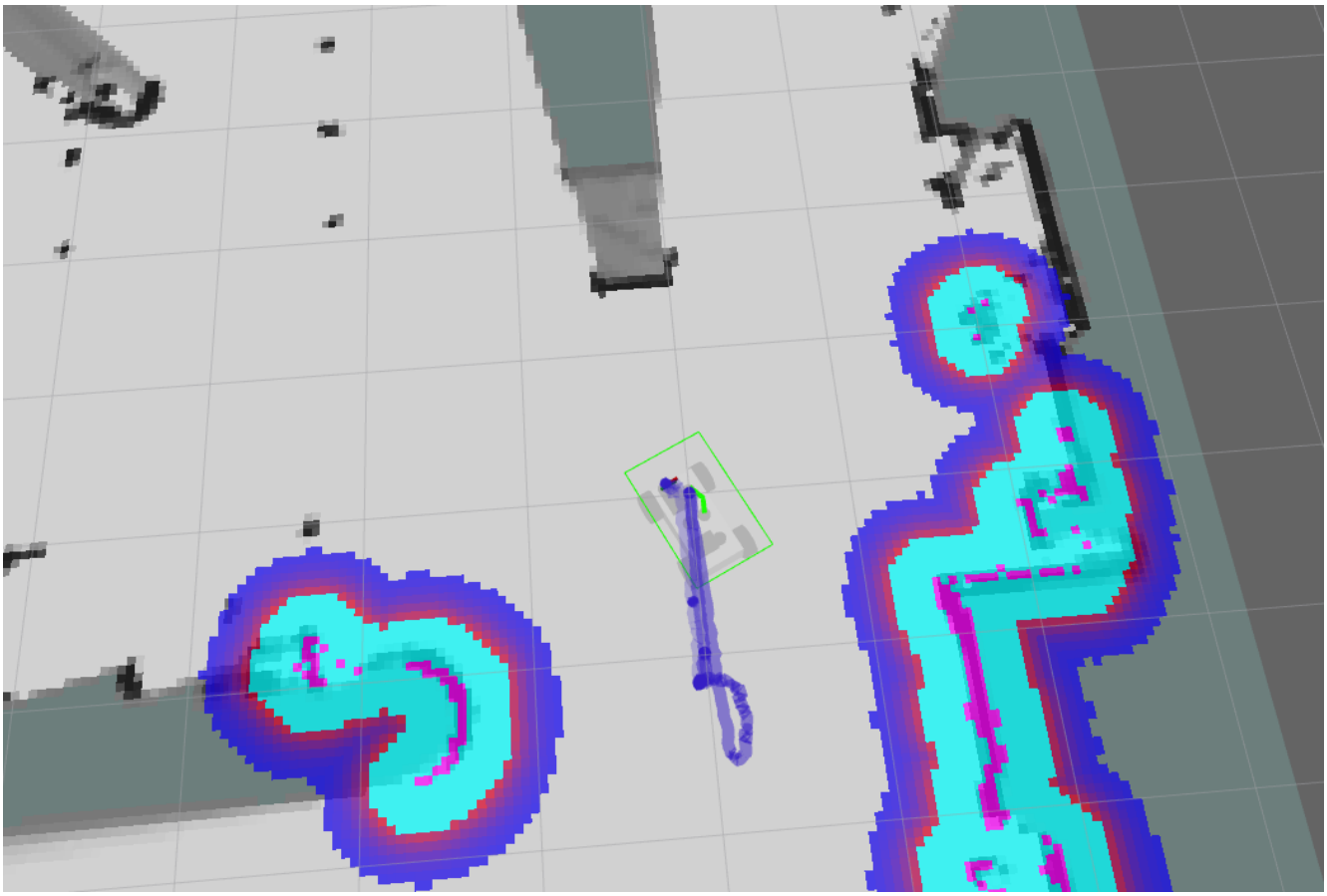

# LIDAR I SLAM PER A LA CONSTRUCCIÓ D'AMRS UTILITZANT ROS

Ton Miserachs Solà
Tutor: Ignasi Charles
2n Bat D
Institut Baix Montseny
Sant Celoni
12/21/2020

## The Three Laws of Robotics:

1) A robot may not injure a human being or, through inaction, allow a human being to come to harm.

2) A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.

3) A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

*Isaac Asimov*, I, Robot, 1942

# Agraïments

Aquest treball forma part d'un curs que tanca una gran etapa de la meva vida. Abans de tancar aquesta etapa, amb el que escriuré a continuació m'agradaria recordar a cadascuna d'aquelles persones que m'han fet ser com sóc.

Moltes de les meves qualitats venen donades per professors i família, no han sorgit del no-res.

Al meu pare li agraeixo el fet de convertir-me en una persona curiosa i inquieta, amb ganes d'aprendre sobre qualsevol tema. Ell em va introduir, quan era ben petit, en el món de l'electrònica i la tecnologia.

Gràcies a la meva mare he descobert la importància de les relacions humanes. Aquestes ens fan ser qui som, ja que l'ésser humà és un ésser social per naturalesa.

Als professors els agraeixo els coneixements adquirits, i no només els acadèmics.

Aquests fets són el que constitueixen una bona persona. No tot és estudiar; sobretot, i per davant de tot, "*s'ha de ser persona*".

Per últim, agrair als que he conegut durant aquests anys de Batxillerat tot el que hem viscut. M'han ajudat moltíssim, encara que potser no ho saben, perquè sempre han estat allà, en les bones i en les dolentes, i sé que amb ells puc parlar de qualsevol cosa. Em permeten i, alhora, em fan ser com sóc. De veritat, gràcies per tot.

# Abstract

Español:

Desde los inicios de la robótica ha habido la necesidad de crear robots móviles capaces de realizar tareas de manera autónoma. Como resultado de ello, uno de los principales problemas a resolver en el campo de la robótica móvil ha sido resolver como generar mapas del entorno de un robot y localizarlo dentro de este mapa. Al hacerlo, podemos conseguir robots totalmente autónomos que se adapten a su entorno y la programación de los cuales no dependa enteramente de su entorno.

Específicamente, los objetivos de este trabajo son aprender cómo funcionan los sensores LIDAR, aprender cómo funciona el proceso SLAM (Simultaneous Localization and Mapping) y aprender cómo funciona ROS para poder desarrollar un robot de bajo coste capaz de generar un mapa de su entorno, localizarse en él y navegar de forma autónoma.

English:

From the beginning of robotics there has been the need to create mobile robots capable of performing tasks autonomously. As a result of this, one of the main problems to be solved in the field of mobile robotics has been to solve how to generate a map of a robot's environment and locate it within this map. By accomplishing this we can make fully autonomous robots that adapt to their environment and whose programming does not fully depend on their environment.

Specifically, the objectives of this work are to learn how LIDAR sensors work, learn how the SLAM (Simultaneous Localization and Mapping) process works and learn how ROS works to be able to develop a low-cost robot that is able to generate a map of its environment, locate itself in it and navigate through it autonomously.

# Índex

# Índex d'il·lustracions

# 1. Introducció

## 1.1 Significat del títol del projecte

Per entendre el títol d'aquest projecte, *"LIDAR i SLAM per a la construcció d'AMRs utilitzant ROS"*, cal conèixer el significat de cadascun d'aquests acrònims.

- LIDAR: De l'acrònim Light Imaging Detection and Ranging, és un tipus de sensor làser utilitzat per a mesurar distàncies que destaca per oferir informació de distàncies en un pla, en dues dimensions.
- SLAM: El terme SLAM surt de l'acrònim Simultaneous Localization and Mapping (mapatge i localització simultanis). Es tracta d'un procés mitjançant el qual un robot és capaç de construir un mapa del seu entorn i, simultàniament, localitzar-se dins d'ell. Per això el robot utilitza diferents sensors per captar informació del seu entorn i diferents algorismes per a construir el mapa i localitzar-s'hi.
- AMR: De l'acrònim Autonomous Mobile Robot, robot mòbil autònom. A diferència d'un AGV (Automated Guided Vehicle), un AMR no necessita un camí preestablert per a navegar (ja sigui una línia pintada a terra, una cinta magnètica, etc.) (Robotics, s.f.) i és capaç de reaccionar adequadament a canvis no esperats de l'entorn, ja que n'està fent un mapa constantment i sap posicionar-s'hi.
- ROS :De l'anglès Robot Operating System, és un middleware robòtic, és a dir, una col·lecció de frameworks de software (marcs de treball) per al desenvolupament de robots. Per a desenvolupar el robot d'aquest projecte s'ha utilitzat aquest middleware.

## 1.2 Objectius

Com s'indica en el títol l'objectiu final d'aquest treball és desenvolupar un robot de baix cost que sigui capaç de generar un mapa del seu entorn, localitzar-s'hi i navegar-hi autònomament.

Per a aconseguir-ho ha fet falta:

- Aprendre com funciona la tecnologia LIDAR.
- Aprendre com funciona la tecnologia SLAM.
- Aprendre com funciona ROS.

## 1.3 Metodologia

Aquest treball s'ha dividit en dues parts: la recerca d'informació per a la implementació de SLAM en un robot físic, i el muntatge del mateix robot.

Per a la primera part s'ha dut a terme una extensa recerca per internet dels conceptes esmentats anteriorment en l'apartat dels objectius, però el coneixement d'aquests conceptes s'ha reforçat al dur a terme la segona part. La segona part d'aquest treball ha estat dissenyar un robot des de zero, muntar-lo i implementar SLAM en ell, per així poder

aconseguir que el robot pugui generar mapes del seu entorn, localitzar-s'hi i navegar a un punt que es vulgui de manera autònoma evitant els obstacles que es vagi trobant.

De les dues parts, la més important, i a la qual s'ha dedicat més temps, és la segona.

# 2. SLAM (Simultaneous Localization and Mapping)

## 2.1. Introducció

La majoria de robots mòbils estan programats per desplaçar-se en ambients coneguts o controlats, si es col·loca el robot en un ambient diferent del que ha estat programat el robot pot arribar a estar perdut completament.

El SLAM és una solució efectiva per a aquests casos. El terme SLAM surt de l'acrònim Simultaneous Localization and Mapping (Mapatge i localització simultanis).

SLAM és una tècnica utilitzada en robòtica que té com a objectiu construir mapes d'un entorn desconegut i, de manera simultània, estimar la posició del robot dins del mapa generat. Per calcular aquests mapes es fan servir entrades de sensors, algoritmes i diferents tipus de càlculs complexes.

Amb el SLAM s'aconsegueixen robots completament autònoms, capaços de navegar entorns desconeguts i en constant canvi.

És important destacar que el SLAM és més un concepte que un únic algoritme. (Blas, 2004)

## 2.2. Història del SLAM

El concepte de SLAM sorgeix el 1986 a la IEEE Robotics and Automation Conference, celebrada a San Francisco. En aquesta època els mètodes probabilístics acabaven d'introduir-se tant en la robòtica com en la IA[1]. Alguns investigadors havien buscat aplicar mètodes d'estimació teòrica per a problemes de mapatge i localització; entre ells s'incloïen Peter Cheeseman, Jim Crowley i Hug Durrant-Whyte. En el transcurs de la conferència es va debatre molt sobre la creació de mapes consistents. Durant aquesta, Raja Chatila, Oliver Faugeras, Randal Smith i més investigadors van contribuir a la conversa.

Arran d'aquesta conversa es va arribar a la conclusió que l'aplicació d'aquests mètodes probabilístics era de vital importància per a la robòtica i que eren un problema que s'havia de resoldre. Al llarg dels següents anys es van publicar alguns dels treballs clau dels temes tractats a la conferència.

El 1990 Smith, Cheeseman i Durrant-Whyte establirien les bases per a descriure la relació entre la localització i els landmark (punts de referència). Un element clau d'aquests treballs va ser mostrar que existeix un alt grau de correlació entre les estimacions de la localització dels landmarks de l'entorn en un mapa i que aquestes correlacions creixerien amb observacions successives. Mentrestant Ayache i Faugeras estaven treballant sobre

[1] IA o AI en anglès, Intel·ligència artificial

mètodes de navegació visuals, i Crowley, Chatila i Laumond en navegació basada en sonar, aplicant algoritmes basats en el filtre de Kalman.

Aquestes dues vessants de la investigació del SLAM tenien molt en comú i poc després van convergir en un estudi sobre marques en l'entorn (landmarks) de Smith R, Self M, Cheeseman P, en un article publicat per aquests mateixos que portava per nom *"Estimating uncertain spatial relationships in robotics"* (Smith R, 1986), en el que es mostra el treball sobre un robot mòbil desenvolupant les seves tasques en un entorn desconegut prenent dades dels landmark. Aquest estudi mostrava que l'estimació de la posició d'un landmark ha d'estar correlacionada amb els altres landmarks, degut als errors en l'estimació de la posició del vehicle que es van acumulant en moure's aquest.

Les implicacions d'aquesta afirmació van ser importants: Una solució consistent al problema de la combinació del mapatge i la localització simultanis implicaven que seria necessari un estat inicial en el qual es trobessin inclosos la posició del robot i la de cada landmark. Per això, el sistema d'estimació hauria d'utilitzar un gran vector d'estats (de l'ordre del nombre de marques mantingudes en el mapa) que creixeria de manera quadràtica amb el nombre de landmarks, tenint un gran cost computacional.

Aquest treball no es va centrar en les propietats de convergència del mapa o en el seu estat estacionari. De fet, es va assumir al moment que els errors en l'estimació del mapa no convergirien i que de fet mostrarien un comportament "random-walk"[2] (Wikipedia) amb un creixement sense límits de l'error. Per tant, donat l'elevat cost computacional del problema de la realització del mapa i sense coneixement del comportament convergent del mapa, les investigacions es van centrar en una sèrie d'aproximacions al problema de la creació consistent d'un mapa, que assumien o inclús forçaven la minimització o eliminació de la correlació entre marques de manera que es reduís el filtre complet de l'algorisme de creació del mapa a una sèrie de filtres landmark-robot desacoblats. Per aquest mateix motiu el treball teòric sobre la combinació de la localització i el mapatge va romandre inactiu temporalment, amb els esforços centrats a tractar la localització i el mapatge com problemes separats.

