin rotades entre sí.

2.7.4 ORB

ORB és un acrònim de *Oriented FAST and Rotation-Aware BRIEF*, que significa que és un algoritme que combina la detecció de característiques *FAST* , i els descriptors *BRIEF*, amb algunes millores en el seu funcionament [11].

Per calcular el *FAST*, es segueixen els mateixos passos que s'han explicat al punt 2.7.2, però s'afegeix l'orientació als punts de característiques. Això es fa calculant l'intensitat del centroide.

- Primer es calcula el moment d'una regió de la imatge: $m_{pq}=\sum_{x,y} x^p y^q I(x,y)$
- Després es calcula el centroide de la regió: $C=\left(\frac{m_{10}}{m_{00}},\frac{m_{01}}{m_{00}}\right)$.
- Es traça el vector $(\vec{O}C)$ que va des del punt de detecció fins el punt central del centroide calculat.
- Es calcula l'angle d'aquest vector amb $\theta=atan^2(m_{01},m_{10})$.

Una vegada que s'ha assignat orientació a les característiques detectades, el detector *BRIEF* fa el test binari, τ, de l'intensitat entre el punts detectats a la regió de la imatge. El test es realitza amb la següent fórmula

$$\tau(p;x,y)=\begin{cases}1 \rightarrow if p(x)<p(y)\\0 \rightarrow if p(x)\geq p(y)\end{cases}$$

El resultat és un vector de 256 tests binaris:

$$f_{256}(p)=\sum_{i\leq i\leq 256} 2^{i-1}\tau(p;x_i,y_i)$$

Per fer que el detector *BRIEF* sigui robust a les rotacions, es dirigeixen els descriptors a l'orientació del punt clau *θ*. Això s'aconsegueix construint una versió dirigida dels tests binaris a la localització de la característica:

$$S=R_\theta S$$

On R_θ es la matriu de rotació i S és igual a:

$$S=\begin{bmatrix} x_1 \dots x_{256} \\ y_1 \dots y_{256} \end{bmatrix}$$

Llavors, el resultat del *BRIEF* dirigit és:

$$g_n(p,\theta)=f_n(p|(x_i,y_i)\in S_\theta)$$

Amb totes aquestes modificacions als dos algoritmes, l'ORB és molt més ràpid que els mètodes SIFT i SURF, que són els altres mètodes que es solen utilitzar. A més, el mètode ORB és d'ús lliure, i en canvi el SIFT i el SURF estan patentats.

2.8 Correspondència de punts característics

L'aparellament de punts característics consisteix en determinar quins punts característics d'entre dues imatges són corresponents, és a dir, corresponen a un mateix punt físic de l'escena.

Es destaquen dos algoritmes a la llibreria OpenCV per aquest propòsit: L'algoritme de força bruta i l'algoritme FLANN [12].

2.8.1 Algoritme de força bruta

Aquest algoritme (en anglès *Brute force matcher*) compara el descriptor des d'un set de *keypoints* (punts clau) de la primera imatge amb tots els de la segona imatge. Generalment, el descriptor amb la menor distància euclidiana s'emparella amb el descriptor de la primera imatge. També s'ha de definir un llindar que faci que aquesta coincidència sigui vàlida o no. És important que el moviment de la càmera entre les dues imatges estigui acotat per afinar aquestes coincidències entre descriptors.

2.8.2 FLANN

És l'acrònim de *Fast Library for Approximate Nearest Neighbours*, i consisteix en un conjunt d'algoritmes que busquen en un grup de dades el veí més proper (*closest neighbour*). L'algoritme crea una estructura en arbre pels descriptors i va buscant les coincidències en les diferents branques de l'arbre utilitzant l'algoritme de veïns propers.

2.9 Estimació del moviment

Una vegada s'han detectat tots els punts característics comuns en la imatge anterior i en l'actual, és moment per estimar el moviment. L'algoritme, a línies generals, el que ha de fer és calcular el moviment relatiu entre la imatge actual i l'anterior. Si es concatenen tots el moviments relatius, s'obté la trajectòria de la càmera.

Els mètodes que existeixen per estimar el moviment relatiu són els següents:

- **2D-a-2D:** Els punts clau de la imatge actual i l'anterior estan en 2D.
- **3D-a-2D:** En aquest cas els punts clau de la imatge anterior estan triangulats i es donen en 3D. En canvi, en la imatge actual les coordenades es donen en 2D.
- **3D-a-3D:** En aquest cas els punts clau en les dues imatges tenen les coordenades en 3D.

Dels tres tipus només s'explicaran els dos primers, perquè el tercer (3D-a-3D) necessita una càmera estèreo, i en aquest treball s'utilitza una càmera monocular.

2.9.1 2D-a-2D

L'implementació d'aquest mètode es basa en unes restriccions geomètriques que apareixen quan una càmera es desplaça. Aquestes restriccions es coneixen per geometria epipolar (*epipolar geometry*) [6].

En el cas de la OV monocular tenim dues imatges I_0 i I_1, que s'han agafat en el temps $k = \{0,1\}$. Si anomenem c_0 i c_1 als centres de las càmeres a l'instant $k = 0$ i $k = 1$ respectivament, en el que c_1 està sotmès a la rotació $\boldsymbol{R}$ i la translació $\boldsymbol{t}$ relativa a c_0. També fem que la projecció del punt 3D $\boldsymbol{X}$ en els plans de les càmeres, és a dir imatges, siguin $\boldsymbol{p}$ i $\boldsymbol{p'}$. Si s'observa la Figura 7 es pot veure que el vectors entre X i els centres de les càmeres (c_0 – X) i (c_1 – X) abasten un pla conegut com el pla epipolar (*epipolar plane*). Com que $\boldsymbol{t}$ també està en aquest pla, es coneix que el *cross product* entre $\boldsymbol{t}$ i $(c_1 - X)$ és el vector normal del pla epipolar, i és ortogonal a $(c_0 - X)$. Això significa que:

$$(c_0 - X)^T (t \times (c_1 - T)) = 0$$

No obstant això, com que se sap que $\boldsymbol{p}$ es paral·lel a $(c_0 - X)$ i $\boldsymbol{Rp'}$ es paral·lel a $(c_1 - X)$, es pot reescriure aquesta formula com:

$$p^T (t \times Rp') = 0$$

Utilitzant la propietat $t \times Rp' = [t]_x Rp'$, on

$$[t]_x = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix}$$

és una matriu simètrica skew, es converteix en:

$$p^T [t]_x Rp' = p^T Ep' = 0; \quad E = t_x R;$$

La matriu E és anomenada la matriu essencial (*essential matrix*). Per tant, el problema de les correspondències 2D-a-2D es redueix a l'estimació de la matriu essencial que satisfà la fórmula anterior per a totes les correspondències de característiques. A la pràctica, no hi ha cap matriu que sigui capaç de resoldre aquesta equació ja que a causa del soroll sempre hi restarà un petit

residu. Es pot abordar aquest problema utilitzant l'algoritme *RANSAC* per reduïr aquest residu gràcies a l'eliminació de falses correspondències (veure secció 2.10).

Hi ha diferents maneres de resoldre la matriu essencial, però si tenim en compte que ***R*** té tres graus de llibertat (*DoF*) i ***t*** té dos graus de llibertat, s'utilitza el mètode solució per cinc punts (*five points solution*), proposat per Nister [13]. Això significa que únicament es necessiten 5 correspondències per estimar la matriu essencial. Generalment, s'intenta escollir el mètode basat en menys correspondències per incrementar l'eficiència.

Una vegada tenim computada la matriu essencial, es necessita extraure ***R*** i ***t***, on:

$$R = U\left(\pm W^T\right)V^T$$

$$t = U\left(\pm W\right)S\,U^T$$

on tenim que:

$$W^T = \begin{pmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

i les matrius U, S, i V es computen utilitzant el mètode de *Singular Value Descomposition (SVD) [14, 15 i 6]*

$$E = USV^T$$

2.9.2 3D-a-2D

A l'anterior punt s'ha explicat com extraure la pose a partir dues imatges i els seus punts característics, que són punts en 2D. Hi ha mètodes que utilitzen els punts característics i els seus punts corresponents en 3D. El problema d'estimar la pose utilitzant aquest mètode és conegut com *Perspective-n-Point* (*PnP*), on *n* es el número de punts característics i les seves correspondències en punts 3D [4].

El mètode 3D-a-2D requereix correspondències en 3 imatges, no en 2 com en el mètode anterior. L'algoritme que s'utilitza en aquest cas és el següent:

1. Computar totes les correspondències a p_{k-2}, p_{k-1} i p_k a les imatges I_{k-2}, I_{k-1} i I_k.
2. Utilitzar el mètode 2D-a-2D per extreure la transformació relativa $T_{k-2,k-1}$ entre les imatges I_{k-2} i I_{k-1}.
3. Utilitzar $T_{k-2,k-1}$ per triangular els corresponents punts 3D a ***X***.
4. Utilitzar ***X*** i p_k en l'algoritme P3P per computar la transformació $T_{k-1,k}$ entre les imatges I_{k-1} i I_k.
5. Computar les característiques corresponents p_{k-1}, p_k i p_{k+1} a través de les imatges I_{k-1}, I_k i la nova imatge I_{k+1}.

6. $k = k + 1$.
7. Saltar al punt 3.

X i ***p*** són vectors de la mateixa mida que contenen les coordenades 3D i 2D respectivament. L'algoritme P3P és un algoritme per trobar correspondències entre poses en 3D. El mètode 2D-a-2D només s'utilitza per inicialitzar les variables.

En aquesta tesi no s'utilitzarà aquest mètode, per tant, no s'entrarà en més detalls.

2.10 Eliminació de falses correspondències

Totes les aproximacions que s'han explicat fins ara de la OV suposen que les correspondències trobades són perfectes. El què passa a la realitat és que s'ha de tenir en compte l'existència de falses correspondències (en anglès *outliers*). Si no es tenen en compte s'obtindrà una matriu essencial incorrecta que donarà una matriu de transformació falsa.

