Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Electrònica Industrial i Automàtica

Títol: Integració de timons en el robot SPARUS II pel control en cinc graus de llibertat

Document: 1. Memòria

Alumne: Èric Pairet Artau

Tutor: Marc Carreras Pérez
Departament: Arquitectura i Tecnologia de Computadors
Àrea: Arquitectura i Tecnologia de Computadors

Convocatòria (mes/any): setembre/2015

ÍNDEX

1. INTRODUCCIÓ 6

1.1. Antecedents 6

1.2. Objecte 9

1.3. Abast 9

2. ESTUDI TEÒRIC DELS TIMONS DE PROFUNDITAT 11

2.1. Els timons de profunditat de l'SPARUS II 11

2.1.1. Compartiment dels timons de profunditat 11

2.1.2. Propietats dels perfils NACA 12

2.2. Model dinàmic teòric 14

2.2.1. Influència del motors sobre els timons de profunditat 16

2.2.2. Forces resultants 18

2.3. Efectes dels timons de profunditat en el robot 19

3. HARDWARE 23

3.1. Servomotors 23

3.1.1. Paràmetres elèctrics teòrics 25

3.1.2. Paràmetres elèctrics reals 26

3.2. Sensor angular 27

3.3. Detecció d'aigua 29

3.4. Placa electrònica 29

4. PROTOCOL DE COMUNICACIÓ 33

4.1. Bases del protocol de comunicació 33

4.1.1. Comandes d'execució 34

4.1.2. Comandes d'usuari 36

4.2. Gestió del bus de comunicació 37

5. FIRMWARE 40

5.1. Fase de configuració general 40

5.2. Fase d'inicialització 41

5.3. Fase principal 41

5.3.1. Gestió del bus 42

5.3.2. Generació dels senyals de control 44

5.3.3. Lectura del sensor d'aigua 46

6. MODELITZACIÓ 48

6.1. Modelització dels motors 48

6.2. Modelització dels timons de profunditat 53

7. CONTROL 58

7.1. Control de posició 59

7.1.1. Control de profunditat amb pitch 64

7.2. Control de velocitat 70

7.3. Control de força 73

7.3.1. Control de força dels motors 74

7.3.2. Control de força del timons de profunditat 77

7.4. Sintonització del control 78

8. PROVES REALITZADES I RESULTATS 81

8.1. Proves parcials 81

8.1.1. Proves de hardware 81

8.1.2. Proves de protocol i firmware 83

8.1.3. Proves d'integració 84

8.1.4. Proves de control 86

8.2. Proves finals 89

8.2.1. Primer assaig general del control 90

8.2.2. Segon assaig general del control 94

8.2.3. Tercer assaig general del control 98

8.2.4. Quart assaig general del control 100

8.2.5. Assaig del control de profunditat amb pitch 102

9. RESUM DEL PRESSUPOST 108

10. CONCLUSIONS 109

11. RELACIÓ DE DOCUMENTS 110

12. BIBLIOGRAFIA 111

13. GLOSSARI 114

A. MANUAL DE LA PLACA ELECTRÒNICA 116

B. CÀLCULS 118

C. PROGRAMACIÓ DEL FIRMWARE 120

C.1. Fitxer de configuració 120

C.2. Programa principal 121

D. PROGRAMACIÓ DEL DRIVER 132

D.1. Fitxers de configuració 132

D.1.1. Configuració port sèrie 132

D.1.2. Configuració actuadors 132

D.2. Programa principal 132

E. PROGRAMACIÓ DEL CONTROL 157

E.1. Fitxer de configuració 157

E.2. Programes principals 160

E.2.1. Programa del PID 160

E.2.2. Programa de l'estructura de control 163

E.2.3. Programa del model dels motors 174

E.2.4. Programa del model dels timons de profunditat 183

1. INTRODUCCIÓ

1.1. Antecedents

Des de l'inici dels temps l'ésser humà ha tingut la necessitat d'entendre el perquè de l'existència de les coses i la raó de tot allò que passa al seu voltant, pel que ha hagut d'estudiar, descobrir i enginyar tot el que a dia d'avui coneixem.

Començant pels aspectes més vitals i imprescindibles fins als avanços tecnològics purament lúdics, tots han sigut fruit de la necessitat d'evolucionar que caracteritza a l'humà. Tot i això no totes les persones han tingut els mateixos camps d'interès.

Al llarg de la història pràcticament tothom, amb més o menys mesura, s'ha interessat amb el que succeeix sobre l'escorça terrestre. Alguns d'aquests també s'han vist atrets pels misteris que s'amaguen en tot el que queda per sobre nostre, incloent-hi l'espai, però ben pocs s'han submergit per a descobrir el que hi ha sota el nivell zero del planeta terra.

Quan sembla que es coneix totalment el planeta on vivim, es pot afirmar que del 71 % de la superfície de la terra, la qual està formada per aigua, se'n té un coneixement pràcticament nul. És més, s'estima que únicament es coneix un 3 % de tot el volum d'aigua existent.

El CIRS (Centre d'Investigació en Robòtica Submarina) és un laboratori que forma part de l'institut VICOROB (Visió per Computador i Robòtica) de la UdG (Universitat de Girona), que des de l'any 1995 es troba al Parc Científic i Tecnològic de la mateixa universitat abocat a la investigació en el camp de la robòtica submarina.

Figura 1. Instal·lacions del CIRS

Amb dues dècades d'investigació, el CIRS ha desenvolupat sis robots submarins i ha participat en varis projectes europeus i nacionals. D'un d'aquests va sorgir el primer robot dissenyat, un ROV (Remotely Operated Vehicle). Aquest tipus de vehicle es caracteritza per disposar d'una connexió física que permet controlar-lo en tot moment.

Degut a les limitacions existents en un ROV, els posteriors robots es van dissenyar com a AUVs (Autonomous Underwater Vehicle), el que significa que no hi ha cap connexió física i per tant, el robot és capaç d'operar de forma autònoma.

A partir de l'experiència adquirida al llarg dels anys i amb els diferents robots, al 2011 es va dissenyar el GIRONA 500 (Generic Intelligent Robot Operated and Navigated Autonomously) (Ribas et al., 2012). Aquest AUV a dia d'avui encara està totalment operatiu.

Figura 2. El GIRONA 500

Al 2013 es va crear l'SPARUS II (Carreras et al., 2013), una versió àmpliament millorada d'un robot dissenyat al 2010, l'SPARUS. Des de llavors aquest robot submarí s'ha millorat de forma ininterrompuda per adequar-se a les exigències dels diferents projectes existents al CIRS. A continuació es pot observar la seva aparença abans d'integrar els timons.

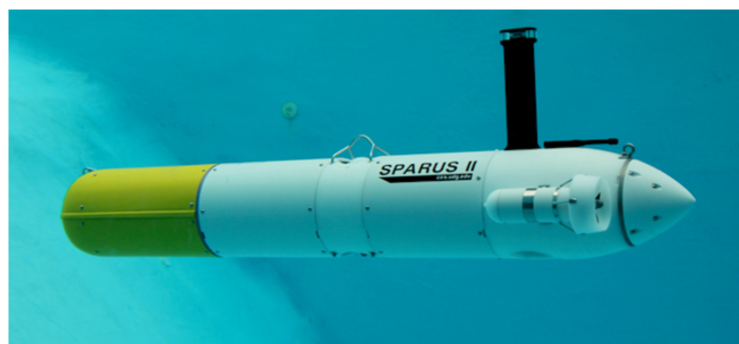

Figura 3. L'SPARUS II

L'SPARUS II és un AUV desenvolupat íntegrament al CIRS que es caracteritza per ser submergible fins a 200 metres de profunditat, ser hidrodinàmic gràcies a la seva forma de torpede, per ser de reduïdes dimensions, per disposar d'una gran autonomia i per ser ajustable segons les necessitats de cada moment tant a nivell de hardware com de software.

