Treball de final de grau

Estudi	Grau en Enginyeria Informàtica
Títol	Algorisme de cerca i seguiment de posidònia oceànica amb un robot submarí
Document	Memòria
Alumne	Alejandro Villanueva Rojo
Tutor **Departament** **Àrea**	Marc Carreras i Guillem Vallicrosa Arquitectura i tecnologia de computadors Arquitectura i tecnologia de computadors
Convocatòria (mes/any)	9 - 2020

1 Introducció, motivacions, propòsit i objectius del projecte

La posidònia oceànica és considerada un bon indicador biològic per a la salut de l'ecosistema marí a les costes del mediterrani. En els darrers 50 anys s'ha observat una disminució poblacional preocupant atribuïda a la contaminació, la pesca i els esports aquàtics, d'entre altres coses [1]. En col·laboració amb el projecte Europeu d'àmbit científic Eurofleets+, es pretén dissenyar un sistema que permeti mapejar dinàmicament la presència de posidònia al fons marí d'una zona determinada, amb l'objectiu de facilitar el posterior estudi a grups de recerca encarregats d'estudiar l'evolució poblacional de la posidònia oceànica i la seva salut al llarg del temps.

En aquest context, s'utilitzarà el robot submarí Girona500, desenvolupat a la UdG, per tal d' adquirir imatges del fons marí. Aquestes imatges seran processades per un mòdul d'intel·ligència artificial ja desenvolupat i instal·lat al vehicle. S'aprofitarà la sortida d'aquest mòdul per mapejar la presència de posidònia al fons marí d'una zona determinada.

L'objectiu doncs, serà aconseguir que el robot autònom submarí explori una zona determinada canviant el comportament de l'exploració segons la sortida rebuda del mòdul d'intel·ligència artificial, amb la finalitat de poder resseguir el contorn del bosc de posidònia i determinar així l'àrea total que abraça. Amb aquest enfocament s'espera reduir el temps d'exploració de la zona en comparació amb el mètode tradicional de lawnmower.

El desenvolupament del mòdul de detecció de posidònia amb intel·ligència artificial no forma part d'aquest treball de final de grau. Aquest mòdul serà desenvolupat de forma paral·lela per un altre equip col·laborador amb el projecte Eurofleets+. Per fer proves i mentre no estigui desenvolupat, es simularà la sortida d'aquest basant-se en una primera versió preliminar del mateix.

Es pretén realitzar la feina al Centre d'Investigació en Robòtica Submarina de la Universitat de Girona, on es podrà aprendre a fer anar el Girona500 a la piscina de proves, sortir a mar per agafar dades i adquirir experiència en el control del robot en un entorn real, i més endavant, tornar a mar a fer proves reals per demostrar el funcionament dels algorismes implementats.

Malauradament, la crisi de la COVID-19 ha endarrerit el desenvolupament del mòdul de detecció de posidònia i ha fet impossible fer les proves finals a mar. Tot i que es van poder realitzar les proves inicials al mar, adquirint dades reals en una zona habitada per la planta marina posidònia, finalment el projecte es desenvoluparà, testejarà i presentarà només en entorn simulat. Les dades experimentals obtingudes s'utilitzaran per fer l'entorn simulat més acurat.

2 Estudi de viabilitat

Els paràmetres que fan el projecte viable tecnològicament són:

- Robot autònom submarí Girona500.
- ROS, clúster de computació heterogeni utilitzat per controlar el Girona500.
- Stonefish, eina de simulació avançada per a la robòtica submarina.
- Existència de tècniques Deep Learning i xarxes neuronals.
- Portàtil personal amb potència suficient per executar simulacions.

Els costos de software per al projecte són nuls, ja que totes les llibreries i programes utilitzats per al disseny i la simulació del projecte son de codi obert i/o d'ús gratuït.

Els costos de personal per al projecte es mostren a les següents taules:

Reunions bisetmanals amb els tutors Març - Agost (1h)			
	Sou/hora	Hores	Cost
Catedràtic	20.39 €	12	244.73 €
Doctor Col·laborador	13.87 €	12	166.43 €
		Total	411.16 €

Sortides a mar previstes (2)			
	Sou/hora	Hores	Cost
Doctor Col·laborador	13.87 €	6 / sortida	166.43 €
Doctor Col·laborador	13.87 €	6 / sortida	166.43 €
		Total	332.85 €

Hores dedicades al desenvolupament de la feina			
	Sou/hora	Hores	Cost
Invest. En Formació	7.60 €	217.5	1.652.32 €

Total	2.396.33 €

Salaris extrets de les retribucions del personal docent investigador laboral amb dedicació exclusiva pel 2019 a la Universitat de Girona [2].

3 Metodologia

La metodologia comprèn la creació, organització i execució de les diferents tasques a realitzar per a considerar finalitzat el projecte.

S'ha escollit adaptar la metodologia SCRUM [3] per a una sola persona i seguint un desenvolupament incremental i iteratiu, IID [4]. SCRUM està dissenyat per a maximitzar l'eficiència i la col·laboració d'equip en el desenvolupament de productes complexes. Tot i així, és totalment vàlid i útil per a projectes d'una sola persona. SCRUM ajuda en l'organització, planificació i revisió del projecte per tal de tenir sempre uns objectius clars i realistes a assolir.

Aquesta metodologia consta de quatre elements principals:

- Product Backlog: Totes les tasques necessàries per a completar el projecte.
- Sprint Backlog: Tasques propostes per completar a curt termini.
- Daily Scrum: Reunions diàries amb l'equip que realitza les tasques.
- Sprint Review: Reunions a curt termini per revaluar els backlogs.

Aquesta adaptació comporta la eliminació de les reunions d'equip diàries, ja que només hi haurà una persona executant les tasques, i aquestes reunions només tenen interès a l'hora de repartir les tasques del Sprint per coordinar l'equip.

A l'inici de cada Sprint i abans de començar a executar les tasques contingudes al Sprint Backlog, ha calgut realitzar un estudi previ per comprovar quines llibreries, mètodes o teories existeixen que puguin ajudar o millorar en la seva execució.

Un cop el Sprint es considera acabat, es prepara una demo clara de la feina afegida al projecte per a la Sprint Review. Aquestes reviews o reunions s'han fet amb els tutors del projecte i han seguit un calendari bisetmanal a partir de la imposició de les mesures pel COVID-19. Abans d'aquestes mesures impostes, les reunions s'executaven un cop assolit l'Sprint proposat, ja que al treballar en el mateix entorn era fàcil trobar un moment per reunir-nos i parlar cada dos o tres dies.

Les Sprint Reviews serveixen per redefinir el Product Backlog i el Sprint Backlog en concordança amb el context canviant provocat pel COVID-19 i els resultats obtinguts al completar els Sprints. Es discuteix si la tasca ha estat completada amb èxit i s'estudien les dificultats trobades, si cal. Un cop avaluada la feina, s'estudien quins canvis cal fer al Product Backlog i quina tasca o tasques s'han de prioritzar. Aquestes tasques prioritzades son les que s'afegiran al Sprint Backlog.

Aquesta metodologia és cíclica i es fan tants Sprints com es considerin necessaris fins que el projecte es consideri acabat.

També s'ha utilitzat GIT en la plataforma de Bitbucket com a eina de control de versions amb la idea de no perdre el progrés entre Sprints. D'aquesta manera també es protegeix el projecte en cas de fallar el disc dur.

4 Planificació

Els diferents Sprints plantejats durant el projecte són els següents:

- **Instal·lació i estudi de l'arquitectura** de control software COLA2 pel Girona500 i el SPARUS II (3 dies).
- **Instal·lació i estudi del simulador** avançat per la robòtica submarina Stonefish (2 dies)
- **Assistir a Deepfield's First Thematic Workshop** on es van realitzar uns tallers d'entrenament amb horari intensiu de l'arquitectura dels AUV al CIRS i del simulador Stonefish (2 dies).
- **Primera immersió i control del Sparus II** per entendre millor el seu funcionament i entrenar per quan s'hagi d'anar a mar (1 dia).
- **Preparar la proposta del TFG** i discutir-la amb els tutors (2 dies).
- **Preparar escenari a Stonefish** senzill amb praderies de posidònia generades a partir de soroll tipus Perlin aleatori (1 setmana).
- **Preparació i sortida a mar** on es va entendre millor el funcionament dels experiments reals i es van agafar dades de GPS i càmera per a altres equips del CIRS, així com per entendre millor la distribució i aparició de la posidònia al fons marí (2 dies).
- **Programar algorisme d'escaneig lawnmower** simple on el AUV pentina una zona determinada d'esquerra a dreta en la seva totalitat (2 setmanes).
- **Programar la simulació del mòdul detector de posidònia** a partir de l'escenari preparat i una càmera a la simulació amb la finalitat de poder fer la resta de la feina sense dependre de tercers per a la completesa d'aquest mòdul (2 setmanes).
- **Programar algorisme d'escaneig per detecció** on el robot comença fent un lawnmower a la zona i quan detecti la presència de posidònia, canviarà el comportament per escanejar només l'interior de la praderia (2 setmanes).
- **Programar mapeig dinàmic** de la praderia de posidònia que s'anirà actualitzant a mesura que el robot vagi explorant (2 setmanes).
- **Crear escenari realista** de praderies de posidònia seguint la línia de platja amb diferents nivells de profunditat basant-se en escanejos de praderies reals (1 setmana).
- **Programar algorisme de detecció de contorns** de les praderies de posidònia per poder mapejar la vora exterior d'aquestes (1 setmana).
- **Programar controlador de seguiment de contorns** que permeti l'exploració completa de la zona (1 setmana).
- **Aplicar millores al detector de contorns** per tenir controlats casos especials (1 setmana).
- **Programar detecció de contorns basada en mapa local** escollint la millor direcció a partir d'un conjunt de cel·les properes candidates (2 setmanes).
- **Millores i proves finals** del controlador i execucions llargues on el vehicle faci tota la volta a la praderia de l'escenari (2 setmanes).

5 Marc de treball i conceptes previs

En aquest capítol s'explicaran els conceptes contextuals, teòrics i tècnics previs per fer entendre al lector el marc d'aquest projecte.

Primerament s'explicarà el context on apareix aquest projecte, és a dir, on neix i cobra sentit dins del mon científic. Seguidament s'exposaran els conceptes teòrics de la navegació autònoma dels robots i en concret dels robots submarins. Per últim es parlarà de les característiques i limitacions dels robots del CIRS, el Girona500 i l'SPARUS II.

5.1 Eurofleets+

Eurofleets+ [5] és un projecte d'iniciativa europea que pretén unir els esforços en investigació de 42 instituts marítims, universitats, fundacions i PIMEs de 24 països diferents d'arreu del món.

Figura 1. Logo del projecte Eurofleets+.

Finançat amb aproximadament 10 milions d'euros i coordinat pel Institut Marítim d'Irlanda, el projecte permet l'accés a una flota avançada de navilis de recerca amb la finalitat de satisfer les necessitats científiques dels seus membres.

Aquest projecte permet que els investigadors que així ho necessitin puguin estudiar multitud de mars i oceans de manera col·laborativa amb altres entitats membres. Posant al seu abast tant navilis amb equipament científic com vehicles autònoms submarins (AUVs) i vehicles d'operació remota (ROVs).

A Girona podem trobar 3 entitats col·laboradores amb el projecte:

- IQUA Robotics S.L.
- Coronis Computing S.L.
- Universitat de Girona

Aquestes 3 entitats, que ja treballen normalment de manera conjunta, presenten diversos projectes a Eurofleets+ amb la finalitat d'aconseguir recursos per prosseguir la investigació i recerca marítima.

Tant IQUA Robotics com la Universitat de Girona, posseeixen vehicles autònoms submarins (AUVs) que poden cedir al projecte per poder col·laborar amb altres investigacions d'altres entitats membres de Eurofleets+.

Aquest TFG forma part d'un projecte conjunt que engloba aquestes 3 entitats de Girona dins d'Eurofleets+ per aconseguir una exploració eficient de la posidònia al fons marí utilitzant robots autònoms submarins. L'objectiu principal d'aquest és aconseguir que el AUV detecti on hi ha posidònia i canviï el seu patró d'exploració en conseqüència amb la finalitat d'optimitzar el temps d'exploració.

5.2 La posidònia al Mar Mediterrani

La posidònia oceànica [6,7] és una planta aquàtica localitzada a les costes del Mar Mediterrani. De reproducció asexual, una mateixa planta pot viure 100 mil anys i tenir fins a 15km d'extensió, fent de la posidònia possiblement l'organisme més gran del món.

Figura 2. Imatge d'una praderia de posidònia oceànica.

Aquest organisme és de vital importància ecològica, proporcionant refugi a multitud d'espècies animals i vegetals i protegint la costa de l'erosió provocada per corrents marines i els temporals. La posidònia és la principal zona de cria per a moltes espècies de peixos del mediterrani, lligant la seva presència o absència a la quantitat de vida marina de la zona. La posidònia oceànica habita a profunditats des de 1 metre fins a 35 metres. Les seves fulles apareixen en grups de 6 o 7 i poden arribar als 1.5 metres d'alçada.

En conseqüència de les corrents marines, les fulles de posidònia pateixen petits trencaments que provoquen l'acumulament de biomassa al fons marí, amortiguant la intensitat de l'erosió del sòl marí. Aquestes petites fractures a les fulles es reparen sense problemes, augmentant així també la quantitat de diòxid de carboni que s'absorbeix de l'aigua. Això comporta, en part, que la posidònia sigui un dels principals organismes mitigants del canvi climàtic.

La importància de la presència de posidònia a les costes del Mar Mediterrani queda reflectida en la Directiva 92/43/CEE relativa a la conservació d'hàbitats naturals i de la fauna i flora silvestre [8].

La posidònia oceànica està sent amenaçada per l'augment de temperatures degut al canvi climàtic, la contaminació de les aigües i els fondejos d'embarcacions a les zones protegides

[9]. Fet que incrementa la necessitat d'estudiar la salut poblacional d'aquesta planta marina a les costes del Mediterrani, ja que si aquesta es veu amenaçada, també estarà en perill la resta de l'ecosistema de la zona.

5.3 El mapeig de posidònia

Actualment, la posidònia oceànica es mapeja a les costes del Mar Mediterrani utilitzant una combinació d'imatges per satèl·lit i imatges obtingudes d'escombrats tipus lawnmower amb embarcacions de recerca.

Aquestes dues tècniques són poc precises. Poden donar una idea general de la quantitat de posidònia de la zona, però són poc fiables a l'hora d'utilitzar-les per comparar l'extensió poblacional al llarg del temps d'una zona mínimament àmplia.

Les imatges per satèl·lit tenen la desavantatge d'una molt baixa resolució i profunditat. La posidònia pot arribar a viure a 35 metres sota el nivell del mar, mentre que les imatges per satèl·lit no aprofundeixen més que uns pocs metres en condicions ideals.

Figura 3. Imatge del satèl·lit SENTINEL-2 de la costa sud de l'illa d'Eivissa del 29 d'Octubre de 2020. Es pot observar la presència de posidònia als primers metres de costa [10].

L'escombrat amb un navili de recerca pot aportar molta més resolució que l'exploració per satèl·lit, però és molt més costós tant pel temps, material i personal invertits. També té el problema de que en zones de profunditat elevada la resolució que es pugui obtenir des del navili es veu molt reduïda per la distància a la càmera i la poca il·luminació. Les condicions del mar també poden afectar en gran mesura als resultats d'aquests tipus d'exploració.

És per això mateix, que per estudiar grans línies de costa no és viable utilitzar un escombrat complert amb navili. El que es fa és seleccionar diverses zones d'interès al llarg de la costa amb un àrea de com a màxim $25m^2$ i fer un estudi estadístic a partir de totes aquestes zones [11] al llarg dels anys.

Els vehicles autònoms submarins poden cobrir aquestes mancances que presenten els mètodes actuals. Es poden obtenir resultats molt més precisos, mapejant el fons marí a una alçada respecte el terra constant. Són molt més eficients energèticament i el fet de que utilitzin energia elèctrica ajuda a conservar el bioma marítim que s'està lluitant per conservar.

De fet, en els últims 15 anys, els AUVs s'han convertit en l'estàndard quan es tracta de mapejar una extensió de terra o paret subaquàtica [12,13,14]. Juntament amb l'evolució de les tecnologies d'identificació d'objectes marins per mitjà de xarxes neuronals [15], els AUV s'han tornat en el mètode idoni per a l'exploració i mapeig de praderies de posidònia oceànica.

5.4 La navegació dels vehicles autònoms submarins (AUV)

Sota l'aigua els mètodes tradicionals de geolocalització com el GPS o mètodes òptics de localització com el LIDAR o l'estereo-visió perden eficàcia o deixen de funcionar. Això comporta que les tecnologies de localització als AUV variïn dràsticament en comparació a les tecnologies utilitzades per vehicles de terra o aeris. Tot i així, els sensors que s'utilitzen als vehicles autònoms, sigui del tipus que sigui, es poden classificar en tres grups diferenciats:

- **Sensors de posicionament global:** Aquests sensors ens donen informació de la posició del vehicle respecte al planeta. Alguns exemples són: GPS, magnetòmetre, altímetre, etc.
- **Sensors de posicionament local:** Aquests sensors ens donen informació de la posició del vehicle respecte a elements propers al mateix. És a dir, ajuden a detectar l'entorn del vehicle. Alguns exemples són: Càmeres, RADAR, LIDAR, sonar, sensor doppler (velocitat i presència d'un objecte extern), etc.
- **Sensors odomètrics:** Aquests sensors ens donen informació del canvi posicional del vehicle respecte al temps. Solen tenir una latència molt més baixa que els sensors anteriors i això permet una predicció de trajectòries molt més fluida que si només s'utilitzessin els sensors de posicionament, generalment. Alguns exemples són: Acceleròmetre, sensor doppler (velocitat pròpia respecte un punt fixe), etc.

Per a una navegació correcte que permeti predir la posició del vehicle amb la major precisió possible, caldrà utilitzar un algorisme que combini la informació de tots els sensors de posicionament i odometria proporcionada pels diferents sensors a l'abast del vehicle. Trobar aquest algorisme no és pas una solució trivial. Existeixen diversos algorismes proposats, desenvolupats i millorats per la comunitat científica, cadascun amb els seus avantatges i inconvenients respecte als altres. Alguns exemples d'algorismes de navegació són: Bayes Filter, Particle Filter, Kalman Filter, Covariance Intersection, SLAM, GraphSLAM, etc. Serà doncs la situació específica a la que s'ha d'enfrontar cada vehicle la que determini l'algorisme a utilitzar, ja sigui un dels anteriorment mencionats, o un completament diferent.

D'aquest algorismes, s'explicarà una variació del filtre de Kalman, el filtre de Kalman estès [16], ja que com es veurà més endavant, és l'algorisme que els AUVs utilitzats en aquest treball porten incorporat.

5.5 Extended Kalman Filter

EKF o filtre de Kalman estès és un algorisme molt utilitzat per resoldre el problema de la localització als vehicles autònoms. Aquest algorisme permet estimar la posició i orientació del vehicle respecte al seu entorn, així com també la posició dels elements que l'envolten i formen part de l'entorn, captats pels sensors que el vehicle porti incorporats. Com a diferència principal amb el filtre de Kalman clàssic cal destacar que aquest serveix per a sistemes lineals mentre que l'EKF permet resoldre el problema de la localització en sistemes no-lineals. Això s'aconsegueix linealitzant el problema localment amb una expansió de Taylor de primer ordre a les equacions del model no lineal [17].

La idea darrera l'algorisme és la de reduir la incertesa dels sensors combinant prediccions de l'espai d'estats amb les mesures dels sensors. D'aquesta manera i mentre els sensors presentin un soroll gaussià de mitjana zero amb variància coneguda i prou petita, el sistema tendirà a reduir l'error i predirà correctament l'espai d'estats.