Una manera de resoldre aquest problema és utilitzant un mètode conegut com *RANSAC* (*Random Sample Consensus*) [16 i 17]. Aquest mètode és un algoritme iteratiu on la idea és agafar una sèrie de *n* punts a l'atzar i utilitzar-los per calcular un model que serà l'hipòtesi de l'iteració. Una vegada fet, s'aplica aquest model a tots els punts i es sumen els errors obtinguts. L'algoritme itera sobre els diferents grups de dades i s'escull la solució en què la major quantitat de dades s'ajusten al model calculat per un número *N* d'iteracions.

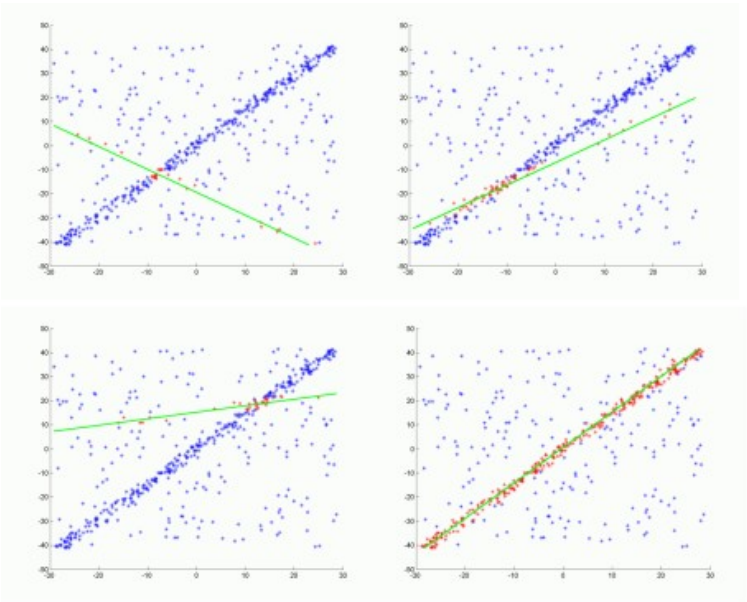

Figura 12: *Exemple aplicació algoritme RANSAC per trobar una línia a partir d'un conjunt de punts amb outliers, obtingut de Wikipedia*

El número d'iteracions necessàries (***N***) per garantir que s'ha trobat una solució correcta amb certa probabilitat es calcula amb:

$$N = \frac{\log(1-p)}{\log\left(1-(1-\epsilon)^s\right)}$$

On *S* es el número de dades mínim requerits per calcular el model, ϵ es el percentatge de *outliers* en les dades, i *p* es la probabilitat d'èxit desitjada. A la pràctica el número *N* es multiplica per un factor de 10 per augmentar la robustesa de l'algoritme. Com es pot intuïr, a més iteracions, més temps de computació es necessita, per tant, és un valor que s'ha d'analitzar per veure com afecta al rendiment.

Com que l'elecció de punts és a l'atzar, el comportament de l'algoritme no és determinístic, és a dir, en dues execucions diferents de l'algoritme es poden obtenir 2 resultats diferents.

Pel cas de l'OV, el model estimat seria el moviment relatiu (*R*,*t*) entre dues posicions successives de la càmera. Per estimar la distància dels punts (*X*, *Y*) al model se li calcula l'error *RMS* (error quadràtic mitja) entre *T(X)* i *Y*. Es considera *outlier* si l'error *RMS* d'aquest es troba fora d'un marge [*-d*, *+d*], amb *d* com un número arbitrari que serà trobat mitjançant ajustament.

Amb aquesta equació es pot veure que a més iteracions, més possibilitats d'èxit.

3 KITTI Vision Benchmark Suite

En aquest punt s'explicarà el *dataset* que s'ha utilitzat per provar l'algoritme.

3.1 Introducció

Al 2012, Andreas Geiger, Philip Lenz i Raquel Urtasun varen presentar un paper amb títul «*Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite*» [18], on parlaven de que en aquell moment hi havia una falta de material per poder provar sistemes de reconeixement òptic, com *optical flow*, odometria visual, SLAM o detecció d'objectes en sistemes de conducció autònoma. Van presentar les captures que havien realitzat a Karlsruhe, Alemanya, amb un cotxe equipat amb una sèrie de sensors, i amb el que havien recorregut 39.2 km agafant dades per odometria visual, dades per optical flow, i 200k objectes 3D anotats (amb un mínim de vianants i cotxes per imatge).

Al 2013, Andreas Geiger, Philip Lenz, Christoph Stiller i Raquel Urtasun van presentar un paper al *International Journal of Robotics Research* (*IJRR*) [19] parlant de les característiques que tenien les dades compartides.

El més interessant de totes aquestes dades es que estan calibrades, sincronitzades i amb una marca de temps, cosa que les fa molt útils per poder provar diferents tipus d'algoritmes.

Totes les dades són disponibles a la web del projecte http://www.cvlibs.net/datasets/kitti/index.php per poder treballar amb elles.

3.2 Plataforma de captura

La plataforma que van utilitzar per realitzar les captures és un «*VW Passat station wagon*», com podem veure a la figura 13.

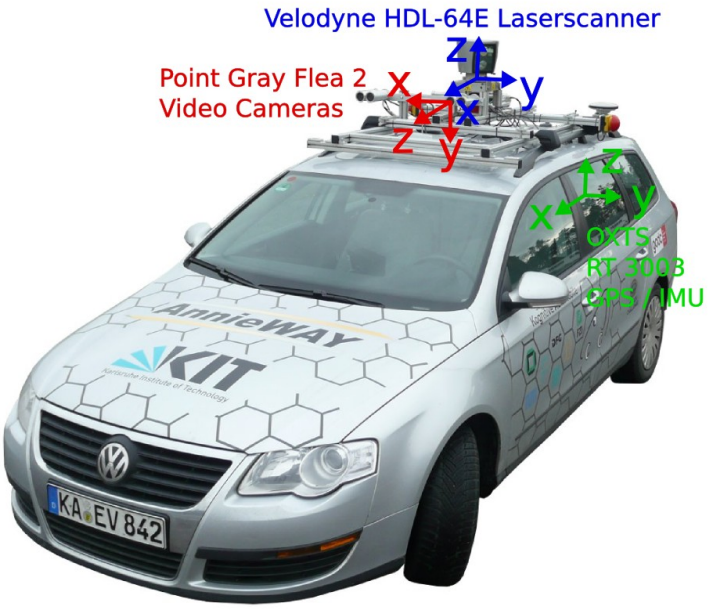

Figura 13: *Plataforma de captura de dades [19]*

La plataforma, com es pot veure a la figura, disposa de diferents tipus de sensors:

- 2 càmeres monocromes PointGray Flea2 (FL2-14S3M-C), de 1,4 Megapixels, global shutter i amb un sensor Sony ICX267 de 1/2".
- 2 càmeres a color PointGray Flea2 (FL2-14S3C-C), de 1,4 Megapixels, global shutter i amb un sensor Sony ICX267 de 1/2".
- 4 Lents Edmund Optics de 4 mm, angle d'obertura de ~90°, angle d'obertura vertical de regió d'interès (ROI) ~35°.
- 1 escàner 3D de laser rotatiu Velodyne HDL-64E, amb un freqüència de 10 Hz, 64 feixos, 0.09 graus de resolució angular, precisió de 2 cm en distància, capaç de recollir ~1.3 milions de punts/segon, amb un camp de visió de : 360° horitzontal, 26.8° vertical, i un rang de 120 m.
- 1 sistema de navegació inercial amb GPS OXTS RT3003, amb 6 eixos, 100 Hz, L1/L2 RTK, i resolució 0.02m /0.1°.

Totes les dades provinents dels sensors es gestionen amb un ordinador amb dos processadors Intel XEON i un disc dur de 4 Terabytes, que utilitzen com a sistema operatiu Linux Ubuntu (versió 64 bits), i una base de dades a temps real per emmagatzemar els resultats.

La col·locació dels diferents sensors al cotxe es pot veure a la Figura 14:

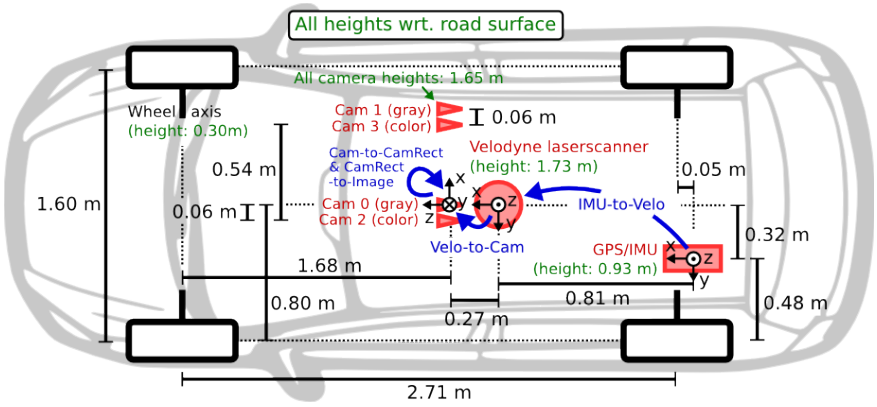

Figura 14: *Posicionament del sensors [19]*

3.3 Dataset per odometria visual

Com ja s'ha comentat, existeixen diversos *datasets* per afrontar diversos problemes de la conducció autònoma, en aquest projecte treballarem amb els que estan preparats per odometria visual.

Hi ha 22 seqüencies amb imatges de càmeres estèreo en format PNG. Les 11 primeres disposen del *ground truth* (és a dir, disposa de les dades de posició i orientació del sistema de navegació inercial de precisió) pensades per entrenar els algoritmes, i per altra banda, n'hi ha 11 més en les que no hi ha *ground truth*, i que estan més pensades per avaluar els algoritmes.