Estructuralment el robot es pot dividir en dues grans parts. Una a la zona anterior, la qual rep el nom de payload, permet instal·lar al vehicle sensors específics imprescindibles per a resoldre una tasca. En aquesta zona l'aigua hi pot entrar lliurament, pel que els sensors, conductors i connexions han d'estar dissenyats per aquest ús. L'altra part del robot consisteix en un cilindre estanc on hi ha tots els sensors navegació, les bateries de lithium-ion, un PC (Personal Computer) i l'electrònica addicional pel funcionament del vehicle.

El robot té tres DOFs (Degree Of Freedom): és propulsat en surge i orientat en yaw a partir de dos motors ubicats horitzontalment a la part posterior del robot i es mou en heave amb un motor disposat verticalment al centre de gravetat del submarí.

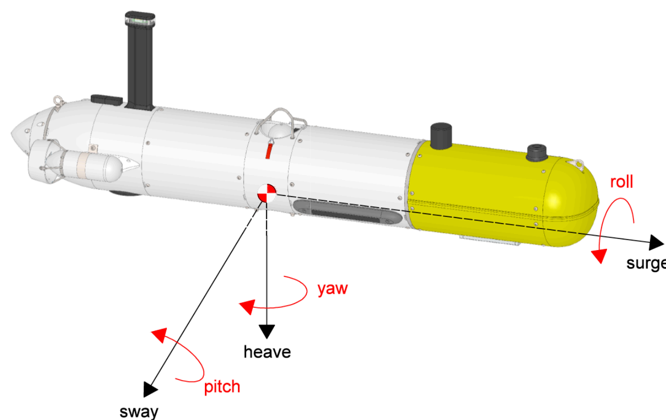

Figura 4. Sistema de coordenades del robot

El cervell del robot és l'ordinador central, que entre altres aspectes, és l'encarregat de la navegació i control del robot, és a dir, de computar totes les dades provinents dels sensors de navegació per a proporcionar un senyal de posicionament i velocitats al sistema de control. Aquest, a partir de les consignes de posició, velocitat o força i del senyal de realimentació esmentat, és capaç de generar consignes pels tres motors.

Per a poder desenvolupar totes aquestes funcions l'ordinador disposa d'una plataforma Ubuntu en la que hi corre el sistema operatiu ROS (Robot Operating System), el qual facilita el desenvolupament d'algorismes destinats a sistemes robotitzats. Aquest sistema operatiu

anomena nodes a cada un dels programes individuals existents, els quals es comuniquen entre ells a través d'informació publicada a diferents tòpics.

Considerant la quantitat de nodes necessaris pel funcionament d'un robot, el CIRS utilitza l'arquitectura COLA[2] (Component Oriented Layered-base Architecture for Autonomy) (Palomeras et al., 2012) per estructurar tot el software desenvolupat pels robots. Aquesta arquitectura distribueix les tasques en grups o capes segons la funció que desenvolupen.

Amb la configuració explicada anteriorment l'SPARUS II no ha aconseguit cobrir les especificacions pel qual va ser dissenyat; quan el robot supera una quarta part de la seva velocitat màxima no és capaç de mantenir-se de forma estable dins de l'aigua.

1.2. Objecte

L'objectiu d'aquest projecte és abastir a l'SPARUS II de dos nous DOFs, el pitch i el roll, a partir de dos timons de profunditat que s'ubicaran a la part posterior del vehicle. La seva finalitat és proporcionar major estabilitat i maniobrabilitat al robot per permetre-l'hi assolir les especificacions pel qual va ser dissenyat i per millorar el rendiment global del vehicle.

Per això s'haurà d'implementar un sistema autosuficient amb els recursos del robot que sigui capaç d'interpretar les comandes provinents de l'ordinador central del robot per tal de posicionar els dos timons de profunditat a la posició desitjada. També s'haurà de definir un protocol de comunicació robust per a l'intercanvi de missatges entre les dues interfícies.

Amb la integració d'aquests dos actuadors al robot, aquest assolirà un total de cinc DOFs, pel que caldrà un sistema de control que gestioni tots els modes de funcionament possibles del vehicle, supervisi cada grau de llibertat i actuï sobre els actuadors conseqüentment.

Finalment, al ser un projecte aplicat, s'haurà de comprovar el correcte funcionament de tots els equips i programes implementats a partir de la realització d'experiments reals.

1.3. Abast

El present projecte abasteix el disseny, muntatge, programació i instal·lació d'una placa electrònica que s'integri totalment amb l'entorn del robot i que realitzi les accions definides a

l'objecte. Tanmateix, inclou la definició i implementació del protocol de comunicació necessari per a la comunicació entre les interfícies. Per altra banda, contempla el disseny, programació i implementació del control de l'SPARUS II en cinc DOFs per a integrar a nivell de software els nous actuadors. Com a últim punt, abasteix l'execució de diferents assaigs per tal de verificar el correcte funcionament de tots els sistemes desenvolupats.

La funcionalitat de la placa electrònica serà interpretar comandes de l'ordinador central del robot per tal de generar el senyal pertinent per al posicionament dels actuadors que controlin les dues pales del timó de profunditat. Aquesta placa també entendrà comandes senzilles per a calibrar individualment els paràmetres crítics de cada una de les aletes. Amb això es podrà garantir un correcte posicionament dels actuadors en funció de les consignes. La mateixa electrònica detectarà la possible presència d'aigua dins la cavitat on s'allotjarà el sistema de gir dels dos timons. En cas d'una intrusió d'aigua, la placa electrònica ho comunicarà al PC per tal que aquest adopti les mesures de seguretat corresponents.

La comunicació entre les dues interfícies es farà a través d'un bus RS-485 ja existent on s'hi troba connectada l'electrònica dels tres motors de l'SPARUS II. Per això es definirà un protocol de comunicació robust que no interfereixi amb el funcionament dels motors i, per a integrar l'electrònica realitzada en el robot, s'escriurà un programa tècnicament anomenat driver. Aquest es programarà en C++ dins de ROS i seguirà l'arquitectura COLA[2].

Tot el control a desenvolupar s'escriurà en C++, dins de ROS i integrat a l'arquitectura COLA[2]. Primerament, es remodelarà l'estructura de control dels tres DOFs ja existents per assolir les consignes de velocitat pel qual el robot va ser dissenyat i s'ajustarà perquè el seu comportament no sigui brusc, conferint-li un millor funcionament en mode autònom. Paral·lelament es desenvoluparà una nova estructura de control pels dos DOFs nous que s'hauran proporcionat al robot, pel que serà de gran importància modelitzar la influència de les pales sobre el vehicle i estudiar la interacció entre actuadors. Conjuntament amb el model actual del robot i les dades provinents de la navegació, aquestes dades ajudaran a extreure un seguit de normes que permetran compatibilitzar condicionadament les dues estructures de control.

Al llarg del projecte es realitzaran varis experiments tant a les instal·lacions del CIRS com en el mar. Amb aquests es corroborarà que cada part funcional descrita anteriorment funciona segons les especificacions marcades, s'ajustarà el sistema de control implementat i, finalment, es demostrarà l'assoliment de l'objectiu principal del present projecte.

2. ESTUDI TEÒRIC DELS TIMONS DE PROFUNDITAT

Abans de desenvolupar una placa electrònica o un sistema de control és imprescindible estudiar el camp al qual s'ha d'aplicar per a conèixer les possibles exigències i limitacions de l'entorn. Per aquest motiu s'ha realitzat un estudi previ de la influència i les necessitats que comporta la instal·lació de dos timons de profunditat a l'SPARUS II.

2.1. Els timons de profunditat de l'SPARUS II

Els timons de profunditat de l'SPARUS II consisteixen en dues pales fabricades amb resina i de forma igual a la definida per l'agència NACA (National Advisory Council for Aeronautics) amb la referència 0015. Cada una d'aquestes té una superfície d'aproximadament 94 cm^2, es troba acoblada en un eix d'acer cromat de 8 mm de diàmetre i la distància estimada entre el centre d'esforços de la pala i el centre de rotació és d'aproximadament 10,5 mm.

Un mòdul annex al robot, el qual no compromet l'estanqueïtat del vehicle, permet l'entrada dels dos eixos i aïlla els futurs actuadors de l'aigua. Abans de la realització d'aquest projecte en el lloc d'aquest mòdul s'hi portava una rèplica fabricada amb una impressora 3D.