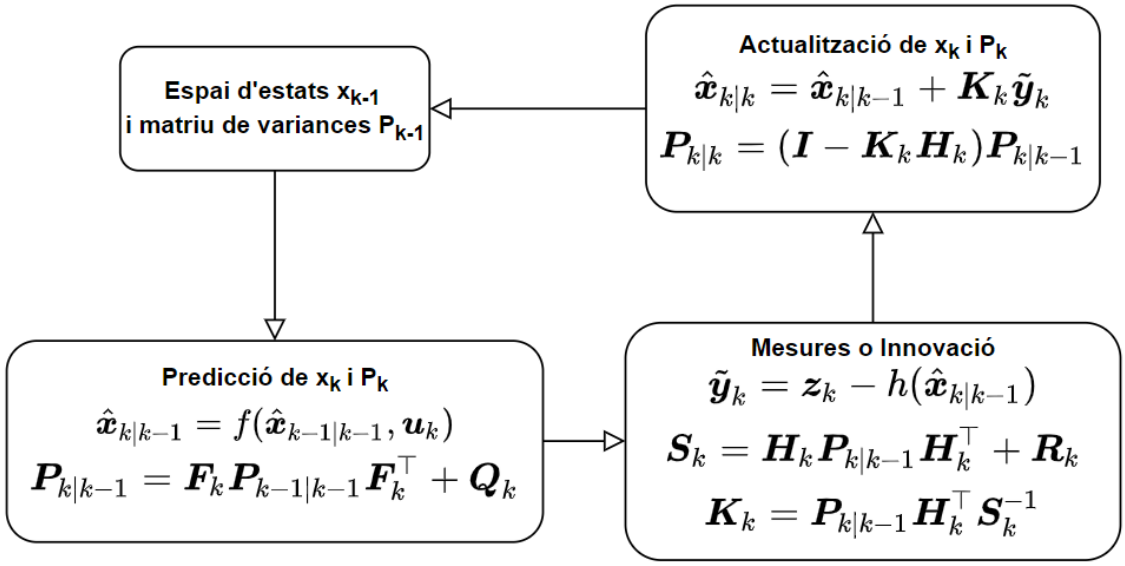

Figura 4. Esquema de l'algorisme de EKF. Fórmules extretes de Wikipedia.

Tot i que EKF ens permet fer convergir l'estimació de l'espai d'estats, al tenir una aproximació per mantenir la linealitat del model, si l'estimació inicial és incorrecte o el model no és prou proper a la realitat, l'estimació divergirà i no serà prou òptima. És per això que normalment es combinen diferents tecnologies sensorials amb l'objectiu de reduir al mínim la variància del sistema, com per exemple combinant sensors odomètrics amb GPS o sensors dels actuadors d'un braç amb càmeres externes.

En el cas dels AUV, el GPS no rep senyal sota l'aigua. En aquest estat la certesa de la posició del vehicle no convergeix, així que es combinen les mesures de la unitat de mesura de inèrcia, el sensor de velocitat Doppler i el sensor de pressió al filtre de Kalman amb l'objectiu de reduir la velocitat en la que la incertesa divergeix en funció del temps.

6 Requisits del sistema

Tot seguit, es descriuen els requisits funcionals i no funcionals del sistema.

6.1 Requisits funcionals

El sistema desenvolupat ha de permetre:

- Detectar existència de posidònia sota el vehicle dins la zona delimitada.
- Aïllar el contorn de la posidònia.
- Determinar una direcció a seguir a partir del contorn trobat.
- Decidir autònomament cap a on moure el AUV per seguir el contorn trobat.
- Crear un mapa dinàmic que s'anirà actualitzant a mesura que l'exploració avança.
- Aquest mapa podrà ser vist per l'operari en cas de disposar de connexió directa amb el vehicle.

6.2 Requisits no funcionals

Els requisits no funcionals dependran de la manera en que s'utilitzi el sistema desenvolupat. És a dir, en condicions reals o simulades, caldran hardware i software diferents.

6.2.1 Requisits de hardware en simulació

- CPU de 4 nuclis amb 2GHz de freqüència o més
- 8 GB de memòria RAM o més
- GPU compatible amb OpenGL 4.3 amb al menys 1GB de VRAM
- Memòria lliure suficient per guardar els les dades recollides (min 60GB)

6.2.2 Requisits de software en simulació

- Ubuntu Linux 18.04
- Python 2.7
- ROS Melodic
- Stonefish, git repo: https://github.com/patrykcieslak/stonefish
- Stonefish_ros, git repo: https://github.com/patrykcieslak/stonefish_ros
- Arquitectura Cola2, git repo: https://bitbucket.org/%7B3bd2a9e6-d535-402d-8113-e038f320e61a%7D/

6.2.3 Requisits de hardware en condicions reals

- Vehicle Girona500 amb:
  - Càmera enfocada al fons marí
  - Mòdul *deep learning* amb Nvidia GPU 2000 Series o més moderna
  - Font d'il·luminació (opcional, dependrà de la profunditat de treball i la visibilitat)

- Embarcació amb grua per manipular l'AUV
- Portàtil amb connexió amb el AUV per configurar i controlar la missió

6.2.4 Requisits de software en condicions reals

Requisits del vehicle:

- Arquitectura Cola2
- ROS Melodic
- Software pel mòdul de *deep learning* amb sortida matricial binària

Requisits al portàtil:

- Linux Ubuntu 18.04
- Python 2.7
- ROS Melodic
- Interfície visual IQUAVIEW (opcional, per situar al vehicle al mapa de la zona)

7 Estudis i decisions

Abans de desenvolupar el sistema d'aquest projecte, és essencial entendre què són exactament l'Sparus II i el Girona500, la seva lògica interna, arquitectura de control i les eines de desenvolupament disponibles. Tal i com s'ha exposat a l'apartat de *Planificació*, aquest procés d'aprenentatge i estudi va durar 2 setmanes. Tot i així, a les pràctiques curriculars i extracurriculars efectuades durant el 3r any de carrera i part del 4rt any es va tenir l'oportunitat d'aprendre alguns dels conceptes. S'ha aprofitat doncs aquest coneixement prèviament adquirit per entendre i aprofundir encara més en l'arquitectura i funcionament del vehicle. En especial, la *Deepfield Thematic Workshop* [18] va aportar en molt poc temps, una quantitat de coneixement i informació sobre el funcionament dels AUV del El Centre d'Investigació en Robòtica Submarina de Girona (CIRS) de gran valor.

7.1 AUV al CIRS

El Centre d'Investigació en Robòtica Submarina de Girona porta més de 20 anys dissenyant, desenvolupant i construint una flota de vehicles autònoms submarins. Els models més moderns a dia d'avui són el Girona500, creat a 2011 i l'Sparus II, creat a 2013. A l'inici del projecte encara no s'havia decidit si per al projecte proposat per Eurofleets+ s'utilitzaria l'Sparus II o el Girona500, així que ambdós van ser estudiats. Això no ha comportat cap canvi important a l'hora de programar la part software del treball, ja que els dos AUV utilitzen la mateixa arquitectura de software.

A continuació, es parlarà dels dos models de vehicle autònom submarí, les seves capacitats i funcionalitats principals.

7.1.1 Sparus II

La primera versió d'aquest robot, Sparus, va ser creada al 2010 i va aconseguir guanyar la competició d'AUVs d'aquell any a Itàlia, European Student AUV by CMRE. Una nova i millorada versió, Sparus II, va ser creada al CIRS al 2013 i comercialitzada per IQUA Robotics a partir del 2016 [19].

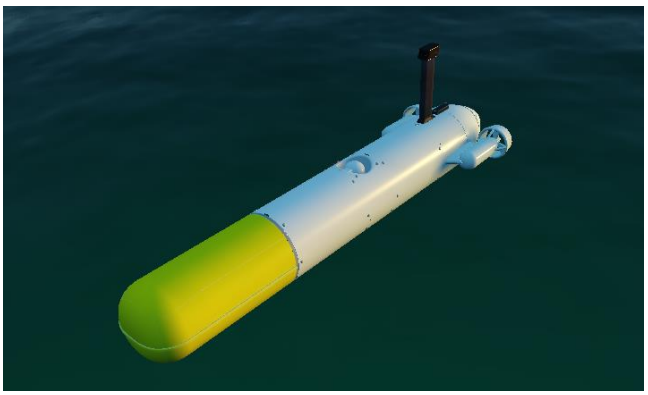

Figura 5. Imatge de l'Sparus II extreta del simulador gràfic Stonefish.

L'Sparus II està pensat per ser un AUV petit, lleuger i de perfil baix. Amb un pes de només 52Kg i una longitud de 1.6 metres. Pot ser transportat i operat per una sola persona. Porta a la punta (secció groga del davant) un compartiment amb un volum de 8 litres dissenyat per

col·locar instruments i sensors específics per a cada missió, de tal manera que es pot personalitzar segons la necessitat de l'usuari en cada moment.

El robot disposa d'una autonomia d'aproximadament 8 hores i pot arribar a operar fins a 200 metres de profunditat. Porta 3 motors, un horitzontal a cada costat i un muntat al centre de gravetat en vertical. Els 2 motors dels costats aporten 2 graus de llibertat (*surge* i *yaw*) i el motor vertical aporta un altre grau de llibertat més (*heave*). Combinant els 3 motors es pot aconseguir un quart grau de llibertat (*pitch*).

A la part superior del casc porta incorporada una antena que li proporciona Wifi, GPS i un indicador LED per augmentar la seva visibilitat i per indicar la quantitat de bateria que li queda al vehicle aproximadament.

D'entre els dos AUV que a dia d'avui fabriquen i venen per encàrrec a IQUA Robotics, l'Sparus II és el més barat dels dos, anunciat com un AUV "low cost".

7.1.2 Girona500

El Girona500, creat l'any 2011, és un vehicle molt més versàtil que l'Sparus II en quant a varietat de projectes que pot realitzar. A diferència de l'anterior, Girona500 pot mantenir una posició estàtica sota l'aigua gràcies als motors laterals que li proporcionen moviment horitzontal (*sway*), sumat als motors posteriors i verticals que aporten moviment frontal (*surge*) i moviment vertical (*heave*) [20].

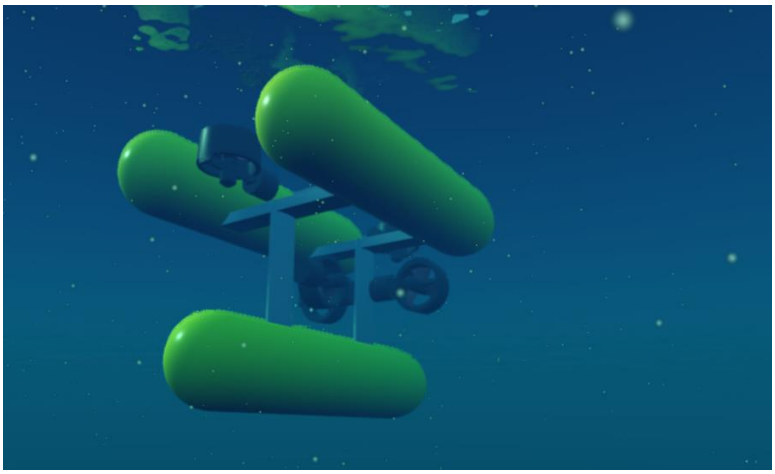

Figura 6. Imatge del Girona500 extreta del simulador gràfic Stonefish.

Aquest robot disposa de 3 compartiments en forma de torpede específicament dissenyats per a reduir la fricció amb l'aigua i augmentar l'eficiència energètica. Els 2 compartiments superiors estan pensats per ser omplerts d'un material lleuger que augmenta la flotabilitat del vehicle, mentre que el compartiment inferior està pensat per portar tot el pes de les bateries i equipament extra per a la tasca específica que calgui fer. Aquesta distribució li dóna molta estabilitat en *roll* i *pitch*, el que comporta que aquest vehicle sigui idoni per a missions d'inspecció o mapeig del fons oceànic, ja que l'estabilitat a la imatge serà molt millor.

Amb una longitud de 1.5 metres, una amplada d'1 metre i una alçada d'1 metre. Aquest AUV es podria considerar de mida mitjana en relació a altres vehicles. Sense equipament extra, té un pes de 140kg, així que tant pel transport com per desplegar el robot a l'aigua caldrà fer servir alguna mena de grua. Degut a la seva mida i tot i tenir més capacitat a les bateries que l'Sparus II, el Girona500 té de base una autonomia de més de 6h.

Igual que l'Sparus II, el Girona500 porta una antena amb Wifi, GPS i LEDS indicadors. Però a diferència del primer, els Girona500 que es fabriquen i venen a IQUA Robotics porten incorporats tots els sensors essencials per a una correcta navegació. Aquests sensors són la unitat de mesura d'inèrcia per mesurar acceleracions direccionals i rotacionals, el registre de velocitat Doppler per mesurar velocitats relatives i el sensor de pressió per determinar la profunditat actual.

Aquest AUV és l'idoni per dur a terme els experiments que es plantegen en aquest projecte degut a la seva gran estabilitat per mantenir la imatge fixa, la seva millor maniobrabilitat, que facilitarà el seguiment del contorn irregular típic de les praderies de posidònia, i un compartiment de càrrega superior, de 35 litres, suficient per acollir el hardware necessari pel mòdul *deep learning* que caldria incorporar.

7.2 ROS

Robot Operating System (ROS) és un conjunt de llibreries i eines dissenyades per al control i disseny de robots, tant mòbils com fixes [21]. Llevat que en el nom apareixen les paraules "sistema operatiu", ROS no és exactament un sistema operatiu, tot i que sí que defineix i governa de quina manera el robot processarà les seves dades.

ROS aporta un sistema heterogeni de nodes de computació amb un gran nivell d'abstracció, allunyant-se de la programació orientada a objectes clàssica, però amb la mateixa idea en ment, separar cada node segons la tasca que realitza. Els nodes no segueixen una jerarquia estricta com succeeix a la programació orientada a objectes. Tots ells són independents i coexisteixen a l'hora. Aquesta implementació de nuclis de computació separats és complementada amb un sistema de subscripció i publicació de missatges que segueix molt de prop el patró de disseny *publisher-subscriber*.

Els nodes de ROS poden ser tant publicadors com subscriptors de diferents tòpics. Aquests tòpics tenen nom únic i són els canals per on circulen els missatges. En un sistema amb molts nodes es pot acabar formant un gran graf amb connexions de complexitat elevada. Tot i així, ROS s'encarrega de gestionar-ho tot i el programador només ha de definir a quins tòpics es subscriu el node i quins missatges publica. També cal destacar que més d'un node pot publicar al mateix tòpic, sense cap problema. Aquests missatges han de tenir una estructura prèviament definida i tot node que utilitzi el mateix missatge, ja sigui per publicar-lo o subscriure's l'haurà de tractar de la mateixa manera. Per exemple, el missatge tipus *geometry_msg/Point.msg* defineix un punt 3D i la seva estructura de dades compren 3 variables; *x*, *y* i *z*, de tipus punt flotant de precisió doble.

Un dels desavantatges de ROS és que no és un sistema operatiu de temps real (RTOS) [22] com el que es pot trobar a màquines o braços industrials com els Stäubli de la Universitat de

Girona. Això implica que la latència de les dades transmeses entre nodes poden patir petits retards de l'ordre de mil·lisegons, típics en la gran majoria de sistemes operatius. Aquests retards només suposen un problema per activitats que requereixen molta precisió i un nivell molt petit d'error. En el cas dels vehicles autònoms això no comporta cap problema important. Tot i així, l'arquitectura de ROS original està patint grans canvis de cara a ROS2, que suportarà RTOS [23].

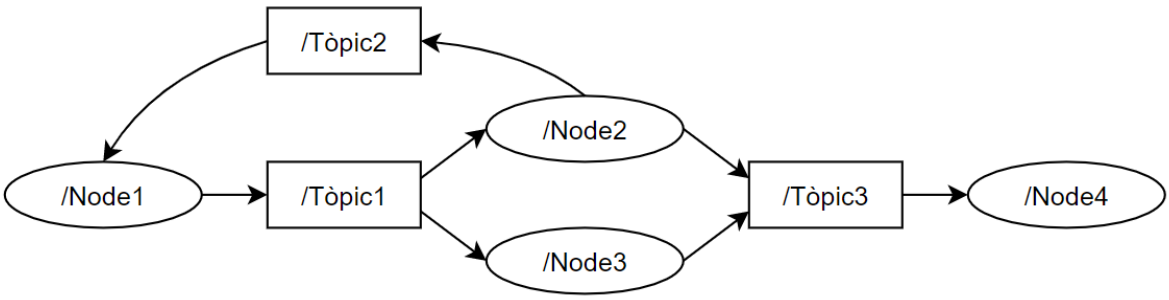

Figura 7. Exemple de graf on es pot observar el tipus d'interconnectivitat que poden assolir els diferents nodes (el·lipses) utilitzant missatges transportats pels tòpics (rectangles). Les fletxes sortints impliquen missatges publicats mentre que les entrants impliquen missatges entrants degut a les subscripcions als tòpics.

ROS incorpora eines molt útils per a analitzar i seguir molt de prop els nodes en execució i els seus tòpics. D'entre elles destaca per sobre l'eina anomenada RViz. Aquesta eina és un visualitzador 3D que permet representar el robot i el que està detectant en l'espai. RViz facilita molt la depuració dels processos dels robots, ja que permet entendre en temps real la informació que s'està transmetent dins del sistema. Amb aquesta eina es pot visualitzar alhora una càmera del robot, el núvol de punts d'un LIDAR, el mapa que està generant el robot i el mateix robot situat a l'espai 3D.

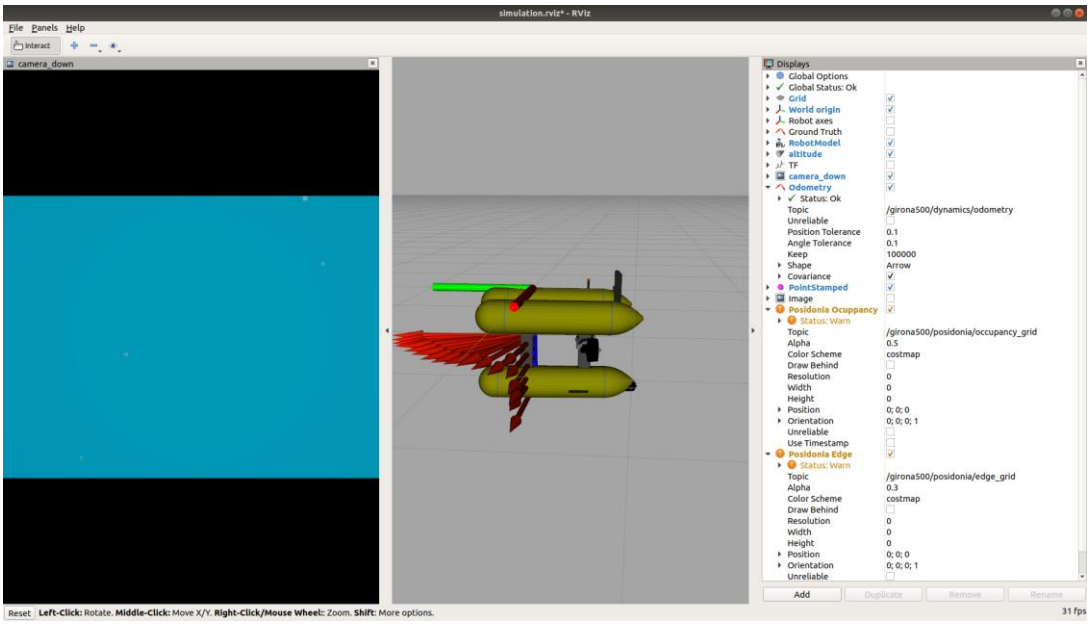

Figura 8. Exemple d'us de l'eina RViz de ROS. A l'esquerra es pot veure la sortida de la càmera del vehicle. Al centre es pot veure el vehicle a l'espai 3D. A la dreta s'aprecien els elements mostrats i les seves propietats.

7.3 Arquitectura del Girona500

L'arquitectura que fa funcionar els AUV del CIRS s'anomena Component Oriented Layer-based Architecture for Autonomy (COLA2). Està formada per un conjunt de llibreries i nodes governen el funcionament dels vehicles tant a baix com alt nivell.

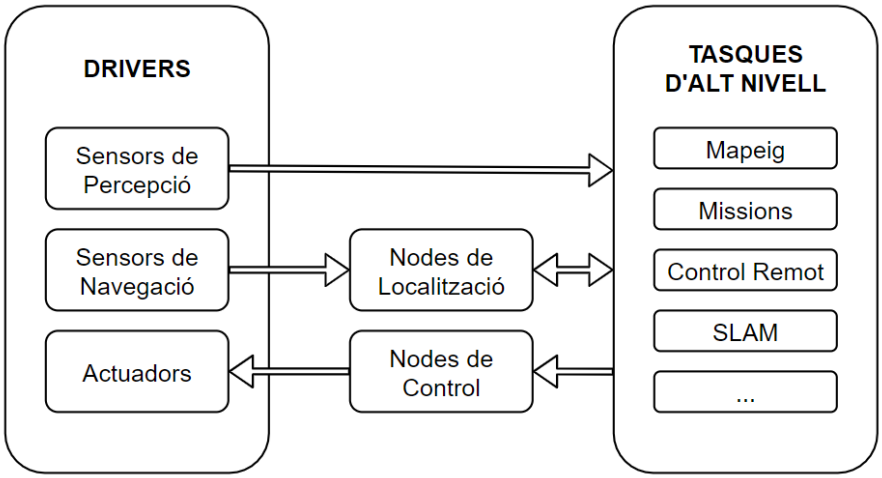

Figura 9. Esquema simplificat del funcionament de l'arquitectura COLA2**.**

Al nivell més baix es troben els *drivers*, que són els encarregats de comunicar-se amb el hardware del vehicle; sensors de percepció, sensors de navegació i actuadors.