Les dades estan dividides segons els seu tipus:

- Imatges en blanc i negre.
- Imatges en color
- Dades del sensor *velodyne*
- Dades de calibratge dels sensors
- Dades de les poses de *ground truth*
- Dades per avaluar el sistema d'odometria.

De totes les dades, el que realment ens interessa són les dades d'imatges en color, les dades de calibratge dels sensors i les poses *ground truth*.

Les imatges estan dividides en seqüències, on en cada seqüència hi un directori per les imatges de cadascuna de les càmeres, un fitxer «*times.txt*» on estan emmagatzemades les marques de temps de totes les imatges i el fitxer «*calib.txt*» on hi ha la calibració de les càmeres (matriu de projecció que inclou calibració intrínsica i extrínsica).

El fitxer de *ground truth* inclou totes les poses en tota la trajectòria.

Amb aquest *dataset* ja podem provar l'algoritme proposat.

4 Implementació

En aquest capítol es descriu com s'ha utilitzat tota la teoria explicada prèviament per implementar l'algoritme d'odometria visual.

Per implementar l'algoritme s'ha creat un node ROS (*Robotic Operating System*) utilitzant C++, amb la llibreria OpenCV per tractar les imatges.

La configuració que s'implementa és amb una càmera monocular i utilitzant la transformació 2D-a-2D.

4.1 Equip de proves

L'equip que s'ha utilitzat per realitzar les proves té les següents característiques:

- Ordinador portàtil ASUS R510V
- Processador: Intel i7-6700HQ
- Memòria RAM: 8 GB DDR4 2133MHz
- Disc dur: 1Tb 5200 rpm mecànic
- Targeta gràfica: Nvidia GeForce 950M 2GB GDDR5
- Sistema operatiu: Linux Ubuntu 16.04 64 bits
- ROS Kinetic
- OpenCV 3.3
- Cuda 9.0

4.2 Nodes ROS

El projecte s'ha implementat en dos nodes ROS que tenen la següent topologia:

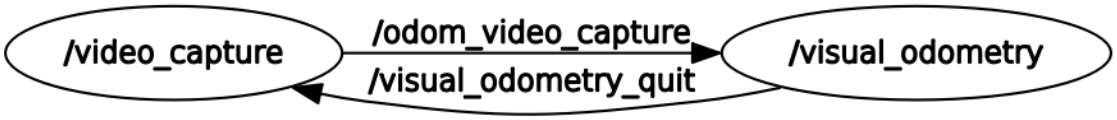

Figura 15: *Nodes ROS*

- **Video Capture:** El node s'encarrega de gestionar les imatges per analitzar. Es pot configurar perquè envii imatges de la webcam o agafar-las d'algun fitxer. En aquest cas, com que s'utilitzen uns *datasets*, les dades provenen de fitxers.
- **Visual Odometry:** Aquí està implementat l'algoritme principal del sistema, on es calcula l'odometria visual.

La connexió entre els nodes es realitza amb els següents missatges:

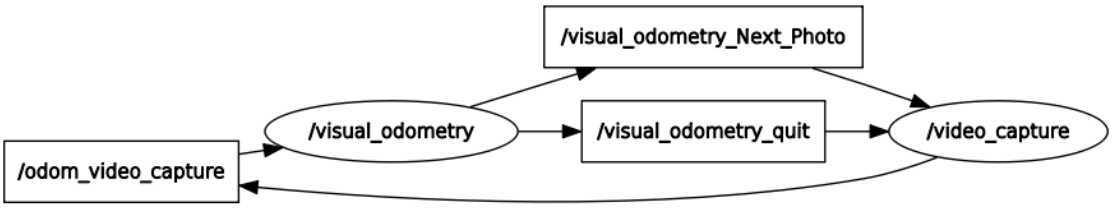

Figura 16: *Topics ROS*

Existeixen tres *topics*:

- ***odom_video_capture:*** És el topic per on s'envien les imatges.
- ***visual_odometry_Next_Photo*:** Per aquest topic el node ***visual_odometry*** demana una nova imatge al node ***video_capture***.
- ***visual_odometry_quit***: *Topic* per enviar un missatge de finalització del node ***video_capture***.

El funcionament del sistema és el següent: El node ***visual_odometry*** demana una foto al node ***video_capture***. Aquest li envia perquè pugui calcular l'odometria visual. En el moment que ha acabat, demana una nova foto utilitzant el *topic* ***visual_odometry_Next_Photo***. Per finalitzar el node, s'utilitza el topic ***visual_odometry_quit***, que envia l'orde d'apagar des del node ***visual_odometry*** al node ***video_capture***.

4.3 Disseny del programa en C++

Per realitzar l'algoritme principal del sistema s'ha implementat tres classes en C++:

- ***orb_track***: Classe on es guarden les dades dels tracks detectats.
- ***orb_tracker***: Classe on s'implementa la detecció i emparellament dels punts característics.
- ***orb_vo***: Classe on s'implementa el càlcul de l'odometria visual.

4.4 Preparació del dataset KITTI

Per provar l'algoritme s'han utilitzat els *datasets* que contenen el *ground truth*, és a dir, que tenen les poses que ha fet el cotxe durant tot el recorregut mesurades pel sistema de navegació GPS + inercial de precisió. El problema que hi ha és que l'algoritme implementat no és capaç de saber la distància entre les mostres, per tant s'ha realitzat un petita funció que calcula l'escala entre mostres a partir del fitxer de *ground truth*.

Aquest problema és conegut com ambigüitat d'escala i és inherent a l'odometria visual monocular.

Per fer les proves s'utilitzen 3 *datasets* diferents

- **Dataset 00:** Les imatges són del cotxe per la ciutat, amb altres cotxes, parades per cedir el pas, etc.
- **Dataset 01:** Aquí el cotxe va per l'autopista.
- **Dataset 09:** Aquí el cotxe es mou per una urbanització amb petites cases i una circulació molt fluida.

S'han escollit aquests tres *datasets* perquè són bastant diferents entre sí i posen l'algoritme a prova en llocs bastant diferents.

4.5 Algoritme OV

L'algoritme que s'ha dissenyat segueix el que hi ha a la Figura 8. Aquest algoritme té quatre passos:

- **Pas 1, Adquisició d'imatges**: En aquest primer pas s'agafa una imatge, es rectifica si és necessari, i s'envia al detector de punts característics. En aquest cas, les imatges obtingudes del *dataset* de KITTI ja estan rectificades.
- **Pas 2, Detecció de punts característics:** Amb la imatge capturada, el següent pas serà buscar tots els punts característics que té la imatge. Per realitzar aquesta tasca s'ha decidit utilitzar un detector ORB, que està implementat a la llibreria OpenCV (https://opencv.org/). La sortida d'aquest detector dóna informació sobre el punts clau on s'han d'extreure les característiques, i que servirà pel següent punt.
- **Pas 3, Emparellament de punts característics:** La sortida de l'anterior pas proporciona uns punts característics de la imatge juntament amb els seus descriptors. En aquest pas es comparen aquests punts i descriptors amb els obtinguts a la imatge anterior. El resultat d'aquesta comparació és un llistat de parelles de punts que s'han detectat en les dues imatges. S'ha tornat a utilitzar una funció de la llibreria OpenCV, en aquest cas el *Brute Force Matching*, que ens dóna tots els valor que es necessiten per poder executar el següent pas.
- **Pas 4, Estimació de moviment:** L'última part per obtenir l'OV es obtenir la matriu de rotació i el vector de translació de les imatges analitzades. Per obtenir la rotació i translació, s'ha de computar la matriu essencial amb les dades obtingudes al pas anterior. Una vegada es finalitzi aquest últim pas, es tornarà al pas 1 per processar la següent imatge.

Durant el desenvolupament del projecte, s'han realitzat diverses iteracions en l'algoritme per millorar els resultats.

Per implementar aquest algoritme, s'han realitzar diverses modificacions en la manera d'extreure i analitzar les característiques, a continuació s'explicaran les diverses iteracions.

4.5.1 Algoritme d'odometria visual

L'algoritme implementat en pseudo-codi és el següent:

```
Codi d'implementació d'Odometría visual

Entrada: Imatge (I), mínimes característiques detectades (minCoincidències)
Sortida: Rotació (R), Translació (t)

/* Primera imatge */
capturaImatge(I);

// Retorna punts i descriptors
[p',d'] ← detectarIComputarCaracterístiques(I);

/*Bucle principal */
while (capturaImatge(I) == TRUE) then
    // Retorna punts i descriptors
    [p,d] ← detectarIComputarCaracterístiques(I);

    // Les característiques que no estiguin a una distància mínima no es computen
    mask ← filtrarCaracterístiques(p,d, p', d');

    // Es computen les millors característiques
    m ← emparellarCaracterístiques(d,d',mask);

    if (tamany(m) < minCoincidències) then
        // No hi ha suficients emparellaments
        [p',d']  ← [p,d];
    end if

    // Computar la matriu essencial
    E ← computarMatriuEssencial(p',p,m);

    // Obtenir la Pose, en forma de matriu de rotació i vector de translació
    [R,t] ← computarRotacióTranslació(E);

    // Guardar els descriptors i punts d'interès
    [p',d'] ← [p,d];
end while
```

Algoritme 1: *Algoritme d'odometria visual*

L'explicació de l'algoritme és la següent:

- S'inicialitza l'algoritme obtenint una primera imatge.
- Es detecten i computen els punts característics de la imatge i es guarden en unes variables per punts i descriptors. Per realitzar aquesta operació s'utilitza la funció de la llibreria *OpenCV* cv::*detectAndCompute()* de la classe *ORBdetector*.