Més tard es va descobrir que el problema combinat de mapatge i localització, una vegada formulat com un problema d'estimació única, era en realitat convergent. És més, es va reconèixer que les correlacions entre els landmarks eren la part més crítica del problema i que, contràriament al que es pensava, com més creixi la correlació millor serà la solució.

Arribat aquest punt, el treball se centraria a aconseguir augmentar l'eficiència computacional i solucionar els problemes associats a la presa de dades al tancament del bucle.

[2] Random-walk: Procés random, aleatori.

A partir de llavors, el SLAM es va convertir en un problema de vital importància per a la robòtica mòbil i s'han dut a terme moltes investigacions en les quals s'han desenvolupat nous algoritmes, més eficients, amb menys error, i que se segueixen desenvolupant actualment.

## 2.3 Estructura del SLAM

El procés del SLAM consisteix en diferents passos. L'objectiu d'aquest procés és generar un mapa de l'entorn i localitzar el robot en ell mitjançant la informació captada pels diferents sensors.

Podem dividir el problema del SLAM en diferents parts: assignació de landmarks, associació de dades, estimació de l'estat, actualització de l'estat i actualització de landmarks. Un cop dividit, hi ha moltes maneres de solucionar cada part (Blas, 2004).

L'algoritme funciona de la manera següent: Quan l'odometria canvia perquè el robot es mou s'actualitza la informació de l'odometria i s'extreuen els landmarks. Si troba nous landmarks, els inclou en el vector d'estats i continua el seu camí. Després de tornar a observar un landmark que ja es trobava en el vector d'estats es tanca el bucle de manera que, coneixent la posició del landmark i del robot en el mapa, la localització del landmark (distància i orientació respecte al robot) i les incerteses degudes al procés de mesurament i posicionament, s'actualitza el vector d'estats, rectificant així el mapa i per tant la posició del robot en ell.

Aquest procés se sol fer utilitzant un EKF (Filtre de Kalman Estès, s'explicarà més endavant). Aquest s'encarrega d'anar actualitzant la posició en la qual el robot creu que està situat dins del mapa, però també s'encarrega d'estimar les posicions relatives dels landmarks dins del mapa i respecte al robot. El procés es pot resumir de la següent manera:

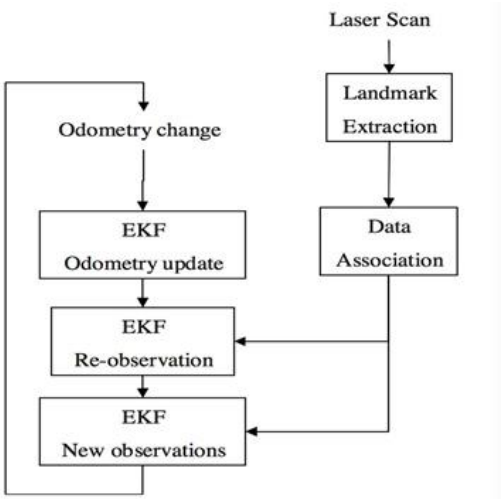


**Figura 1 Esquema del procés SLAM**

## 2.4 Sensors

Un robot mòbil es caracteritza pel fet de tenir un/uns actuadors que li permeten moure's. Aquests poden ser motors amb hèlixs, en cas d'un robot aeri, motors amb rodes, en cas d'un robot terrestre, etc.

Per aconseguir un robot autònom, a part dels actuadors, és necessari que el robot sigui capaç de calcular la velocitat a la qual es desplaça i detectar els objectes que té al voltant.

Per això s'utilitzen diferents tipus de sensors.

### 2.4.1 Sensor làser

El sensor làser és un dels més utilitzats actualment en robòtica, per la seva precisió, fiabilitat i rapidesa. Per calcular distàncies amb el làser existeixen dos mètodes:

- TOF[3]: Mètode en el qual s'emet un feix de llum polsat (normalment amb una longitud d'ona fora de l'espectre visible) i es mesura el temps fins que es torna a rebre a causa de la reflexió d'aquest en un objecte. D'aquesta manera la distància fins a l'objecte serà $D = \frac{1}{2}cT$, on D és la distància, T el temps des de l'emissió fins a la recepció i c la velocitat de la llum.
- Phase-shift[4]: Mètode en el qual s'emet un raig làser continu, la potència del qual va modulant sinusoidalment, cap a l'objectiu. Quan l'ona arriba a l'objectiu, part d'ella continua per refracció i una altra part és reflectida, motiu pel qual es produeix un canvi de fase en l'ona reflectida. Comparant-la amb l'ona de referència emesa inicialment podem saber la distància que ha recorregut, trobant així la distància del sensor fins a l'objecte. Aquest mètode no és pràctic per mesurar distàncies grans.

[3] Time of flight, temps de vol.
[4] Desfasament.

Per calcular la distància utilitzant el phase-shift s'analitza la funció d'autocorrelació del senyal elèctric i òptic utilitzant quatre mostres A1, A2, A3, A4 desfasades 90 graus cadascuna. Trobem la fase (Φ) amb la fórmula següent:

$$\phi = arctan\frac{(A1 - A3)}{(A2 - A4)}$$

I amb la fase podem calcular la distància de la següent manera:

$$D = \frac{(c \cdot \phi)}{(4 \cdot \pi \cdot F_m)}$$

On c és la velocitat de la llum, Φ la fase calculada amb l'equació anterior i Fm la freqüència amb la qual ha estat modulada l'ona emesa. (Wikipedia)

Alguns dels principals avantatges d'aquests sensors són la seva rapidesa, considerada de reacció instantània; la precisió en les mesures, ja que en els sensors actuals s'arriba a una precisió de fins 10mm aproximadament i una precisió angular d'entre 0,2º i 0,25º; i la seva immediatesa, ja que les dades del sensor poden ser interpretades directament.

Entre els seus inconvenients cal destacar el seu elevat cost, el fet que donen informació sobre un pla, que pel seu funcionament no poden detectar alguns materials transparents, com el vidre, i que es veuen afectats per les condicions lumíniques de l'entorn.

Tot i això, és considerat com el millor sensor per mesurar distàncies sobre superfície planes.

#### 2.4.1.1 LIDAR

De l'acrònim anglès Light Imaging Detection and Ranging, és el sensor làser més utilitzat en robòtica actualment. Existeixen sensors LIDAR de diferents tipus amb diferents rangs angulars de detecció, però els més comuns són els de dues dimensions.

Un LIDAR 2D consisteix en un emissor, i un detector làser que giren sobre un mateix pla. Utilitzant el TOF i pel fet d'estar girant, aquest és capaç de rebre mesures de les múltiples direccions compreses en el pla en el qual es troba el sensor.

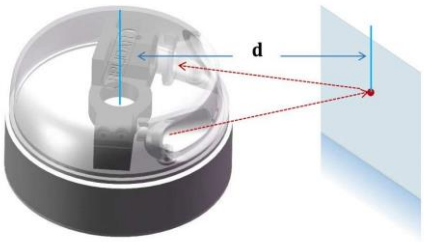


**Figura 2 Sensor LIDAR**

### 2.4.2 Sensor d'ultrasons (Sonar)

De la mateixa manera que el sensor làser, és molt utilitzat en robòtica, però en aquest cas pel seu cost. La base de funcionament dels sonar o sensors d'ultrasons és la mateixa que els sensors làser (TOF), però amb la diferència que en comptes d'enviar un feix de llum s'envia una ona sonora de l'ordre dels ultrasons. És per això que el seu cost no és tan elevat.

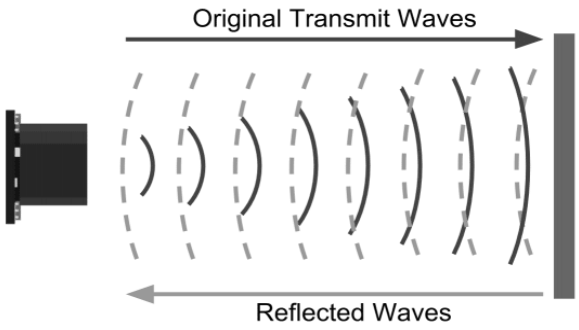


**Figura 3 Sensor d'ultrasons**

Els principals avantatges d'aquests sensors són la seva precisió, que poden treballar en entorns amb qualsevol mena d'il·luminació, i el seu reduït cost.

Aquests tenen una gran quantitat d'inconvenients que cal destacar:

- Reflexions múltiples: Les ones sonores poden ser reflectides per més d'un objecte abans d'arribar al receptor, causant lectures errònies.
- *Crosstalk*: Quan treballem amb més d'un d'aquests sensors pot ocórrer que els sensors rebin ones que procedeixin d'altres sensors.
- Resolució angular baixa.
- Rang limitat: El rang sol anar dels 0,2 m als 10 m, pel que no serveixen per mesurar distàncies molt grans.

- Velocitat de propagació de l'ona: La velocitat del so en l'aire és d'aproximadament 340 m/s, la qual és notablement inferior a la de la llum, que és d'aproximadament 300.000 km/s. És per això que no s'obtenen els resultats tan ràpidament.

Per intentar solucionar alguns d'aquests problemes es pot fer servir un conjunt d'aquests sensors mitjançant tècniques de triangulació, basades en regles trigonomètriques.

### 2.4.3 Visió

La visió, a diferència del làser i el sonar, permet detectar distàncies i objectes en l'entorn. Les càmeres proporcionen imatges, sigui en color o en blanc i negre, i aquestes imatges proporcionen molta informació que, un cop processada, pot servir per identificar objectes, formes i inclús distàncies si es compta amb dues imatges o més d'una càmera.

El funcionament de la majoria de càmeres TOF es basa en el phase-shift per calcular la distància.

Els principals avantatges són la gran quantitat d'informació que proporcionen, la capacitat d'obtenir informació en 3D i el seu funcionament com a sensor passiu (no cal que emeti cap mena d'ona, a diferència del làser i el sonar). No obstant això, són dispositius que, en proporcionar tanta informació que s'ha de processar, requereixen una capacitat de computació elevada, solen ser cars i es veuen molt influenciats per la lluminositat de l'entorn.

#### 2.4.3.1 Microsoft Kinect

El sensor Kinect és un sensor desenvolupat per a la Xbox de Microsoft, i permet al jugador interaccionar amb la consola sense la necessitat d'un comandament.

El Kinect, a part de ser utilitzat per videojocs, també és un sensor molt utilitzat per a la robòtica per les seves característiques.

El Kinect incorpora una càmera RGB, un sensor de profunditat IR i un micròfon bidireccional. Tots aquests sensors que incorpora es poden utilitzar per detectar objectes en 3D, per reconeixement facial i per comandaments de veu.

El rang de detecció d'aquest sensor és d'1,2 a 3,5 m de profunditat, 57,5º horitzontalment i 43,5º verticalment, més un moviment vertical de ±27º gràcies a un pivot motoritzat.

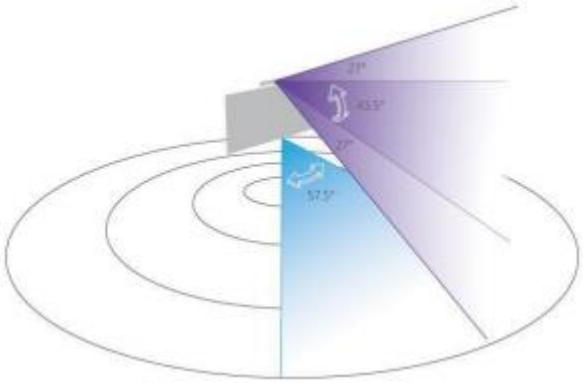

**Figura 4 Rang detecció Kinect**

El funcionament es basa en un sensor infraroig emissor i receptor combinat amb un sensor CMOS. El sensor infraroig envia un feix de llum, i utilitzant el TOF, el sensor CMOS[5] i el receptor IR[6] el reben i es calcula així la distància a la qual es troben els objectes a detectar.

### 2.4.4 Odometria

L'odometria estudia l'estimació de la posició d'un vehicle amb rodes durant la navegació. Amb ella podem fer una estimació de la trajectòria del robot, estimant la posició relativa del sistema respecte a la posició inicial.

L'odometria es tracta d'un càlcul incremental que inevitablement fa que els errors s'acumulin, és per això que la precisió d'aquesta és essencial per a un bon SLAM. Un error inferior a 2 cm/m i menor de 2º per cada 45º és el recomanable perquè el SLAM es calculi bé.

A part de servir per estimar la posició del robot, també serveix per trobar les distàncies als landmarks i així localitzar-los correctament dins del mapa.

Els tipus de sensors més comuns, i que se solen utilitzar en conjunt, per obtenir l'odometria són els següents:

- IMU (Inertial Measurement Unit): és un conjunt de dos detectors diferents; un acceleròmetre i un giroscopi. L'acceleròmetre mesura acceleracions en tres eixos, i el giroscopi mesura acceleracions angulars.
- Encoders: Els encoders van afegits als motors i mesuren la velocitat angular d'aquests. Utilitzant aquesta informació i el radi de les rodes es calcula el desplaçament del robot.

[5] Sensor que converteix fotons en electrons per al seu processament digital. Fet servir en moltes càmeres.
[6] Infrared, infraroig.

## 2.5 Landmarks

Els landmarks, o punts de referència, són punts que de l'entorn que el robot utilitza per estimar la seva posició. Aquests han de ser fàcils de distingir de la resta d'objectes o punts de l'entorn.