- **Drivers de sensors de percepció:** Llegeixen les dades sense processar dels sensors de percepció instal·lats, les processen si cal i publiquen tota la informació segmentada en diferents tòpics, cada tòpic amb el seu tipus de missatge diferent. S'espera que aquests missatges els rebin nodes d'alt nivell encarregats de prendre decisions amb algorismes més sofisticats, segons les dades rebudes.
- **Drivers de sensors de navegació:** Llegeixen les dades dels sensors de navegació instal·lats i les publiquen en tòpics de manera que puguin ser interpretades pels nodes de localització. Unitat de mesura d'inèrcia (IMU), registre de velocitat doppler (DVL), sensor de velocitat de so (SVS) i sistema de posicionament global (GPS) són alguns dels sensors de navegació incorporats al Girona500 i interpretats per aquests drivers.
- **Drivers actuadors:** Aquests drivers són els encarregats de comunicar als motors instal·lats al vehicle les ordres que dicten els nodes de control. Gestionen el flux de corrent cap als actuadors segons els missatges procedents dels nodes de control.

A un nivell intermedi es troben tots els nodes encarregats de la localització i el control del vehicle.

- **Nodes de localització:** S'encarreguen de combinar totes les dades publicades pels drivers dels sensors de navegació al filtre de Kalman estès (EKF), ja explicat anteriorment. D'aquesta manera, aquests nodes publicaran missatges que contindran les dades de localització concretes del vehicle. Aquestes dades les faran servir els nodes d'alt nivell per l'execució de la missió o els experiments que es preparin. La variància dels sensors s'especificarà individualment en un fitxer de configuració per a que els càlculs puguin resultar acurats.

- **Nodes de control:** Aquests nodes permeten controlar el vehicle manualment amb teclat o comandament, executar test dels motors i maniobres preestablertes. A aquest grup també hi pertany el node *Captain*, tot i que aquest últim també executa tasques d'alt nivell com és el visitar un conjunt de coordenades en ordre i executar diferents accions a cada punt (missions).

A les tasques d'alt nivell es trobarien el mapejat del fons o parets submarines, missions d'exploració i aquest mateix projecte. Aquest nodes són els que normalment s'han de modificar o crear de nous amb l'objectiu de preparar el vehicle per a un experiment o tasca específica.

Existeixen 2 tipus de nodes més que no es poden classificar en cap de la resta de grups pel rol especial que exerceixen.

- **Nodes de registre:** Encarregats d'estar subscrits a una gran quantitat de tòpics per deixar constància del que està passant en tot moment. Pel que en cas d'error del sistema o necessitar depurar algun node, pot resultar una informació molt útil a consultar.
- **Nodes de seguretat:** Aquests nodes tenen una funció molt concreta, assegurar la integritat estructural i funcional del robot en tot moment. Estan subscrits a tòpics com la profunditat del vehicle, la temperatura interna, la quantitat de bateria restant o sensors que indiquen si hi ha una fuita d'aigua al compartiment on es troba l'electrònica del vehicle. Si detecten dades anormals i considerades perilloses, enviaran un missatge de prioritat crítica als drivers dels actuadors per iniciar un ascens d'emergència del vehicle.

7.4 Control del Girona500

Amb la quantitat de nodes que poden demanar al vehicle que executi un moviment o un altre, cal un control estricte sobre aquests missatges per a que les instruccions no siguin contradictòries i garantir el correcte funcionament del vehicle.

A més, l'arquitectura permet 4 tipus de missatges diferents a l'hora d'enviar consignes de moviment pels motors:

- **world_waypoint_req:** Coordenades *x*, *y*, *z* i *yaw* respecte a l'origen de coordenades. S'indica el punt i els nodes de control s'encarreguen de portar al vehicle al punt desitjat de la manera més eficient possible.
- **body_velocity_req:** Velocitats desitjades *u*, *v* i *w* en m/s, i *r* en rad/s. El vehicle mantindrà les velocitats demanades sempre que el missatge es continuï enviant.
- **body_force_req:** 2 vectors tridimensionals, força en N i parell de forces en N*m. El vehicle mantindrà les forces demanades sempre que el missatge es continuï enviant. Aquest tipus de missatges resulten útils en cas d'utilitzar un braç robot acoblat al vehicle per contrarestar les forces que exerceixi el braç i mantenir l'estabilitat, per exemple.

- **setpoints:** Matriu de propulsió, especificant quins motors s'activen i la seva potència en el rang [-1, 1]. Aquest tipus és massa baix nivell i està pensat només per ser utilitzat pels propis nodes de control.

El control de les consignes de moviment es realitza en cascada. Es calcularan equivalències fins que només quedin consignes tipus *setpoint*. Es calcularà la velocitat necessària per assolir la posició especificada a les consignes de tipus *world_waypoint_req, convertint-les en body_velocity_requests*. Per a aquestes es calcularà la força necessària per assolir la velocitat especificada, depenent de la velocitat actual. Aquest càlcul transformarà totes les consignes anteriors a *body_force_request.* Per últim, mitjançant la funció de caracterització dels motors, aquestes consignes seran traduïdes a tipus *setpoint*. Cada consigna tindrà associada una prioritat, segons quin node l'hagi emès. Aquesta prioritat dependrà del programador en cas d'afegir nodes que enviïn consignes de moviment. Normalment s'utilitzarà la prioritat *normal*. Per defecte estaran per sobre de la prioritat normal les consignes de teleoperació i les de seguretat del vehicle.

Totes les consignes s'acumulen en una cua i 10 vegades per segon (10Hz), s'agafen totes elles i es combinen de la següent manera:

- Per ordre de prioritat, les consignes bloquejaran els eixos que requereixen, inhabilitant-los per a la resta de consignes.
- Si una consigna de menor prioritat demana la utilització d'eixos que han sigut bloquejats, la consigna serà rebutjada.

D'aquesta manera, és possible combinar consignes que demanen al vehicle moure's cap endavant a una certa velocitat, mantenir una alçada respecte al fons constant i girar cap a un costat, per exemple. No serà possible però, mantenir una alçada constant i enfonsar-se 10 metres alhora. La consigna de més prioritat serà la que faci efecte sobre el vehicle. Això permet tenir consignes de seguretat com la de mantenir una distància mínima amb el terra, que evitarà que el vehicle col·lideixi amb el terreny per error de l'operari.

7.5 Operació del Girona500

Els AUV del CIRS permeten la seva operació manual amb connexió directa i per Wifi. Per a la connexió directa es requereix un cable 'umbilical' compatible amb la tecnologia Ethernet que obre un bus de dades entre un ordinador i el vehicle. Per a la connexió sense fils s'utilitza una petita boia amb una antena Wifi connectada al vehicle amb un cable 'umbilical'. Això permet mantenir la connexió amb el vehicle a una profunditat relativament baixa (aprox 10m, depenent de la llargada del cable). En condicions normals, el senyal Wifi només és capaç de penetrar uns pocs centímetres a l'aigua abans de perdre potència, així que sense la boia, el vehicle només pot utilitzar el Wifi de l'antena pròpia quan és a la superfície.

Amb connexió establerta, l'operari pot comandar el vehicle en mode de teleoperació utilitzant el teclat o més còmodament fent servir un comandament amb joysticks.

Tot i que encara és possible operar al vehicle amb coordenades cartesianes enviant consignes de posició *x*, *y* i *z* als node *pilot*, no és gens pràctic per a un usuari sense experiència prèvia. Existeix una aplicació amb interfície visual anomenada IQUAview desenvolupada per l'empresa IQUA Robotics [24] que facilita molt aquesta tasca i que és utilitzada tant per usuaris novicis com experimentats.

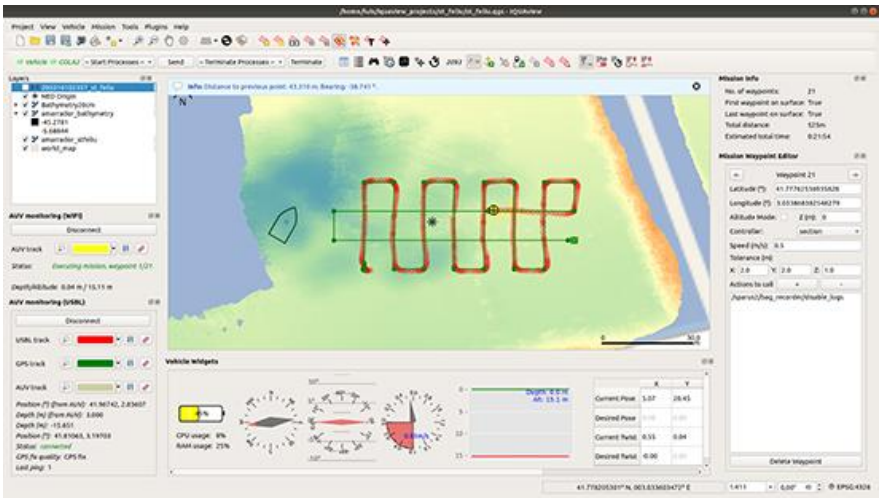

Figura 10. Interfície gràfica IQUAview per un control òptim del vehicle.

La interfície no només serveix per moure el vehicle a un punt determinat. Com es pot observar a la figura anterior, es mostren diversos indicadors de l'estat del vehicle com la bateria o la velocitat actual. També permet generar un conjunt de punts a seguir (missions) on es poden especificar accions a cada punt com engegar una gravació amb la càmera o un escàner sonar.

7.6 Simulador Stonefish

Stonefish és una eina avançada de simulació gràfica centrada en robòtica marina [25]. A diferència del simulador més senzill que incorpora la llibreria COLA2, aquest computa la complexa hidrodinàmica de cossos geomètrics i l'absorció lumínica de l'aigua segons la longitud d'ona. Aconseguint així la tonalitat blava que es pot observar comunament en fons marins reals, ideals per testejar algorismes d'escaneig de de fons com el proposat a aquest projecte.

La llibreria també modela la inèrcia dels diferents elements del vehicle, de tal manera que permet simular el comportament quasi real dels AUV segons la carga que portin. Si aquests tenen un braç robòtic acoblat, simularà correctament les forces inercials i el moviment real que patiria un vehicle al canviar el seu centre de masses dinàmicament.

Altres dinàmiques modelades a la llibreria són: flotabilitat, fricció segons el material, col·lisions modelades com impulsos, simulació atmosfèrica, corrents, onades, vent i múltiples sensors com càmeres i *multi-beam*.

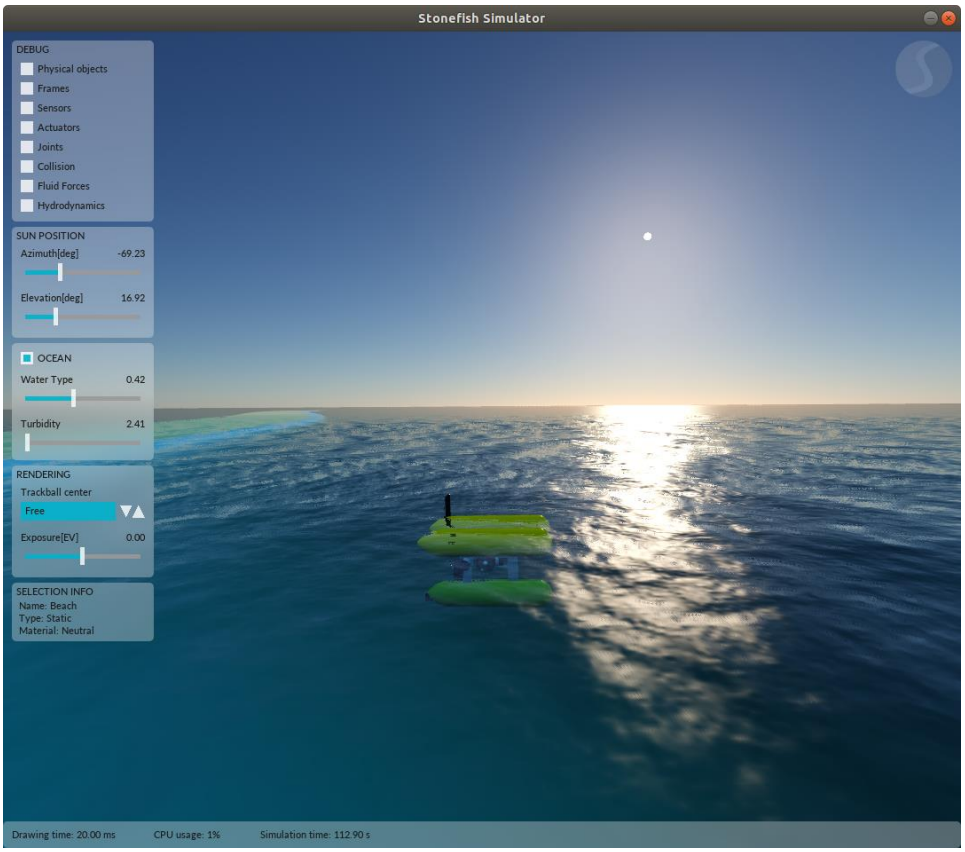

Figura 11. Imatge extreta del simulador gràfic Stonefish on es pot apreciar un model lumínic realista amb reflexions i dispersió. A mesura que augmenta la profunditat es pot apreciar l'enfosquiment de l'aigua. També es poden observar onades a la superfície.

Existeix el paquet per ROS, *Stonefish_ros,* que porta una interfície per a la llibreria Stonefish que permet la utilització d'aquest simulador amb l'arquitectura ROS. Aquest paquet doncs també és necessari per a simular el projecte localment i testejar els algorismes.

7.7 Python2.7

Python és un llenguatge de programació interpretat, multiplataforma i d'alt nivell. Aquest llenguatge és molt conegut per la seva facilitat d'ús i de lectura.

Les llibreries de ROS són compatibles amb C++ i Python. S'ha escollit Python per la seva gran facilitat pel prototipatge. En aquest projecte i pels algorismes desenvolupats, no és crític el temps d'execució i per tant s'ha mirat més per la comoditat per programar i depurar.

A més, al realitzar pràctiques durant un any i mig a IQUA Robotics de programador Python, va ser molt més còmode continuar programant en Python.

En comptes d'utilitzar Python3.8, una versió molt més moderna i generalment més eficient, s'ha escollit programar en Python2.7 perquè la versió de ROS Melodic instal·lada als vehicles està configurada per només funcionar amb Python2.7. Per aquest motiu tots els paquets requerits per al funcionament dels vehicles, els paquets de la seva arquitectura software i els paquets ROS de la llibreria Stonefish han sigut desenvolupats en python2.7.

Per totes aquestes raons, s'ha decidit utilitzar Python2.7 en comptes de C++.

8 Anàlisi i disseny del sistema

Per a dissenyar el sistema de nodes que permetrà la cerca i mapeig dinàmics de posidònia al vehicle autònom, caldrà primerament analitzar les necessitats del sistema.

Al tractar-se de dissenyar un mòdul afegit a l'arquitectura del vehicle, no és necessari crear cap interfície gràfica, ja que no afegeix cap tipus d'interacció amb l'usuari tret de configurar les dades d'inici als diferents fitxers de configuració.

El paquet ROS que cal dissenyar ha de contenir els diferents fitxers Python dels nodes, fitxers tipus *launch,* que iniciaran els nodes i els seus paràmetres inicials, i els fitxers necessaris per configurar la simulació.

- **Fitxers de node:** Aquests contindran tots els algorismes rellevants per a l'execució del projecte. Escrits en Python2.7, especificaran les llibreries a utilitzar i tindran com a mínim una classe definida, específica per a cada node, amb tots els algorismes necessaris per portar a terme les seves funcions.
- **Fitxers launch:** Aquest fitxers, amb format "xml" i extensió ".launch", donaran inici als nodes programats. També permeten definir paràmetres globals ROS (*rosparam*) als que podran accedir tots els nodes en execució. Aquests paràmetres seran molt útils a l'hora de definir les diferents variables del sistema, com el nom del vehicle a utilitzar, les característiques de la càmera instal·lada al vehicle o l'alçada respecte el fons marí a la que es vol executar l'experiment.
- **Fitxers de simulació:** seran tots aquells fitxers afegits que el simulador *Stonefish* requereixi per a un correcte funcionament de l'escenari. Aquest fitxers són els *URDF* o Universal Robot Descriptor Format, els fitxers de definició de geometria dels objectes tridimensionals, els fitxers de textura d'aquests objectes tridimensionals i per últim els fitxers d'escenari, amb extensió ".scn". Aquests últims contindran tota la configuració de l'escenari, és a dir, especificaran la situació de cada element, el seu tipus de material, la textura i el coeficient de fregament de la superfície de l'objecte.

Pel que fa al disseny dels fitxers launch s'ha decidit crear-ne dos. El primer executa tota l'arquitectura del vehicle i Stonefish amb l'escenari preparat. El segon s'encarrega de crear tots els paràmetres tipus rosparam que faran servir els nodes del projecte i d'engegar els mateixos nodes.

8.1 L’escenari simulat

Aquest escenari té com objectius ajudar a provar els algorismes i replicar la manera més fidel possible l’entorn i distribució reals en el que es troba la posidònia a la natura. Al repositori “cola2_stonefish” del CIRS [26] es poden trobar exemples de fitxers .launch, .scn i els models tridimensionals dels vehicles Sparus II i Girona500 amb les seves textures corresponents. Així doncs, per a dissenyar l’escenari caldrà, a més d’adaptar els exemples de codi obert anteriors, dissenyar el terreny que escanejarà el vehicle i afegir-lo als fitxers de configuració d’escenari.

Per dissenyar un terreny que sigui lo més semblant possible a l’hàbitat de la posidònia, caldrà modelar una extensió de terreny irregular des de la costa fins a les profunditats màximes on es sol trobar posidònia. Aquest terreny serà modelat utilitzant Blender [27].

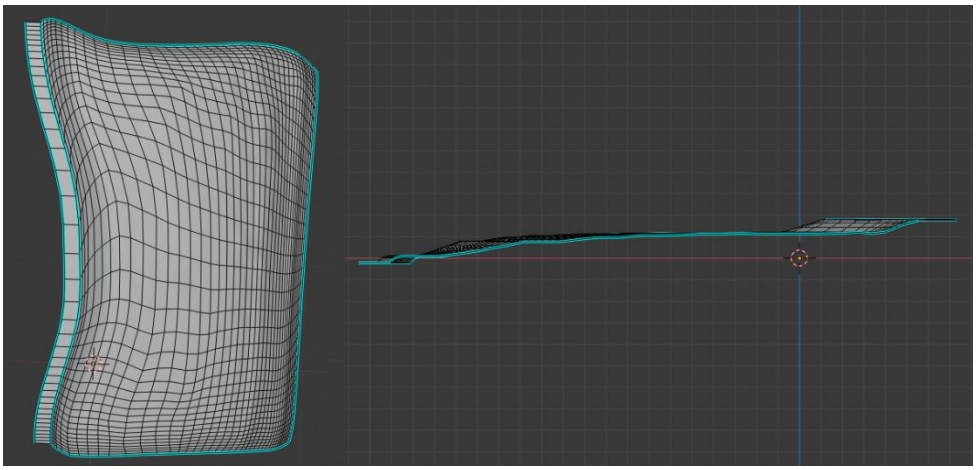

Figura 12. Imatge del disseny d’un terreny de costa irregular i realista. A l’esquerra des d’un punt de vista zenital i a la dreta de perfil. Les mesures del terreny són 250 metres d’amplada, 400 metres de llargada de costa i 20 metres de profunditat.

Un cop modelat el terreny, cal afegir-hi una textura, ja que la finalitat d’aquest és que el vehicle escanegi el fons i detecti la planta de posidònia utilitzant la càmera que porta incorporada. Aquest fons es dotarà d’una textura groga i blava, representant el color groc la presència de la planta de posidònia i el color blau representant l’absència d’aquesta. La finalitat de la textura és facilitar la simulació del mòdul de Deep Learning posterior explicat en detall a la següent secció.