Figura 5. Mòdul dels timons de profunditat acoblat a l'SPARUS II amb les pales NACA 0015

Cal remarcar que tota la part mecànica explicada ja estava fabricada al moment d'iniciar el present estudi, tot i que fins aleshores no s'havia instal·lat mai al robot.

2.1.1. Compartiment dels timons de profunditat

El compartiment dels timons de profunditat és un bloc delrin modelat seguint la forma hidrodinàmica de la part posterior del vehicle, lloc on s'ha d'incorporar. Aquest conté un buit

d'aproximadament 75 x 48 x 36 mm, el qual està reservat per a la instal·lació dels actuadors i els corresponents sistemes mecànics d'acoblament que han de rotar els eixos. Conseqüentment, el mòdul no és estanc per si mateix, pel que una junta tòrica ubicada al contorn del buit se n'encarrega quan aquest compartiment s'uneix a la resta del robot. L'entrada dels dos eixos en el mòdul es fa a través de dues juntes tòriques rotatives, les quals permeten la rotació d'aquests eixos al mateix temps que eviten l'entrada d'aigua dins del compartiment.

Per passar els conductors necessaris pels actuadors des del robot fins al compartiment hi ha dimensionat un passa murs; aquest, un cop s'hagin passat els conductors, s'emplenarà de resina per garantir un total aïllament entre el robot i el mòdul en qüestió. Al esdevenir un compartiment totalment aïllat del robot caldrà comprovar l'existència d'aigua.

Un cop muntades totes les parts mecàniques que fan possible que l'SPARUS II disposi de dos timons de profunditat, el centre d'esforços estimat de cada una de les pales queda a uns 65 cm del centre de gravetat i a uns 14 cm de l'eix longitudinal del robot.

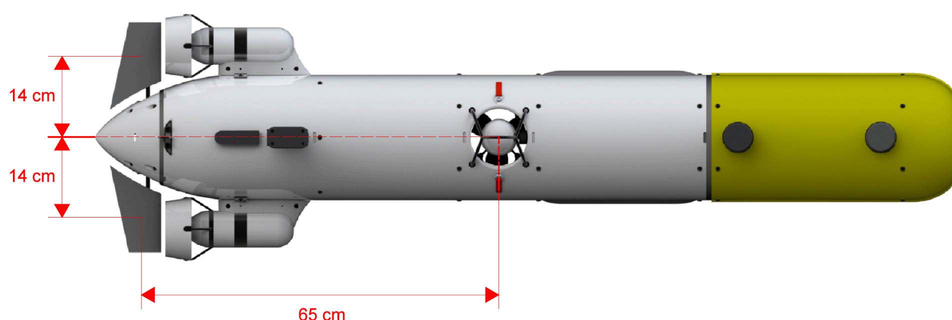

Figura 6. Vista de planta amb la ubicació de les pales respecte l'eix de coordenades de l'SPARUS II

A més, com es pot observar, cada pala queda ubicada a darrera d'un motor horitzontal, pel que serà d'especial importància estudiar la influència de cada motor sobre el comportament de la corresponent aleta. A priori s'ha pogut mesurar que el diàmetre de la tubera dels motors és de 98 mm i la distància entre aquesta i la part més pròxima del respectiu timó de profunditat és de 8 mm.

2.1.2. Propietats dels perfils NACA

El conjunt de seccions reunides sota l'estàndard NACA van ser analitzades per la institució aeronàutica anomenada amb el mateix nom. L'objectiu d'aquest estudi va ser caracteritzar

diferents perfils de pala per tal de poder determinar el seu comportament envers l'aire sense necessitat de modelar el seu comportament a partir d'experiments realitzats dins un túnel del vent. Així doncs, l'agència NACA va parametritzar per a diferents angles d'atac, o el que és el mateix, per a diferents angles d'incidència de l'aire sobre la pala, el comportament de cada perfil principalment amb dos coeficients adimensionals: el de sustentació o C_L i el de resistència o C_D.

Tal i com s'ha esmentat anteriorment, les pales del timó de profunditat de l'SPARUS II segueixen el perfil definit com NACA 0015, el qual es caracteritza per tenir una forma simètrica i per ser hidrodinàmicament eficient per angles d'atac reduïts. La dinàmica dels esmentats coeficients característics en funció de l'angle d'atac de la pala envers l'aire és la mostrada a continuació (Sheldahl et al., 1981).

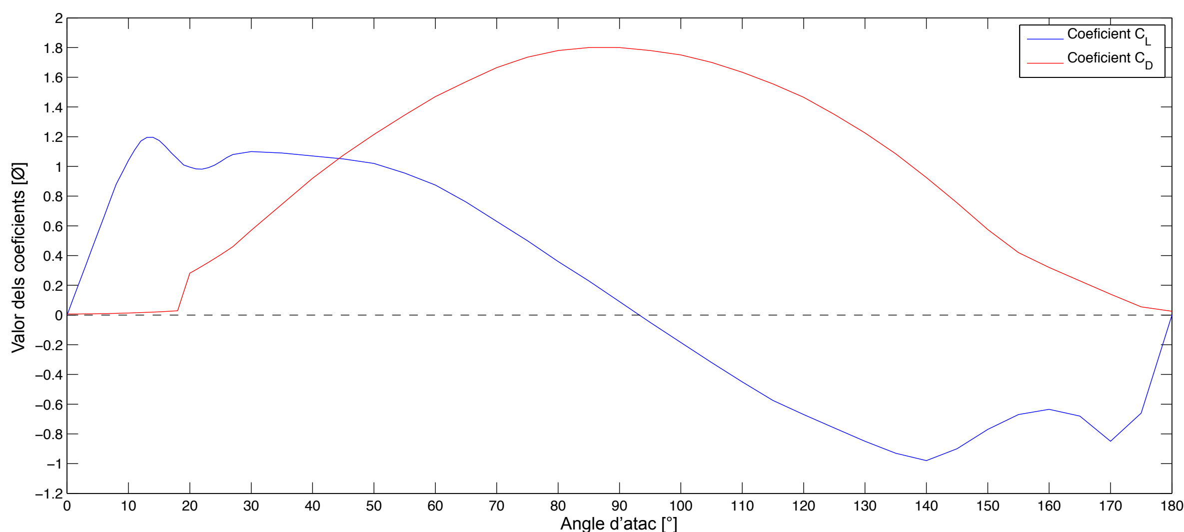

Figura 7. Coeficients característics del perfil NACA 0015 en aire

Malgrat que inicialment aquests perfils únicament van ser estudiats per situacions on estiguessin envoltats d'aire, amb el transcurs del temps ha sorgit la necessitat de saber el seu comportament en ambients aquosos, com en el present projecte. Per això, algunes institucions científiques s'han dedicat a fer els mateixos estudis que va fer l'agència NACA però per els ambients anteriorment esmentats; la incompressibilitat de l'aigua fa que el seu comportament sigui totalment diferent.

A diferència de l'abundància de dades sobre els perfils NACA utilitzats en ambients d'aire, la informació existent a la xarxa sobre els mateixos perfils en ambients aquosos és molt escassa. Tanmateix, el seu anàlisi es troba limitat fins a l'angle d'atac òptim, el qual es

defineix pel valor màxim del coeficient C_L. Els valors que prenen aquests coeficients característics en funció de l'angle d'atac de la pala envers l'aigua es troben representats a continuació (Folger et al., 1958).

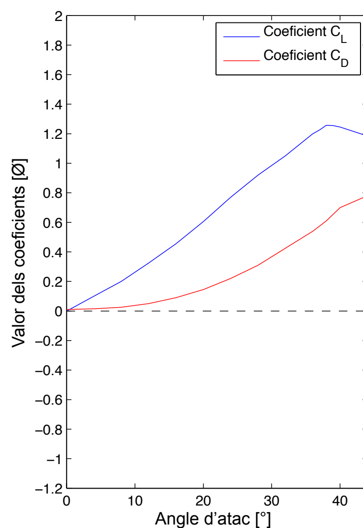

Figura 8. Coeficients característics del perfil NACA 0015 en aigua

Comparant els valors i dinàmiques dels coeficients fins els 45 º d'angle d'atac es pot apreciar que, segons el medi on es troben les pales, el coeficient C_L canvia considerablement mentre que el C_D és semblant. Cal destacar que segons aquestes dades experimentals els perfils NACA 0015 en aire tenen un angle d'atac òptim de 16 º, mentre que per ambients aquosos aquest és igual a 36 º.