Perquè el robot pugui estimar la seva posició correctament els landmarks han de ser reobservables i accessibles pel sensor des de diferents posicions. És per això que el primer cop que el sensor detecta un landmark només el situa en el mapa, només quan el torna a veure el pot utilitzar com a referència per a situar-se i estimar la posició del robot dins del mapa.

Un altre factor a tenir en compte és que els landmarks han de ser estacionaris, ja que si es mouen no serveixen per estimar la posició del robot en un moment determinat, pel fet que la seva posició actual després del moviment i la que el robot té emmagatzemada serien diferents. (Blas, 2004)

En resum, un bon landmark:

- Ha de ser observable des de diferents punts.
- S'ha de poder distingir fàcilment.
- Ha de ser fix.

### 2.5.1 Extracció de landmarks

Un cop s'ha decidit quins punts de l'entorn són candidats a ser landmarks s'ha d'extreure d'alguna manera fiable dels sensors del robot. Per a això existeixen molts mètodes, entre els que veurem RANSAC i Spike Landmarks.

#### 2.5.1.1 RANSAC

RANSAC, de l'anglès Random Sampling Consensus, és un mètode matemàtic que es pot utilitzar per a aconseguir línies a partir d'un sensor làser. Aquestes línies poden ser utilitzades com a landmarks. És uns dels mètodes més utilitzats per l'extracció de landmarks amb sensors làser. Funciona molt bé en entorns interiors, ja que en aquests és habitual que el làser observi línies rectes, ja que aquestes són formades per les parets.

RANSAC troba aquestes línies prenent dades de manera aleatòria del sensor làser i fent servir una aproximació per mínims quadrats (Wikipedia, s.f.) per trobar la línia que millor s'ajusta als resultats obtinguts. Una vegada realitzat això, RANSAC comprova quants resultats del làser corresponen a la mateixa línia. Si el nombre és superior a un llindar preestablert (anomenat consensus) es pot dir amb certesa que s'ha trobat una línia recta. (Blas, 2004)

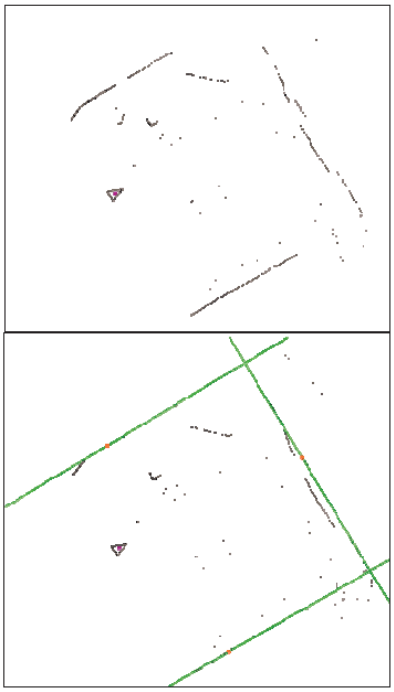

**Figura 5 Línies creades amb RANSAC, els valors atípics no influeixen en el resultat**

### 2.5.1.2 Spike Landmarks

Aquest mètode utilitza extrema [7] en les dades de distància del sensor làser per a identificar landmarks. En aquest procés, el robot analitza les dades de distància en l'àrea visible buscant canvis significatius de distància per localitzar zones de canvis extrems. El robot pot distingir aquestes zones com landmarks. Utilitzant aquest mètode, sovint s'extreuen objectes com ara potes de taula i cantons d'escriptori com a punts de referència.

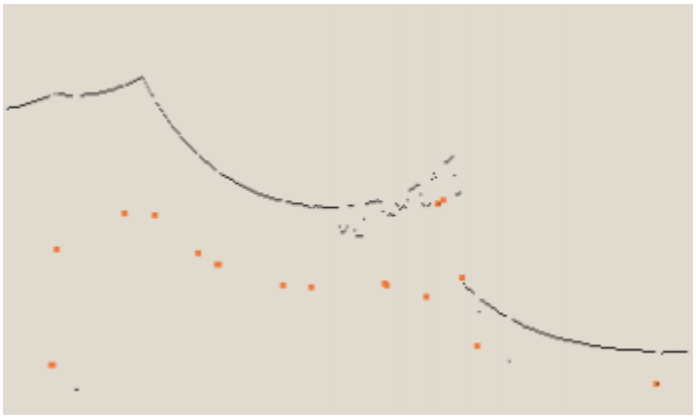

**Figura 6 Spike landmarks. Els punts de color taronja són potes de taula extretes com a landmarks**

[7] Extrema: en càlcul, qualsevol punt en què el valor d'una funció és més gran (un màxim) o més petit (un mínim).

Aquest mètode per extreure landmarks funciona molt bé en entorns amb molt de desordre, ja que hi ha moltes extrema disponibles per a extreure com a landmarks. Falla en entorns més "suaus" on no hi ha diferències significatives de distància perquè l'algoritme no és capaç d'extreure l'extrema. (Macalester College, 2013)

## 2.6 Associació de dades

Un cop s'han obtingut els landmarks del sistema s'ha de buscar algun mètode per assegurar que quan es torna a veure un landmark sigui el mateix. Per això, s'estableixen unes regles que s'han de complir per a considerar un punt com a landmark, ja que en el cas que no es donés una correspondència adequada dels landmarks es provocaria un error fatal en el sistema, el robot es pensaria que es troba en una posició completament diferent.

Els problemes que podem trobar al moment de l'associació de les dades són els següents:

- És possible que no tornis a observar landmarks al llarg del temps
- Pots identificar alguna cosa com un landmark però no tornar-la a veure més.
- Pots associar de manera errònia un landmark a un altre vist prèviament.

Com ja s'ha dit en l'apartat dels landmarks, aquests ha de ser reobservables. Per aquest motiu els dos primers casos no són vàlids com a landmarks. Encara que facis servir un molt bon algoritme d'extracció de landmarks eventualment trobaràs aquestes situacions, així que és millor definir unes regles a complir.

La primera regla serà que no considerem un punt com a landmark fins que no hagi estat reobservat N vegades. D'aquesta manera assegurem que es tracta d'un punt que podem tornar a veure. Si el sistema agafés punts no reobservables com a landmarks, a part d'introduir errors en ell, requeriria més potència de càlcul de manera innecessària, a l'estar fent més gran el vector d'estat amb una dada sense importància.

Un dels mètodes per aconseguir això es denomina *nearest-neighbor approach*, aproximació al veí més proper en català, i funciona d'aquesta manera:

1. Quan s'obté un nou escaneig làser s'extreuen tots els landmarks visibles.
2. S'associa cada landmark extret amb el seu landmark més proper que ha estat vist més de N vegades en la base de dades.
3. Es passa aquest parell d'associacions (landmark extret i landmark de la base de dades) per un procés de validació.
    a. Si el parell passa aquest procés, ha de ser el mateix landmark reobservat, així que s'incrementa el nombre de vegades que s'ha vist.
    b. Si el parell no passa el procés de validació, s'afegeix aquest landmark a la base de dades com un landmark nou, i s'estableix N=1, ja que s'ha vist per primer cop.(Blas, 2004)

# 3. EKF-SLAM

El filtre de Kalman estès va ser el primer algoritme implementat en el problema del SLAM, ja que molts investigadors feien servir aquest mètode per models de navegació en robots mòbils.

L'EKF (filtre de Kalman estès) està basat, com el seu nom indica, en el filtre de Kalman, algoritme desenvolupat per Rudolf Emil Kálmán. Durant una visita de Kalman al NASA Ames Research Center, Stanley F. Schmid va descobrir la utilitat que té el filtre de Kalman per problemes d'estimació de trajectòries no lineals (com la de la major part dels robots). El principal inconvenient era que el filtre de Kalman estava destinat a estimacions de sistemes lineals, i aquí és on va aparèixer el filtre de Kalman estès. El filtre de Kalman estès linealitza els models no lineals mitjançant sèries de Taylor multivariades. Schmid va incorporar aquest filtre en el sistema de navegació del projecte Apollo.

L'EKF és considerat l'estàndard per excel·lència en l'estimació d'estats no lineals. (Wikipedia, s.f.)

L'EKF s'encarrega d'estimar la posició del robot a través de l'odometria i dels landmark. A més, l'EKF ha de fer això tenint en compte una estimació del mapa en el qual es va movent, generant incertesa en totes les dades. Per fer aquesta estimació el filtre utilitza una forma de control de retroalimentació: el filtre estima l'estat del sistema en un instant donat i després obté retroalimentació en forma de mesures amb soroll dels sensors.

Com tot SLAM, el procés que realitza aquest filtre es pot dividir en els passos següents que s'executen en bucle:

- Actualitza la posició actual del robot amb les mesures d'odometria.
- Actualitza la posició estimada fent servir els landmark reobservats.
- Afegeix nous landmarks.

A part d'aquests passos també en fa un de previ en el qual s'inicialitzen variables i s'introdueixen els primers landmark obtinguts pel sensor.

L'EKF es basa en una sèrie de matrius, aquestes no s'explicaran en aquest treball.

# 4. ROS (Robot Operating System)

ROS (de l'anglès Robot Operating System) és un middleware robòtic, és a dir, una col·lecció de frameworks de software (marcs de treball) per al desenvolupament de robots. S'anomena Robot Operating System, però, de fet, no es tracta d'un sistema operatiu. Malgrat això ROS ofereix serveis com els que pot oferir un sistema operatiu: abstracció de hardware, controladors de dispositius a baix nivell, pas de missatges entre processos i manejament de paquets.

Es tracta d'un sistema de codi obert, l'objectiu del qual és aprofitar l'ús de codi ja existent per a fer més fàcil la programació de robots complexos i robusts.

## 4.1 Història de ROS

Des dels inicis de la robòtica ha existit la necessitat d'un framework col·laboratiu obert per al desenvolupament de software per a robots, és per això que existeixen molts projectes amb aquesta finalitat, com MRPT, MRDS, CARMEN o ROS.

Les primeres parts del que més endavant seria ROS sorgeixen el 2007, a la universitat de Stanford. Eric Berger i Keenan Wyrobek, estudiants de la universitat que treballaven en el laboratori de robòtica de Kenneth Salisbury, dirigien el programa Stanford Personal Robotics Program. ROS va començar com un projecte personal d'aquests estudiants, com un intent d'eliminar la situació de "*reinventing the wheel* " (reinventar la roda) la qual patia la robòtica. Els problemes principals eren els següents:

- Es dedicava massa temps a tornar a implementar la infraestructura de software que fa falta per construir algoritmes robòtics complexos (bàsicament drivers pels sensors i actuadors, i mitjans de comunicació entre diferents programes dins del mateix robot).

- Es dedicava molt poc temps en desenvolupar programes robòtics intel·ligents que estiguessin basats en aquesta infraestructura.

Aquest fet el podem veure clarament en una de les diapositives que Eric i Keenan van utilitzar per trobar espònsors pel seu projecte.

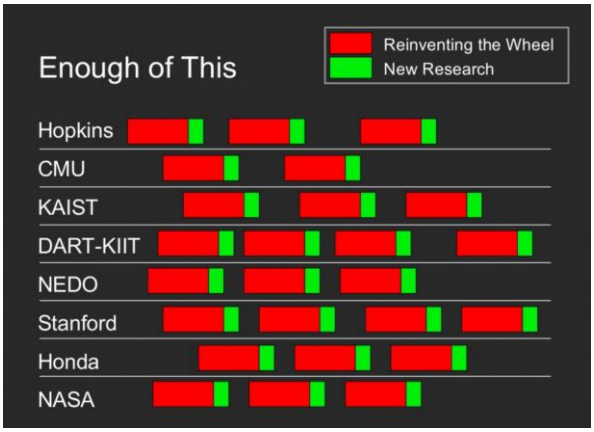


**Figura 7 Diapositiva d'Eric i Keenan**

És aquí on destaca el programa de Stanford Personal Robotics Program, que tenia com a objectiu principal el desenvolupament d'un framework que servís com a base per proporcionar un punt de partida i unes estructures bàsiques per poder desenvolupar plataformes robòtiques més complexes sobre aquest mateix, sense necessitat d'estar "reinventant la roda" constantment.

## 4.2 Sistema d'arxius de ROS

El sistema d'arxius de ROS es pot distribuir en aquestes capes:

- Paquets: Són la unitat principal per organitzar tot el programari de ROS. Aquests poden contenir rutines de processos d'execució (nodes), llibreries, arxius de configuració, etc. Els paquets són la part més atòmica de ROS, és a dir, la cosa més essencial que hi ha dins el sistema d'arxius.
- Metapaquets: Són paquets especialitzats per a una funció. Estan formats per un conjunt de diferents paquets relacionats entre si.
- *Package Manifests*: Dins l'arxiu package.xml, conté informació bàsica sobre el paquet, com el seu nom, versió, descripció, llicències, dependències, etc.
- Repositoris: Molts dels paquets de ROS es mantenen fent servir *Version Control System* (VCS); com Git, Subversion (svn), Mercurial, etc. Els repositoris són conjunts de paquets que comparteixen un VCS comú.
- Tipus de missatge (*msg types*): Es defineixen dins la carpeta msg dins d'un paquet, amb l'extensió .msg (my_package/msg/MyMessageType.msg), i defineix l'estructura de dades dels missatges a ROS.
- Tipus de serveis (*Service types*): Es defineixen dins la carpeta srv dins d'un paquet, amb l'extensió .srv (my_package/srv/MyServiceType.srv), i defineix l'estructura de dades de sol·licitud i resposta per als serveis a ROS.