Per a aconseguir una textura el més realista i fidel possible a la realitat s’ha utilitzat com a model un mosaic generat d’un escaneig real de posidònia realitzat a la costa d’Aigua Blava. Aquest mosaic s’ha quadruplicat utilitzant la mateixa imatge reflectida en horitzontal i vertical, per ampliar la varietat de dades disponible al mapa final. Un cop fet això, s’ha binaritzat a mà utilitzant l’eina gratuïta i de codi obert Gimp [28] utilitzant diversos pinzells amb textures i transparències diferents. D’aquesta manera s’espera introduir variabilitat pel posterior processament d’imatge i així aconseguir un resultat amb més soroll, més proper al que donaria el mòdul de deep learning.

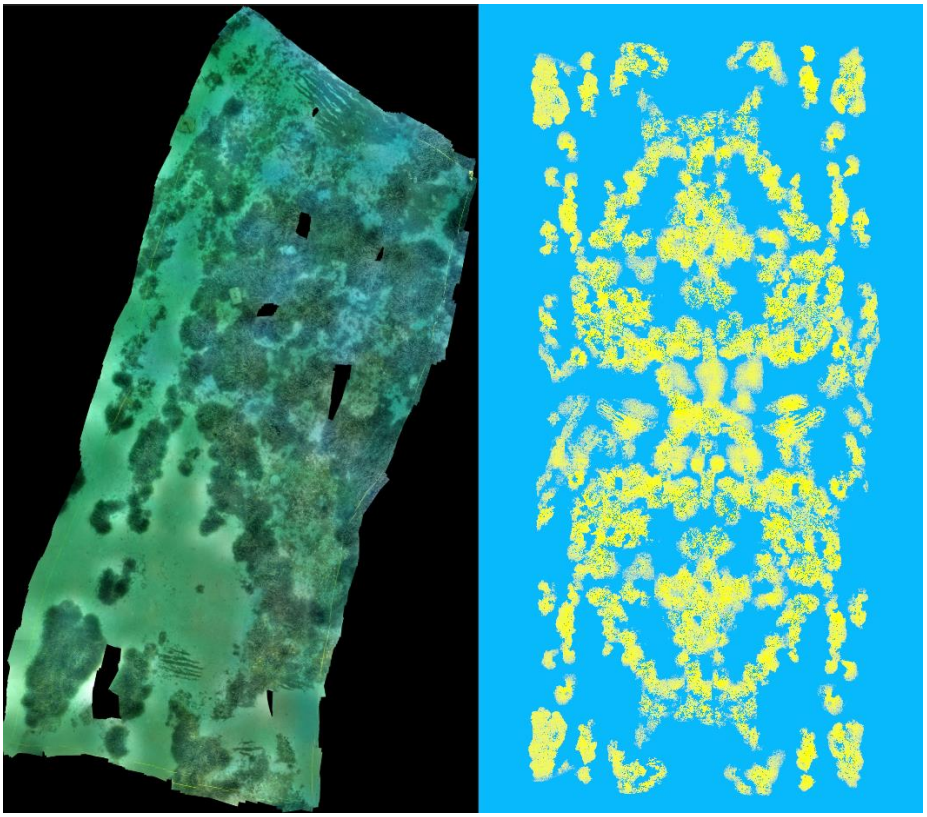

Figura 13. A l'esquerra el mosaic resultat de l'escaneig d'un fons amb posidònia a Aigua Blava [29]. A la dreta el mapa sintetitzat a partir de replicar 4 vegades el mosaic amb reflexions horitzontal i vertical i després binaritzat a mà.

La intenció final és la de crear una praderia de posidònia lo més semblant a la realitat possible. Per això, un cop s'obté la imatge de la dreta a la anterior figura, s'utilitzen els diferents patrons generats per construir una gran praderia amb varietat i sense repeticions. Així s'aconsegueix un escenari variat i s'assegura la robustesa del sistema final.

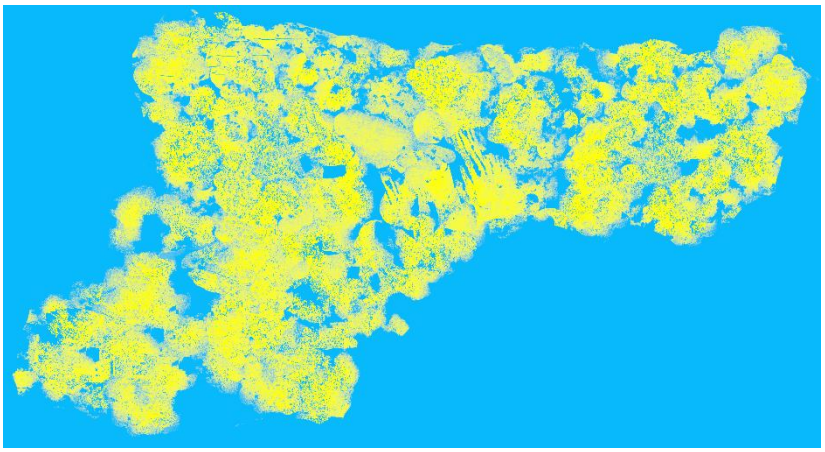

Figura 14. Praderia dissenyada a partir de patrons de posidònia reals.

Un cop s'ha dissenyat el fons marí, caldrà mapejar aquest fons com a textura del terreny tridimensional.

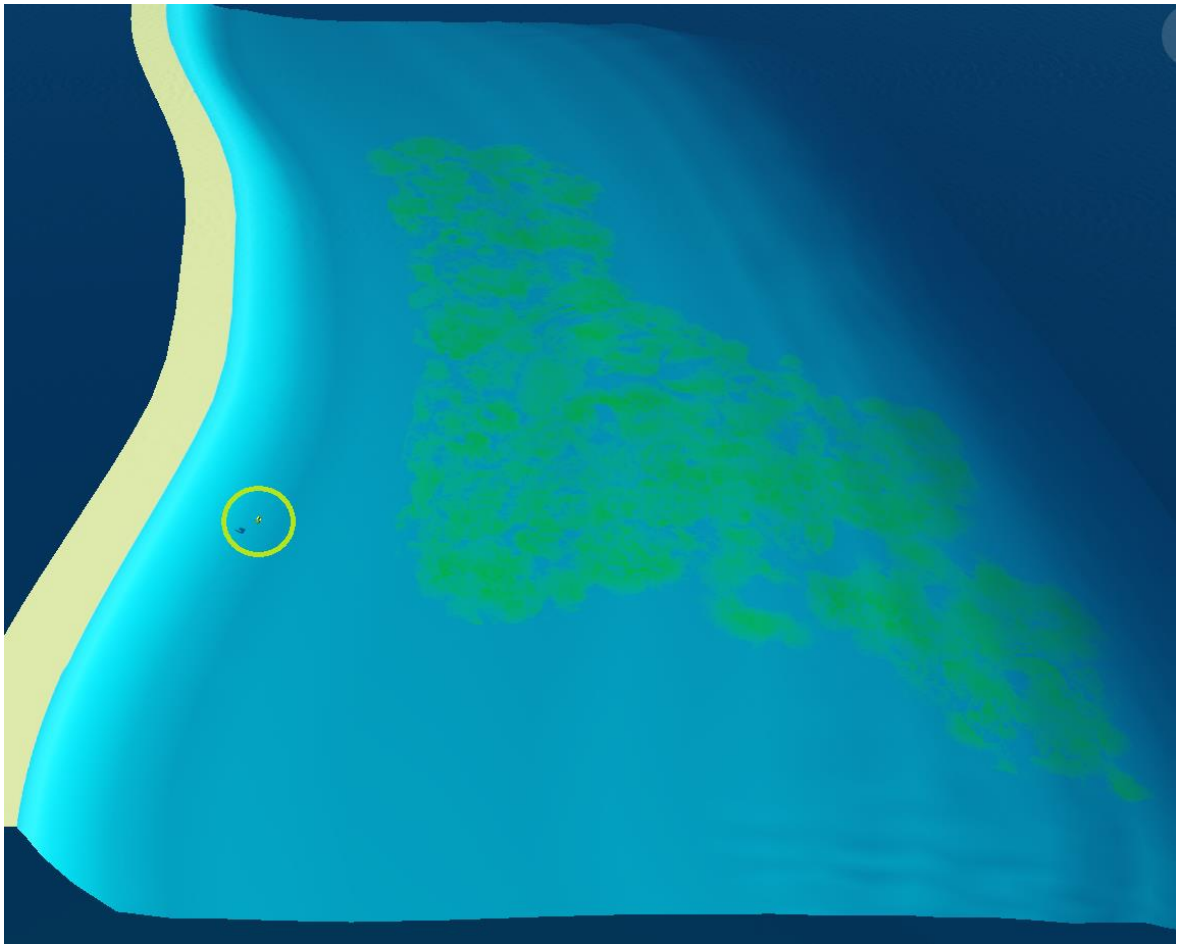

Figura 15. Escenari final. S'ha afegit color a la part del terreny amb profunditat zero com a detall visual. Es pot apreciar encerclat en verd el vehicle, 1.5 metres de llargada, que ajuda a posar en perspectiva la mira de l'escenari.

8.2 Disseny dels nodes

Per a dissenyar el sistema que permetrà fer el seguiment i mapejat del contorn de la praderia de posidònia calen tres funcions. La funció d'anàlisi d'imatge que simularà la sortida del mòdul d'intel·ligència artificial, la funció de mapeig i la funció del control de la navegació. Per tant, caldrà dissenyar tres nodes.

- **Node de detecció d'imatge:** Aquesta funció, tot i no entrar dins del projecte, és necessària per testejar el comportament dels algorismes i el sistema complet.
- **Node de mapeig:** Tindrà la funció de crear, actualitzar i publicar el mapa de detecció de posidònia, segons l'entrada del detector i la posició que ens doni el node de localització del vehicle "/girona500/navigator".
- **Node de control de la navegació:** Segons les dades locals del mapa dinàmic, decidirà el pròxim moviment del vehicle.

Cada funció es programarà en un node independent. També es crearà un quart node amb l'única funció de depurar els nodes de mapeig i navegació de manera visual.

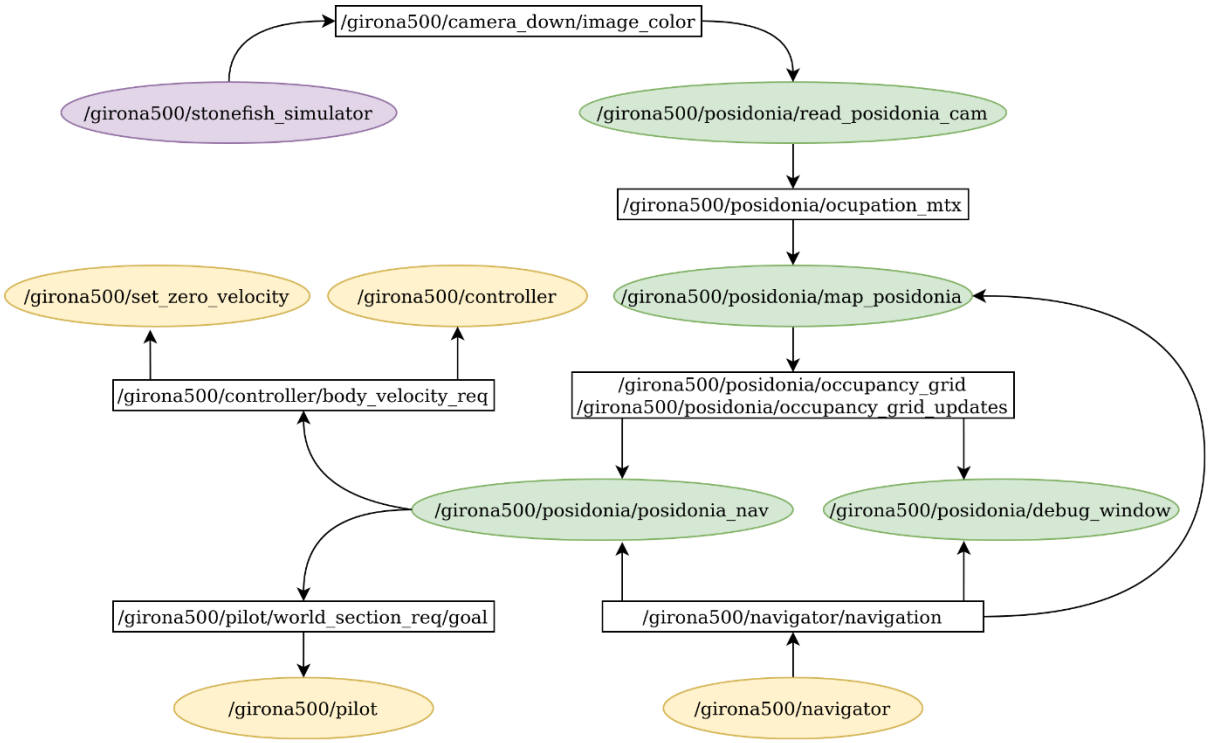

Figura 16. Diagrama de nodes i tòpics del projecte. De color verd es troben els nodes mencionats que es dissenyaran per aquest projecte. De color groc els nodes de l'arquitectura del vehicle que interactuaran amb els nodes anteriors. De color lila el node del simuladors Stonefish. Els rectangles representen tòpics de ROS i les fletxes representen subscripcions i publicacions d'aquests.

S'ha decidit anomenar a tots els nodes i tòpics amb el mateix prefix "/girona500/posidonia/" per a mantenir un cert ordre. D'aquesta manera, és molt més senzill entendre quins elements formen part d'aquest paquet.

8.2.1 Disseny del node de detecció d'imatge

Com s'ha explicat anteriorment, aquest projecte està pensat per poder funcionar conjuntament amb un mòdul d'intel·ligència artificial, desenvolupat independentment, per detectar la posidònia al fons marí. Per dissenyar i testejar un bon comportament del vehicle caldrà doncs simular aquesta sortida.

Aquesta sortida, segons s'ha parlat amb els membres de l'equip del CIRS que s'encarreguen del mòdul mencionat, serà d'una matriu binària de mida segons la resolució i precisió del sistema desenvolupat. Per tant, s'ha de modelar un sistema que tingui com a sortida una matriu d'uns i zeros de mida N.

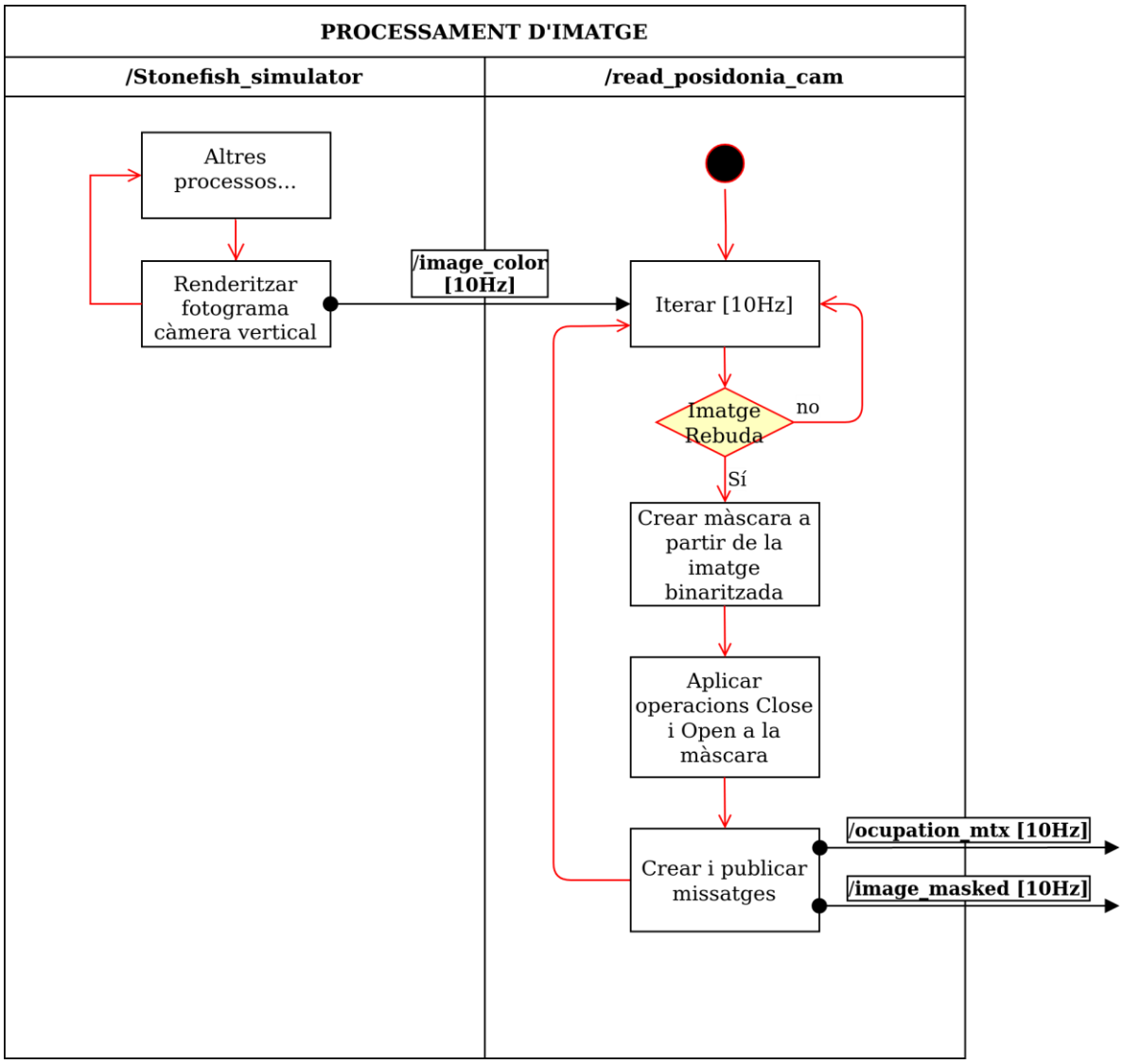

Figura 17. Diagrama d'activitats de la interacció del node "/read_posidonia_cam" amb el simulador Stonefish. El tópic "/ocupation_mtx" serà la sortida de matriu binària que es busca. El tòpic "/image_masked" serà un missatge de la imatge a color, combinada amb la matriu resultant, que facilitarà la depuració.

8.2.2 Disseny del node de mapeig

Tota la informació sobre la detecció que es rep és important i s'ha de tractar. Aquestes deteccions quedaran reflectides en un mapa d'ocupació dinàmic per caselles, que s'anirà omplint a mesura que el vehicle prengui mesures del fons. Aquest mapa s'haurà d'anar publicant periòdicament amb l'objectiu que tant el node de navegació com l'operari, en cas de que hi hagi connexió, rebin les últimes modificacions.

La mida de les caselles del mapa d'ocupació serà determinada a l'inici de l'execució als fitxers de configuració. Dependrà de l'algorisme el fer l'equivalència entre les caselles de la matriu de detecció i les caselles del mapa.