- Per accelerar i millorar el resultat de l'emparellament de punts característics, es realitza una operació prèvia en la que es calcula la distància entre tots els punts característics detectats en aquesta i l'anterior imatge, i només es seleccionen els que estiguin a menys d'una distància determinada. Aquesta operació redueix el número de falsos emparellaments, perquè intenta netejar el màxim de punts erronis, però en contraprestació, si la velocitat de la càmera provoca un desplaçament dels píxels major que la distància màxima, es poden eliminar moltes coincidències.
- L'emparellament de punts característics busca els punts més semblants entre els resultats de les dues imatges, tenint en compte els valors de la màscara. Per realitzar aquesta operació s'utilitza la funció *cv::match*().
- Si el número de emparellaments és menor a un mínim, en aquest cas 5, el càlcul de la matriu essencial no es podrà realitzar correctament, per tant, els valors s'ignoren i es guarden els anteriors.
- Es computa la matriu essencial amb els valor obtinguts. Per realitzar aquesta operació s'utilitza la funció *cv::findEssentialMat()*.
- S'obté la Pose a partir de la matriu essencial. La funció utilitzada es *cv::recoverPose()*.
- S'emmagatzemen els valor per a la següent iteració.

Aplicant aquest algoritme als tres *datasets* de proves, el resultat és:

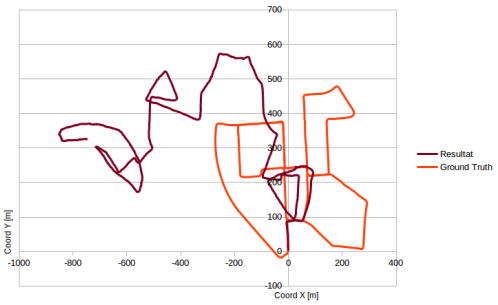

Figura 17: *Dataset 00*

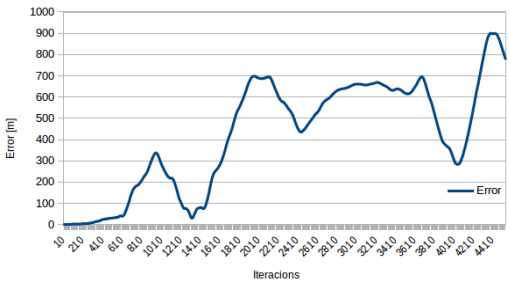

Figura 19: *Error de Posició Dataset 00*

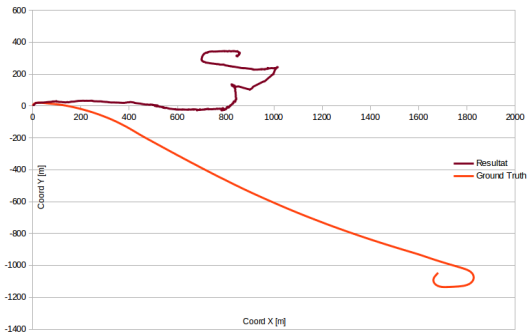

Figura 20: *Dataset 01*

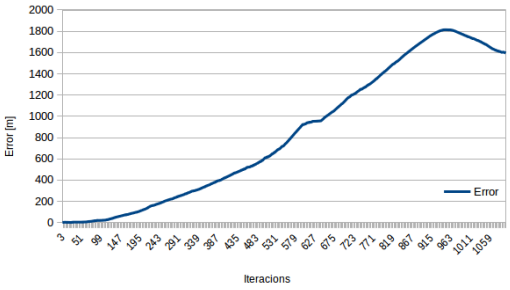

Figura 18: *Error de posició Dataset 01*

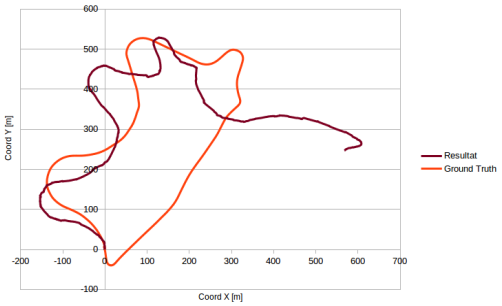

Figura 21: *Dataset 09*

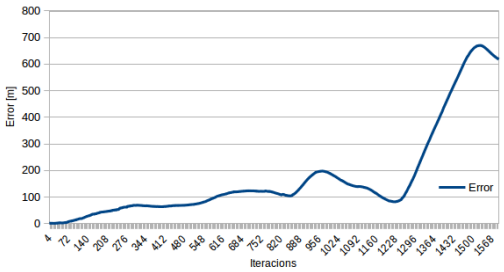

Figura 22: *Error de posició Dataset 09*

Per mesurar la velocitat de computació de l'algoritme, es calcula la velocitat d'imatges per segon (*FPS* o *Frames per second*) que pot implementar l'algoritme:

Dataset	Màxim FPS	Mínim FPS	Mitja FPS	Distància recorreguda GT	Distància recorreguda calculada	Error distància punt final
00	22,62	0,12	9,3	3716,29 m	3430,32 m	775,63 m
01	27,21	2,48	15,74	2446,88 m	2428,38 m	1595,66 m
09	29,88	1,84	14,38	1701,57 m	1679,23 m	618,17 m

Taula 1: *FPS i distància recorreguda primera iteració*

A la taula també es pot veure la distancia que ha recorregut realment el cotxe (utilitzant el valors que apareixen al fitxer de *Ground Truth*), la velocitat que s'ha estimat que ha recorregut el cotxe i la distancia entre els punts finals del recorregut real i l'estimat.

Com es pot veure tant a les figures 17, 20 i 21 l'algoritme no segueix la línia de *ground truth*, i a les poques mostres ja comença a mostrar una deriva molt important.

Les gràfiques que es poden veure a les figures 18, 19 i 22 mostren la diferència entre la posició que estima l'algoritme amb el *Ground Truth*. El que es pot veure és que l'error és acumulatiu, i va augmentant, en general, conforme avança la distància recorreguda. Encara que els errors dels *datasets* 00 i 09 són elevats, de l'ordre de 700 metres, el que més error té és el *dataset* 01, que té un error de fins 1,8 km. És un error mes elevat perquè també el cotxe es mou més ràpid.

Les imatges adquirides amb el *dataset* KITTI es van obtenir a una velocitat de captura de 10 Hz aproximadament, per tant, en els tres casos la xifra d'FPS és correcta, però es veu que de tant en tant hi ha baixades de FPS, tal i com es pot veure en aquest gràfic de les mesures de FPS al *dataset* 09:

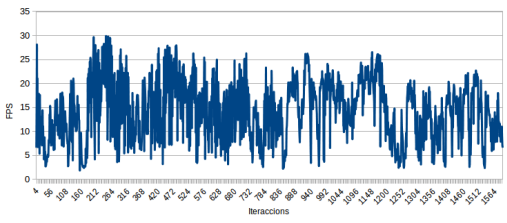

Figura 23: *FPS durant la computació del Dataset 09*

D'aquesta primera iteració de la implementació de l'algoritme podem concloure que es necessiten millores, tal i com s'explica a les següents iteracions.

4.5.2 Segona iteració

A la primera iteració s'han utilitzat els valors per defecte de totes les funcions. En aquesta iteració es volen ajustar els paràmetres de configuració de les funcions d'OpenCV que s'utilitzen.

La funció que més es pot ajustar és la funció de creació del detector ORB. Aquesta funció té aquests paràmetres per defecte [20]:

- ***nfeatures = 500:*** Número màxim de característiques que emmagatzema.
- ***scaleFactor = 1.2f:*** Factor per controlar la relació entre els nivells de la piràmide per l'anàlisi de la imatge.
- ***nlevels = 8:*** Quantitat de nivells de la piràmide.
- ***EdgeThreshold = 31:*** Aquesta és la mida de la vora on no es detecten les característiques. Hauria de coincidir, aproximadament, amb el paràmetre ***patchSize***.
- ***firstLevel = 0:*** Ha de ser 0.
- ***WTA_K = 2:*** El número de punts que produeix cada element del descriptor BRIEF.
- ***scoreType = ORB::HARRIS_SCORE: é***s el tipus d'algoritme que s'utilitza per classificar els punts característics.
- ***patchSize = 31:*** Mida del retall que utilitza el descriptor orientat BRIEF.
- ***fastThreshold = 20:*** És el llindar pel detector FAST.

De totes aquestes opcions, es van modificar els següents paràmetres:

- ***nfeatures = 3000***
- ***EdgeThreshold = 20***
- ***patchSize = 20***

Els resultats obtinguts es poden veure a les següents captures:

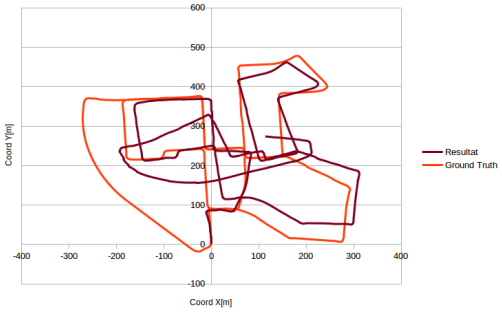

Figura 24: *Dataset 00*

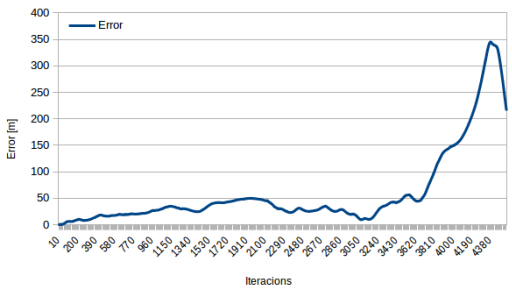

Figura 25: *Error de Posició Dataset 00*

Figura 26: *Dataset 01*

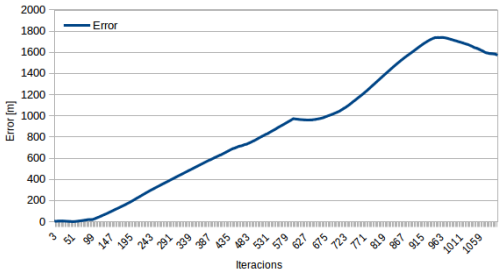

Figura 27: *Error de posició Dataset 01*

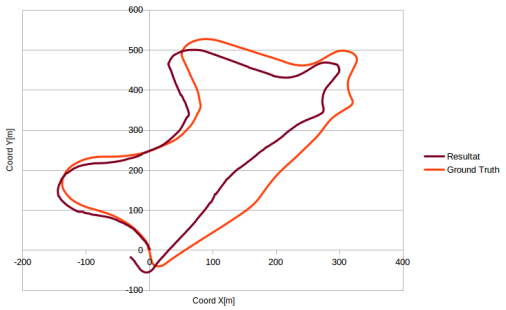

Figura 28: *Dataset 09*

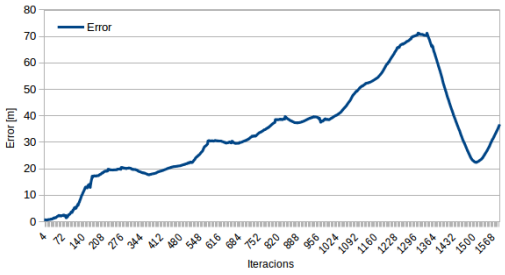

Figura 29: *Error de posició Dataset 09*

En aquest cas, el FPS i les distàncies recorregudes són:

Dataset	Màxim FPS	Mínim FPS	Mitja FPS	Distància recorreguda GT	Distància recorreguda calculada	Error distància punt final
00	6,17	0,94	2,6	3716,29 m	3494,15 m	215,51 m
01	6,46	1,14	3,97	2448,59 m	2421,42 m	1572,8 m
09	6,17	0,94	2,6	1701,57 m	1655,31 m	36,79 m

Taula 2: *FPS i distància recorreguda segona iteració*

Dels resultats podem extreure diverses conclusions:

- El resultat de l'aplicació de l'algoritme amb els *datasets* 00 i 09 (figures 24 i 28) ha millorat bastant respecte a l'anterior iteració.
- En canvi, el *dataset* de l'autopista (figura 26) no ha millorat molt respecte l'anterior iteració.
- L'error de posició sí que ha millorat als *datasets* 00 i 09 (figures 25 i 29), on arriba a 350 i 90 metres d'error màxim, en canvi, l'error al *dataset* 01(figura 27) es manté igual.
- Els FPS han caigut en picat, i ara hi ha imatges que fins i tot tarden 1 segon a ser processades. El punt positiu respecte l'anterior iteració és que ara els FPS son més estables:

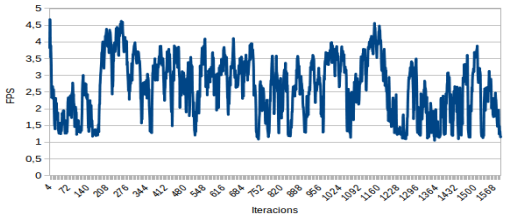

Figura 30: *FPS durant la computació del Dataset 09*

Veiem que les modificacions han millorat el comportament, però encara no és del tot satisfactori, i a més, el temps per computar és massa elevat.

4.5.3 Tercera iteració

Durant les proves de les dues primeres iteracions es va veure que a vegades els punts característics estaven molt centrats en una zona de la imatge, i això podia ser una de les causes dels errors que s'havien trobat fins ara. Per tant es va trobar interessant dividir la imatge en trossos, i aplicar els passos de detecció i emparellament separadament per cada tros, de manera que es detectessin punts en tota la imatge d'una forma el més homogènia possible. Aquesta modificació permet tenir menys punts detectats en tota la imatge, però fa que aquests punts estiguin repartits en tota la imatge, provocant una millor definició del problema geomètric que ha de complir les restriccions epipolars. L'algoritme modificat és:

Codi d'implementació d'Odometría visual

Entrada: Imatge (*I*), mínimes característiques detectades (*minCoincidències)*
Sortida: Rotació *(R)*, Translació (*t*)

```
/* Primera imatge */
capturaImatge(I);

// Bucle per trossejar la imatge
for (trossosImatge) fes
    // Es prepara un troç de la imatge
    I_tros ← dividirImatge(I,trossosImatge)

    // Retorna punts i descriptors
    [p_tros[trossosImatge],d_tros[trossosImatge] ]
        ← detectarIComputarCaracterístiques(I_tros);
end for

// Adjunta els resultats
[p',d'] ← unirDesciptorsIPunts(p_tros[trossosImatge],d_tros[trossosImatge] )

/*Bucle principal */
while (capturaImatge(I) == TRUE) then
    // Bucle per trossejar la imatge
    for (trossosImatge) fes
        // Es prepara un troç de la imatge
        I_tros ← dividirImatge(I,trossosImatge)
```

```
        // Retorna punts i descriptors
        [p_tros[trossosImatge],d_tros[trossosImatge] ]
              ← detectarIComputarCaracterístiques(I_tros);
    end for

    // Adjunta els resultats
    [p,d] ← unirDesciptorsIPunts(p_tros[trossosImatge],d_tros[trossosImatge] )
    // Les característiques que no estiguin a una distància mínima no es computen
    mask ← filtrarCaracterístiques(p,d,p',d');

    // Es computen les millors característiques
    m_res ← emparellarCaracteristiques(d,d',mask);

    // Es computen les millors característiques
    m ←emparellarCaracterístiques(d, d', mask);

    if (tamany(m) < minCoincidencies) then
        // No hi han suficients emparellaments
        [p',d']  ← [p,d];
    end if

    // Computar la matriu essencial
    E ← computarMatriuEssencial(p',p,m);

    // Obtenir la Pose, en forma de matriu de rotació i vector de translació
    [R,t] ← computarRotacióTranslació(E);

    // Guardar els descriptors i punts d'interes
    [p',d'] ← [p,d];
end while
```

Algoritme 2: *Algoritme d'OV modificat per trossejar l'imatge*

En aquesta iteració el número de punts característics a detectar es pot reduïr de les 3000 que hi havia a l'anterior iteració, a 50 per a cada tros de la imatge, i els resultats obtinguts es poden veure a les següents captures:

Els resultats obtinguts es poden veure a les següents captures:

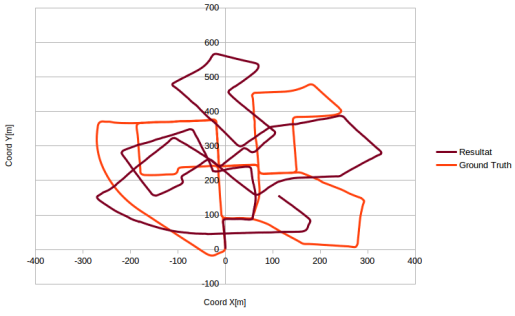

Figura 31: *Dataset 00*

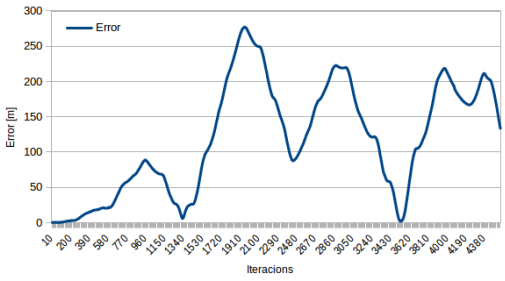

Figura 32: *Error de Posició Dataset 00*

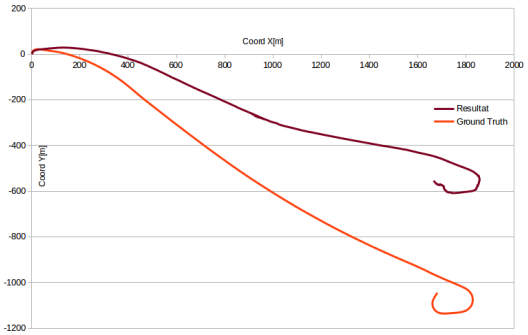

Figura 33: *Dataset 01*

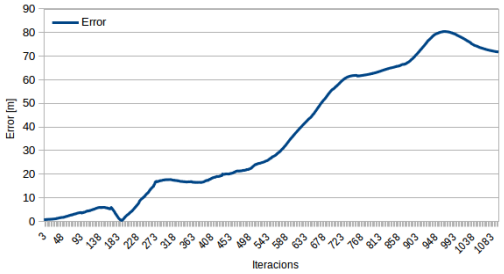

Figura 34: *Error de posició Dataset 01*

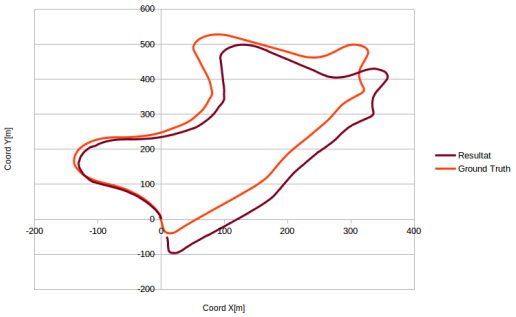

Figura 35: *Dataset 09*

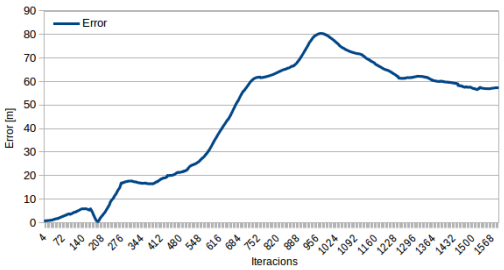

Figura 36: *Error de posició Dataset 09*

A la tercera iteració els FPS i les distàncies recorregudes són:

Dataset	Màxim FPS	Mínim FPS	Mitja FPS	Distància recorreguda GT	Distància recorreguda calculada	Error distància punt final
00	5,07	1,23	3	3716,29 m	3670,97 m	132,23 m
01	6	1,46	4,03	2446,88 m	2434,94 m	494,6 m
09	4,73	1,18	2,79	1700,84 m	1674,78 m	57,35 m

Taula 3: *FPS i distància recorreguda tercera iteració*

Observant el resultats obtinguts, podem notar que:

- El resultat obtingut al *dataset* 00 (figura 31)ha empitjorat respecte l'anterior iteració. Es veu que hi ha més error de rotació, i per tant, el resultat està una mica girat, i l'error de translació (figura 32) és una mica millor que el de l'anterior interacció.
- El resultat del dataset 09 (figura 35) ha millorat un mica respecte a l'anterior exemple. L'error no es tan gran,
- Però el que millora substancialment és el resultat del dataset 01 (figura 33), on es veu com ara sí que segueix el track, i l'error només és per *offset*. No hi ha errors que canvien la direcció del track com en l'anterior iteració.
- L'error en distància recorreguda com l'error de posició ha millorat en tots els casos.
- El temps de computació no ha millorat, segueix sent sent bastant elevat, i a més no és gens estable, els valors tenen massa fluctuació.