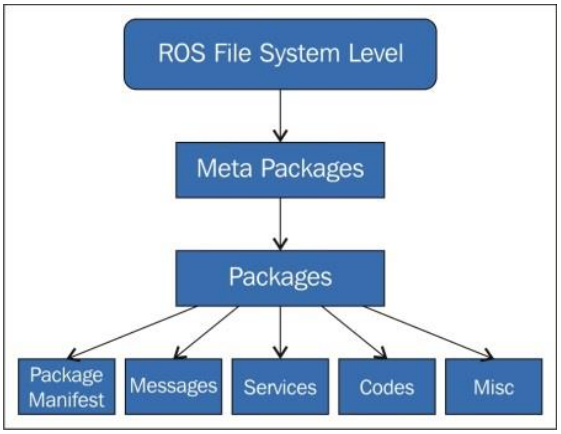


**Figura 8 Sistema d'arxius de ROS**

## 4.3 Sistema de còmput de ROS

El sistema de còmput de ROS està basat en un graf format per una xarxa bidireccional (peer-to-peer) de processos que processen dades junts. Els conceptes bàsics d'aquest graf són els següents:

- Nodes: Són els processos que realitzen càlculs. Cada node sol estar destinat a una funció essencial del robot, ja sigui recollir dades d'un sensor làser, controlar els motors, localitzar el robot, etc; formant així una estructura modular del sistema.

  Aquests nodes es comuniquen entre ells a través dels Topics, i solen estar escrits en C++ o en Python.

- Master: El master controla tot el procés, de manera que, gràcies a ell, els nodes poden trobar-se, intercanviar missatges o invocar serveis.

- Servidor de paràmetres: S'encarrega d'emmagatzemar la informació en una localització central. És una part del master.

- Missatges: Els diferents nodes es comuniquen entre ells amb aquests. Un missatge és simplement una estructura de dades. Poden incloure tipus de dades primitives (enter, coma flotant, booleà, etc.) i cadenes (en forma de matriu) d'aquestes.

- Topics: Els missatges es transporten a través d'un sistema de publicador/subscriptor. Els nodes envien missatges publicant-los en un topic, i els reben subscrivint-se a un topic. El topic és un nom que es fa servir per identificar el contingut del missatge que transporta aquest. Els nodes subscrits a un topic tindran accés a la informació publicada per altres nodes en aquell topic. Cada node pot publicar i estar subscrit a múltiples topics sense cap problema. Com que els nodes que publiquen i se subscriuen a un topic no tenen per què conèixer-se entre ells, el que es busca amb l'ús de diferents topics és separar la producció d'informació del seu consum. Es pot entendre un topic com un bus de missatges.

- Serveis: En alguns robots un sistema publicador/subscriptor no és suficient, es necessita un sistema del tipus pregunta/resposta, els serveis. Aquest

complementa als topics, i està format per dues parts: un missatge que és el que pregunta i un altre que és el que respon. En els topics, els nodes publiquen informació, que pot ser llegida per un subscriptor o no. En el cas dels serveis es fan peticions que necessiten una resposta, existint una dependència real entre els dos nodes a través dels serveis.

A diferència dels topics, un node només pot cridar un servei sota un nom particular. El nom d'un topic és assignat arbitràriament, segons com el vulgui anomenar l'usuari, mentre que els noms dels serveis són assignats directament per ROS.

- Bags: Els bags (de l'anglès bossa), són una manera de guardar els missatges publicats un topic i repetir-los si és necessari. Són un mitjà molt important per a guardar dades, com dades de sensors, que poden ser difícils de recollir, però són necessàries per desenvolupar i provar el funcionament del robot.

L'estructura del sistema de còmput de ROS és la següent: El master dóna un nom al sistema. Aquest guarda la informació de registre dels nodes als diferents topics i serveis. De la mateixa manera els nodes es comuniquen amb el master per indicar la seva informació de registre, i n'obtenen informació sobre els altres nodes registrats i estableixen les connexions apropiades entre ells. El master també comunicarà amb els nodes quan la informació de registre canviï, permetent així una creació dinàmica de connexions quan s'executen nodes nous.

D'altra banda, els nodes es connecten directament entre ells. El master només dóna la informació sobre els topics i serveis als quals es connecta cada node. El protocol de connexió entre nodes més habitual en ROS és TCPROS, basat en el socket estàndard TCP/IP.

El fet que els nodes es comuniquin entre ells a través de topics permet que el sistema estigui completament desacoblat, ja que un node se subscriu a un topic sense saber si algú hi està publicant o publica sense saber si ningú està subscrit. Això permet que els nodes puguin ser iniciats, reiniciats o destruïts sense necessitat d'introduir cap error en el sistema. Per aquest motiu, en l'arquitectura de ROS, els noms juguen un paper molt important: cada node, topic, servei i paràmetre té un nom propi que l'identifica.

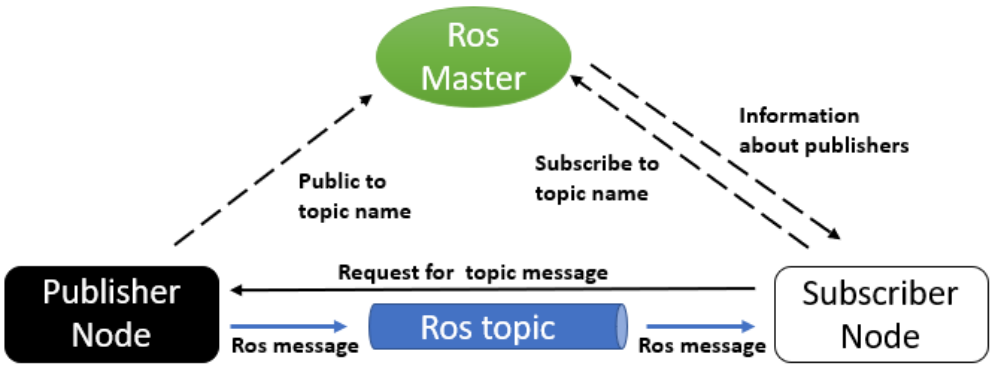


**Figura 9 Sistema de còmput de ROS**

## 4.4 Eines de ROS

ROS ofereix una gran quantitat d'eines pel desenvolupament de projectes de robòtica, però les més utilitzades són aquelles que permeten obtenir resultats gràfics, siguin mapes o grafs de les connexions entre els nodes. Les dues més utilitzades són Rviz i Rqt.

### 4.4.1 Rviz

De l'anglès ROS visualizer, es tracta un visualitzador 3D. Aquest és un paquet de ROS i permet visualitzar en temps real la posició del robot, les dades dels diferents sensors, els diferents mapes, etc. La majoria d'informació que es transmet a través dels missatges es pot visualitzar d'una manera o altra utilitzant Rviz.

A part de mostrar aquestes dades ens permet detectar errors entre els diferents nodes i paquets i així poder depurar els diferents errors.

A través de Rviz podem veure múltiples processos en una única pantalla, fet indispensable pel desenvolupament de robots. (Ku, 2012)

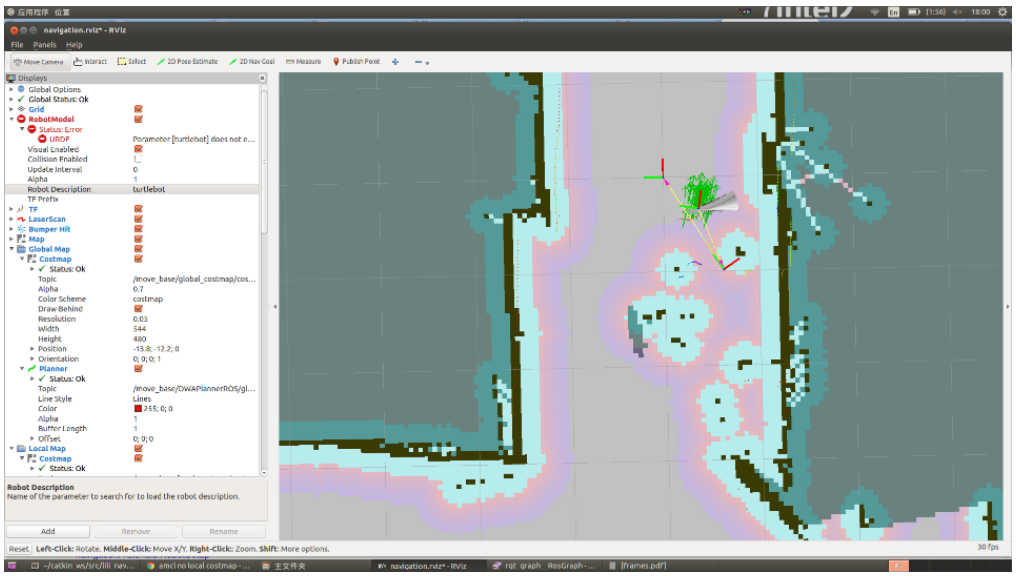


**Figura 10 Finestra de Rviz**

### 4.4.2 Rqt

Es tracta d'una eina de ROS que permet crear eines GUI[8] en forma de plugins. Permet controlar diferents aspectes de ROS mitjançant una interfície gràfica més còmoda. Alguns dels plugins de rqt permeten visualitzar de manera gràfica les connexions establertes entre els nodes i topics.

És una eina molt versàtil, ja que ofereix la possibilitat que l'usuari creï els seus propis plugins escrits en C++ o Python.

[8] De l'anglès graphical user interface, interfície gràfica d'usuari

Rqt està format per 3 metapaquets:

- Rqt: conté el nucli de l'eina.
- Rqt_common_pluguins: conté els plugins més comuns; com rqt_graph, que mostra un graf amb les connexions entre nodes i topics, i rqt_reconfigure, que permet reconfigurar diferents paràmetres mentre s'executa ros.
- Rqt_robot_plugins: conté eines per interactuar amb un robot durant el seu funcionament.

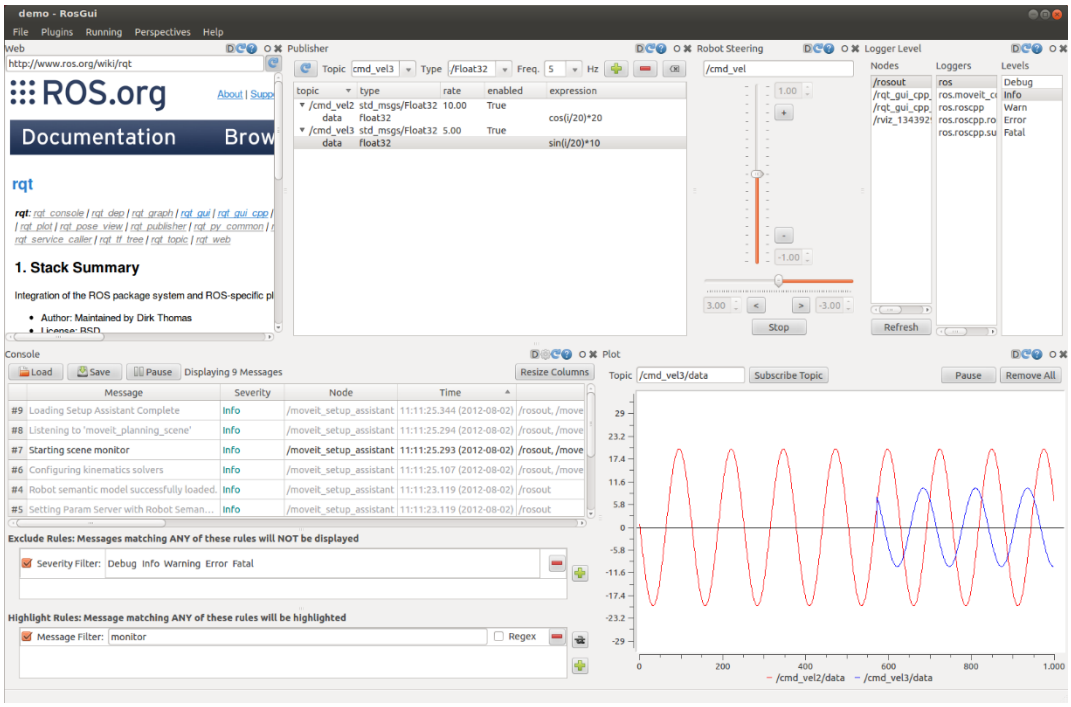


**Figura 11 Finestra de rqt**

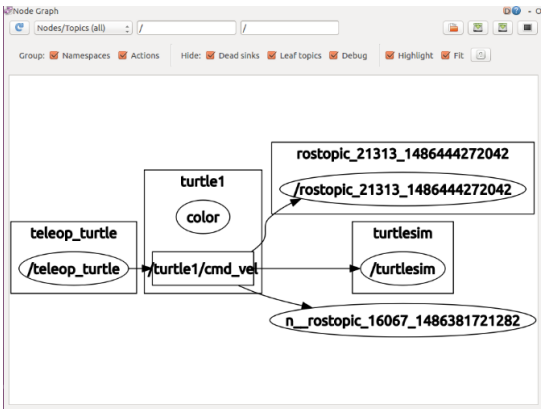


**Figura 12 Finestra de rqt_graph**

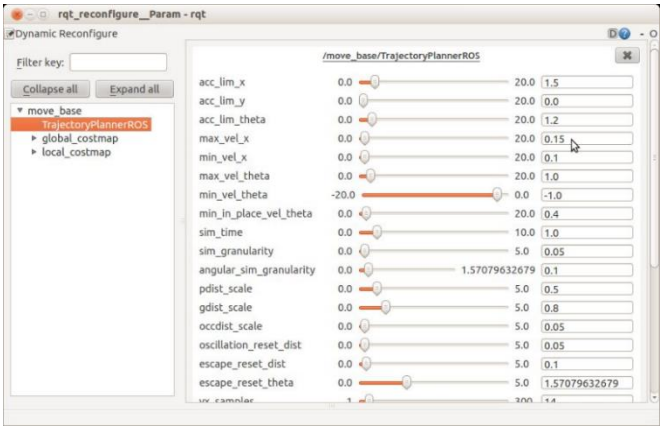


**Figura 13 Finestra de rqt_reconfigure**

En conclusió, ROS és un sistema l'essència del qual es pot resumir en aquests punts:

- Es tracta d'un sistema peer-to-peer. Aquest fet permet que els diferents processos dels nodes puguin ser executats en diferents màquines, formant així una xarxa de diferents màquines governades per un master.
- Multitud d'eines. ROS ofereix diferents eines que permeten desenvolupar sistemes robòtics més còmodament.
- Multillenguatge. ROS pot treballar en diferents llenguatges de programació de manera simultània: C++, Python, i LISP
- Modular. Gràcies a l'estructura per paquets de ROS es pot reutilitzar codi d'altres projectes, inclús donant-li un ús diferent pel qual va ser creat originalment.
- Codi lliure i obert. El codi font complet de ROS està disponible al públic. A part, es distribueix sota la llicència BSD[9] que permet tant el desenvolupament de projectes comercials com no comercials.