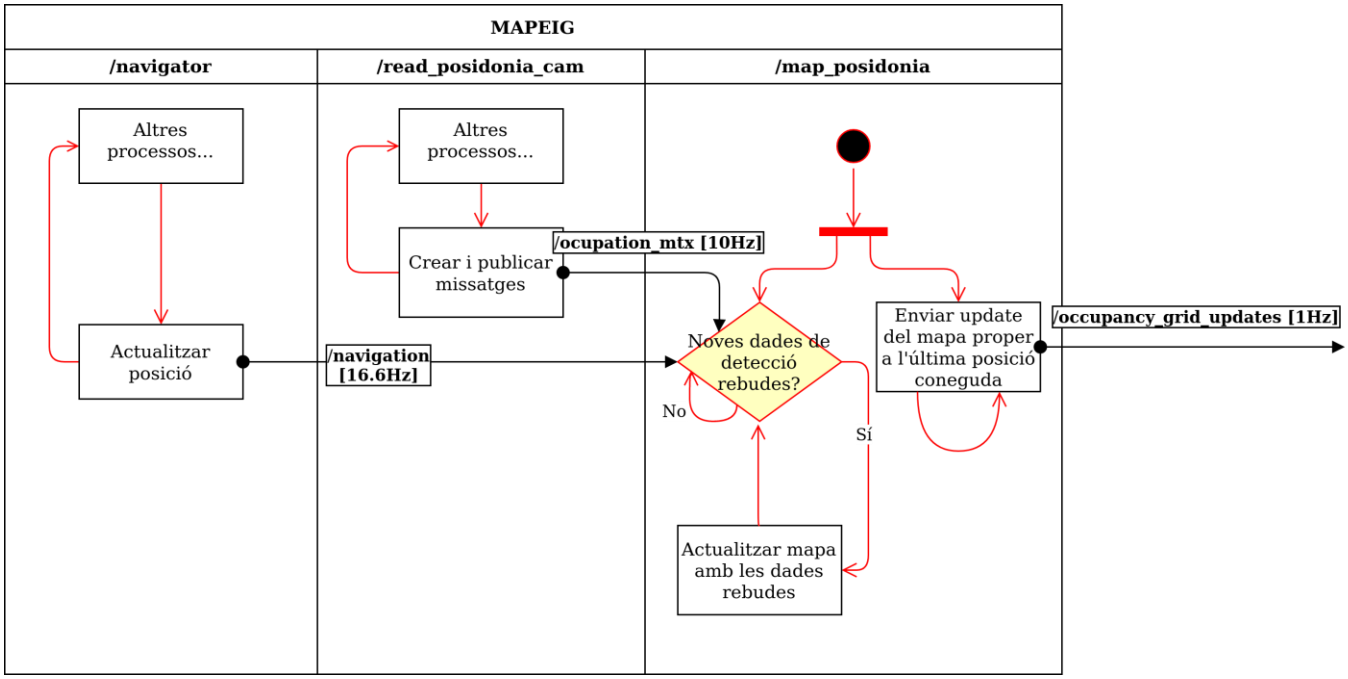

Figura 18. Diagrama d'activitats del mapejat dinàmic que realitza el mòdul "map_posidonia". De manera paral·lela actualitza el mapa amb les dades rebudes i envia un tall del mapa proper a l'última posició coneguda com a "update".

Per reduir errors de detecció, el mapa s'actualitza deu vegades per segon mentre que el missatge d'actualització s'envia tan sols una vegada per segon. D'aquesta manera es minimitza l'efecte que pot tenir el soroll en el resultat final.

8.2.3 Disseny del node de control de la navegació

El node de navegació serà l'encarregat de dirigir tota la missió d'escaneig. Un cop iniciats els processos, i mentre no hi hagi posidònia detectada, el node de navegació efectuarà una maniobra "lawnmower". Aquest tipus de maniobra rep el seu nom per utilitzar el mateix tipus de trajectòria que els tallagespa de jardí. Proporciona, segons la configuració, una cobertura molt complerta d'un àrea específica.

Amb aquesta maniobra lawnmower es pretén trobar el contorn d'una praderia de posidònia. Un cop aquesta detecció sigui positiva, es podrà iniciar l'algorisme per resseguir aquest contorn i mapejar tota la praderia.

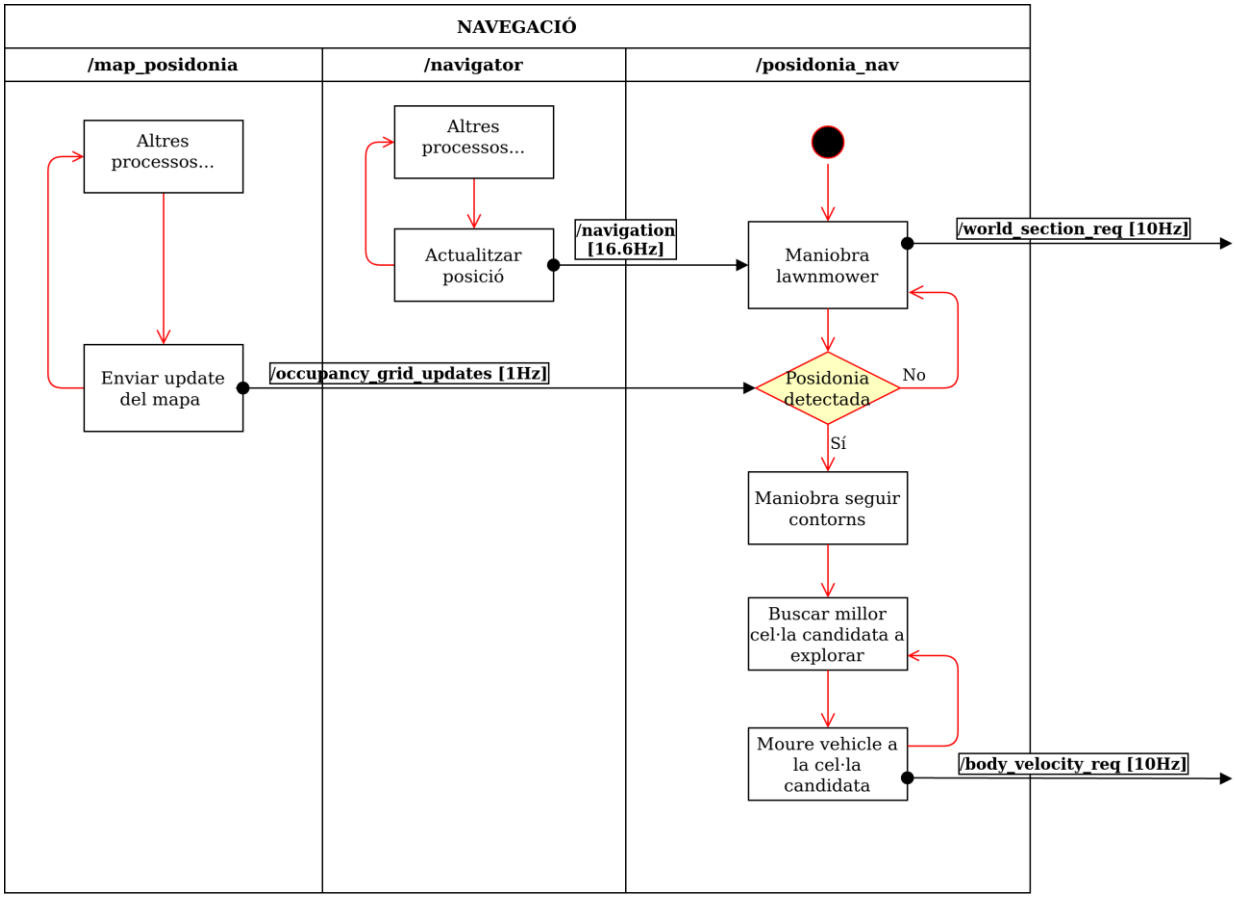

Figura 19. Diagrama d'activitats del node "posidonia_nav" per a la navegació. Un cop detectada la posidònia per primer cop, es passarà a efectuar la maniobra de seguir els contorns fins a finalitzar l'execució.

Es pot observar en el diagrama que el node "posidonia_nav" envia 2 consignes de moviment diferents segons l'estat en el que es troba. Això és degut a que al seguir els contorns, es requereix un control molt més precís del vehicle. Però per efectuar la maniobra lawnmower serà suficient el càlcul dels diferents punts de la trajectòria i anar enviant aquests punts al controlador a mesura que avanci el vehicle.

9 Implementació i proves

Durant la implementació del projecte, no tot ha anat com s'esperava. Alguns dissenys inicials que en principi semblaven poder funcionar, un cop implementats no donaven el resultat esperat. En aquest apartat es destacarà un d'aquest problemes i alguns dels algorismes més rellevants del projecte.

9.1 Problemes de la primera implementació del control de navegació

El disseny plantejat a l'apartat anterior no ha sigut sempre d'aquesta manera. En un principi, es pretenia crear un node que analitzés la matriu de sortida de la càmera ja processada. Aquest node publicava un missatge de detecció de posidònia indicant el percentatge de celes amb detecció positiva respecte a les cel·les amb detecció negativa i la direcció en la qual es trobava la major densitat de posidònia detectada. Aquest missatge el rebia el node controlador per determinar cap a on calia moure el vehicle per poder seguir correctament el contorn.

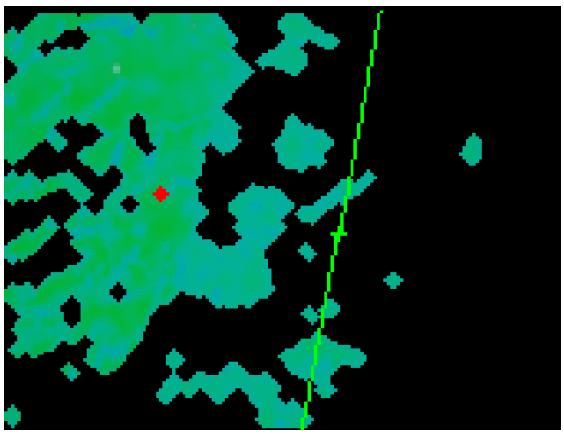

Figura 20. Imatge amb finalitat de depuració. És el resultat de la imatge de la càmera multiplicada per la matriu binària de detecció. El punt vermell representa el centre de masses de la posidònia detectada. La línia verda indica la trajectòria aproximada que cal seguir per explorar el contorn de la praderia.

A partir del processament de la detecció resultant del mòdul "/read_posidonia_cam" el node de control disposa de tota la informació necessària per poder seguir correctament els contorns. Tot i així, aquest disseny no té un bon comportament quan es detecten més d'una illa de posidònia, com es pot observar a la següent imatge.

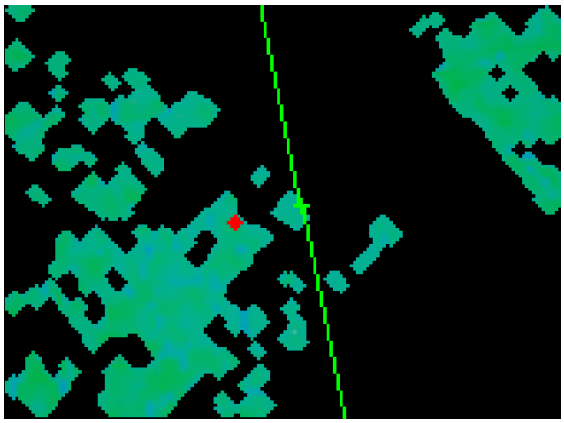

Figura 21. Imatge amb finalitat de depuració. Es pot observar con tant a l'esquerra com a la dreta es detecten petites illes de posidònia. L'algorisme troba un centre de masses i un contorn a seguir incorrectes.

Va ser possible millorar l'algorisme de detecció del contorn per aconseguir superar aquest problema en concret. Tot i així, i degut a la naturalesa sorollosa de la presència de posidònia a l'escenari i al mon real, es va decidir replantejar el disseny. A la següent imatge es pot observar el resultat parcial que es va obtenir.

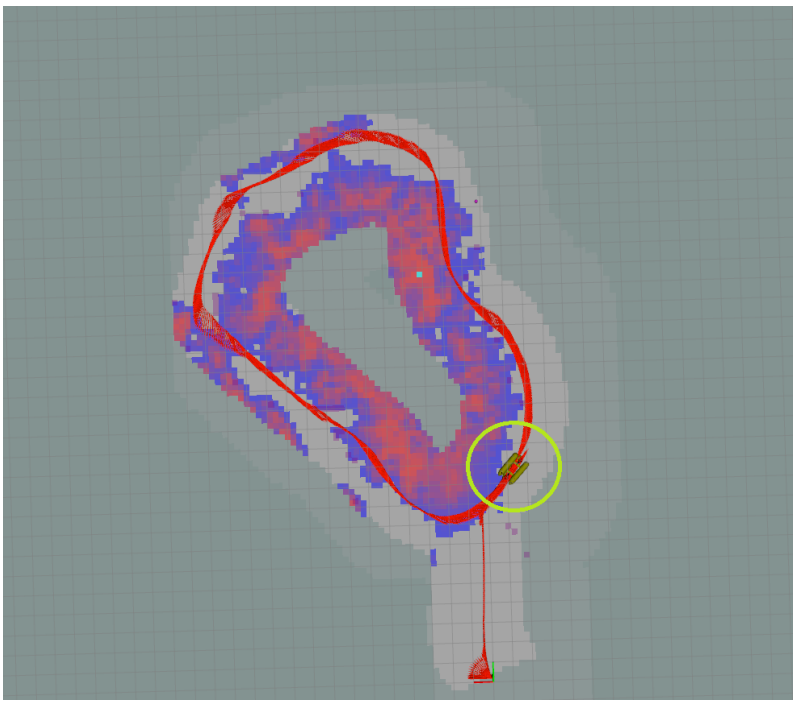

Figura 22. Mapa resultant del primer disseny. El vehicle segueix el contorn de la illa de posidònia. Imatge extreta de RViz.

El resultat, tot i que no el desitjat, ha ajudat a entendre que el disseny de la decisió del moviment era incorrecte. Com la càmera està orientada de manera zenital, és molt complexa predir quina és la direcció que seguirà el contorn de la praderia (no el de la illa individual). En aquest punt es va decidir utilitzar les dades del mapa i no les de la càmera per decidir un rumb. Això permet tenir una visió més general de la zona que envolta el robot i poder prendre una decisió més encertada. Aquest enfoc serà explicat en la secció *Implantació i resultats*.

9.2 Algorismes rellevants

Durant la implementació del sistema, alguns dels algorismes i problemes a resoldre han sigut més difícils de tractar que altres. En aquest apartat es destaquen dos situacions que han requerit una atenció especial i han resultat especialment complexes.

9.2.1 Adaptar la matriu binària de la detecció de posidònia per actualitzar el mapa

Per motius d'eficiència, la quantitat de cel·les que composen la matriu de detecció serà superior a la que aquesta ocupi al mapa. Caldrà reduir la mida de la matriu de detecció, perdent el mínim d'informació possible.

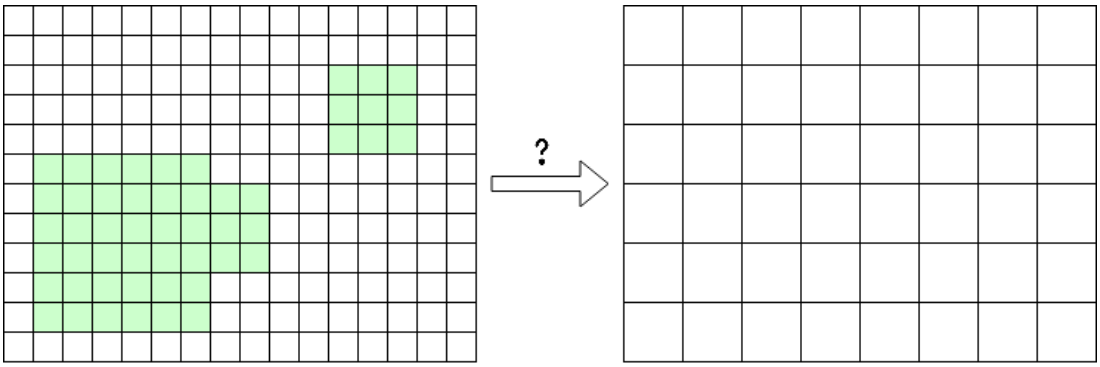

Figura 23. Ajut visual per entendre el problema plantejat. La graella de l'esquerra representa la matriu de detecció de posidònia, publicada pel node "/read_posidonia_cam" en simulació. La graella de la dreta representa el tall del mapa que equival al camp visual de la càmera.

Aquest problema es complica més quan es té en compte que l'orientació de la matriu de detecció vindrà donada per l'orientació del vehicle, mentre que el mapa només pot tenir una orientació.

Seguidament es detalla l'algorisme utilitzat per resoldre aquest problema d'equivalència en pseudocodi.

```
# Caldrà comprimir la matriu de detecció, redüint el seu nombre de cel·les
ràtio_vertical = n_files_detecció / n_celes_mapa_vertical
ràtio_horitzontal = n_columnes_detecció / n_celes_mapa_vertical

# Primer la matriu es comprimirà verticalment, sumant les files de la matriu
de detecció que equivalen a la mateixa fila en el mapa entre sí.
Crear Matriu (aux1, n_celes_mapa_vertical, n_columnes_detecció)
Per i de 0 a n_celes_mapa_vertical, fer
      index_fila = i * ràtio_vertical
      Per j de 0 a ràtio_vertical, fer
            # Suma de files
            aux1[i] = aux1[i] + matriu_detecció[index_fila + j]
      Fi per
```

```
Fi per


# La matriu aux1 ara es comprimirà en horitzontal, sumant les columnes de la matriu que equivalen a la mateixa columna en el mapa entre sí
Crear Matriu (m_det_compr, n_celes_mapa_vertical, n_celes_mapa_horitzontal)
Per i de 0 a n_celes_mapa_vertical, fer
      Per j de 0 a n_celes_mapa_horitzontal, fer
            valor = 0
            index_columna = j * ràtio_horitzontal
            # Suma de columnes
            Per k de 0 a ràtio_horitzontal, fer
                  valor = valor + aux1[i][index_columna + k]
            Fi per
            m_det_compr[i][j] = valor
      Fi per
Fi per

# Seguidament es normalitza el valor de les cel·les per l'interval [0,1].
m_det_comp = m_det_comp / (ràtio_horitzontal * ràtio_vertical)

# S'ha passat de tenir una matriu booleana de mida nxm a una matriu de valors tipus float de mida pxq molt més petita. Ara caldrà alinear aquesta matriu sobre el mapa. Caldrà trobar les posicions x i y de cada cel·la en relació al vehicle, rotar aquestes coordenades respecte el vehicle segons la seva orientació i trobar quina cel·la del mapa té les coordenades més properes a aquesta.

x_matriu = mida_cela * n_celes_mapa_vertical / 2
y_matriu = mida_cela * n_celes_mapa_horitzontal / 2

Per i de 0 a n_celes_mapa_vertical, fer
      Per j de 0 a n_celes_mapa_horitzontal, fer
            # Obtenir posició de la cel·la respecte al vehicle en metres
            x_cela = x_vehicle + x_matriu –i * mida_cela
            y_cela = y_vehicle + y_matriu –j * mida_cela

            # Rotar el punt de la cel·la trobada respecte al punt del vehicle
            p_cela = Punt(x_cela, y_cela)
            p_vehicle = Punt(x_vehicle, y_vehicle)
            p_cela = RotarPunt (p_cela, p_vehicle, angle_vehicle)

            # Trobar cel·la del mapa més propera al punt trobat
            p_mapa = TrobarCelaPropera(p_cela)

            # Actualitzar mapa amb les dades de detecció de la cel·la
            ActualitzarMapa(p_mapa, m_det_comp[i][j])
```

```
      Fi per
Fi per
```

La complexitat d'aquest algorisme es pot calcular amb la següent aproximació:

- Compressió de la detecció vertical:

$$Gv(n_v, p_v, r_v) = p_v \cdot r_v \cdot n_v, on\ p_v \cdot r_v = n_v, O(n_v^2)$$

- Compressió de la detecció horitzontal:

$$Gh(p_v, p_h, r_h) = p_v \cdot p_h \cdot r_h, on\ p_h \cdot r_h = n_h, O(p_v \cdot n_h)$$

- Alineació de la matriu al mapa i actualització:

$$A(p_v, p_h) = p_v \cdot p_h, O(p_v \cdot p_h)$$

On:

- n_v i n_h son les mides horitzontals i verticals de la matriu de detecció
- p_v i p_h són les mides horitzontals i verticals de la matriu comprimida
- r_v i r_h són els ratis horitzontals i verticals calculats

Per a l'ús en que s'ha dissenyat aquest algorisme, on les mides de la matriu de detecció són molt d'un a dos ordres de magnitud superiors a les mides de la matriu comprimida, l'ordre aproximat d'aquest algorisme serà de $O(n_v^2)$.

9.2.2 Trobar cel·les candidats per explorar

S'ha dissenyat el node "/posidonia_nav" per poder fer que el vehicle segueixi el contorn de la praderia de posidònia. Aquest analitza la informació sobre el seu voltant que publica el node "/map_posidonia" en busca de possibles cel·les candidates per explorar. Aquestes cel·les són candidates a ser possibles cel·les de contorn de la praderia, però que són al costat de cel·les inexplorades i encara no hi ha suficient informació per determinar si ho són o no.

Per a trobar les cel·les candidates a l'exploració, l'algorisme s'ha basat en la tesis doctoral del Dr. Eduard Vidal, "Online 3D View Planning for Autonomous Underwater Exploration" [30]. Més concretament, en la secció "2D Frontier-based Viewpoint Generation for Exploring and Mapping Underwater Environments", on es tracta la generació de punts d'exploració de frontera per a l'exploració i mapeig en entorns subaquàtics.