2.2. Model dinàmic teòric

Una pala exposada a un fluid en moviment experimenta una força originada per la diferència de pressions existent entre la superfície superior i inferior del perfil. Aquestes pressions són el resultat de l'acceleració experimentada pel caudal màssic que transcorre la superfície del timó de profunditat, tal i com es pot demostrar amb el principi de Bernoulli.

La força neta resultant es sol descompondre en un eix longitudinal i un altre de transversal a la posició de repòs de la pala o angle d'atac zero, tot obtenint una força de sustentació L i una de resistència D (Molland et al., 2007). La força de sustentació és la component perpendicular a la direcció del fluid, mentre que la força de resistència és la que conserva la mateixa direcció i sentit que el líquid, tal i com es pot observar a la Figura 9, la qual anomena U_0 a l'aigua d'entrada i α a l'angle d'atac de la pala.

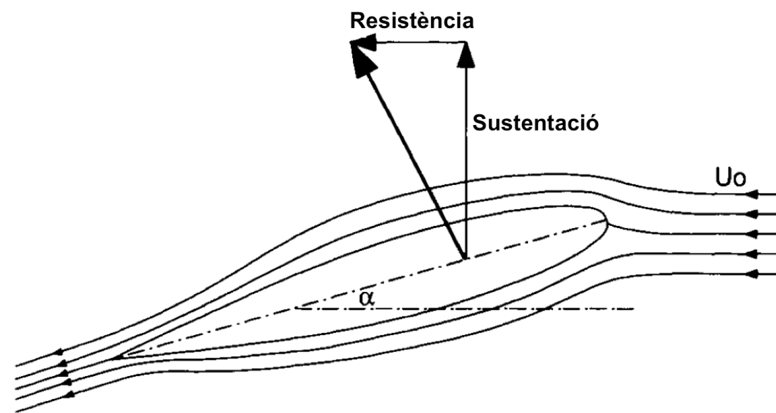

Figura 9. Components de la força originada per una pala (Molland et al., 2007)

Aquestes dues components mantenen una relació directa amb els coeficients C_L i C_D que caracteritzen el comportament d'una pala segons l'angle d'atac de la mateixa.

$$L = \frac{1}{2} \cdot C_L \cdot \rho \cdot A_F \cdot U_0^2 \qquad \text{(Eq. 1)}$$

$$D = \frac{1}{2} \cdot C_D \cdot \rho \cdot A_F \cdot U_0^2 \qquad \text{(Eq. 2)}$$

on,

L = força de sustentació, en newtons

C_L = coeficient de sustentació, adimensional

D = força de resistència, en newtons

C_D = coeficient de resistència, adimensional

ρ = densitat del fluid, en kilograms partit per metre cúbic

A_F = àrea de la pala, en metres quadrats

U_0 = velocitat del fluid incident a la pala, en metres partit per segon

En el cas de l'SPARUS II cal considerar que la velocitat del fluid incident a les pales no és la mateixa que el propi robot degut a la disposició de les aletes darrera dels motors horitzontals del vehicle.

2.2.1. Influència del motors sobre els timons de profunditat

La disposició d'un timó de profunditat respecte la resta d'un vehicle és de gran interès a la hora de modelitzar el seu comportament. En els casos on l'aigua arriba lliurament fins a les pales, la seva velocitat es pot saber a partir de les lineals i angulars del vehicle.

En casos com el del present projecte, però, la velocitat de l'aigua es veu alterada per l'existència d'un motor, pel que determinar la velocitat d'incidència del fluid a la pala no és tant directe; cal saber la velocitat a la que surt l'aigua del motor i com aquesta minva d'acord amb la distància existent entre el motor i la pala. Com que la separació entre actuadors és d'únicament 8 mm, per aquest cas s'ha pogut aproximar que la velocitat del líquid incident a la pala és igual a la que abandona la tubera del motor.

Així doncs, la velocitat que pren un fluid a la sortida d'un motor és funció de la velocitat d'entrada del líquid i de la força que exerceix el motor sobre el medi (Molland et al., 2007).

$$U_1 = \left[U_{0T}^2 + \frac{2 \cdot T}{\rho \cdot A_T} \right]^{\frac{1}{2}} \qquad \text{(Eq. 3)}$$

on,

U_1 = velocitat de sortida del fluid en el motor, en metres partit per segon

U_{0T} = velocitat d'entrada del fluid en el motor, en metres partit per segon

T = força d'empenyiment realitzada pel motor, en newtons

ρ = densitat del fluid, en kilograms partit per metre cúbic

A_T = àrea de la tubera del motor, en metres quadrats

Respecte la força d'empenyiment transmesa al líquid pel motor, aquesta es troba condicionada no únicament per la velocitat angular de gir de la hèlix sinó també per la velocitat a la que l'aigua entra a l'actuador (Carlton, 2012). Aquesta relació es defineix segons la següent equació.

$$T = C_1 \cdot \rho \cdot N^2 \cdot D^4 - C_2 \cdot U_{0T} \cdot \rho \cdot N \cdot D^3 \qquad \text{(Eq. 4)}$$

on,

T = força d'empenyiment realitzada pel motor, en newtons

C_1 = coeficient estàtic característic del conjunt motor i hèlix, adimensional

C_2 = coeficient dinàmic característic del conjunt motor i hèlix, adimensional

N = velocitat angular de gir del motor, en hertzs

D = diàmetre de la tubera del motor, en metres

U_{0T} = velocitat d'entrada del fluid en el motor, en metres partit per segon

ρ = densitat del fluid, en kilograms partit per metre cúbic

Estudis anteriors realitzats al CIRS havien modelitzat el conjunt motor i hèlix existent abans de la realització d'aquest projecte únicament considerant el terme estàtic, ja que era una aproximació de l'Equació 4 totalment vàlida per a velocitats baixes.

Tot i això, com que amb la integració dels timons de profunditat es creu que el vehicle assolirà satisfactòriament la velocitat de 2 m/s, s'ha vist oportú modelar el conjunt motor i hèlix considerant la influència de la velocitat d'entrada de l'aigua, ja que aquesta pren especial importància per valors alts.

De l'Equació 4, els coeficients C_1 i C_2 poden determinar-se teòricament amb un complex estudi sobre sistemes de propulsió; tanmateix, aquests també es poden conèixer a partir

d'una sèrie d'assajos. Pel primer coeficient cal fer experiments estàtics que consisteixin en fer girar el motor a una velocitat angular fixa i mesurar la força real que exerceixen. Un cop ajustat C_1, el segon coeficient és deduïble executant el mateix procediment però amb diferents velocitats d'entrada d'aigua en el motor.

Altres efectes deguts a la interacció entre els motors i els timons de profunditat són presents a la realitat. Per exemple, la força efectiva exercida per un motor es troba condicionada a la dificultat experimentada pel fluid al abandonar la tubera del motor, la qual està determinada per la posició de les pales. Tanmateix, la sortida d'aquest líquid presenta una forma helicoïdal, pel que considerar que la seva naturalesa és rectilínia és únicament una aproximació. Per aquest estudi, però, al parlar d'un robot de petites dimensions tots aquests efectes s'han pogut menysprear.

2.2.2. Forces resultants

Un cop definides totes les variables de les Equacions 1 i 2 i considerant les capacitats màximes del robot, s'ha pogut calcular la força màxima de sustentació i de resistència a la que es sotmetrà cada una de les pales un cop instal·lades a l'SPARUS II.