[9] BSD (Berkeley Software Distribution), tipus de llicència de software.

# 5. Construcció d'un AMR utilitzant ROS

Per dur a terme el procés del SLAM es pot fer de dues maneres: simulant un robot i un ambient per a aquest o utilitzant un robot i entorn materials, físics. Tot robot consta de dues parts diferenciades: el hardware i el software. Aquests s'expliquen tot seguit.

## 5.1 Hardware

El Hardware fa referència als components físics que conformen el robot, ja siguin electrònics, mecànics, etc.

En aquest apartat es descriuran i analitzaran els components que formen el hardware del robot realitzat per a aquest projecte.

### 5.1.1 Jetson Nano

Es tracta d'un ordinador (SoC[10]) de baix cost i mida reduïda. El Jetson Nano està format per un mòdul principal que és el que conté el processador i la memòria RAM. Aquest mòdul es connecta a través d'un port DDR4 SODIMM 0.5MM com el que utilitzen les memòries RAM DDR4 per a ordinadors portàtils a una placa de desenvolupament, que és on trobem els diferents reguladors de voltatge per alimentar el mòdul principal i les diferents sortides i entrades que ofereix la placa: els ports USB per connectar-hi diferents perifèrics, el port HDMI i Display Port per a la sortida d'imatge, el port d'ethernet RJ45, el port micro-USB i jack DC 5.5x2.1 per a l'alimentació, el port CSI per a connectar càmeres i els pins GPIO[11].

Perquè funcioni només cal posar-li una targeta microSD amb el sistema operatiu, en aquest cas una distro[12] de Linux basada en Ubuntu 18.04 i adaptada al xip Tegra del Jetson Nano, donar-li corrent i connectar-la a internet per fer la configuració inicial del software.

**Figura 14 Soc Jetson Nano**

[10] System on a chip: circuit integrat que conté tots els components i circuits necessaris en una mateixa placa.
[11] General-purpose input/output: pins genèrics d'entrada i sortida del xip, programables per l'usuari
[12] Una distro de Linux (Linux distribution) és un sistema operatiu basat en un kernel de Linux específic.

Les característiques clau del Jetson Nano són les següents:

- GPU: 128-core NVIDIA Maxwell™ architecture-based GPU
- CPU: Quad-core ARM® A57
- Video: 4K @ 30 fps (H.264/H.265) / 4K @ 60 fps (H.264/H.265) encode and decode
- Camera: MIPI CSI-2 DPHY lanes, 12x (Module) and 1x (Developer Kit)
- Memory: 4 GB 64-bit LPDDR4; 25.6 gigabytes/second
- Connectivity: Gigabit Ethernet
- OS Support: Linux for Tegra®
- Module Size: 70mm x 45mm
- Developer Kit Size: 100mm x 80mm

Al Jetson Nano hi van connectats una targeta wifi per USB, el Rplidar per usb, i la placa Pixhawk pel port UART[13] que es troba en els pins GPIO.

S'ha triat aquesta placa per la seva gran potència, no només de processament, sinó també gràfica. En un futur es vol afegit IA al robot, i molts Soc no solen incloure xips gràfics dedicats, i per a aplicacions robòtiques que facin servir intel·ligència artificial es necessita un xip gràfic dedicat (GPU).

### 5.1.2 Pixhawk

Pixhawk és una família de plaques basades en Arduino utilitzades en el món del ràdio control per a dotar als diferents vehicles teledirigits de capacitats de navegació autònoma assistida per GPS.

En aquest projecte s'ha decidit utilitzar una controladora Pixhawk, específicament la Pixhawk 3 Pro de Drotek, per a fer de controlador low-level (a baix nivell) dels motors. Quan es connecten motors a un Soc no se solen connectar els drivers dels motors directament als GPIO del Soc, sinó que s'utilitza un microcontrolador complementari connectat a aquest (una Pixhawk en aquest cas). Es fa així per treure càrrega computacional innecessària del processador principal, que en aquest robot fa els càlculs del SLAM i corre ROS.

Les característiques més importants de la Pixhawk 3 Pro són les següents:

- Size: 71 x 49 x 23 mm (with case).
- 6-14 PWM servo outputs.
- R/C inputs for CPPM, Spektrum / DSM and S.Bus.

[13] Universal asynchronous receiver-transmitter, protocol de comunicació bidireccional per a connectar dos dispositius i que es comuniquin entre ells.

- Two I2C ports
- Two external SPI ports.
- Two CAN Bus interfaces.
- Analog inputs for voltage / current of two batteries.
- IMU: ICM-20602, isolated from vibrations.

També s'ha decidit utilitzar aquesta placa pel fet que integra un IMU i que el software que corre (ardurover) incorpora un EKF que, mitjançant la informació dels encoders dels motors i de l'IMU, proporciona la informació de l'odometria al Jetson Nano per un port UART connectada al port marcat com TELEM1. El Jetson Nano és capaç de transmetre i rebre informació de la Pixhawk gràcies al paquet de ROS anomenat MAVROS, que utilitza el protocol MAVLink per comunicar-se amb la controladora. El Jetson Nano i la Pixhawk s'utilitzen en conjunt.

**Figura 15 Drotek Pixhawk 3 Pro**

És important destacar que s'ha desconnectat el mòdul GPS de la Pixhawk, ja que l'objectiu d'aquest projecte és guiar el robot únicament amb la informació visual del LIDAR, però amb ajuda de l'IMU de la Pixhawk per a l'odometria.

### 5.1.3 Motors i rodes

Com s'explicarà més endavant en l'apartat de disseny del robot, el que es buscava en el desenvolupament d'aquest projecte era crear una plataforma robòtica de mides reduïdes perquè així pogués maniobrar en espais reduïts (com seria una casa), i per reduir el pes i el cost del mateix.

Per aquests motius s'han escollit uns motors DC[14], ja que no es necessitava el parell que ofereixen els motors Brushless[15] i si s'aparellen amb uns encoders, ofereixen molt bona precisió.

[14] Motor DC: Motor de corrent continu.
[15] Motors Brushless: motors sense escombretes; fet que els diferencia, principalment, dels DC.

S'han utilitzat quatre motors Pololu 37Dx70L de 12V amb reductor 50:1 i encoder de 64 CPR[16](només en els motors de darrere), ja que tenen un parell més que suficient per moure el robot amb comoditat, són relativament petits i tenen un reductor encapsulat, que no està desprotegit.

Juntament amb aquests motors s'han utilitzat unes rodes Dagu Wild Thumper de 120x60mm

**Figura 17 Pololu 37Dx70L**

**Figura 17 Rodes Dagu Wild Thumper 120x60mm**

### 5.1.4 Driver vnh2sp30

Per connectar els motors a la Pixhawk es necessita un element de control anomenat preactuador. Els preactuadors serveixen per governar actuadors (motors en aquest cas) que requereixen molt a potència per a funcionar. El seu ús és indispensable perquè les sortides d'un microcontrolador solen proporcionar molt poca potència (normalment uns 40mA a 5V) i els motors necessiten més potència (5.5A a 12V, 12W).

En aquest cas s'ha optat per utilitzar 4 drivers vnh2sp30. El vnh2sp30 és un driver de motors fabricat per ST que consisteix bàsicament d'un pont-H. Un pont-H serveix per alimentar una càrrega (en aquest cas un motor) de manera que es pugui regular el sentit i intensitat del corrent que travessa aquesta càrrega. Activant els transistors oposats podem canviar el sentit del corrent que travessa la càrrega (Figura 18).

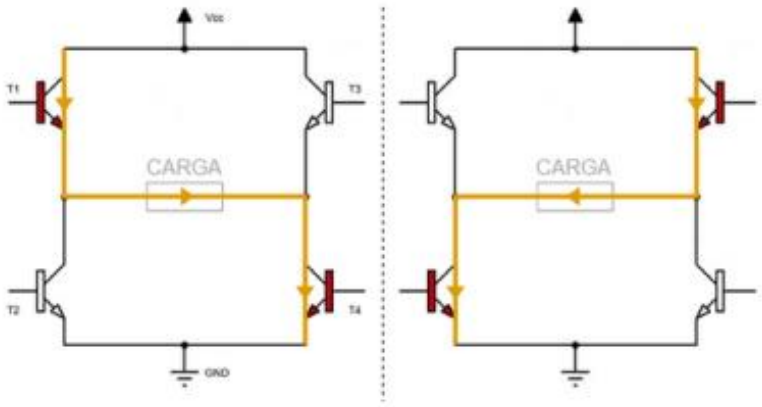


**Figura 18 Funcionament pont-H**

[16] CPR: Cycles per revolution, nombre de senyals elèctrics per revolució. Indica com és de precís un encoder.

Com podem veure en el datasheet (ST, 2017) d'aquest driver el driver incorpora diferents components electrònics per evitar que els quatre transistors es puguin activar alhora creant així un curtcircuit, i agrupa les connexions en pins d'activació per triar la direcció de gir (INA i INB) i en un pin per regular el voltatge que arriba als motors (PWM), aconseguint així poder triar la direcció i velocitat de gir del motor. Els pins INA i INB segueixen aquesta taula de la veritat (ST, 2017):

| $IN_A$ | $IN_B$ | $DIAG_A/EN_A$ | $DIAG_B/EN_B$ | $OUT_A$ | $OUT_B$ | Operating mode |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | H | H | Brake to $V_{CC}$ |
| | 0 | | | | L | Clockwise (CW) |
| 0 | 1 | | | L | H | Counterclockwise (CCW) |
| | 0 | | | | L | Brake to GND |

**Figura 19 Taula de la veritat vnh2sp30**

#### 5.1.3.1 Arduino

La controladora Pixhawk no és capaç d'interactuar directament amb aquests drivers, ja que les sortides PWM[17] que té funcionen de manera que hi ha una posició neutra, 1500us, que representa absència de senyal de sortida (motors parats) i segons s'augmenta aquest senyal positivament/negativament s'indica el sentit i velocitat de gir del motor.

Exemple: Un senyal de 2000us representa velocitat màxima en sentit positiu, 1000us representa velocitat màxima en sentit negatiu, 1750us representa la meitat de velocitat en sentit positiu, etc..

En canvi, el driver segueix la taula de la veritat vista anteriorment per controlar el sentit i un senyal PWM per controlar la velocitat. Requereix 3 pins d'entrada per governar un únic motor, però la Pixhawk només ofereix un pin de sortida per cada motor.

Per solucionar aquest problema, tot i no ser el més adequat, s'ha afegit al robot una placa Arduino Mega que s'encarrega de rebre els senyals de la Pixhawk de cada motor i transformar-los en altres que els drivers puguin entendre (el codi es pot robar annexat en un repositori de Github). No és el més convenient perquè s'està introduint un element més al robot ,que pot ser una altra font d'errors; petits retards entre el moment que l'algoritme del SLAM envia el senyal als motors fins que realment s'executa el moviment, etc.

Una solució més bona hauria estat connectar l'Arduino directament al Jetson Nano per així utilitzar el paquet ROSSERIAL_PYTHON per governar l'Arduino des de ROS. Aquesta idea es va descartar perquè complicava més la programació del robot, ja que els controladors PID i l'EKF per a controlar el robot de manera precisa que incorpora la Pixhawk estan molt ben optimitzats, i programar-los de zero per incorporar-los directament en l'Arduino portaria més temps del que es volia dedicar a la tasca que ROS pogués governar els motors. Governar els motors és només una petita part del que ha de fer el robot, i en aquest projecte es volia donar més èmfasi l'algoritme del SLAM.

[17] Pulse width modulation: Modulació per amplada de polsos, tipus de senyal elèctric.

### 5.1.5 RPLidar

En els apartats anteriors s'ha vist que per SLAM podem fer servir, principalment dos tipus de sensors per obtenir els landmarks: sensors làser i sensors de visió.

En aquest projecte s'ha decidit utilitzar un sensor làser, concretament LIDAR.

S'ha utilitzat LIDAR pel fet que proporciona informació en 2D. Els sensors LIDAR de dues dimensions que tenen molt bona resolució angular i longitudinal són molt cars, però el sensor adquirit, el RPLidar A1M8, té molt bones característiques pel seu preu reduït.

Les seves característiques més notables són les següents:

- Measurement Range (m): 0.15-12
- Angular Range (°): 0-360
- Measurement Resolution (mm): <0,5 within 1.5m, <1% of actual distance for whole measurement range.
- Angular Resolution (°): ≤1 when scanning at 5.5Hz
- Time for single measurement (ms): 0.5
- Measurement Frequency (Hz): minimum of 2000, typical of ≥4000 and maximum of 8000
- Scan Frequency (Hz): minimum of 5, typical of 5.5 and maximum of 10

El seu preu reduït de 70 € i aquestes característiques fan que sigui un molt bon sensor per començar a fer projectes robòtics que incorporin SLAM.

A part dels motius exposats anteriorment, també s'ha escollit aquest sensor pel fet que existeix un paquet de ROS anomenat rplidar, que ha desenvolupat el mateix fabricant del sensor (SLAMTEC).

Aquest paquet es connecta al RPLidar per USB i ens dóna les dades d'escaneig en un missatge de ROS amb format sensor_msgs/LaserScan Message (ROS wiki, 2019).