La idea radica en identificar les cel·les explorades amb detecció positiva i que són veïnes de cel·les inexplorades i de cel·les explorades amb detecció negativa. Un cop identificades, s'escollirà el candidat més idoni a explorar. Aquesta idoneïtat vindrà donada per la distància de la cel·la al vehicle i l'angle entre aquesta i la direcció actual del vehicle. S'expressa matemàticament amb la següent funció de cost:

$$\beta = -atan2\left(v_y - p_y, v_x - p_x\right) - \varphi$$

$$c = \|v_{xy} - p_{xy}\| + \omega \cdot \frac{\left|\frac{\pi}{2} - \beta\right|}{\frac{\pi}{2}}$$

On 'v' és el punt on es troba el vehicle, 'p' es el punt de la cel·la candidata i 'ω' és un factor de pes determinat experimentalment. El candidat escollit serà el que tingui un menor cost.

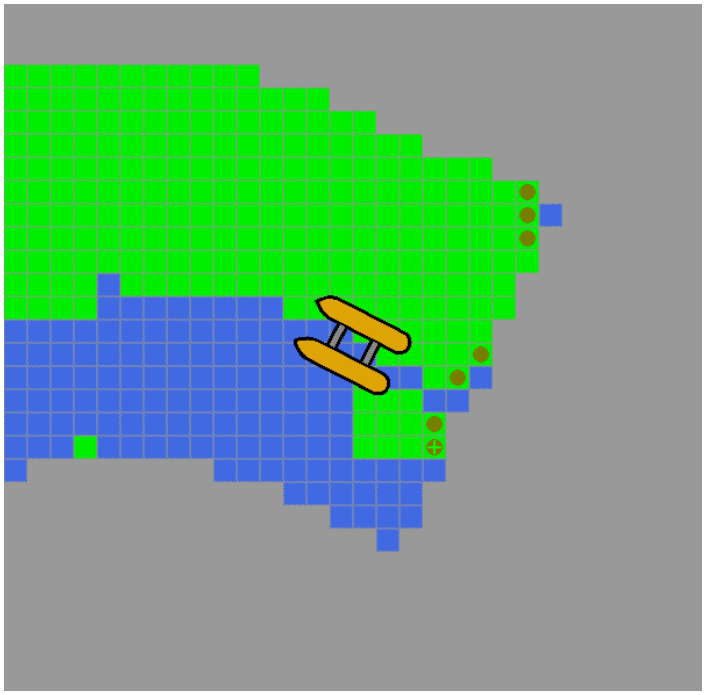

Figura 24. Imatge creada pel node "/debug_window" amb l'objectiu de depurar l'algorisme de cerca de candidats. Es pot observar que totes les cel·les amb detecció de posidònia que compleixen els requisits per ser candidates estan marcades amb un cercle. El candidat escollir té una creu verda a l'interior del cercle.

A la figura apareix una cel·la isolada a la part esquerra, que tot i complir les condicions per a ser candidata, no ho és. Això és degut a que es filtren de l'algorisme les possibles cel·les candidates que no tenen cap cel·la adjacent de posidònia. Això redueix l'error de decisions influenciat per falsos positius o soroll.

Seguidament es detalla l'algorisme utilitzat per resoldre aquest problema en pseudocodi.

```
# Trobar candidats a la matriu mapa en veïnatge a 8
candidats = Vector()
Per cada fila de mapa, fer
      Per cada columna de mapa, fer
            Si HiHaDetecció(mapa, fila, columna), fer
                  # Trobar els 8 veins de la cel·la
                  veins = TrobaVeinsA8(mapa, fila, columna)

                  # Comprovar si hi ha algun veí inexplorat i algun sense
                  detecció
                  Si   HiHaVeiInexplorat(veins)   i   HiHaVeiBuit(veins)   i
                  HiHaVeiDetectad(veins), fer
                        # Calcular angle del contorn
                        cd = TrobarCentreDeteccions(veins)
                        cb = TrobarCentreBuits(veins)
                        angle = -atan2(cd, cb)
```

```
                        # Afegir candidat a la llista
                        candidat = (fila, columna, angle)
                        candidats.afegir(candidat)
                  Fi si
            Fi si
      Fi per
Fi per

# Un cop s'ha obtingut la llista, es troba el candidat amb menor cost
Per cada candidat de candidats, fer
      c = CalcularCost(candidat)
      Si cost no existeix o c < cost, fer
            cost = c
            escollit = candidat
      Fi si
Fi per
```

10 Implantació i resultats

A continuació es descriuran en detall les implementacions dels nodes desenvolupats i els seus resultats tant individuals com de funcionament conjunt.

10.1 Implantació del node “/read_posidonia_cam”

Com ja s’ha explicat, el projecte dependrà d’un mòdul d’intel·ligència artificial per funcionar al mar en un entorn real. En l’entorn simulat no es disposa d’aquest mòdul i caldrà simular-lo de la manera més fidel possible. Aquesta serà la funció principal del node “/read_posidonia_cam”.

Gràcies a les dades que es van prendre en la sortida al mar del Febrer, els companys del CIRS encarregats del disseny i implementació d’aquest mòdul van poder crear un vídeo de demostració d’aquest.

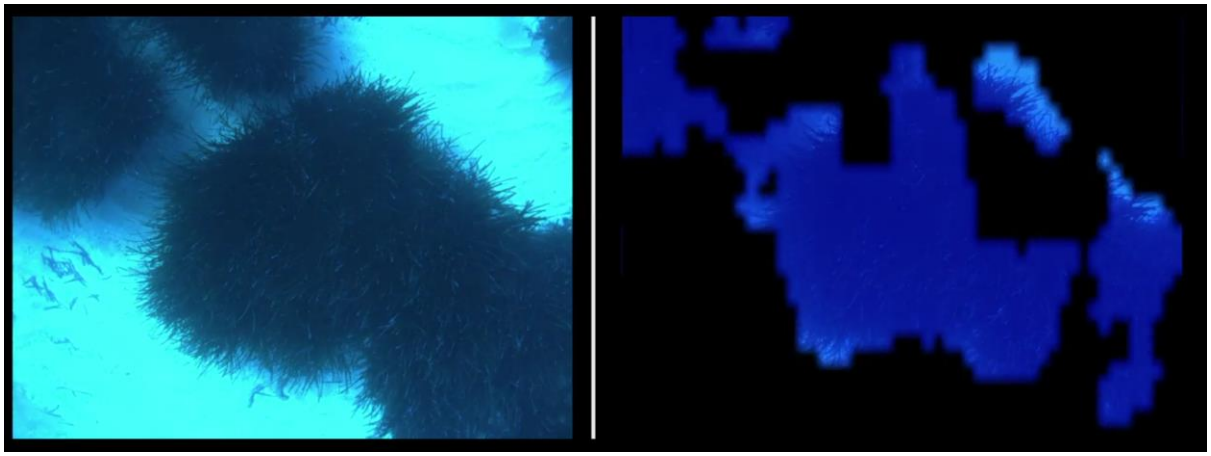

Figura 25. Fotograma extret del vídeo demostració creat per Coronis Computing demostrant les capacitats del mòdul d’intel·ligència artificial. A l’esquerra imatge de la planta de posidònia. A la dreta la sortida del detector Deep Learning multiplicada sobre la imatge original.

Aquest vídeo demostració s’ha fet servir com a referència per simular la detecció amb el node “/read_posidonia_cam/”. Partint de l’observació de que la intel·ligència artificial dóna resultats imperfectes i amb soroll, es vol implementar una detecció d’imatge amb un nivell similar de precisió. També és per aquest motiu que es va dissenyar l’escenari amb una textura bicolor i alhora una elevada entropia. Això permetrà implementar un anàlisi d’imatge senzill i amb resultat sorollós.

Per a implementar aquesta detecció d’imatge, cal primer entendre què veu la càmera del vehicle i perquè.

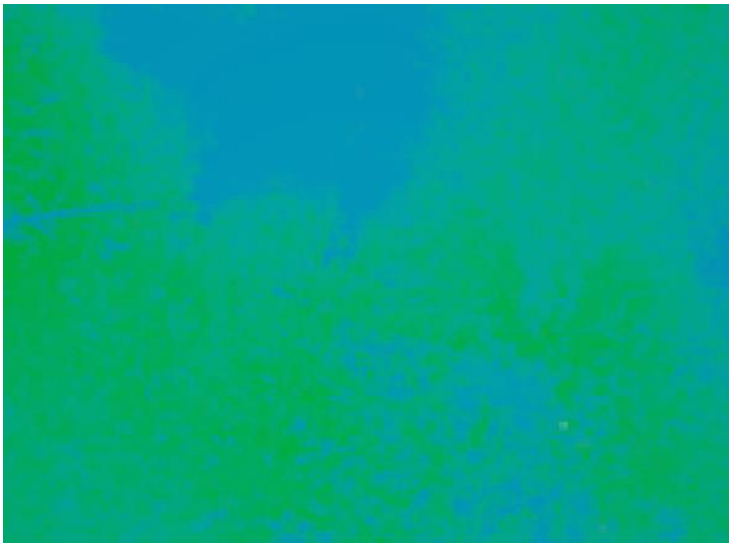

Figura 26. Imatge de la càmera publicada pel simulador.

Sota l'aigua , la llum pateix dispersió segons la seva terbolesa i una absorció de la llum depenent de la longitud d'ona. Aquest efecte és també simulat a Stonefish, i per tant, s'ha de tenir en compte.

El node "/read_posidonia_cam" doncs, realitzarà una binarització de la imatge seguit d'un postprocessat bàsic per reduir el soroll final i fer que la imatge de sortida s'apropi més a la del vídeo demostració.

La binarització es realitza filtrant pels colors dels píxels. Experimentalment s'ha trobat el millor llindar per a la binarització provant diferents valors. Tots aquells píxels amb un valor menor a 130 pel color blau o un valor superior a 175 pel color verd es consideren com a píxels de posidònia. Aquest filtre es realitza a l'espai de color RGB i no a l'espai HSV. En visió per computador, típicament s'utilitza l'espai HSV per filtrar per tonalitat, independentment de la intensitat o la saturació. Filtrar per RGB implica que els resultats seran inconsistents segons canviï la il·luminació de la escena o la saturació de la textura. Aquest resultat és òptim, ja que es busca que la detecció sigui imperfecta per simular una major quantitat d'errors de detecció, de igual manera que succeiria a la realitat.

Per al processament posterior, s'han encadenat dos processos de morfologia matemàtica per millorar la qualitat del resultat. Primer es realitza una operació de 'closing', que equival a fer una dilatació seguida d'una erosió a tots els píxels de la imatge. Aquesta operació permet reduir el soroll de la detecció a dins de les zones de posidònia. Seguidament s'efectua un 'opening' a la imatge, l'operació dual de 'closing'. L'opening aconsegueix eliminar les deteccions aïllades amb una mida de pocs píxels, considerades soroll.

El processament de la imatge amb aquestes dues operacions utilitzant un operador circular de radi 3 té com a efecte, a més de reduir el soroll de la detecció, suavitzar els contorns detectats. Tant l'operador com l'ordre de les operacions s'han determinat experimentalment, amb la intenció de trobar la combinació que donés un resultat més semblant a la detecció del vídeo demostració.

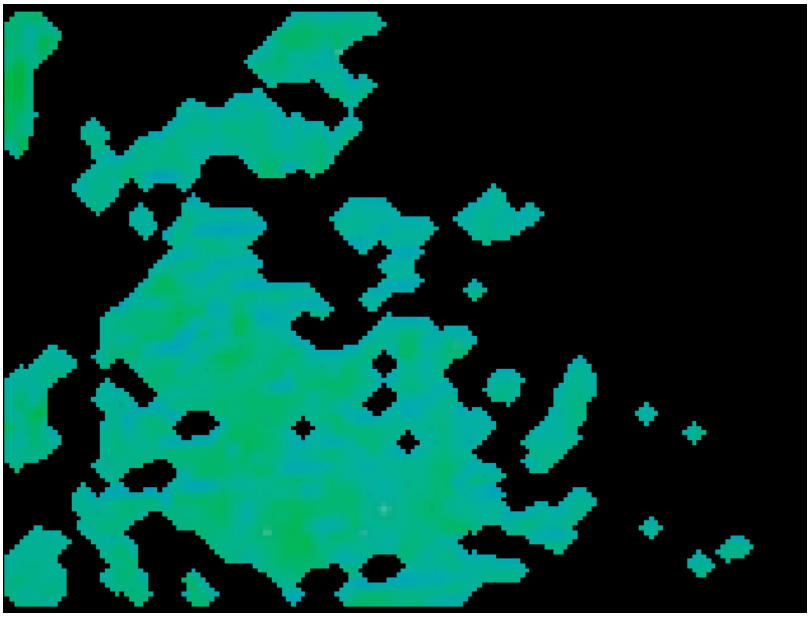

Figura 27. Imatge creada per a la depuració del node “/read_posidonia_cam”. Es poden observar patrons similars als de la figura 23, encara amb una mica de soroll però amb clústers ben definits.

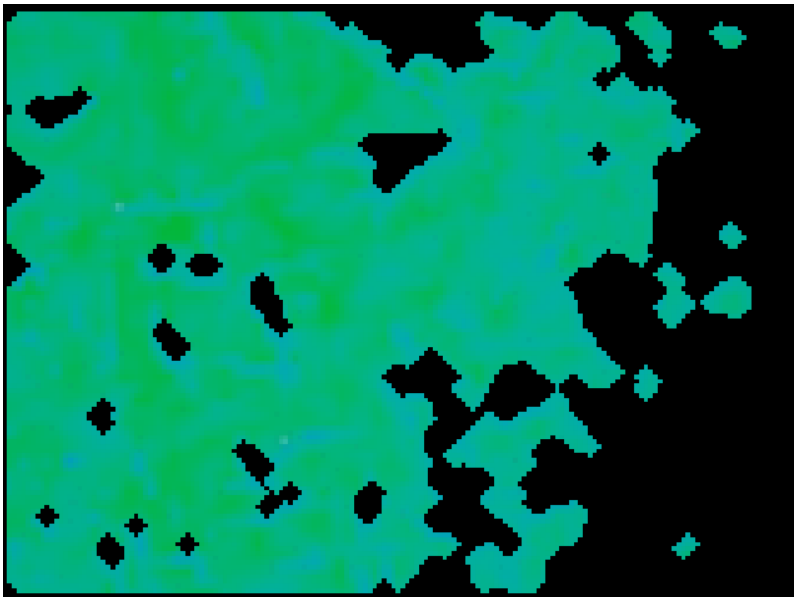

Figura 28. Un altre exemple de detecció de posidònia postprocessada. Aquest exemple s’ha realitzat sobre una zona a l’escenari d’alta densitat de posidònia.

A les imatges anteriors es pot apreciar que tot i que han sigut postprocessades, encara hi apareixen artefactes i soroll. Aquest és el resultat desitjat, ja que la intenció és que l’escenari simulat sigui prou realista.

10.2 Implantació del node “/map_posidonia”

El mapa d’ocupació dinàmic s’ha dissenyat per tenir una mida de cel·la segons els paràmetres definits i les característiques de la càmera. Aquesta mida de cel·la dependrà de l’altitud respecte al fons, el camp de visió de la càmera i un nombre de cel·les ‘n’ que definirà quantes cel·les hi caben a cada imatge. D’aquesta manera, suposant una altitud i càmera constants, es podrà variar la densitat i precisió del mapa només amb el nombre de cel·les que hi caben a la imatge, definit al fitxer launch. Aquesta densitat afectarà en gran mesura al rendiment del programa. La mida de la cel·la serà inversament proporcional al quadrat de cel·les totals al mapa i per tant, a memòria.

Aplicant la següent fórmula es pot obtenir la mida en metres de les cel·les del mapa.

$$mida = 2 \cdot h \cdot \frac{tan\left(\frac{fov}{2}\right)}{n}$$

On h és l’alçada respecte el fons, fov és el camp de visió horitzontal de la càmera en radians i n és el nombre de cel·les que cobrirà una imatge de la càmera en horitzontal. Per a h=5m, fov=0.96rad (55º) i n=16 cel·les, la mida resultant és de 32.5cm.

S’ha de tenir molt en compte que tots aquests valors són entrats al fitxer de configuració, i poden tenir un efecte important en el consum de memòria i processament de la màquina. Tant l’àrea a escanejar com el nombre de cel·les que ha de contenir cada fotograma de la imatge.

Àrea d'escaneig (Km2)	Cel·les càmera	Mida cel·la (m2)	Cel·les totals	Espai a memòria (MB)
0.36	8	0.423	850,216	10
0.36	16	0.106	3,400,864	41
0.36	32	0.026	13,603,454	163
0.36	64	0.007	54,413,817	653

Taula 1. Taula que reflecteix l’efecte del nombre de cel·les que conté un fotograma de la càmera sobre la memòria que ocupa el programa.

Es pot extreure de la taula que no només patirà la memòria, si no que les operacions que iterin per totes les cel·les del mapa seran molt costoses, en cas d’escanejar un àrea extensa o de tenir una resolució del mapa molt elevada.

D’aquestes operacions però, només n’hi ha una a l’inici de l’execució. Aquesta s’encarrega de publicar el mapa sencer sense explorar en un missatge tipus “OccupancyGrid”. Aquest missatge es publica per a informar a RViz de l’existència del mapa i que pugui actualitzar-lo amb les subseqüents publicacions de missatges “OccupancyGridUpdate” que farà el node “/map_posidonia”

Les cel·les de la matriu del mapa dinàmic estan formades per parelles de dades. El valor de la cel·la i el nombre d’observacions d’aquesta (float, int). Cada vegada que es rep un missatge

de detecció i aquest afecta a una cel·la, el valor detectat se sumarà al valor actual de la cel·la i el valor d'observació s'augmentarà en un.

El mapa s'actualitza d'aquesta manera per augmentar la robustesa del resultat front a la naturalesa sorollosa que tenen les dades d'entrada. Aconseguint així que les deteccions provinents de soroll aleatori tendeixin a valor zero. Un segon avantatge és que les zones amb una densitat baixa de posidònia quedaran registrades al mapa amb un valor per sota d'u però per sobre de zero. Això quedarà reflectit al mapa visual de RViz i es podran identificar posteriorment les zones amb baixa i alta densitat de posidònia.

Per publicar una actualització del mapa, s'utilitzen missatges tipus "OccupancyGridUpdate". Aquests missatges estan compostos d'una part del mapa total. Aquesta manera de notificar les actualitzacions del mapa evita enviar-lo sencer i estalvia molts recursos. La mida d'aquest missatge d'actualització vindrà donada pel paràmetre de configuració "update_radius", en metres.

El mapa s'actualitza amb les dades de detecció 10 vegades per segon, però publicarà el missatge d'actualització només una vegada per segon, ja que no és un missatge crític que calgui enviar ràpidament i així s'estalviaran recursos.

El resultat d'aquesta implementació es pot observar amb la següent imatge.

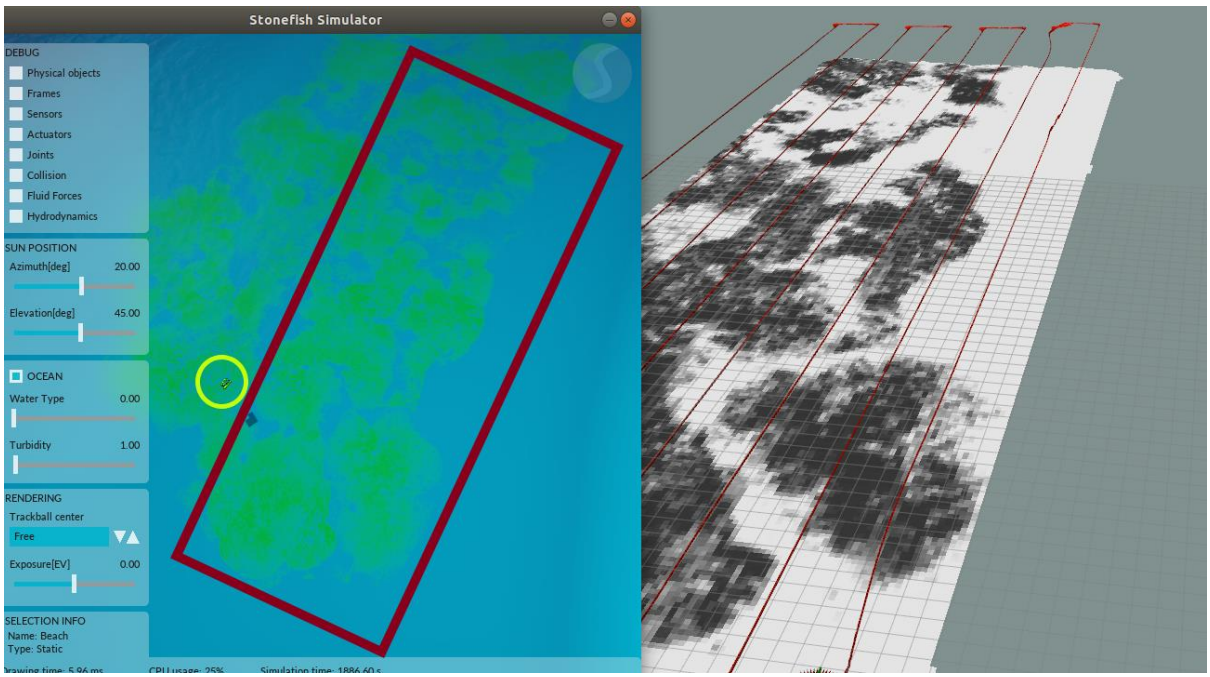

Figura 29. A l'esquerra es pot veure la finestra del simulador. El vehicle encerclat en color verd. La zona explorada està destacada amb un rectangle vermell. A la dreta es pot veure la finestra de RViz, on arriben tots els missatges d'actualització del node "/map_posidonia".