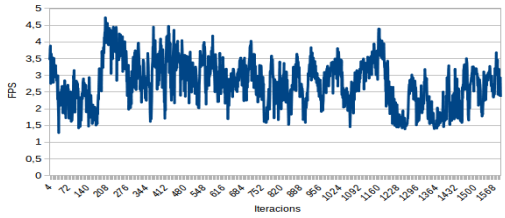

Figura 37: *FPS durant la computació del Dataset 09*

Encara que s'ha millorat molt la part de la detecció, i ara es més regular en els tres tipus de entorns, el temps de processat és massa variable i elevat.

4.5.4 Quarta iteració.

Les anteriors iteracions han estat centrades en aspectes del detector de característiques, i en aquesta iteració els canvis es realitzaran en el tipus d'emparellament. Fins ara s'ha utilitzat un detector del tipus ***match()***, però en aquesta ocasió s'utilitzarà el ***knnMatch()***. Aquest detector troba els millors k emparellaments de cada grup de *querys*. El més interessant d'aquest tipus d'emparellament és que retorna els emparellaments ordenats per distància, de menor a major, i de tal manera, es poden filtrar fàcilment els millors emparellaments. En aquest cas, i per la manera que funciona aquest descriptor, l'algoritme ha de ser modificat.

A l'algoritme 3 s'ha eliminat la màscara que s'utilitzava a l'emparellament, i ara es fa una anàlisi de tots els emparellaments per trobar els millors. Per filtrar les mostres s'utilitza una variable anomenada «*Nearest Neighbour Distance Ratio*», que es pot traduïr com la relació de distància al veí més pròxim, i s'aplica com la relació entre les distàncies del millor emparellament i la distància del segon millor emparellament:

$$matches[0].distance < 0.8 \cdot matches[1].distance$$

Perquè sigui correcte, la relació entre mostres les dues distàncies ha de ser menor a 0,8, encara que es pot arribar a 0,6. A l'algoritme s'utilitza 0,8.

Codi d'implementació d'Odometría visual

Entrada: Imatge (*I*), mínimes característiques detectades (*minCoincidencies)*
Sortida: Rotació *(R)*, Translació (*t*)

```
/* Primera imatge */
capturaImatge(I);

// Bucle per trossejar la imatge
for (trossosImatge) fes
    // Es prepara un troç de la imatge
    I_tros ← dividirImatge(I,trossosImatge)

    // Retorna punts i descriptors
    [p_tros[trossosImatge],d_tros[trossosImatge] ]
          ← detectarIComputarCaracterístiques(I_tros);
end for

// Adjunta els resultats
[p',d'] ← unirDesciptorsIPunts(p_tros[trossosImatge],d_tros[trossosImatge] )

/*Bucle principal */
while (capturaImatge(I) == TRUE) then
    // Bucle per trossejar la imatge
    for (trossosImatge) fes
        // Es prepara un troç de la imatge
        I_tros ← dividirImatge(I,trossosImatge)

         // Retorna punts i descriptors
        [p_tros[trossosImatge],d_tros[trossosImatge] ]
              ← detectarIComputarCaracterístiques(I_tros);
    end for

    // Adjunta els resultats
    [p,d] ← unirDesciptorsIPunts(p_tros[trossosImatge],d_tros[trossosImatge] )

    // Es computen les millors característiques
    m_res ← emparellarCaracteristiques(d,d');

    // Es filtren les millors característiques de l'emparellador
    m ←obtenirMillorsEmparellaments(m_res, ratio_distància);

    if (tamany(m) < minCoincidencies) then
        // No hi han suficients emparellaments
        [p',d']  ← [p,d];
    end if

    // Computar la matriu essencial
    E ← computarMatriuEssencial(p',p,m);

    // Obtenir la Pose, en forma de matriu de rotació i vector de translació
    [R,t] ← computarRotacióTranslació(E);

    // Guardar els descriptors i punts d'interes
    [p',d'] ← [p,d];
end while
```

Algoritme 3: *Algoritme d'OV modificat utilitzant l'emparellador Knn*

Els resultats que s'han obtingut són:

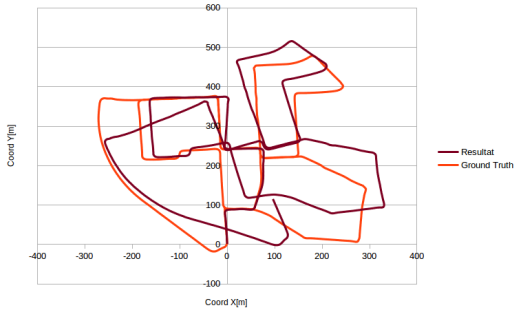

Figura 38: *Dataset 00*

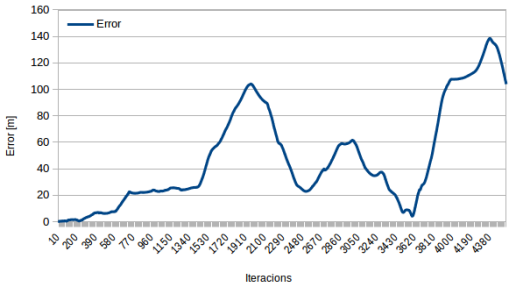

Figura 39: *Error de Posició Dataset 00*

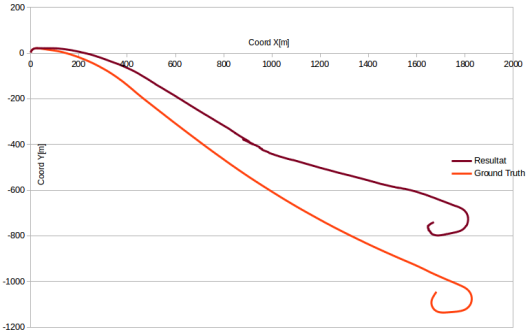

Figura 40: *Dataset 01*

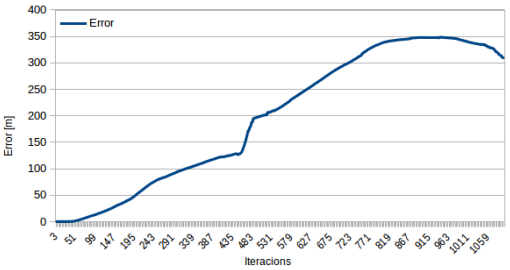

Figura 41: *Error de posició Dataset 01*

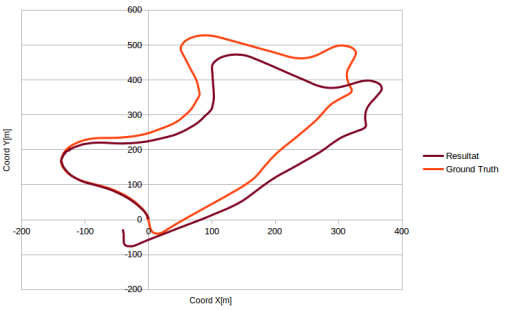

Figura 42: *Dataset 09*

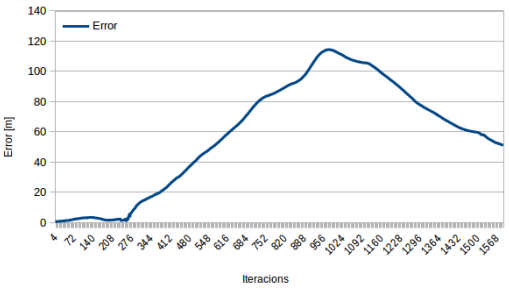

Figura 43: *Error de posició Dataset 09*

A la quarta iteració els FPS i les distàncies recorregudes són:

Dataset	Màxim FPS	Mínim FPS	Mitja FPS	Distància recorreguda GT	Distància recorreguda calculada	Error distància punt final
00	4,8	1,38	2,9	3716,29 m	3673,04 m	103,95 m
01	5,6	1,85	3,91	2446,88 m	2435,29 m	308,21 m
09	4,3	1,18	2,83	1700,84 m	1676,54 m	51,2 m

Taula 4: *FPS i distància recorreguda quarta iteració*

L'interessant d'aquesta iteració és:

- El resultat obtingut al *dataset* 00 (figura 38) ha millorat molt i ara l'error de rotació es molt més petit. L'error de posició (figura 39) també s'ha reduït.
- El resultat obtingut al *dataset* 01 i 09 (figures 40 i 42) es pot observar que ha empitjorat respecte l'anterior iteració, ara hi ha més error al *dataset* 09 (figura 43) , i al *dataset* 01 (figura 41) hi ha un moment on detecta una rotació que a l'anterior iteració no feia.
- Aquest algoritme és molt mes estable en FPS. La gran majoria de mostres estan al voltant dels 3 FPS, i és molt més estable que les altres iteracions, on s'apreciaven els increments molt erràtics en els temps de computació.