**Figura 20 Sensor RPLidar A1M8**

## 5.2 Disseny de l'AMR

Un cop escollit el hardware que es volia utilitzar per al robot es va procedir a dibuixar un xassís per l'AMR (Robotics, s.f.) amb el programa Autodesk Fusion 360 per a imprimir-lo en 3D.

En començar aquest projecte es volia utilitzar un hardware ben diferent, que ja es tenia: uns motors brushless 6374 de 192Kv, unes rodes més grans, etc. És per això que el primer disseny de robot (que es pot trobar en el repositori de Github del projecte) es va enfocar en aquests components. Es va descartar aquest disseny pel fet que constava d'un cos principal format per una xapa de ferro 3mm que es volia tallar amb làser, però un robot amb aquell disseny resultaria molt pesat, car i seria una mica massa gran per entorns petits com una casa.

Finalment, un cop amb el hardware final decidit (que és el que s'ha vist a l'apartat anterior) es va arribar al següent disseny:

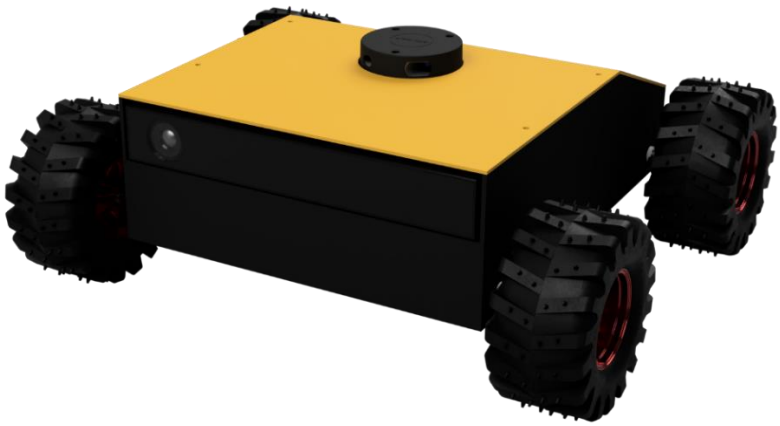

**Figura 21 Robot dissenyat. Amb tapa.**

**Figura 22 Robot dissenyat. Sense tapa.**

Com es pot observar s'ha dissenyat tenint en compte que tots els components sigui fàcilment accessibles i que es tracti d'un AMR de mides reduïdes perquè així pugui maniobrar en espais reduïts (com seria una casa), que sigui relativament barat, i que estigui construït íntegrament amb PLA imprès amb una impressora 3D, per abaratir el cos del xassís.

Com es pot veure en la figura 21, també s'ha dissenyat perquè més endavant es pogués incorporar un sensor Kinect de Xbox. Aquest fet es detallarà en l'apartat de treball futur.

Cal destacar que per facilitar la impressió del xassís, el cos principal es va dividir en 6 seccions i la tapa en 4 (consultar model 3D que es pot trobar en el repositori de Github).

El robot final, un cop construït es veu així:

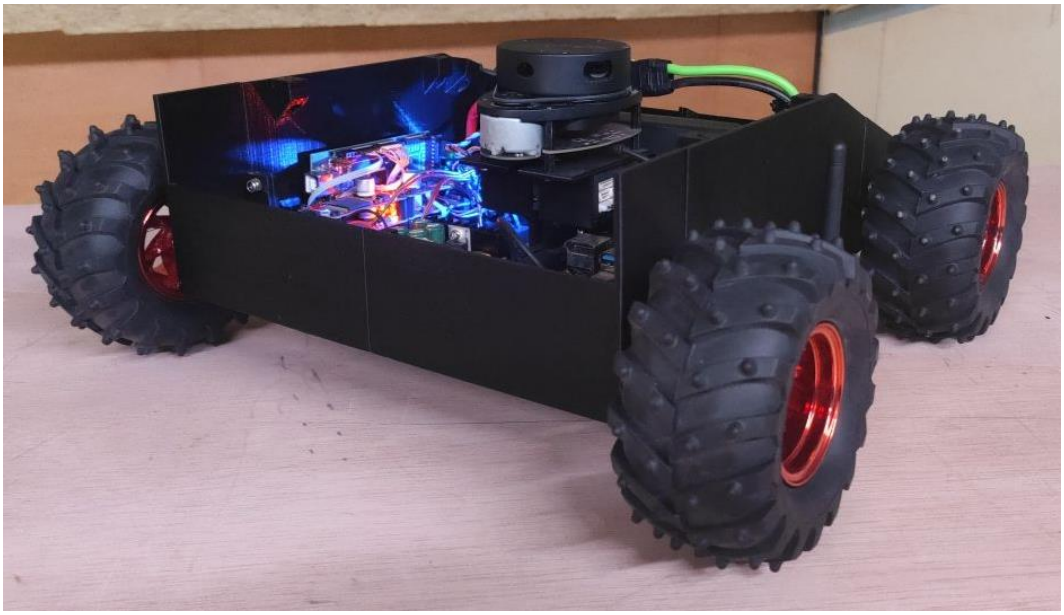

**Figura 23 Robot físic**

## 5.3 Software

El software fa referència al codi sobre el qual funciona el robot. En aquest apartat es descriuran i analitzaran les diferents eines de software utilitzades en el desenvolupament del robot d'aquest projecte, així com la manera com s'han instal·lat i configurat.

### 5.3.1 Sistema Operatiu Ubuntu 18.04

El sistema operatiu del Jetson Nano sobre el qual està desenvolupat tot aquest projecte és Linux. Com ja s'ha mencionat anteriorment la imatge de Linux que corre el Jetson Nano és una distro de Linux basada en Ubuntu 18.04 i adaptada al xip Tegra d'aquest.

Instal·lar el sistema operatiu és molt fàcil. Es baixa la imatge del sistema operatiu de la pàgina de desenvolupadors de NVIDIA i, seguint els passos que s'indiquen en aquesta, es formata la targeta microSD i s'hi escriu la imatge del sistema operatiu utilitzant el programa Balena Etcher. Un cop fet això simplement es connecta per RJ45 i ja es pot configurar remotament des de qualsevol ordinador.

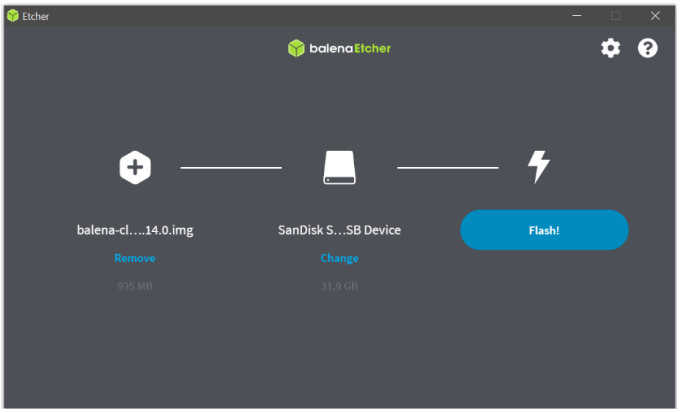


**Figura 24 Finestra de Balena Etcher**

### 5.3.2 ROS

Pel desenvolupament del robot s'ha decidit utilitzar ROS, pel fet que es tracta d'un projecte open source[18] amb una gran comunitat darrere.

#### 5.3.2.1 Instal·lació de ROS

Com que ROS està instal·lat sobre Ubuntu 18.04, hem de córrer la versió ROS Melodic, que és la distro de ROS recomanada per a aquesta versió d'Ubuntu. El procés d'instal·lació de ROS Melodic el trobem de manera detallada a la wiki de ROS (ROS wiki, 2019).

[18] Open source, de codi obert.

#### 5.3.2.1.1 Configuració dels repositoris d'Ubuntu

El primer pas per a la instal·lació de ROS és configurar els repositoris d'Ubuntu per permetre "restricted", "universe" i "multiverse". Això ho podem fer des de la finestra paràmetres de software d'Ubuntu.

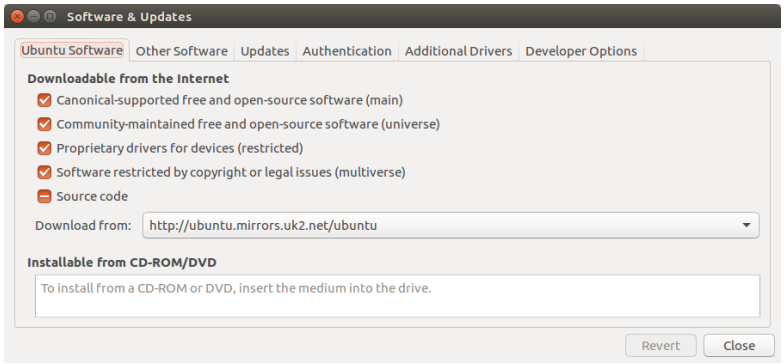


**Figura 25 Finestra paràmetres de software d'Ubuntu.**

#### 5.3.2.1.2 Configurar sources.list

Aquesta configuració ens permetrà que el Jetson Nano pugui acceptar software de packages.ros.org. Per configurar- fem córrer el següent comandament:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
 > /etc/apt/sources.list.d/ros-latest.list'
```

#### 5.3.2.1.3 Configurar les claus

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E
6BADE8868B172B4F42ED6FBAB17C654
```

#### 5.3.2.1.4 Instal·lació

Primer actualitzarem el paquet Debian:

```
$ sudo apt update
```

Després, com que el Jetson Nano estarà connectat a un altre ordinador que farà de master, però l'objectiu és que tot el software s'executi en el Jetson Nano, instal·larem ROS amb el següent comandament:

```
$ sudo apt install ros-melodic-desktop
```

D'aquesta manera s'instal·larà ROS, rqt, Rviz i la majoria de les llibreries necessàries.

#### 5.3.2.1.5 Configuració de l'entorn

És recomanable que les variables de l'entorn de ROS siguin afegides automàticament cada cop que s'obre un terminal nou. Per tant correrem aquest comandament:

```
$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrcsource ~/.bashrc
```

#### 5.3.2.1.6 Dependències per construir paquets

Fins a aquest punt s'ha instal·lat tot el necessari per executar paquets base de ROS. Per treballar amb ROS s'ha de crear un espai de treball. Per crear i controlar aquests espais existeixen moltes eines, i una de les més utilitzades és rosinstall. Rosinstall s'instal·la de la següent manera:

```
$ sudo apt install python-rosdep python-rosinstall python-rosinstall-generator pyth
on-wstool build-essential python-rosinstall-generator stow protobuf-compiler qt4-qm
ake qt4-dev-tools
```

Tot seguit instal·lem i inicialitzem rosdep:

```
$ sudo apt install python-rosdep

$ sudo rosdep init

$ rosdep update
```

Ara tot està llest per començar un projecte de ROS

### 5.3.2.2 Creació de l'espai de treball

Creem l'espai de treball (workspace) on s'instal·laran tots els paquets específics per al projecte, fitxers de configuració, etc.

Utilitzarem el comandament següent:

```
$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/

$ catkin build
```

Per acabar executem aquest comandament que configurarà el sistema Linux perquè ROS sàpiga que aquest serà l'espai de treball que ha d'utilitzar.

```
$ source devel/setup.bash
```

### 5.3.2.3 Configuració xarxa ROS

Anteriorment ja s'ha explicat que ROS pot executar-se en diferents màquines de manera simultània. Cal connectar les diferents màquines entre elles perquè es puguin comunicar, i hi ha d'haver un master que controli a les altres màquines.

En aquest projecte s'han utilitzat dues màquines:

- El Jetson Nano: Controla el LIDAR, els motors, etc.
- Ordinador amb Ubuntu: Fa de master, rep i mostra el mapa utilitzant Rviz, envia els missatges de punts als quals navegar i permet controlar el robot manualment amb el node teleop_twist_keyboard.

Les dues màquines es connecten a una mateixa xarxa wifi i es comuniquen mitjançant TCPROS. Per indicar quina màquina és el master i quina "l'esclau" fem el següent a cada màquina:

En el Jetson Nano

```
$ export ROS_IP="192.168.1.222"
$ export ROS_MASTER_URI="http://1.223:11311"
```

En el master:

```
$ export ROS_IP="192.168.1.223"
$ export ROS_MASTER_URI="http://1.223:11311"
```

192.168.1.222 i 192.168.223 són, respectivament, les adreces IP del Jetson Nano i de l'ordinador master.

## 5.4 Implementació del SLAM

Un cop tenim creat el workspace procedim a instal·lar els diferents paquets necessaris per a implementar SLAM en l'AMR.

El primer pas és instal·lar MAVROS, per així poder comunicar la Pixhawk amb ROS

```
$ cd ~/catkin_ws
$ catkin init
$ wstool init src
$ rosinstall_generator --rosdistro melodic mavlink | tee /tmp/mavros.rosinstall
$ rosinstall_generator --upstream mavros | tee -a /tmp/mavros.rosinstall
$ wstool merge -t src /tmp/mavros.rosinstall
$ wstool update -t src -j4
$ rosdep install --from-paths src --ignore-src -y
$ sudo bash ./src/mavros/mavros/scripts/install_geographiclib_datasets.sh
```

Tot seguit instal·larem el node que controlarà el RPLidar:

```
$ cd $HOME/catkin_ws/src
$ git clone https://github.com/Slamtec/rplidar_ros.git
```

Després instal·larem el node de Google Cartographer, que serà el que calcularà el SLAM. Google Cartographer és el sistema de SLAM que Google utilitza en Google Street View per mapejar interiors d'edificis, i és per aquest motiu, i pel fet que pot generar mapes en 2D i 3D, que s'ha decidit utilitzar aquest sistema en el robot del projecte. Abans de baixar-nos el node necessitem uns paquets de Linux:

```
$ sudo apt-get install python-wstool python-rosdep ninja-build
```

Tal com s'indica a la documentació de Cartographer, un cop s'han instal·lat aquests paquets de Linux reinicialitzem el workspace amb wstool, baixem els fitxers de Cartographer i obtenim el codi per a les dependències:

```
$ cd $HOME/catkin_ws

$ wstool merge -t src  https://raw.githubusercontent.com/cartographer-project/cartographer_ros/master/cartographer_ros.rosinstall

$ wstool update -t src
```

Instal·lem les dependències:

```
$ sudo rosdep init

$ rosdep update

$ rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -y
```

Cartographer utilitza la llibreria abseil-cpp, l'instal·lem:

```
$ src/cartographer/scripts/install_abseil.sh
```

Un cop s'ha baixat els fitxers de Cartographer I s'han instal·lat les seves dependències, clonarem el paquet de ROS robot_pose_publisher. Aquest paquet de ROS ens permet publicar la posició del robot en relació amb el mapa fent servir transformacions TF (wiki, ROS wiki, 2015). Ens baixem el paquet:

```
$ cd $HOME/catkin_ws/src

$ git clone https://github.com/GT-RAIL/robot_pose_publisher.git
```

Cal observar que tots els paquets s'instal·len en el directori  $HOME/catkin_ws/src.