Per això, primer ha calgut determinar la velocitat del líquid incident a les pales a partir de l'Equació 3. Per una banda, la velocitat d'entrada de l'aigua als motors s'ha considerat igual a la del robot, el valor màxim teòric del qual és de 2 m/s, tot i que fins a dia d'avui no s'hagin superat els 0,5 m/s de manera satisfactòria. Respecte la força realitzada pels motors per mantenir de forma estacionària el vehicle a 2 m/s, aquesta correspon a uns 30,27 N per cada actuador segons estudis realitzats anteriorment al CIRS; en aquest punt del projecte no es disposava de les hèlixs noves, pel que no es va poder determinar a través de l'Equació 4. Finalment, agafant com a densitat de l'aigua el valor de 1000 kg/m^3 i calculant l'àrea de la tubera del motor a partir dels 98 mm de diàmetre anteriorment esmentats, s'ha obtingut que la velocitat màxima a la que sortirà l'aigua del motor serà d'aproximadament 3,47 m/s.

El següent pas ha consistit en determinar els coeficients característics dels perfils NACA 0015 pels punts de treball màxims; per la força de sustentació aquest es troba a un angle d'atac de 36 º, ja que el coeficient C_L pren un valor màxim de 1,25, mentre que per la força de resistència aquest es troba a un angle d'atac de 90 º i s'ha estimat que el seu coeficient C_D té un valor d'aproximadament 1,7. Amb el resultat anterior, agafant com a densitat de

l'aigua el valor de 1000 kg/m^3 i 94 mm^2 per a l'àrea de pala, s'han aplicat les Equacions 1 i 2 obtenint una L_{max} de 70,74 N i una D_{max} de 96,21 N.

Tot i que aquests càlculs donen una idea de les tensions lineals màximes que es podrà trobar sotmesa una pala del timó de profunditat, per a dimensionar els seus actuadors és més útil considerar el parell màxim resultant de la força neta. Considerant que no interessa posicionar la pala a més angle que l'òptim, ja que el coeficient C_L disminueix, s'ha avaluat la força neta en aquest punt de treball, sent de 78,22 N. Tot i això, d'aquesta força únicament la component perpendicular a la pala genera un parell, la qual té un valor de 76,97 N i s'aplica en el centre d'esforços de la pala, és a dir, a uns 10,5 mm del centre de rotació, provocant un moment angular màxim de 0,81 Nm. Aquest moment haurà de considerar-se al moment de dimensionar els actuadors dels timons de profunditat.

2.3. Efectes dels timons de profunditat en el robot

Les forces originades pels timons de profunditat de l'SPARUS II tenen un efecte directe en els seus DOFs; per qualsevol angle d'atac les pales generaran unes forces de resistència orientades en sentit contrari al surge del robot, mentre que la força de sustentació influenciarà en el pitch i/o en el roll dependentment al posicionament dels timons.

Abans de tot, però, per a avaluar l'efecte de les forces de sustentació en el vehicle ha calgut definir un conveni per al posicionament dels timons. Per això, els angles d'atac que provoquen forces de sustentació positives respecte el heave del robot, s'han definit com a angles positius, i contràriament, com a negatius.

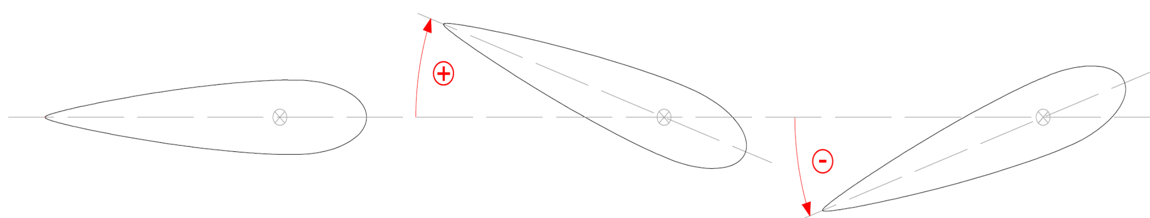

Figura 10. Conveni pel posicionament. D'esquerra a dreta, angle d'atac zero, positiu i negatiu

A més, d'aquesta manera l'aplicació de les equacions que determinen la força de sustentació resulta directe, ja que el sentit de la força de sustentació queda determinat pel diagrama característic del perfil NACA 0015, en el qual els angles d'atac negatius es caracteritzen per coeficients negatius.

Per tant, considerant que els dos timons de profunditat poden ser accionats mecànicament de manera individual, cal diferenciar dos modes de funcionament base: quan les dues pales tenen el mateix angle d'atac i, per contra, quan una pala està posicionada oposadament a l'altra respecte l'angle d'atac zero.

Posicionant els timons de profunditat amb el mateix angle d'atac s'aconsegueixen dues forces de sustentació orientades amb el mateix sentit, les quals provoquen un moment en el pitch i cap efecte en el roll. Dit d'una altra manera, angles d'atac positius generaran únicament moments positius en el pitch, i contràriament, parells negatius.

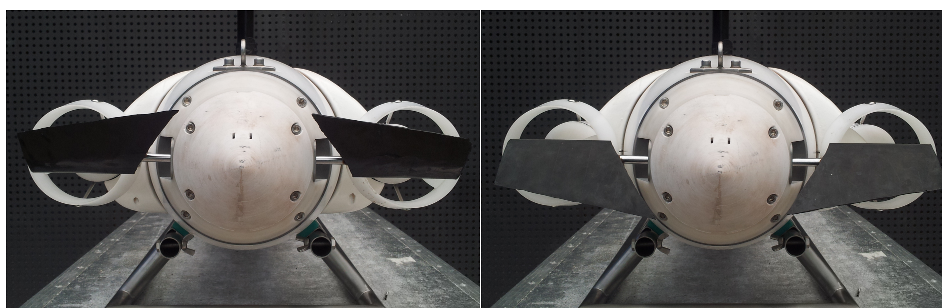

Figura 11. Moments en pitch segons posicionament. A l'esquerra, moment positiu. A la dreta, moment negatiu

Per altra banda, posicionant les dues pales de forma oposada respecte l'angle d'atac zero, s'aconsegueixen dues forces de sustentació de sentit oposat que generaran un moment en el roll i, considerant que la velocitat incident a les dues pales és la mateixa, cap efecte en el pitch. Dit d'una altra manera, amb un angle d'atac positiu per la pala dreta i negatiu per la pala esquerra únicament s'obtindrà un moment positiu en el roll, i contràriament, un de negatiu en el mateix DOF.

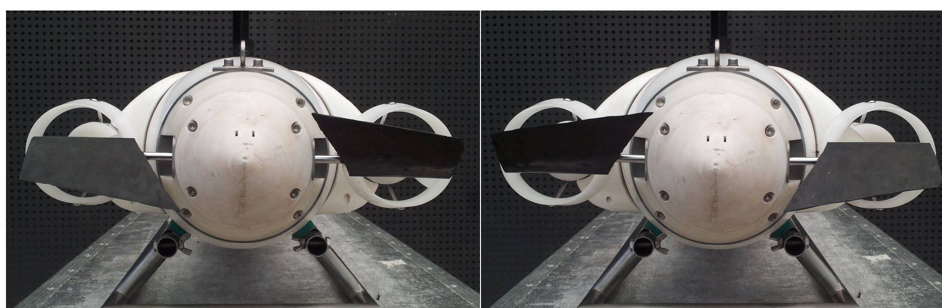

Figura 12. Moments en roll segons posicionament. A l'esquerra, moment positiu. A la dreta, moment negatiu

Tanmateix, els timons poden actuar en el pitch i en el roll simultàniament, tot posicionant-se diferencialment per accionar en roll respecte qualsevol angle diferent al d'atac zero. A tall

d'exemple, posicionant la pala esquerra a -30 º i la dreta a 10 º, el robot experimentaria un moment positiu en roll al mateix temps que un de negatiu en pitch.