Es pot apreciar que el patró de la posidònia a l'interior del rectangle vermell a la finestra del simulador és molt semblant al patró mapejat pel node "/map_posidonia". Les diferents tonalitats de grisos representen els diferents valors de detecció al mapa. En aquest cas i per testejar la robustesa del mapeig, s'ha efectuat una maniobra de lawnmower.

10.3 Implantació del node “/posidonia_nav”

Aquest node haurà de rebre informació de la navegació i l’estat de les accions dels nodes de l’arquitectura COLA2, “/navigator” i “/pilot”. També rebrà informació sobre el mapa local a mesura que el node “/map_posidonia” publiqui actualitzacions.

Inicialment, el node crearà un conjunt de punts per a executar una maniobra lawnmower segons la configuració que es detecti als paràmetres inicials. Aquests inclouen orientació inicial, elongació, separació entre passades, etc. Mentre no es detecti posidònia, s’anirà executant la maniobra lawnmower, amb l’objectiu de trobar-ne.

Els punts a seguir es publicaran al tòpic “/pilot/world_section_req/goal” amb el tipus de missatge “WorldSectionActionGoal” propi de la llibreria COLA2.

Aquest missatges contenen:

- Un número identificatiu per diferenciar-los.
- Un punt inicial des d’on es comença l’acció.
- Un punt final on acabarà l’acció.
- La tolerància de proximitat per a considerar que s’ha arribat al punt.
- El mode de mantenir la profunditat, en altitud o profunditat.
- En temps de seguretat per considerar que hi ha un problema si el vehicle no hi arriba a temps al punt final.
- Velocitat de navegació

Amb tota aquesta informació, el node “/pilot” de l’arquitectura executarà l’acció indicada. El missatge seguirà sent enviat a una freqüència de 10Hz fins que l’acció sigui completada. Un cop sigui completada, el node “/pilot” publicarà al tòpic “/pilot/world_section_req/result” amb un missatge tipus “WorldSectionActionResult”. Aquest missatge contindrà el resultat de l’acció, que pot ser que l’acció ha acabat correctament, que ha sigut cancel·lada, o que el temps per executar l’acció s’ha exhaurit.

En cas de que l’acció es completi correctament, el node “/posidonia_nav” crearà i publicarà un nou missatge tipus “WorldSectionActionGoal” amb el següent punt de la maniobra lawnmower.

En qualsevol moment d’aquest procés, si suficient posidònia és detectada com per no considerar-se soroll, es cancel·larà tota la maniobra i es canviarà d’estratègia per seguir el contorn de la praderia.

A la següent imatge es pot observar el resultat d’una trajectòria lawnmower executada pel node “/posidonia_nav”. S’ha desactivat la detecció de posidònia per poder observar el funcionament de la trajectòria correctament sense que s’interrompi.

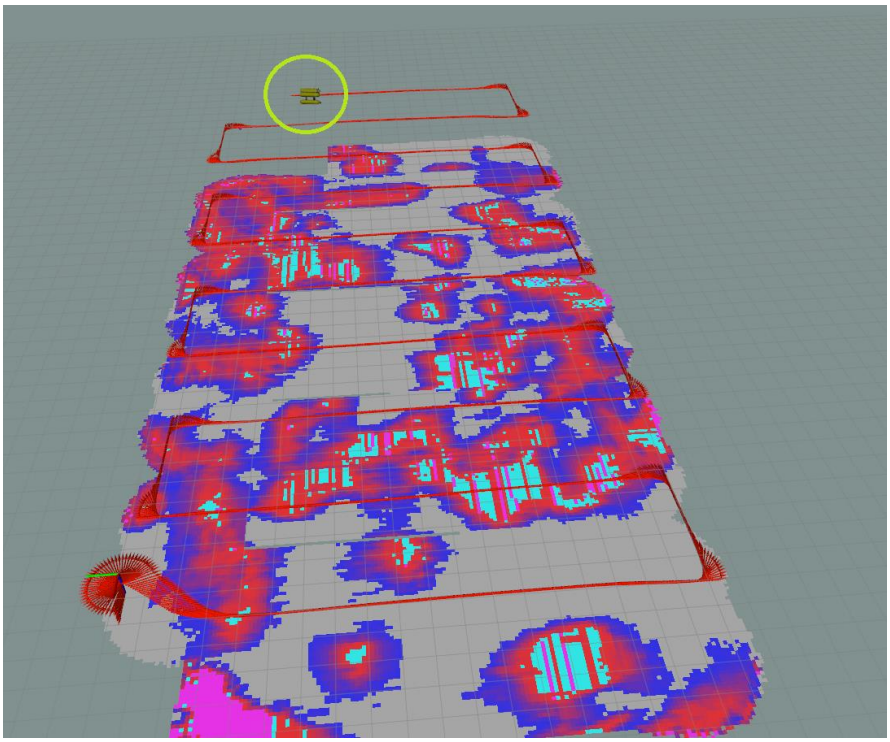

Figura 30. Captura de la finestra de RViz on es pot observar el vehicle encerclat en verd i la trajectòria tipus lawnmower que està seguint. A diferència de la figura 27, en aquesta imatge el mapa és representat en escala de colors, per detectar més fàcilment les diferències en densitat de la praderia de posidonia.

Per cancel·lar l'acció que el vehicle està realitzant en aquell moment, cal publicar en el tòpic "/pilot/world_section_req/cancel" un missatge tipus "GoalID". Aquest tipus de missatge no precisa de cap tipus de contingut. Per assegurar-se de que el node "/pilot" cancel·la correctament l'acció, aquest missatge s'envia 10 vegades durant 1 segon.

Un cop l'acció ha sigut cancel·lada, es pot iniciar l'estratègia de buscar candidats per a l'exploració de contorns de la praderia. Aquesta exploració donarà prioritat sempre als candidats que es trobin a la dreta del vehicle. Així es pot assegurar que el vehicle deixarà el contorn sempre a la seva esquerra i que la trajectòria sempre acabi sent circular. La implementació i algorísmica de la cerca de cel·les candidates a exploració ja han sigut explicades a la secció d'algorismes.

Un cop el vehicle ha decidit quina cel·la ha d'explorar, cal que es mogui en aquesta direcció. Donat el fet de que les cel·les a explorar normalment són cel·les frontera trobades just a l'extrem de la detecció, per a explorar-les caldrà ser precís. No tindrà sentit moure el vehicle 3 metres per a explorar una cel·la que ja es troba a l'extrem del camp visual de la càmera. Per aquest motiu, no es farà servir el missatge tipus "WorldSectionActionGoal" com al executar la maniobra lawnmower.

En aquest cas, es farà servir un control de més baix nivell amb missatges tipus "BodyVelocityReq" publicats al tòpic "/controller/body_velocity_req". Tal i com indica el nom del tòpic, el node que rebrà i interpretarà aquest missatge serà el node "/controller".

Aquest tipus de missatge està format per les velocitats desitjades en cada un dels eixos cartesians i angulars. També hi contindrà les consignes booleanes dels eixos que es faran servir per a la maniobra.

Les velocitats indicades al missatge seran regulades segons la distància i orientació del vehicle respecte a la cel·la que es vol explorar. Les fórmules que dictaran aquest control són les següents:

$$v_x = d * e^{-(2\varphi)^2}$$
$$\omega_z = \frac{\varphi}{5}$$

On:

- v_x és la velocitat a l'eix x del robot, que equival al moviment frontal. Com a màxim serà de 1.0 m/s
- ω_z és la velocitat angular en l'eix z. Com a màxim serà de 0.4 rad/s.
- φ és l'angle format entre la direcció actual del vehicle i la línia que creua el vehicle i el punt destí.
- d és la distància des de la posició del vehicle fins al punt destí.

Aquestes fórmules s'han trobat de manera experimental.

Cal dir que si bé aquest mètode per seguir el contorn de la praderia de posidònia és més efectiu que el primer mètode, comentat a la secció de implementació i proves, no és infal·lible. En certes ocasions, normalment provocades pel soroll de la detecció, el vehicle es mourà cap a una direcció on deixaran d'existir possibles candidats. En aquest punt, l'algorisme no pot continuar.

Per ajudar a l'algorisme a continuar en aquesta situació, s'ha mantingut implantat també el primer mètode per seguir contorns. El mètode que utilitza directament la matriu de detecció publicada pel node "/read_posidonia_cam". Aquest mètode tindrà una prioritat secundària i només afectarà al control del vehicle en cas de que aquest no tingui cel·les candidates a exploració.

El resultat d'aquesta combinació és que el vehicle és totalment capaç de recórrer els 1.200 metres del perímetre de la praderia de posidònia en 64 minuts satisfactòriament. En la següent seqüència d'imatges es pot observar en detall aquest fet.

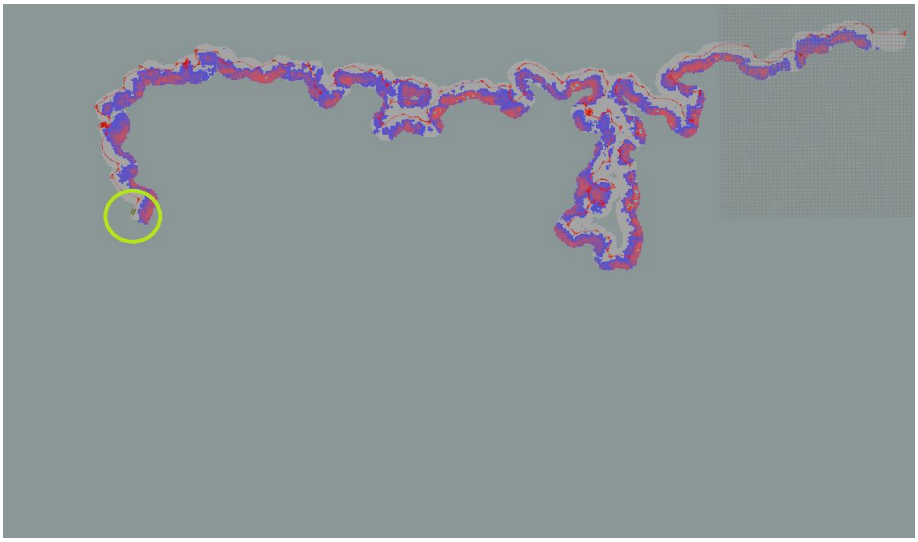

Figura 31. Exploració de la praderia, 1/3. Aproximadament 25 minuts. El vehicle es troba a l'interior del cercle verd.

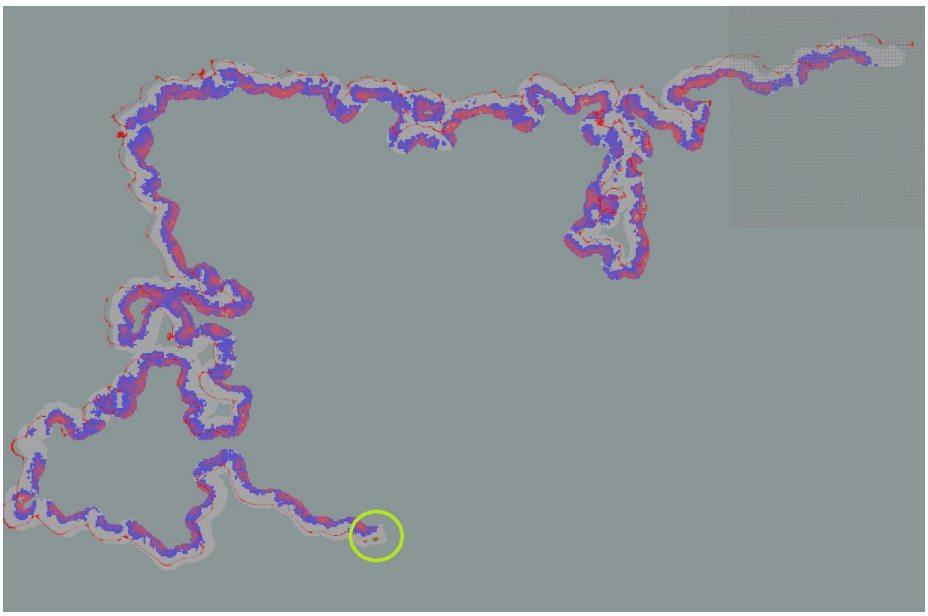

Figura 32. Exploració de la praderia, 2/3. Aproximadament 40 minuts. El vehicle es troba a l'interior del cercle verd.

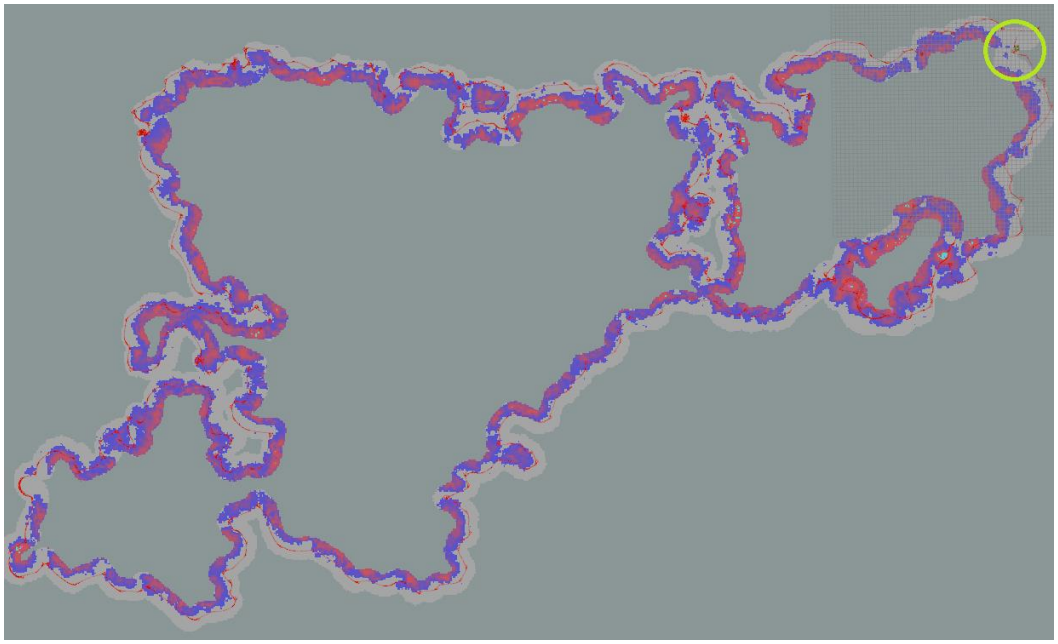

Figura 33. Exploració de la praderia, 3/3. Aproximadament 64 minuts. El vehicle es troba a l'interior del cercle verd. S'ha recorregut tot el contorn de la praderia.

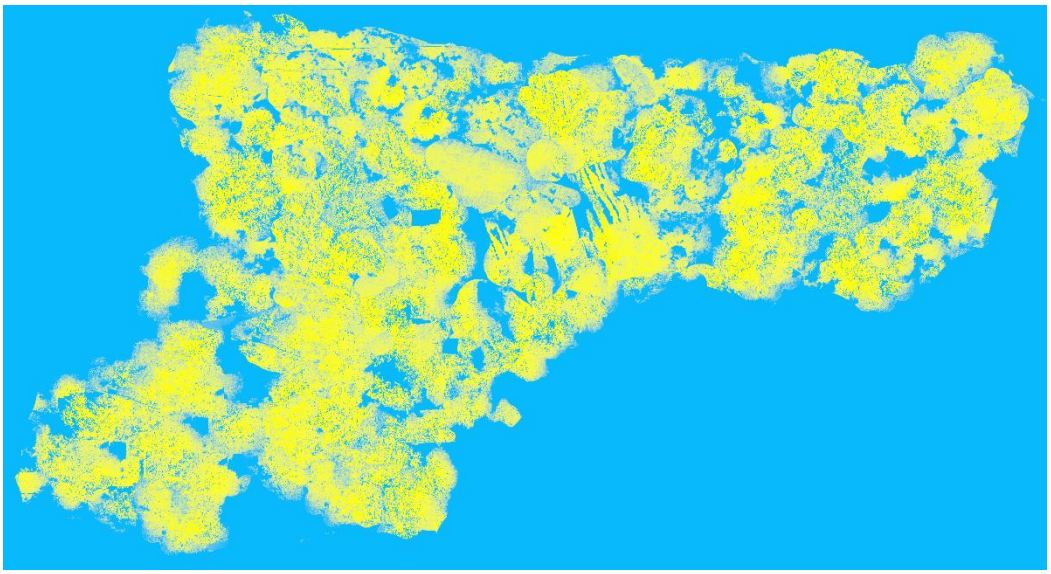

Figura 34. Imatge de la textura utilitzada pel terreny de la simulació. Afegida per facilitar la comparació amb el resultat final.

10.4 Comparant resultats amb el Lawnmower

Quan es tracta de mapejar un àrea específica, la trajectòria tipus lawnmower és per norma general la millor estratègia a seguir, ja que cobreix tots els punts de l'àrea definida i només passa per tots ells un cop. Ara bé, si l'àrea a mapejar és desconeguda, com és el cas de trobar les zones on hi ha posidònia i mapejar només aquestes, el lawnmower deixa de ser el mètode idoni. S'espera que amb el mètode implantat de seguir els contorns de la praderia de posidònia el temps d'exploració de la mateixa es redueixi dràsticament en comparació a executar un lawnmower sense coneixement previ de la zona.

Per comparar aquests dos mètodes, s'ha executat una maniobra de lawnmower rectangular al mateix escenari i cobrint tota la praderia de posidònia. Utilitzant la interfície gràfica IQUAview, s'ha preparat aquesta missió tipus lawnmower a la mateixa àrea que anteriorment s'ha executat l'algorisme de seguir contorns. Això és possible gràcies a que IQUAview registra la trajectòria del vehicle en coordenades georeferenciades i permet guardar-les i representar-les al mapa. Amb la eina que incorpora per crear missions tipus lawnmower, s'ha creat una just cobrint l'àrea amb posidònia.