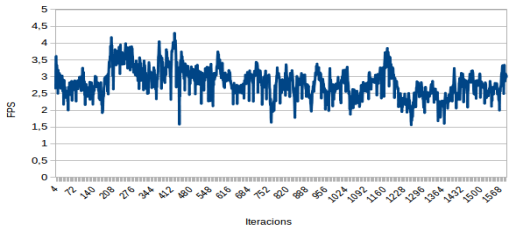

Figura 44: *FPS durant la computació del Dataset 09*

Aquesta quarta iteració, encara que no es molt millor en detecció, sí que és molt més estable en FPS, el que faria que es pogués implementar un algoritme amb una velocitat constant (de uns 3 Hz) i les estimacions de posició resultants serien bastant estables en temps.

Fent proves amb aquest tipus de detector s'ha vist que hi ha un problema bastant greu quan el cotxe està aturat. Si el cotxe està aturat a un lloc on hi creuen cotxes per davant, com per exemple en una senyal de cedir el pas o un *STOP*, si el detector troba pocs punts, apareix un error molt elevat de rotació, i a vegades, en alguna iteració s'ha vist que el cotxe arribava a tenir un error de rotació de gairebé 180 graus. Per això, el detector ha d'agafar 200 punts de cada finestra de detecció.

4.5.5 Cinquena iteració

Les quatre iteracions anteriors han deixat patent que si es vol generar una odometria més o menys acurada, el temps de computació augmenta de tal manera que si les mostres venen a temps real, no es podrà extreure l'odometria correctament.

Per intentar disminuïr aquest temps de computació, es buscarà l'ajuda de la *GPU* (*General Purpose Unit* o Unitat de processament Gràfic).

Algunes de les *GPU's* de la marca NVIDIA tenen compatibilitat amb unes llibreries anomenades *CUDA* (*Compute Unified Device Architecture* o Arquitectura unificada de dispositius de computació), que fa referència a una plataforma de computació en paral·lel que permet codificar en C algoritmes perquè siguin executats per la GPU [21].

Aquesta llibreria intenta explotar els avantatges de la GPU respecte a la CPU per a la computació en paral·lel. Les targetes gràfiques disposen de multitud de fils d'execució simultanis, i per tant poden fer tantes operacions en paral·lel com fils puguin computar. Per tant, si l'aplicació està dissenyada per *multi-threading* (multi fil), el temps de computació disminuirà. Si es veu un exemple, la targeta que porta l'ordinador on s'han fet les execucions té 640 CUDA cores a una freqüència de 915 MHz. Una targeta d'última generació, com per exemple la 1080 Titan XP té 3840 CUDA cores a una freqüència de 1582 MHz.

En aquesta iteració s'han utilitzat les llibreries d'OpenCV optimitzades per utilitzar les llibreries CUDA. El que fa és implementar l'algoritme de detecció de característiques i emparellament a la GPU, ja que són un tipus d'operació que poden aprofitar la computació en paral·lel d'una manera bastant natural.

Implementar aquest algoritme aprofitant la GPU és tan senzill com pujar les imatges a la memòria de la GPU i canviar l'instrucció per una que crida a funcions que s'executen a la GPU. Els únics canvis que s'han realitzat han sigut a la configuració:

- L'emparellador en comptes de utilitza la distancia euclidiana (*NORM_L2*), s'utilitza la distancia *Hamming (NORM_HAMMING*) perquè no s'ha aconseguit fer anar el detector de distancia euclidiana.
- *nfeatures = 400*
- *EdgeThreshold = 10*
- patchSize = 10

Els resultats que s'han obtingut són els següents:

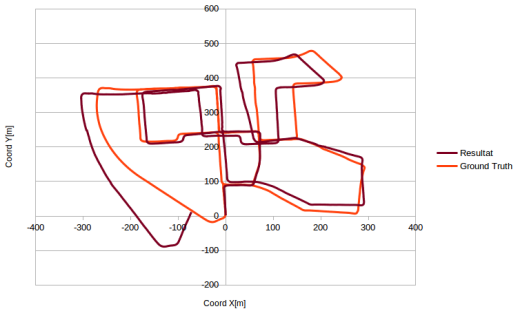

Figura 45: *Dataset 00*

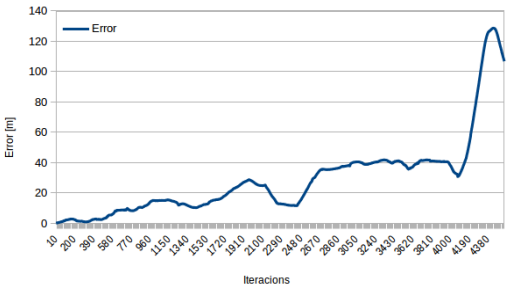

Figura 46: *Error de Posició Dataset 00*

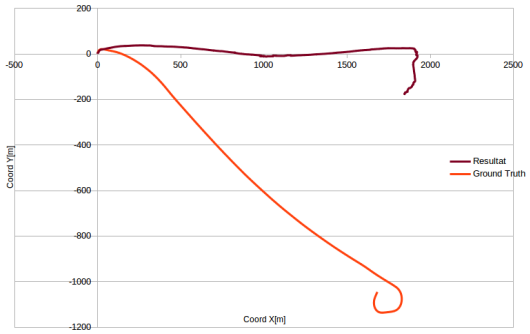

Figura 47: *Dataset 01*

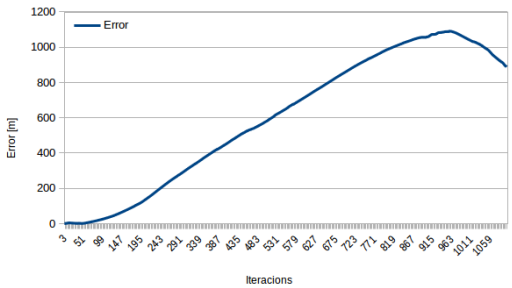

Figura 48: *Error de posició Dataset 01*

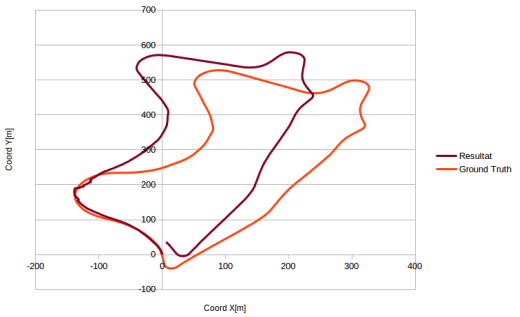

Figura 49: *Dataset 09*

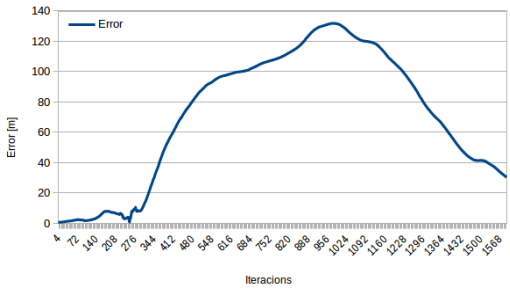

Figura 50: *Error de posició Dataset 09*

A la cinquena iteració els FPS i les distancies recorregudes són:

Dataset	Màxim FPS	Mínim FPS	Mitja FPS	Distància recorreguda GT	Distància recorreguda calculada	Error distància punt final
00	8,88	1,74	5,7	3716,29 m	3697,76 m	106,9 m
01	10,47	4,35	7,76	2446,88 m	2441,25 m	888,5 m
09	8,4	2,51	5,92	1700,84 m	1654,61 m	30,41 m

Taula 5: *FPS i distància recorreguda cinquena iteració*

Els punts que s'han observat són:

- El resultat obtingut amb el *dataset* 00 (figura 45) ha millorat molt i ara l'error de rotació és encara més petit. També l'error de translació ha millorat, encara que arriba a tenir un error molt semblant a l'anterior iteració (figura 46).
- El resultat obtingut amb el dataset 01 (figura 47) ha empitjorat considerablement, l'error és molt més gran que l'anterior, encara que l'error (figura 48) no arriba als nivells de les primeres iteracions.
- El resultat obtingut amb el dataset 09 (figura 49) també ha empitjorat, encara que l'error és molt semblant (figura 50)
- Però el sistema es molt més ràpid que l'anterior, de l'ordre de 2 vegades més ràpid. Amb aquest tipus de detector es podria anar sense molts problemes amb una freqüència de refresc d'uns 6 Hz.

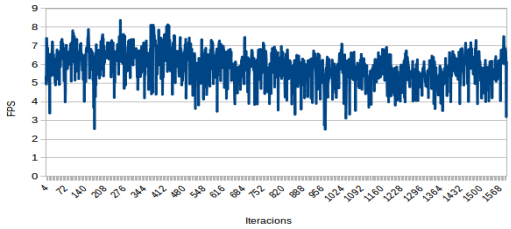

Figura 51: *FPS durant la computació del Dataset 09*

Encara que aquesta iteració no té la qualitat que tenia la quarta iteració, realitzar una computació en paral·lel utilitzant la CPU i la GPU ha proporcionat una velocitat de computació que ens podria permetre utilitzar aquest sistema en algun dispositiu real.

5 Conclusions i treball futur

5.1 Conclusions

La complexitat d'aquest projecte ha estat dividida en dos parts molt diferenciades: La part de conceptualització i la part d'implementació.

La primera part va començar amb l'estudi el problema i totes les matemàtiques necessàries per poder formular-lo. La segona part ha sigut la part més intensa, ja que s'havien d'implementar totes aquestes matemàtiques i conceptes en codi, provar-ho, corregir els problemes i tornar a provar-ho. És un temps de desenvolupament molt variable, ja que hi han hagut problemes que s'han solucionat ràpidament, i altres que han tardat dies o fins i tot setmanes per corregir o ajustar.