Les esmentades accions dels timons de profunditat en els DOFs del robot es poden escriure en forma d'equació. Tanmateix, com és interessant contextualitzar el seu efecte segons les característiques del robot, s'ha aprofitat per actualitzar el model teòric ja existent del robot (Vidal, 2014), el qual es basa en una equació diferencial vectorial que considera totes les forces incidents en un robot submarí amb sis DOFs (Fossen, 2011).

$$M \cdot \dot{v} + C(v) \cdot v + D(v) \cdot v + g(\eta) = \tau + \tau_p \qquad \text{(Eq. 5)}$$

on,

η = vector de posició i orientació del robot, unitats del SI

v = vector de velocitats del robot, unitats del SI

$\dot{v}$ = vector d'acceleracions del robot, unitats del SI

M = matriu amb els termes de massa i inèrcia característics del robot, unitats del SI

C = matriu amb els termes de Coriolis característics del robot, unitats del SI

D = matriu amb els termes de fregament característics del robot, unitats del SI

g = vector de les forces de gravetat i de flotació incidents al robot, unitats del SI

τ = vector de les forces exercides pels actuadors del robot, unitats del SI

τ_p = vector de les forces exercides per les pertorbacions, unitats del SI

Per a considerar les forces originades pels timons ha calgut modificar el vector de les forces exercides pels actuadors del robot, tal i com es mostra a l'Equació 6.

$$\tau = \begin{pmatrix} T_L + T_R - D_L - D_R \\ 0 \\ T_v \\ d_r \cdot \left(-L_L + L_R\right) \\ d_p \cdot \left(L_L + L_R\right) \\ d_y \cdot \left(T_L - T_R\right) \end{pmatrix} \qquad \text{(Eq. 6)}$$

on,

τ = vector de les forces exercides pels actuadors del robot, unitats del SI

T_L = força d'empenyiment realitzada pel motor esquerre, en newtons

T_R = força d'empenyiment realitzada pel motor dret, en newtons

D_L = força de resistència realitzada pel timó esquerre, en newtons

D_R = força de resistència realitzada pel timó dret, en newtons

L_L = força de sustentació realitzada pel timó esquerre, en newtons

L_R = força de sustentació realitzada pel timó dret, en newtons

d_r = 0,14 m. Distància en Y entre el centre del robot i el centre d'esforços dels timons

d_p = 0,65 m. Distància en X entre el centre del robot i el centre d'esforços dels timons

d_y = 0,16 m. Distància en Y entre el centre del robot i el centre dels motors

Cada element de la matriu anterior representa un DOF del robot, pel que a través d'aquesta es pot verificar que amb la integració dels timons de profunditat l'SPARUS II aquest assolirà dos DOFs addicionals, el pitch i el roll, aconseguint un total de cinc DOFs. L'únic DOF no controlat serà el sway.

3. HARDWARE

Per a poder posicionar les dues aletes dels timons de profunditat a partir de les consignes provinents de l'ordinador central de l'SPARUS II s'ha dissenyat i implementat una placa electrònica. Aquesta té com a funció fer d'interfície de posicionament entre els servomotors, actuadors escollits per a moure els eixos dels timons, i l'ordinador a través del bus RS-485.

La mateixa placa electrònica llegeix l'estat del sensor d'aigua per tal de saber la presència no desitjada d'aigua dins el compartiment dels timons de profunditat. Aquesta informació es comunica a l'ordinador del robot perquè adopti les mesures corresponents.

El següent diagrama de blocs representa el hardware ja existent en el robot abans d'iniciar el present projecte i que s'ha hagut de considerar al moment de seleccionar i dissenyar tot el que s'explica en aquest capítol. A més, de color vermell, indica tot el hardware incorporat per a la integració dels dos timons de profunditat.

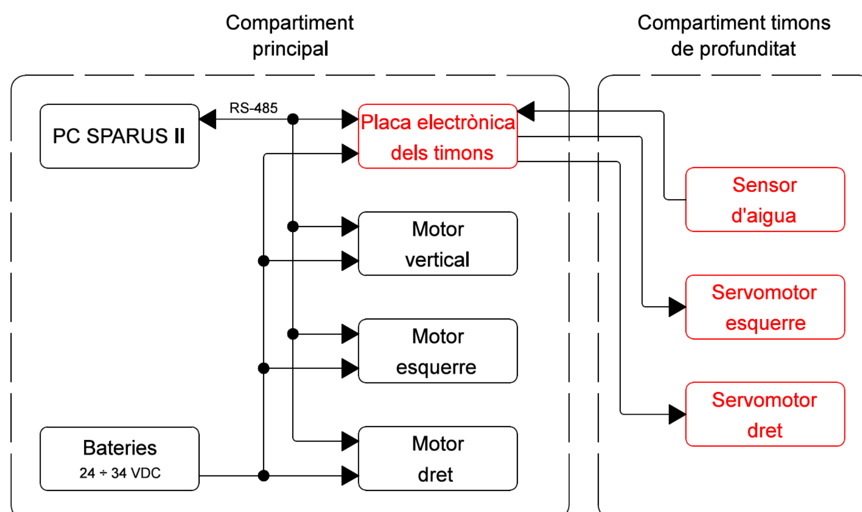

Figura 13. Diagrama de blocs del elements utilitzats i afegits en el robot

3.1. Servomotors

Un servomotor és un actuador format per un motor elèctric de corrent contínua i un sistema de control de posició, el qual permet ubicar l'eix a un angle de gir determinat. Aquest control de posició integrat dins el servomotor es realitza amb un microcontrolador i un sensor de posició angular, el qual és solidari amb l'eix de sortida. A la Figura 14 es poden apreciar els diferents components citats, on el sensor de posició angular és un potenciòmetre.

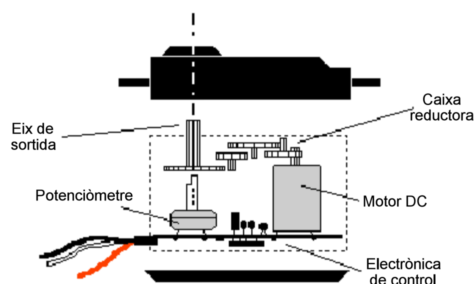

Figura 14. Components d'un servomotor

Per a moure les aletes del timó s'ha escollit fer-ho amb dos servomotors SAVÖX SC-1251MG, ja que ofereixen un parell lleugerament superior als 0,81 Nm necessaris segons els càlculs desenvolupats anteriorment i presenten un bon compromís entre consum, mida i pes, factors molt importants en un AUV. A més, les seves dimensions són suficientment reduïdes com per a ser instal·lats dins del compartiment estanc ja dissenyat pels timons de profunditat.

Propietat	Valor
Dimensions [mm]	40,8 x 20,2 x 25,4
Tensió d'alimentació [VDC]	4,8 a 6,0
Consum màxim [A]	2,0 a 3,0
Consum en buit [mA]	150 a 180
Velocitat [rad/s]	12,93 a 11,63
Parell [N·m]	0,68 a 0,88
Rang d'obertura [º]	160
Sistema de control i freqüència [Hz]	PWM – 250 a 300

Taula 1. Característiques del servomotor SAVÖX SC-1251MG

Com es pot apreciar a la taula anterior, els valors corresponents al consum màxim, al consum en buit, a la velocitat i al parell no estan representats per un valor únic, sinó en un rang; depenent de la tensió d'alimentació, la qual pot anar des dels 4,8 fins als 6 VDC, les propietats esmentades prendran un valor característic o altre.

Tot i que es volia un actuador que proporcionés informació sobre la posició angular real assolida per l'eix de sortida per tal de poder detectar possibles problemes en el seu funcionament, no s'ha trobat cap servomotor al mercat que complís les prestacions anteriorment citades i que a més donessin aquest senyal de sortida.

3.1.1. Paràmetres elèctrics teòrics

Pel bon funcionament d'aquests servomotors és essencial alimentar-los a la tensió nominal i proporcionar-los-hi un senyal de control coherent a les característiques indicades pel fabricant. Com que es necessita un parell superior a 0,81 Nm, segons càlculs, s'ha hagut d'alimentar els actuadors a 6 VDC, el que implica un consum màxim de 3 A per cada un.

El senyal de control que interpreten aquests servomotors s'anomena PWM (Pulse Width Modulation) i es basa amb una sèrie de polsos, la duració dels quals, és proporcional a l'angle de posicionament. La freqüència d'aquest senyal no influeix en el posicionament de l'eix, però el fabricant recomana mantenir-la entre 250 i 333 Hz per tal d'evitar problemes d'inestabilitat, vibracions i pèrdua de parell.