Cal fer una petita modificació en el codi del node robot_pose_publisher, en el fitxer robot_pose_publisher.cpp.

```
$ cd $HOME/catkin_ws/src/robot_pose_publisher/src
$ gedit robot_pose_publisher.cpp
```

Obrim un editor de text i modifiquem la línia 40 perquè es vegi d'aquesta manera, canviant el "false" per un "true":

```
nh_priv.param<bool>("is_stamped", is_stamped, true);
```

Per iniciar el node de Google Cartographer hem de crear un fitxer .launch en el directori del node. El fitxer .launch és el que ROS "crida" per executar cada node.

```
$ cd $HOME/catkin_ws/src/cartographer_ros/cartographer_ros/launch
$ gedit cartographer.launch
```

Hi copiarem el text que es pot trobar en el repositori de Github del projecte en el mateix directori /catkin_ws/src/cartographer_ros/cartographer_ros/launch.

Per configurar els diferents paràmetres de Cartographer crearem un fitxer .lua el contingut del qual també es pot trobar en el repositori de Github annexat.

```
$ cd $HOME/catkin_ws/src/cartographer_ros/cartographer_ros/configuration_files
$ gedit cartographer.lua
```

Amb els paquets instal·lats fins ara ja es pot executar Cartographer, crear mapes amb ell i que el robot s'hi localitzi. Ara instal·larem el stack de navegació, necessari per poder enviar punts a l'AMR i que aquest calculi la ruta més bona, per arribar a l'objectiu evitant obstacles, a la velocitat corresponent, etc.

Primer instal·larem el stack de navegació de ROS:

```
$ sudo apt-get install ros-melodic-navigation
```

Una vegada instal·lat el paquet de navegació ens baixem el fitxer següent i l'extraiem (ardupilot, n.d.):

```
$ cd ~/catkin_ws/src
$ wget https://github.com/ArduPilot/companion/raw/master/Common/ROS/ap_navigation.zip
$ unzip ap_navigation.zip
```

Per acabar configurem MAVROS perquè enviï i rebi els diferents topics amb el nom necessari.

```
$ roscd mavros
$ cd launch
$ sudo gedit node.launch
```

Obrim un editor de text i modifiquem la línia 12 perquè es vegi d'aquesta manera, canviant el "v2.0" per "v1.0":

```
<arg name="fcu_protocol" default="v1.0" />
```

Afegirem les línies següents després de <rosparam command="load" file="$(arg config_yaml)" />:

```
<remap from="/mavros/vision_pose/pose" to="/robot_pose" />
<remap from="/mavros/setpoint_position/local" to="/move_base_simple/goal" />
<remap from="/mavros/setpoint_velocity/cmd_vel_unstamped" to="/cmd_vel" />
```

Obrim un editor de text en el document apm.launch:

```
$ sudo gedit apm.launch
```

Modifiquem la línia 10 perquè es vegi d'aquesta manera, canviant el "v2.0" per "v1.0":

```
<arg name="fcu_protocol" default="v1.0" />
```

També la línia 5 configurar el port UART al qual s'ha connectat la Pixhawk al Jetson Nano:

```
<arg name="fcu_url" default="//dev/ttyTHS1:115200" />
```

Un cop tenim tots els fitxers necessaris en el workspace muntarem els paquets i inicialitzarem de nou el workspace:

```
$ cd $HOME/catkin_ws
$ catkin build
$ source devel/setup.bash
```

Ja està tot llest per executar el SLAM.

## 5.5 Execució del SLAM

Un cop instal·lat tot el programari necessari ja el podem executar.

Primer de tot iniciem el nucli de ROS a la màquina master des del workspace catkin_ws:

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roscore
```

Un com executat roscore, executem els nodes necessaris en diferents terminals del Jetson Nano per connectar el RPLidar amb ROS, connectar la Pixhawk utilitzant MAVROS, córrer Google Cartographer i el stack de navegació:

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roslaunch rplidar_ros rplidar.launch
$ roslaunch mavros apm.launch fcu_url:=udp://:14855@
$ roslaunch cartographer_ros cartographer.launch
$ roslaunch ap_navigation ap_nav.launch
```

Per moure el robot, enviem una posició GPS falsa a la Pixhawk, amb l'ajuda d'un programa en Python, perquè es pensi que està en un lloc concret i canviem el mode a GUIDED, perquè així respongui als missatges /cmd_vel que ROS envia amb la velocitat lineal i angular a la qual s'ha de moure en cada moment. També cal "armar" la controladora perquè activi la sortida cap als motors:

```
$ python set_origin.py
$ rosrun mavros mavcmd -set_mode GUIDED
$ rosrun mavros mavsafety arm
```

Una vegada el robot respon al missatge /cmd_vel podem controlar-lo manualment amb el teclat per començar a construir el mapa utilitzant el node teleop_twist_keyboard. Executem el següent comandament en la màquina master.

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Per veure com es va construint el mapa podem utilitzar Rviz mentre anem conduint el robot

Un cop el robot ja s'ha desplaçat per l'entorn i l'ha mapejat podem utilitzar el botó "2D Nav Goal" de Rviz per fer que el robot navegui autònomament, i evitant obstacles, posició a la qual volem que el robot.

### 5.5.1 Configuració dels paràmetres de navegació

El stack de navegació de ROS està format per diferents paquets, entre els quals destaquen move_base, costmap_2d i dwa_local_planner.

Costmap_2d crea dos "costmaps", dos mapes que representen el cost (dificultat) de travessar diferents àrees d'un mapa (wiki, ROS wiki, 2018):

- Global_costmap: costmap que es fa servir per planejar la ruta a seguir de manera global, sense tenir en compte els obstacles momentanis.
- Local_costmap: costmap de menys resolució (de mida més reduïda), utilitzat per modificar la ruta programada utilitzant el global_costmap si s'observa algun canvi en l'entorn com podria ser una persona movent-se, etc.

Amb aquests costmaps el paquet dwa_planner pot planejar la ruta a seguir perquè el robot pugui navegar fins a un punt determinat.

Aquests dos paquets s'han de configurar per a aconseguir una navegació eficient, es pot fer de dues maneres:

1. Modificar els fitxers de configuració del directori `~/catkin_ws/src/ap_navigation/params` abans d'executar roscore i tots els altres nodes.
2. Modificar els valors un cop s'estan executant tots els nodes utilitzant rqt_reconfigure.

# 6. Resultats

Un cop configurat el robot s'ha conduït manualment per un entorn i els resultats obtinguts han estat els següents:

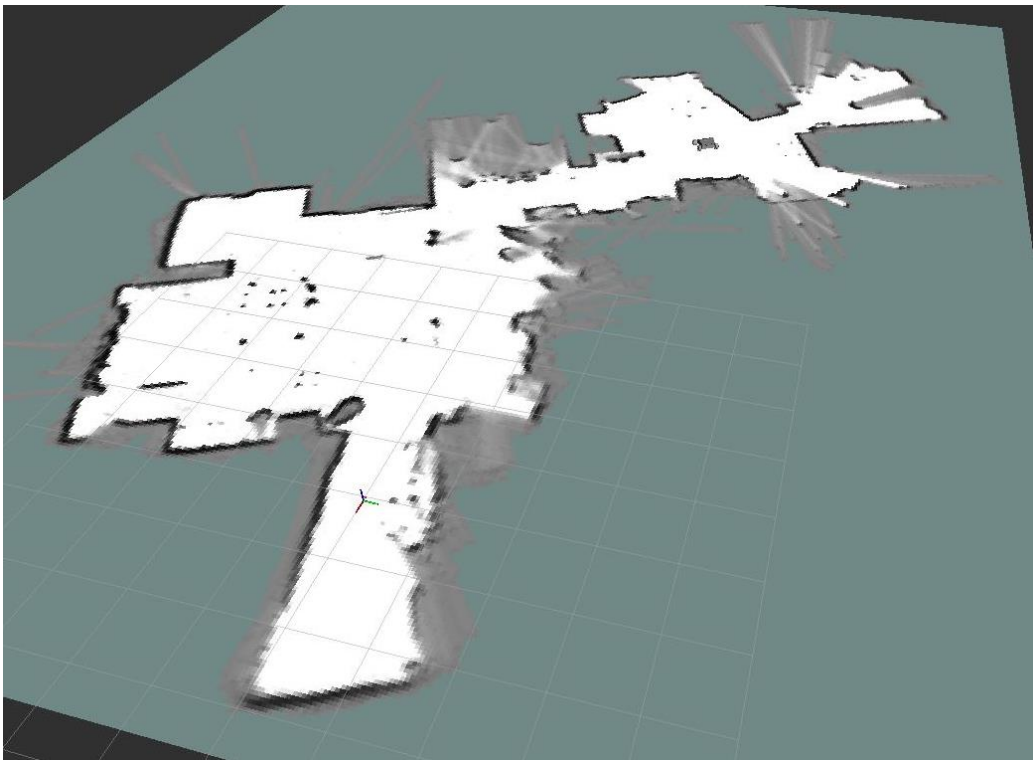

**Figura 26 Mapa generat per Cartographer vist en perspectiva**

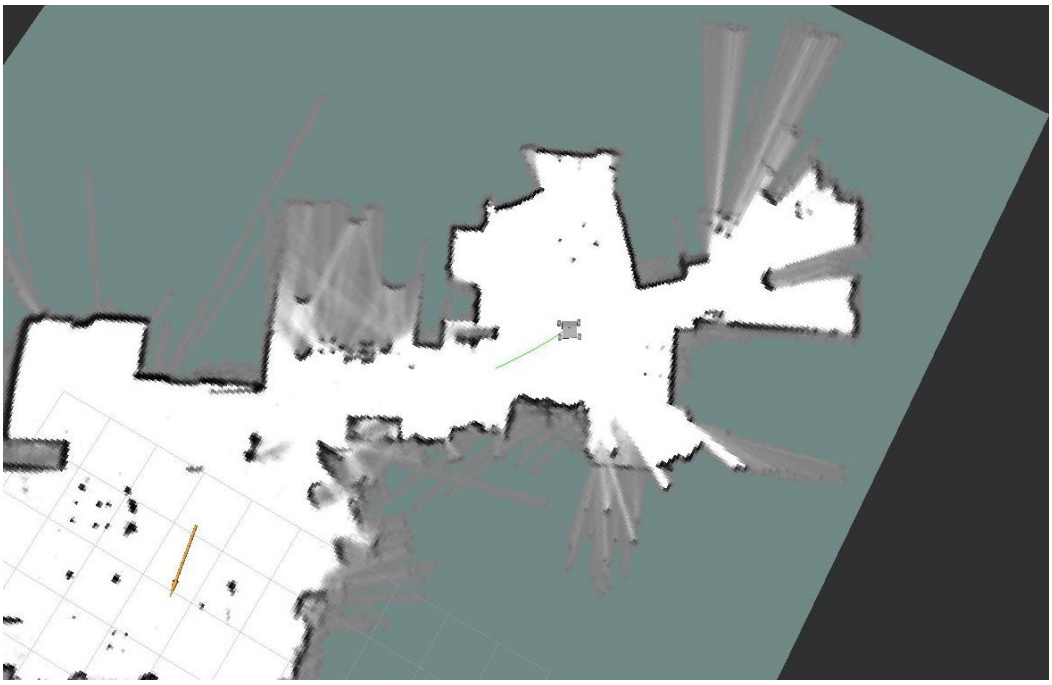

**Figura 27 Mapa on es pot veure en verd la ruta calculada per anar al punt marcat amb la fletxa groga**

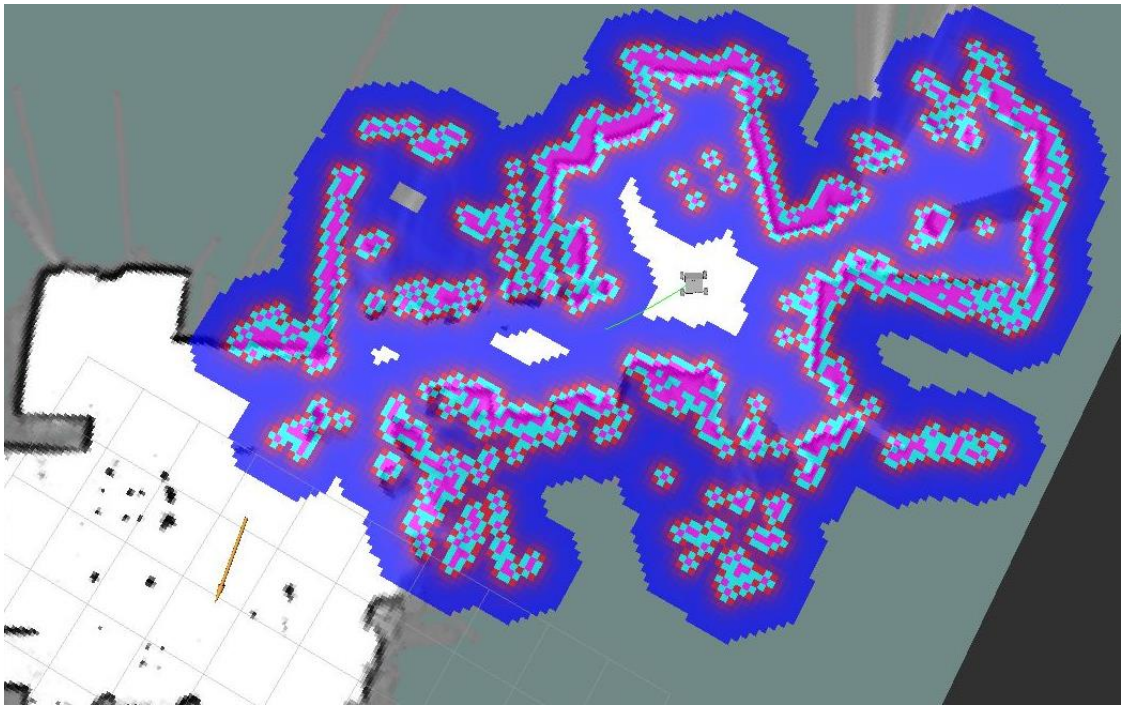

**Figura 28 Costmap global**

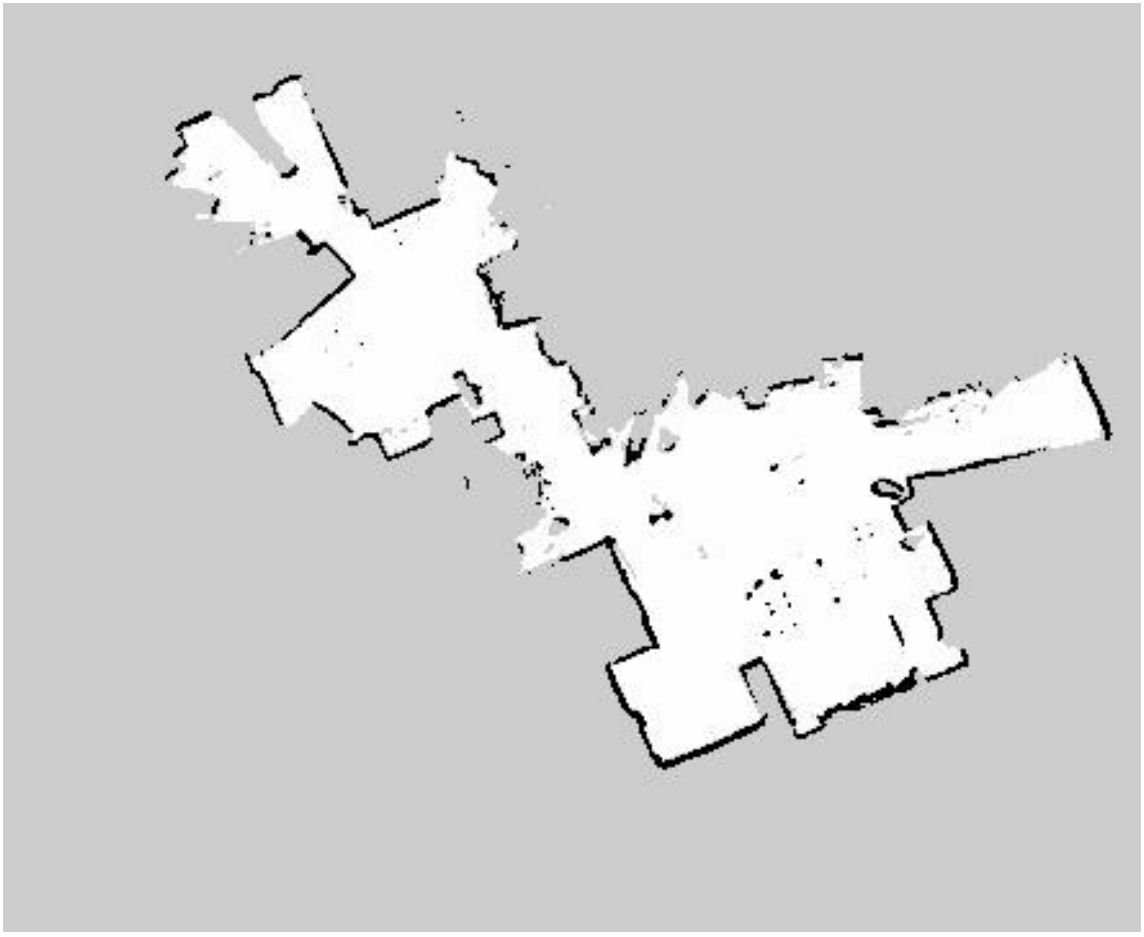

**Figura 29 Mapa final generat amb el paquet de ROS map_server**

Com es pot veure, el robot desenvolupat és capaç de generar mapes de manera bastant precisa, localitzar-se dins d'aquests i navegar a un punt qualsevol del mapa generat tenint en compte els diferents obstacles.

L'entorn per on s'ha mogut el robot per generar aquest mapa és el següent:

**Figura 30 Menjador**

**Figura 31 Passadís 1**

**Figura 32 Sala d'estar**

**Figura 33 Passadís 2**

Abans d'executar el procés del SLAM s'ha situat el robot en el passadís de la figura 31, és per això que veiem que en aquesta zona del mapa (consultar figura 26) apareix un eix de coordenades que representa la transformació (TF) anomenada "odom"(publicada pel node robot_pose_publisher), que és el punt agafat com a punt de referència pel procés del SLAM.

En fixar-se en el mapa generat, i comparant-lo amb l'entorn real, podem veure que no només es mapegen les parets, sinó que també queden registrats amb més detall petits objectes com les potes de les taules, cadires i escriptori, el cotxe de la figura 33, etc. Abans de començar aquest projecte no s'esperava obtenir resultats tan precisos.

En els trams on hi ha vidre, degut a les reflexions del làser en aquest, el resultat no és tan exacte.

Cal destacar que les figures 26, 27 i 28 són extretes directament de Rviz, mentre que la 29 s'ha aconseguit mitjançant el següent comandament:

```
$ rosrun map_server map_saver -f mapname
```

# 7. Conclusions

S'han assolit tots els objectius plantejats:

- Aprendre com funciona la tecnologia LIDAR.
- Aprendre com funciona la tecnologia SLAM.
- Aprendre com funciona ROS.
- Muntar un AMR utilitzant ROS.

S'ha dedicat molt de temps a aprendre en detall com funcionen LIDAR, SLAM i ROS, i el resultat final d'aquest treball reflecteix això, ja que per desenvolupar el robot ha fet falta saber com funcionen LIDAR, SLAM i ROS.

Tot i haver aconseguit els objectius inicials, si es tornés a fer aquest treball de nou, es faria d'una altra manera.

Com s'ha esmentat anteriorment, aquest projecte cercava construir un robot sense gastar molts diners, ja que es tenia un pressupost limitat. Per reduir els costos s'ha utilitzat material que ja es tenia, i tot i haver estat un bon repte es podria haver fet d'una manera més eficient.

Com ja es disposava de la controladora Pixhawk, que de per si ja és cara, es va decidir utilitzar-la juntament amb MAVROS per a controlar els motors. Però es va trobar el problema comentat en l'apartat 5.1.4, la Pixhawk no pot governar els drivers dels motors directament. Per a solucionar-ho es va utilitzar un Arduino connectat entre aquests perquè poguessin funcionar conjuntament.

Una solució més bona hauria estat connectar l'Arduino directament al Jetson Nano per així utilitzar el paquet ROSSERIAL_PYTHON per governar l'Arduino des de ROS. Aquesta idea es va descartar perquè complicava més la programació del robot, ja que els controladors PID i l'EKF per a controlar el robot de manera precisa que incorpora la Pixhawk estan molt ben optimitzats, i programar-los de zero per incorporar-los directament en l'Arduino portaria més temps del que es volia dedicar a la tasca que ROS pogués governar els motors. Governar els motors és només una petita part del que ha de fer el robot, i en aquest projecte es volia donar més èmfasi l'algoritme del SLAM.

# 8. Treball futur

Després de desenvolupar aquest robot i haver aconseguit implementar SLAM en ell es vol seguir avançant en aquest projecte.

Es vol arribar a desenvolupar un robot col·laboratiu que tingui una base mòbil que incorpori SLAM, la qual integri un braç robòtic de 6 graus de llibertat que tingui una mà completament articulada a l'extrem (de la mateixa manera que un braç humà). Aquest braç es podria desplaçar amunt i avall i constaria amb el suport d'una càmera de profunditat per a poder detectar objectes mitjançant IA i així conèixer la posició d'aquests. La informació de distància de la càmera s'utilitzaria per a poder agafar els objectes reconeguts, aplicant un model cinemàtic invers (Wikipedia, s.f.) al braç.

El robot es veuria similar a aquest, però incorporant el braç robòtic amb la mà:

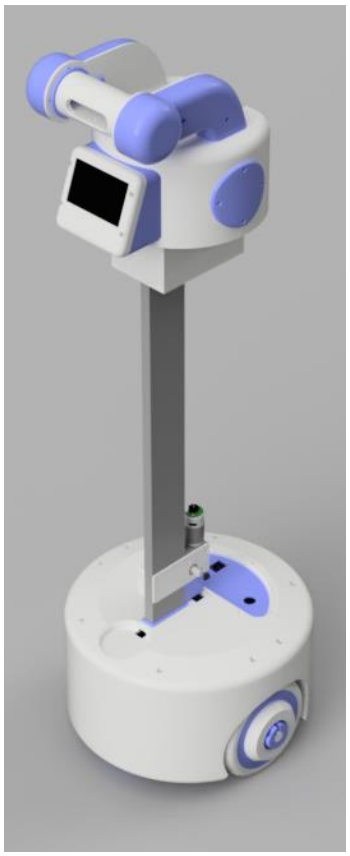

**Figura 30 Really useful robot de James Bruton**

# 9. Bibliografia

ardupilot. (sense data). *Ardupilot docs*. Recollit de https://ardupilot.org/dev/docs/ros-object-avoidance.html

Blas, S. R. (2004). *SLAM for Dummies*. Recollit de https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslam_blas_repo.pdf

Google. (sense data). *Cartographer*. Recollit de https://opensource.google/projects/cartographer

Ku, L. Y. (18 / 11 / 2012). *the Serious Computer Vision Blog*. Recollit de https://computervisionblog.wordpress.com/2012/11/18/rviz-a-good-reason-to-implement-a-vision-system-in-ros/

Macalester College. (5 / 2013). Recollit de An Analysis of Simultaneous Localization and: https://digitalcommons.macalester.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1030&context=mathcs_honors#:~:text=Spike%20Landmark%20Extraction%20uses%20extrema,locate%20areas%20of%20extreme%20change.

Robotics, W. (sense data). *Waypoint Robotics*. Recollit de AMR vs AGV: A Clear Choice for Flexible Material Handling: https://waypointrobotics.com/blog/amr-vs-agv/#:~:text=An%20Autonomous%20Mobile%20Robot%20or,for%20physical%20guides%20or%20markers.

ROS wiki. (2 / 8 / 2019). *ROS wiki*. Recollit de http://wiki.ros.org/rplidar

Smith R, S. M. (1 / 1986). *ResearchGate.* Recollit de Estimating Uncertain Spatial Relationships in Robotics: https://www.researchgate.net/publication/221405213_Estimating_Uncertain_Spatial_Relationships_in_Robotics

ST. (17 / 1 / 2017). Recollit de https://www.st.com/resource/en/datasheet/vnh3sp30-e.pdf

wiki, R. (14 / 12 / 2015). *ROS wiki*. Recollit de robot_pose_publisher: http://wiki.ros.org/robot_pose_publisher

wiki, R. (1 / 10 / 2018). *ROS wiki*. Recollit de costmap_2d: http://wiki.ros.org/costmap_2d

Wikipedia. (sense data). Recollit de Càmeres Time of Flight: https://ca.wikipedia.org/wiki/C%C3%A0meres_Time_of_Flight

Wikipedia. (sense data). Recollit de Extended Kalman Filter: https://en.wikipedia.org/wiki/Extended_Kalman_filter

Wikipedia. (sense data). *Least squares*. Recollit de https://en.wikipedia.org/wiki/Least_squares#:~:text=The%20method%20of%20least%20squares,results%20of%20every%20single%20equation.

wikipedia. (sense data). *Wikipedia.* Recollit de Random Walk: https://en.wikipedia.org/wiki/Random_walk

Wikipedia. (sense data). *Wikipedia*. Recollit de Inverse kinematics: https://en.wikipedia.org/wiki/Inverse_kinematics

# 10. Annexos

Pressupost:

| Component | €/unitat | Unitats | Total component (€) |
|---|---|---|---|
| Motor 37Dx70L amb encoder | 32.59 | 2 | 65.18 |
| Motor 37Dx54L sense encoder | 20.35 | 2 | 40.7 |
| 2 rodes 120x60mm | 12.2 | 2 | 24.4 |
| NVIDIA Jetson Nano | 90 | 1 | 90 |
| Drotek Pixhawk 3 Pro | 189 | 1 | 189 |
| RPLidar A1M8 | 70 | 1 | 70 |
| Arduino Mega | 13.99 | 1 | 13.99 |
| XL4015 Step Down 300W | 8.99 | 1 | 8.99 |
| Driver VNH2SP30 | 3.41 | 4 | 13.64 |
| | | Total | 515.9 |

Repositori Github del projecte on es pot trobar e codi i els models 3D dels robots disasenyats: https://github.com/TonMise/ROS_robot