Hi han hagut varies coses que al final no s'han pogut implementar però que si que s'ha desenvolupat codi, com per exemple la detecció de track per implementar el *Bundle Adjustment* [5], però al final no hi ha hagut temps per poder continuar.

De totes maneres, ha sigut un projecte molt enriquidor, i la complexitat que m'ha obert la porta a aprendre noves tècniques i algoritmes per poder superar els problemes.

5.2 Treball futur

Fins aquí s'ha vist la implementació proposada per un sistema d'odometria visual. Aquesta implementació, com s'ha vist, es pot millorar, perquè encara és poc precisa i el temps de computació és molt elevat.

Després de veure els resultats de la quarta i cinquena iteració, s'ha vist que es pot arribar a un resultat bastant decent, però el temps de computació és massa elevat per poder garantir una resposta correcta amb vehicles o robots que circulin a alta velocitat. Crec que la continuació natural d'aquest projecte seria investigar als algoritmes CUDA per poder realitzar més operacions a la targeta gràfica, disminuint el temps de càlcul a la CPU.

Un dels problemes que han aparegut a partir de la quarta iteració ha sigut l'error de rotació quan el cotxe era a un cediu el pas o a un *STOP*. Crec que s'hauria de treballar en algun tipus d'algoritme que detectés que el cotxe està aturat, per minimitzar els errors que ens hem trobat. Aquest algoritme, com he pogut intuïr a les execucions, permetria reduïr la quantitat de mostres a detectar, i per tant, disminuiria el temps de computació.

Continuant el desenvolupament, a l'algoritme proposat per *Davide Scaramuzza* parla d'una implementació addicional de l'algoritme que es diu *Bundle Adjustment (BA)*. El *BA* és el concepte de minimitzar l'error de retroprojecció de les funcions triangulades respecte a diverses característiques i imatges diferents. Això es fa optimitzant la matriu de transformació i les característiques triangulades fins que el punts reprojectats produeixin un error mínim en aquestes imatges. La complexitat de BA augmenta amb la quantitat d'imatges i les

característiques triangulades que s'utilitzen en l'optimització, i per tant,augmentant també el temps d'execució.

Es va començar a desenvolupar aquest algoritme, detectant els tracks i emmagatzemant-los, però al final no s'ha pogut seguir. S'hauria d'investigar si afegint aquest algoritme es millora tant el resultat que es poden reduïr el número de punts característics a detectar, i per tant, es redueix el temps d'execució.

Una altra cosa que és pot fer és intentar implementar un algoritme d'SLAM, per aprofitar la propietat de tancament de bucle, i per tant, que el sistema pogués corregir la seva posició cada vegada que passés per un lloc prèviament visitat.

Òbviament, totes aquestes millores tenen un cost computacional que s'hauria d'analitzar . Per tant, crec que hi ha feina per endavant si es vol fer una odometria més acurada.

Si es vol entrar en millores de hardware, potser es podria utilitzar una FPGA per poder realitzar el processat en paral·lel de les imatges i reduïr el temps d'execució.

6 Bibliografia

[1] ALEX G. QUINCHIA, GIANLUCA FALCO, EMANUELA FALLETTI, FABIO DOVIS AND CARLES FERRER. "A COMPARISON BETWEEN DIFFERENT ERROR MODELING OF MEMS APPLIED TO GPS/INS INTEGRATED SYSTEMS". SENSORS (2013): 13(8)

[2] VAN SICKLE, JAN. *DEMOS, GPS MODERNIZATION AND GNSS*. HTTP://JANVANSICKLE.COM/GPS-MODERNIZATION-AND-GNSS/ (CONSULTA: 27 DE NOVEMBRE DE 2017).

[3] MOHAMMAD O.A. AQEL, MOHAMMAD H. ARHABAN, M. IQBAL SARIPAN AND NAPSIAH BT. ISMAIL. "REVIEW OF VISUAL ODOMETRY: TYPES, APPROACHES, CHALLENGES, AND APPLICATIONS". SPRINGERPLUS (2016): 5(1897) WEB: HTTPS://SPRINGERPLUS.SPRINGEROPEN.COM/ARTICLES/10.1186/S40064-016-3573-7

[4] DAVIDE SCARAMUZZA AND FRIEDRICH FRAUNDORFER. "VISUAL ODOMETRY: PART I: THE FIRST YEARS AND FUNDAMENTALS". IEEE ROBOTICS & AUTOMATION MAGAZINE (2011): DECEMBER 2011

[5] DAVIDE SCARAMUZZA AND FRIEDRICH FRAUNDORFER. "VISUAL ODOMETRY: PART II: MATCHING, ROBUSTNESS, OPTIMIZATION, AND APPLICATIONS". IEEE ROBOTICS & AUTOMATION MAGAZINE (2012): JUNE 2011

[6] RICHARD HARTLEY AND ANDREW ZISSERMAN. (2000). MULTIPLE VIEW GEOMERTY IN COMPUTER VISION. CAMBRIDGE: CAMBRIDGE UNIVERSITY PRESS.

[7] ROS WIKI. *HOW TO CALIBRATE A MONOCULAR CAMERA*. HTTP://WIKI.ROS.ORG/CAMERA_CALIBRATION/TUTORIALS/MONOCULARCALIBRATION (CONSULTA: 27 DE NOVEMBRE DE 2017).

[8] OPENCV PYTHON TUTORIALS. HARRIS CORNER DETECTION. HTTPS://DOCS.OPENCV.ORG/3.0-BETA/DOC/PY_TUTORIALS/PY_FEATURE2D/PY_FEATURES_HARRIS/PY_FEATURES_HARRIS.HTML (CONSULTA: 27 DE NOVEMBRE DE 2017).

[9] OPENCV PYTHON TUTORIALS. FAST ALGORITHM FOR CORNER DETECTION. HTTPS://DOCS.OPENCV.ORG/3.0-BETA/DOC/PY_TUTORIALS/PY_FEATURE2D/PY_FAST/PY_FAST.HTML (CONSULTA: 27 DE NOVEMBRE DE 2017).

[10] M. CALONDER, V. LEPETIT, M. OZUYSAL, T. TRZCINSKI, C. STRECHA, AND P. FUA. "COMPUTING A LOCAL BINARY DESCRIPTOR VERY FAST". IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 2012

[11] OPENCV PYTHON TUTORIALS. ORB (ORIENTED FAST AND ROTATED BRIEF). HTTP://OPENCV-PYTHON-TUTROALS.READTHEDOCS.IO/EN/LATEST/PY_TUTORIALS/PY_FEATURE2D/PY_ORB/PY_ORB.HTML (CONSULTA: 27 DE NOVEMBRE DE 2017).

[12] OPENCV PYTHON TUTORIALS. FEATURE MATCHING. HTTPS://DOCS.OPENCV.ORG/3.0-BETA/DOC/PY_TUTORIALS/PY_FEATURE2D/PY_MATCHER/PY_MATCHER.HTML (CONSULTA: 27 DE NOVEMBRE DE 2017).

[13] D. NISTER. "AN EFFICIENT SOLUTION TO THE FIVE-POINT RELATIVE POSE PROBLEM". PROC. CVPR03, 2003, PP. II: 195–202.

[14] OPENCV DOCUMENTATION. CAMERA CALIBRATION AND 3D RECONSTRUCTION – RECOVER POSE. HTTPS://DOCS.OPENCV.ORG/TRUNK/D9/D0C/GROUP__CALIB3D.HTML#GADB7D2DFCC184C1D2F496D8639F4371C0 (CONSULTA: 27 DE NOVEMBRE DE 2017).

[15] OPENCV DOCUMENTATION. CAMERA CALIBRATION AND 3D RECONSTRUCTION–DECOMPOSE ESSENTIAL MAT.HTTPS://DOCS.OPENCV.ORG/TRUNK/D9/D0C/GROUP__CALIB3D.HTML#GA54A2F5B3F8AEAF6C76D4A31DECE85D5D (CONSULTA: 27 DE NOVEMBRE DE 2017).

[16] M. A. FISCHLER AND R. C. BOLLES. "RANDOM SAMPLE CONSENSUS: A PARADIGM FOR MODEL FITTING WITH APPLICATIONS TO IMAGE ANALYSIS AND AUTOMATED CARTOGRAPHY". COMMUN. ACM, VOL. 24, NO. 6, PP. 381–395, 1981.

[17] FLORES, P., BRAUN, J. (2011). «ALGORITMO RANSAC: FUNDAMENTO TEORICO». HTTP://IIE.FING.EDU.UY/INVESTIGACION/GRUPOS/GTI/TIMAG/TRABAJOS/2011/KEYPOINTS/FUNDAMENTO RANSAC.PDF (CONSULTA: 27 DE NOVEMBRE DE 2017).

[18] GEIGER, A, LENZ, P, URTASUN, R. "ARE WE READY FOR AUTONOMOUS DRIVING? THE KITTI VISION BENCHMARK SUITE". CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), PP. 3354 – 3361. IEEE (2012).

[19] GEIGER, A., LENZ, P., STILLER, C., & URTASUN, R. "VISION MEETS ROBOTICS: THE KITTI DATASET". INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, 32, 1231–1237 (2013).

[20] KAEHLER, A. AND BRADSKI, G. (2015). LEARNING OPENCV 3: COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY. O'REILLY MEDIA.

[21] NVIDA. «¿QUE ES CUDA?». HTTP://WWW.NVIDIA.ES/OBJECT/CUDA-PARALLEL-COMPUTING-ES.HTML (CONSULTA: 27 DE NOVEMBRE DE 2017).

[22] D. NISTER, O. NARODITSKY, J. BERGEN. "VISUAL ODOMETRY". COMPUTER VISION AND PATTERN RECOGNITION. CVPR 2004.