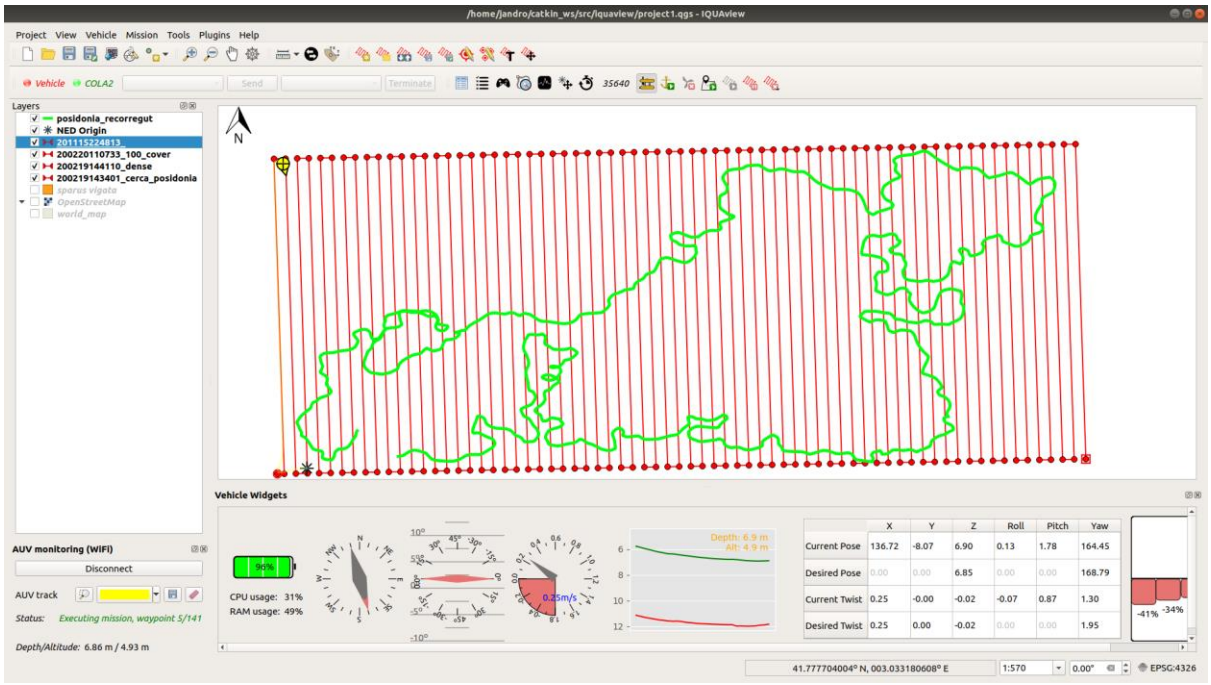

Figura 35. Interfície gràfica IQUAview. Es pot observar al panell principal la trajectòria que ha seguit el vehicle per seguir el contorn de posidònia en verd. La maniobra lawnmower creada en vermell.

S'ha creat aquesta maniobra amb una un solapament entre passades del 50%. Aquest és un valor típic per a aquesta maniobra. La separació resultant entre passades és de 4 metres. El temps total per a executar la maniobra és de aproximadament 6 hores. Mentre que el temps que ha trigat el vehicle en recórrer el perímetre de la praderia és de només 1 hora.

Si finalment calgués escanejar tota la praderia i no només el contorn, serà molt més eficient primer escanejar el contorn i després preparar una maniobra lawnmower adaptada al perfil

de la praderia, ja coneixent els contorns. Així s'estalviarà tot el temps d'escaneig de zones on no hi ha posidònia.

11 Conclusions

El sistema desenvolupat pel Girona500 compleix amb l'objectiu inicial de ser capaç de seguir i mapejar autònomament una praderia de posidònia.

La mida i disseny de l'escenari implementat han comportat que sigui tot un repte aconseguir que el vehicle l'explorés sencer. Les formes aparentment aleatòries típiques de la posidònia han dificultat molt l'exploració. Ha calgut invertir moltes hores testejant el comportament del vehicle per a trobar la combinació de paràmetres apropiats per una exploració òptima i completa.

També ha sigut molt útil tenir dues implementacions d'exploració alhora. Mantenir el primer disseny d'exploració, tot i que no del tot eficaç, encara és útil. Quan el mètode de buscar candidats a explorar falla i no troba cap candidat idoni, el vehicle és capaç de seguir el contorn de la zona ja explorada fins a trobar cel·les inexplorades candidates.

Idealment, es podria analitzar el mapa sencer un cop el vehicle no trobés cap punt candidat en el mapa local per continuar l'exploració, però amb la mida de l'escenari plantejat, això no seria possible fer-ho en temps real.

Tot i que al final s'ha treballat menys amb els vehicles reals del que s'esperava, s'han pogut obtenir bons resultats que fàcilment es podran integrar al vehicle i serviran sense dubte a experiments futurs en la matèria.

Amb aquest treball he pogut aprendre moltíssim sobre els vehicles autònoms submarins i dels companys i companyes del CIRS. Tot i que el meu temps al CIRS va ser breu per la crisi de la COVID-19, vaig poder aprendre molt de tots ells.

12 Treball futur

Com a principal treball futur faltaria testejar el sistema desenvolupat en un entorn real amb un detector real. Això, tot i que va ser plantejat a l'inici del projecte, a principis d'any, de seguida es va veure que no acabaria sent possible per factors externs al mateix.

Caldria modificar el codi per fer-lo funcionar amb el mòdul d'intel·ligència artificial, en el cas de que no fos compatible d'entrada. També caldran múltiples sortides al mar per aconseguir resultats reals i modificar els algorismes i fórmules en concordança. Tot i que el simulador intenta ser realista, no ho és del tot. Així doncs, és molt probable que calgui modificar el sistema un cop es comencin a fer proves al mar. Si ha de canviar molt o poc encara està per veure.

A 22 de Novembre, les actualitzacions que han patit tant l'arquitectura COLA2 com el simulador Stonefish impedeixen l'execució correcte del codi desenvolupat. És per això que com a treball futur caldrà fer el projecte compatible amb els canvis esmentats.

També m'hauria agradat afegir l'exploració de praderies de posidònia com a funcionalitat al programa de codi obert IQUAview. Ja que vaig estar fent pràctiques a IQUA Robotics afegint diferents funcionalitats a l'aplicació. Es podria afegir, per exemple, que el mapa dinàmic fos rebut per la interfície visual IQUAview i mostrat per pantalla juntament al mapa de la zona. Així l'operari podria assegurar-se de que el vehicle no pateix perill per passar a prop de zones poc profundes o obstacles coneguts.

13 Bibliografia

1. Marba, Nuria & Diaz-Almela, Elena & Duarte, Carlos. (2014). Mediterranean seagrass (Posidonia oceanica) loss between 1842 and 2009. Biological Conservation. 176. 183–190. 10.1016/j.biocon.2014.05.024.
2. UDG – Taules Retributives. (Online, 15-08-2020) https://www.udg.edu/ca/transparencia/Transparencia/Personal/Plantilla/Retribucions
3. Scrum – What is Scrum? (Online, 19-08-2020) https://www.scrum.org/resources/what-is-scrum
4. Wikipedia – Iterative and incremental development. (Online, 24-08-2020) https://en.wikipedia.org/wiki/Iterative_and_incremental_development
5. Eurofleets+ - Project Information. (Online, 24-08-2020) https://www.eurofleets.eu/project-information/
6. Wikipedia - Posidonia Oceanica. (Online, 24-08-2020) https://en.wikipedia.org/wiki/Posidonia_oceanica
7. Enciclopedia Of Life – Mediterranean Tapeweed. (Online, 24-08-2020) https://eol.org/pages/1089001
8. Directiva 92/43/CEE del Consejo – Relativa a la conservación de los hábitats naturales y de la fauna y flora silvestres https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:1992L0043:20070101:ES:PDF (Online, 24-08-2020).
9. EOS – SENTINEL-2 landviewer. (Online, 02-11-2020) https://eos.com/landviewer/?aoi_location=default&lat=38.86456&lng=1.34318&z=13&tool-timelapse=&id=S2B_tile_20201029_31SCD_0&b=Red,Green,Blue&anti
10. MasMallorca – Posidonia oceanica: Destrucción por fondeos y su concepción como delito ambiental en las Illes Balears. https://www.masmallorca.es/wp-content/uploads/2018/09/2018_04_23_Morelle_Posidonia-Delito_Baleares.pdf (pdf, 23-04-2018).
11. RAC/SPA - UNEP/MAP, 2014. Monitoring protocol for Posidonia oceanica beds. By Guala I, Nikolic V, Ivesa L, Di Carlo G, Rajkovic Z, Rodic P, Jelic K. Ed. RAC/SPA - MedMPAnet Project, Tunis. 37 pages + annexes
12. R. M. Eustice, O. Pizarro and H. Singh, "Visually Augmented Navigation for Autonomous Underwater Vehicles," in IEEE Journal of Oceanic Engineering, vol. 33, no. 2, pp. 103-122, April 2008, doi: 10.1109/JOE.2008.923547.
13. Ramp, S.R., Davis, R.E., Leonard, N.E., Shulman, L., Chao, Y., Robinson, A., Marsden, J., Lermusiaux, P.F., Fratantoni, D.M., Paduan, J.D., et al., 2009. Preparing to predict: The secound autonomous ocean sampling network (AOSN-II) experiment in the Monterey bay. Deep Sea Res. II: Top. Stud. Oceanogr. 56 (3), 68-86.
14. Ridao, P., Carreras, M., Ribas, D. and Garcia, R. (2010), Visual inspection of hydroelectric dams using an autonomous underwater vehicle. J. Field Robotics, 27: 759-778. doi:10.1002/rob.20351
15. Moniruzzaman M., Islam S.M.S., Bennamoun M., Lavery P. (2017) Deep Learning on Underwater Marine Object Detection: A Survey. In: Blanc-Talon J., Penne R., Philips W., Popescu D., Scheunders P. (eds) Advanced Concepts for Intelligent Vision Systems. ACIVS 2017. Lecture Notes in Computer Science, vol 10617. Springer, Cham. https://doi.org/10.1007/978-3-319-70353-4_13

16. Wikipedia - Extended Kalman Filter, (Online, 25-08-2020) https://en.wikipedia.org/wiki/Extended_Kalman_filter
17. Cyrill Stachniss - Extended Kalman Filter. (Online, 7-09-2020) http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam04-ekf.pdf
18. CIRS Blog – DEEPFIELD Thematic Workshop: Autonomous Underwater Intervention. (Online, 16-09-2020) https://cirs.udg.edu/deepfield-thematic-workshop-autonomous-underwater-intervention/
19. CIRS – Sparus II AUV. (Online, 23-10-2020) https://cirs.udg.edu/auvs-technology/auvs/sparus-ii-auv/
20. IQUA Robotics – Girona500 AUV. (Online, 24-10-2020) http://iquarobotics.com/girona-500-auv
21. ROS – About ROS. (Online, 25-10-2020) https://www.ros.org/about-ros/
22. Wikipedia – Real-time operating system. (Online, 25-10-2020) https://en.wikipedia.org/wiki/Real-time_operating_system
23. ROS – Real-time programming in ROS 2. (Online, 26-10-2020) https://index.ros.org/doc/ros2/Tutorials/Real-Time-Programming/
24. IQUA Robotics – IQUAview Graphical User Interface. (Online, 27-10-2020) http://iquarobotics.com/iquaview-graphical-user-interface
25. Patryk Cieślak, "Stonefish: An Advanced Open-Source Simulation Tool Designed for Marine Robotics, With a ROS Interface", In Proceedings of MTS/IEEE OCEANS 2019, June 2019, Marseille, France
26. udg_cirs – cola2_stonefish. (Online, 04-11-2020) https://bitbucket.org/udg_cirs/cola2_stonefish/src/master/
27. Blender – Blender. (Online, 04-11-2020) https://www.blender.org/download/
28. Gimp – GNU Image Manipulation Program. (Online, 05-11-2020) https://www.gimp.org
29. A.Barcelona, J. Colomer, M. Soler, N. Gracias and T. Serra “Meadow fragmentation determines Posidonia oceanica density at the edge of nearby gaps”, submitted to Estuarine, Coastal and Shelf Science, 2020
30. Vidal Garcia, Eduard & Palomeras, N. & Istenič, Klemen & Gracias, Nuno & Carreras, Marc. (2020). Multisensor online 3D view planning for autonomous underwater exploration. Journal of Field Robotics. 10.1002/rob.21951.

14 Manual d'instal·lació

Per instal·lar i executar el software desenvolupat en aquest projecte caldrà primerament instal·lar ROS Melodic.

Per instal·lar ROS Melodic caldrà executar els següents passos.

- Afegir els paquets de ros.org a la llista de sources d'Ubuntu:
  - `sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'`

- Afegir la clau de l'aplicació dels servidors d'Ubuntu:
  - `sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654`

- Instal·lació de ROS Melodic:
  - `sudo apt update`
  - `sudo apt install ros-melodic-desktop-full`

- Instal·lació de paquets extres necessaris per l'arquitectura COLA2:
  - `sudo apt install ros-melodic-rosbridge-server ros-melodic-joy lm-sensors lcov python-ruamel.yaml`

- Afegir les variables d'entorn de ROS per defecte a bash:
  - `echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc`
  - `source ~/.bashrc`

- Es crearà un Catkin Workspace que contindrà els paquets de ROS que caldran per a l'execució:
  - `mkdir -p ~/catkin_ws/src`
  - `cd ~/catkin_ws/`
  - `catkin_make`

- Afegir les variables d'entorn del Catkin Workspace per defecte a bash:
  - `echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc`
  - `source ~/catkin_ws/devel/setup.bash`

Seguidament, caldrà instal·lar l'arquitectura COLA2 dels vehicles. Per assegurar compatibilitat, es descarregaran les versions compatibles amb el projecte desenvolupat.

- En cas de que no tenir GIT instal·lat, executar:
  - `sudo apt install git`

- Instal·lar cola2_lib:
  - `cd ~`
  - `git clone --depth 1 --branch 3.2.2 https://bitbucket.org/iquarobotics/cola2_lib.git`
  - `cd cola2_lib`
  - `mkdir build`

- `cd build`
- `cmake ..`
- `make`
- `sudo make install`

- Descarregar tots els paquets necessaris de l'arquitectura al directori de Catkin Workspace:
  - cd ~/catkin_ws/src
  - git clone --depth 1 --branch 3.2.0 https://bitbucket.org/iquarobotics/cola2_msgs
  - git clone --depth 1 --branch 3.2.3 https://bitbucket.org/iquarobotics/cola2_core
  - git clone --depth 1 --branch 3.2.0 https://bitbucket.org/iquarobotics/cola2_lib_ros
  - git clone --depth 1 --branch 3.2.1 https://bitbucket.org/iquarobotics/cola2_girona500
  - git clone --depth 1 --branch 3.2.0 https://bitbucket.org/iquarobotics/girona500_description

- Compilar els paquets per completar la instal·lació de l'arquitectura:
  - `cd ~/catkin_ws`
  - `catkin_make`
  - `source ~/catkin_ws/devel/setup.bash`

Arribats a aquest punt, caldria comprovar que tot s'ha instal·lat correctament.
Per comprovar-ho, s'executarà l'arquitectura amb el simulador bàsic que porta per defecte.

- Per iniciar el simulador incorporat a l'arquitectura:
  - `roslaunch cola2_girona500 sim_start.launch`

Aquesta comanda obrirà tots els nodes de l'arquitectura i RViz on es podrà veure al vehicle representat en 3D.

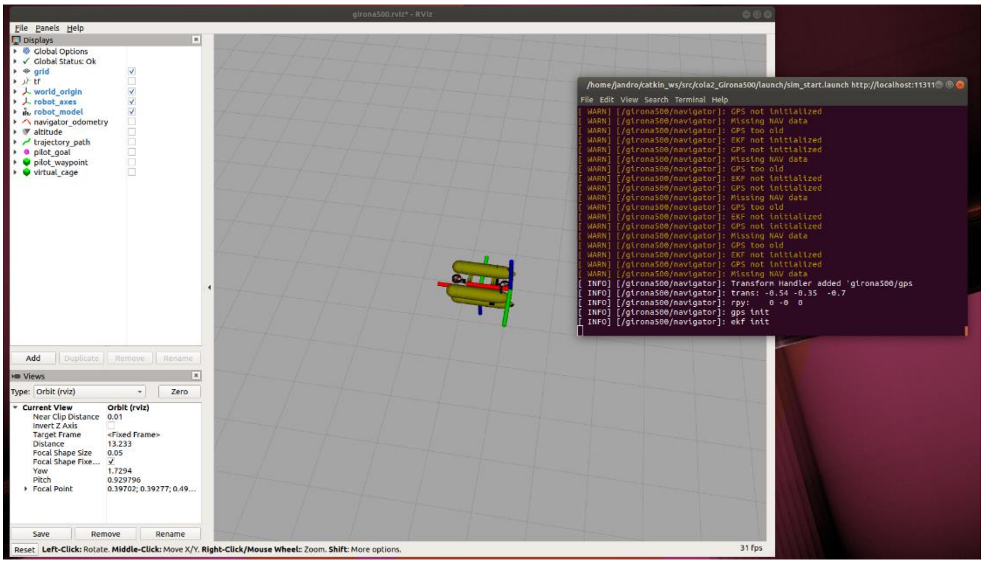

Figura 36. Visualitzador RViz i arquitectura COLA2 en funcionament. S'obrirà automàticament al executar la comanda anterior.

A continuació caldrà instal·lar l'eina de simulació Stonefish i les seves dependències.

- Dependències d'Stonefish:
  - `sudo apt-get install libglm-dev`
  - `sudo apt-get install libsdl2-dev`

- Instal·lació d'Stonefish:
  - `cd ~`
  - `git clone https://github.com/patrykcieslak/stonefish.git`
  - `cd stonefish`
  - `git checkout dd6a3c29eaaf005b10b74b7270d005b4ee0c2036`
  - `mkdir build`
  - `cd build`
  - `cmake ..`
  - `make -j4`
  - `sudo make install`

- Instal·lació del paquet ROS d'Stonefish:
  - `cd ~/catkin_ws/src`
  - `git clone https://github.com/patrykcieslak/stonefish_ros`
  - `cd stonefish_ros`
  - `git checkout bfe0862c00080601c18da3f5fd7277d257e711f2`
  - `cd ~/catkin_ws`
  - `catkin_make`
  - `source ~/catkin_ws/devel/setup.bash`

- Per últim, caldrà descarregar el paquet ROS desenvolupat per aquest projecte:
  - `cd ~/catkin_ws/src`
  - `git clone https://bitbucket.org/Jandro19/posidonia_search_sim.git`
  - `cd ~/catkin_ws`
  - `catkin_make`
  - `source ~/catkin_ws/devel/setup.bash`

Un cop estigui tot instal·lat, només caldrà executar el l'escenari simulat i els nodes desenvolupats.

- Per executar el projecte, executar les següents comandes en 2 terminals diferents:
  - `roslaunch posidonia_search_sim posidonia_search_simulation.launch`
  - `roslaunch posidonia_search_sim find_edges.launch`

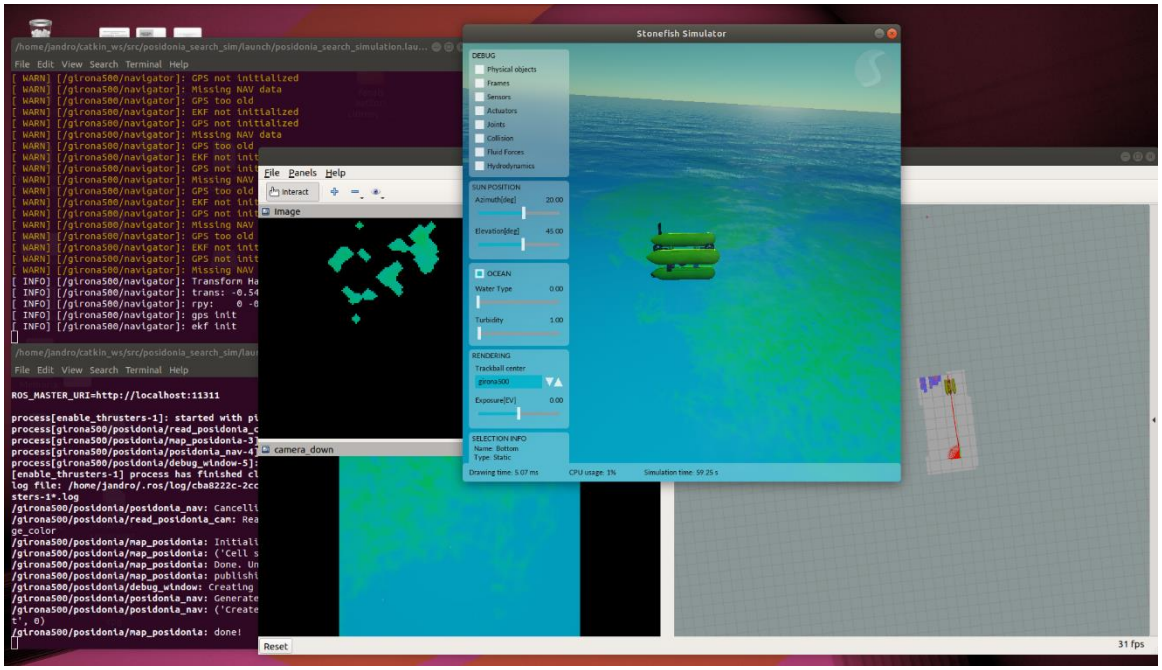

Figura 37. Al executar les comandes anteriors s'obriran les finestres de la imatge. La finestra del simulador Stonefish i la finestra del visualitzador RViz amb la configuració del projecte ja definida.