Cal tenir especial cura amb la duració dels polsos, ja que cal respectar les indicacions donades pel fabricant en quan a durada mínima i màxima d'aquests, 700 i 2.300 µs respectivament. En el cas de proporcionar als actuadors polsos de durada no compresa dins el rang recomanat, l'eix de sortida quedaria sense parell.

A continuació es mostren diferents posicionaments teòrics d'un servomotor SAVÖX SC-1251MG segons el senyal de control proporcionat.

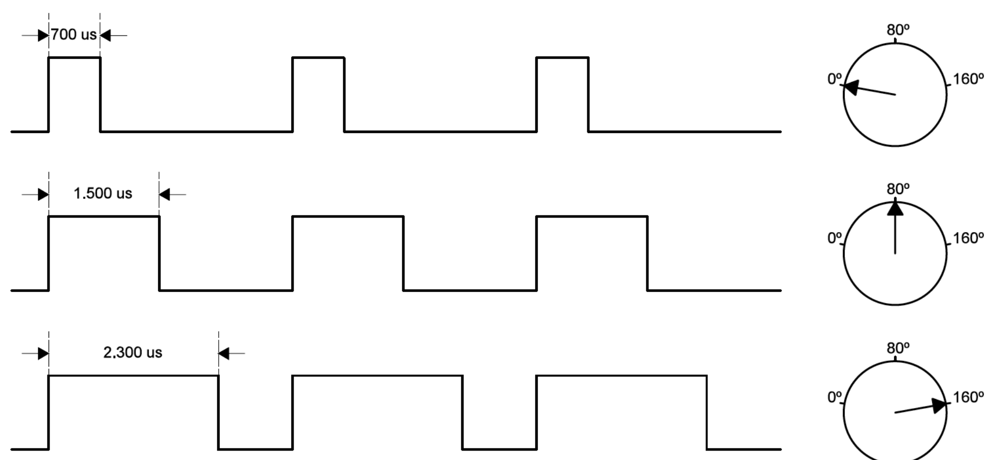

Figura 15. Posicionament dels servomotors segons el senyal de control aplicat

El conductors de l'alimentació conjuntament amb el de control estan integrats en un connector elèctric ràpid com el que es mostra a la Figura 16. La secció de tots els conductors és de 1 mm^2, el que s'adequa al corrent màxim que hi ha de passar multiplicat

pel factor corrector de 1,25 a aplicar als conductors d'alimentació dels motors segons la ITC-BT-47 del REBT (Reglament Electrotècnic de Baixa Tensió). El cable de color taronja és el conductor de control, el vermell l'alimentació i el negre la massa.

Figura 16. Connector d'un servomotor

3.1.2. Paràmetres elèctrics reals

Per a familiaritzar-se amb els servomotors es va decidir experimentar a través d'una plataforma Arduino. Aquesta va permetre escriure ràpidament el codi necessari per fer moure l'eix dels actuadors a diferents angles i així extreure els seus paràmetres elèctrics reals. Per una part es va observar que la freqüència del senyal de control no produïa efectes d'inestabilitat, vibracions i pèrdua de parell si el seu valor estava comprès entre 43 i 380 Hz, un rang significativament major al teòric.

Per altra banda, la resta de paràmetres trobats no s'ajustaven als teòrics, obtenint diferències bastant significatives. A més, per cada actuador que es testejava s'obtenien valors diferents. La taula que segueix mostra els paràmetres elèctrics teòrics i els reals dels dos servomotors utilitzats, a més d'indicar el rang d'obertura real de cada actuador.

	t angle mínim [μs]	t angle central [μs]	t angle màxim [μs]	Rang [º]
Valors teòrics	700	1.500	2.300	160
Servomotor esquerre	765	1.559	2.365	136
Servomotor dret	791	1.534	2.241	134

Taula 2. Comparació dels paràmetres teòrics amb els reals

Abans d'experimentar amb el servomotors es veia la necessitat de disposar d'unes comandes, les quals permetessin calibrar les aletes i conseqüentment corregir els errors

derivats del muntatge. A més d'això, amb els resultats obtinguts es va reforçar la necessitat d'aquestes.

També es va detectar un aspecte a considerar en el moment de fer el codi del microcontrolador; com que la disposició dels actuadors dins del mòdul del timó de profunditat no és igual, tal i com es pot veure a la següent figura, en el moment de generar el senyal de control per a l'aleta dreta s'ha hagut de pensar que tot el servomotor està girat.

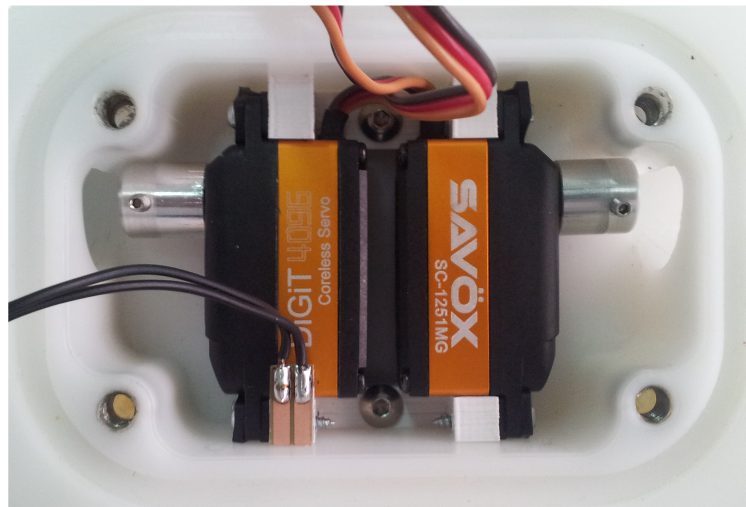

Figura 17. Disposició dels servomotors dins el compartiment del timó

Aquesta ubicació comporta que l'angle mínim s'assolirà amb el temps de pols màxim i l'angle màxim amb el temps de pols mínim.

3.2. Sensor angular

Tenir informació sobre la posició angular real de l'eix de sortida del servomotor pot ser de gran utilitat. A part de permetre corregir possibles errors de posicionament induïts pel control intern del servomotor, pot ajudar a identificar un suposat desacoblament mecànic del timó de profunditat per tal que el robot prengui les mesures de seguretat corresponents. A més de detectar els errors esmentats, aquest senyal permetria implementar un sistema de calibratge automàtic, el qual ajudaria a eliminar els errors comesos amb el calibratge manual i contribuiria a que el canvi d'actuadors, en cas de ruptura, fos més ràpid i fàcil.

Per a tot això, malgrat no trobar cap servomotor que tingués tant les prestacions requerides per a l'aplicació com l'esmentat senyal de feedback, s'ha estudiat la possibilitat d'incorporar

en els dos eixos un sensor que oferís aquesta informació. Després d'estudiar els productes existents en el mercat, s'ha determinat que el sensor que s'ajusta millor a les necessitats és el MTS-360 de la casa MOUSER.

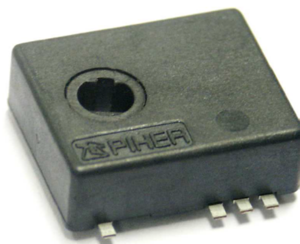

Figura 18. Sensor angular MOUSER MTS-360

Aquest sensor angular de reduïdes dimensions es caracteritza per la seva tecnologia contactless, el que li confereix una llarga vida útil i la capacitat de rotar 360 º. A més, permet programar-li la dinàmica del senyal de sortida entre diferents possibilitats i definir-li zones mortes. Altres característiques destacables que han determinat la selecció d'aquest sensor davant altres són les que es poden veure a continuació.

Propietat	Valor
Dimensions [mm]	17,0 x 18,0 x 6,0
Tensió d'alimentació [VDC]	5,0 ± 10 %
Consum [mA]	8,5
Senyal de sortida	Analògic, PWM o SPI
Resolució [bits]	12 (Analògic i PWM) o 14 (SPI)
Vida útil [cicles]	50.000.000
Linealitat	1 %

Taula 3. Característiques del sensor angular MOUSER MTS-360

Tot i que el MTS-360 s'adequa a les necessitats dels timons de profunditat del robot, aquest no s'ha pogut instal·lar a causa d'una limitació mecànica no resoluble durant el temps d'execució d'aquest projecte; l'eix que uneix cada servomotor amb la seva pala s'hauria de reduir fins als 4 mm de diàmetre i mecanitzar en forma de D per tal d'acoblar-se correctament en el sensor angular. Com que tot això comportaria redissenyar el compartiment estanc ja existent, finalment s'ha deixat la implementació d'aquest sensor com a treball futur.

3.3. Detecció d'aigua

La detecció d'aigua en el compartiment on s'allotgen els servomotors és un requisit imprescindible en la implementació dels timons de profunditat. En cas de filtrar-se aigua dins d'aquest mòdul cal actuar adequadament per tal de mantenir la integritat del hardware.

El sensor d'aigua utilitzat consisteix en una petita PCB (Printed Circuit Board) que té dues pistes rectes aïllades entre elles. El circuit imprès es munta de manera que un extrem de cada pista quedi a la part més baixa del compartiment, tal i com es pot apreciar a la Figura 17. D'aquesta manera, en cas de l'existència d'aigua, les dues pistes queden comunicades com a conseqüència de la baixa resistivitat d'aquest líquid.

Per tant, el propi sensor no indica si dins el compartiment hi ha aigua, és una electrònica addicional la que permet diferenciar els dos estats. Aquesta estratègia és la utilitzada en tots els detectors implementats en el CIRS, pel que aquesta PCB no s'ha hagut de dissenyar.

3.4. Placa electrònica

Considerant les propietats dels servomotors, el sensor d'aigua i les limitacions de l'entorn en el que s'havia d'integrar tot aquest hardware, amb l'ajuda del software Altium Designer s'ha dissenyat una placa electrònica descriptible segons el diagrama de blocs següent.

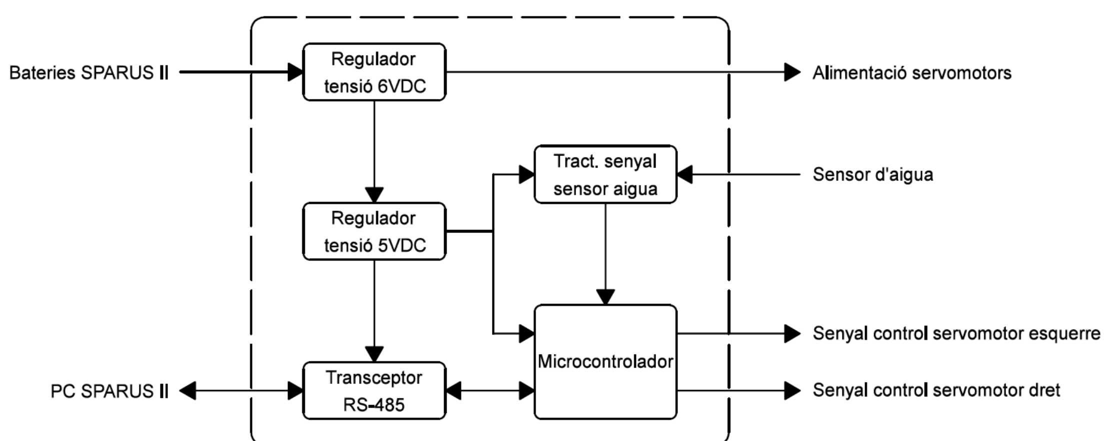

Figura 19. Diagrama de blocs de la placa electrònica dels timons de profunditat

Com es pot apreciar, aquesta placa únicament dependrà dels recursos del robot, el qual era un dels requisits de disseny de la placa electrònica.

Per a cobrir totes les necessitats de processament s'ha escollit el microcontrolador PIC16F1829. Aquest es caracteritza per admetre des de 1,8 fins a 5 VDC per a la seva alimentació, per un baix consum i per incorporar una UART (Universal Asynchronous Receiver Transmitter), indispensable per a la comunicació sèrie amb l'ordinador del robot. A més, disposa de 256 bytes de memòria EEPROM (Electrically-Erasable Programmable Read-Only Memory), 17 pins d'entrada o sortida digitals i 4 timers de 8 bits i 1 de 16 bits.

Tot i que el microcontrolador escollit disposa d'un oscil·lador intern de 32 MHz, s'ha utilitzat un senyal extern de 11,0592 MHz provinent d'un cristall de quars. A partir d'aquest senyal el microcontrolador és capaç de generar un senyal de rellotge pel bus RS-485 idèntic al seu baudrate teòric, evitant la possible pèrdua d'informació en trames formades per un gran nombre de bytes.

La funció del transceptor RS-485 és gestionar la comunicació entre el bus i la UART del microcontrolador, pel que s'ha escollit l'integrat DS75176BTM. Únicament connectant-lo al bus i a la UART del microcontrolador i indicant-li quan ha d'estar en mode receptor o emissor, la comunicació entre els dispositius ja és factible. La seva alimentació és a 5 VDC i els seus terminals connectats al bus disposen de tercer estat per aïllar-se'n i deixar-lo lliure per a les comunicacions existents entre altres dispositius.

Per garantir els nivells de tensió en cada línia del bus, a la sortida del transceptor s'ha dimensionat un pull-up i un pull-down per a la línia A i B del bus RS-485 respectivament. A més, per a evitar possibles rebots dels senyals emesos s'ha ubicat una resistència de final de línia. Amb aquesta electrònica addicional al transceptor de línia s'assegura el bon funcionament del bus de comunicació.

S'ha implementat un circuit pel tractament del senyal del sensor d'aigua consistent en un transistor PNP, un NPN i les seves corresponents resistències. Com que el sensor està allotjat en un ambient molt sorollós, les resistències s'han ajustat perquè el circuit sigui poc sensible, evitant que les possibles tensions induïdes a través del propi sensor o conductors originin falses alarmes.

Amb aquest circuit s'aconsegueix que quan no hi ha aigua dins de la cavitat del timó de profunditat, la sortida d'aquest circuit està en estat alt, és a dir, 5 VDC. En canvi, quan es detecta la presència d'aigua, la sortida pren una tensió de 0 VDC. Amb aquesta lògica de

tensions es garanteix la seguretat del vehicle, ja que en cas d'haver una avaria en el circuit el robot ho interpretarà com una presència d'aigua, actuant com a una situació d'emergència i evitant tornar a posar el robot operatiu fins arreglar l'avaria.

Finalment, per alimentar tant els servomotors com tota l'electrònica i circuits integrats explicats anteriorment, s'ha requerit una part de potència que proporcioni tant 5 com 6 VDC. Per una banda, s'ha triat el regulador de tensió PTN78020W per obtenir 6 VDC a partir de la tensió de les bateries del robot, la qual varia aproximadament des dels 28 fins als 32 VDC. El regulador seleccionat admet un rang de tensions d'entrada comprés entre 7 i 36 VDC, un rang de tensions de sortida configurable des dels 2,5 fins als 12,6 VDC i un corrent de sortida de 6 A.

Tot i que el regulador de forma automàtica ajusta el seu funcionament segons la tensió d'entrada, cal indicar-li quina és la tensió desitjada a la sortida. Per això, entre el pin quatre d'aquest component i la massa cal connectar una resistència de valor igual al resultant de la següent equació.

$$R_{SET} = 54.900 \cdot \frac{1{,}25}{V_O - V_{min}} - R_P \quad \text{(Eq. 7)}$$

on,

R_{SET} = valor de la resistència d'ajustament de la tensió de sortida, en ohms

V_O = tensió desitjada a la sortida, en volts

V_{min} = tensió mínima a la sortida del dispositiu, en volts

R_P = 6,49 kΩ. Valor estipulat pel fabricant d'una resistència interna

Realitzant el càlcul anterior surt un valor òhmic igual a 13.117,14 Ω, pel que s'aproxima el seu valor al més proper que es troba comercialment, 13,1 kΩ. Aquest ajustament comporta que el valor de tensió a la sortida sigui teòricament 3 mV major al desitjat, fet menyspreable. La resistència escollida tindrà una tolerància del 1 % amb el fi d'evitar una gran variació respecte la tensió desitjada.

Pel bon funcionament d'aquest regulador de tensió s'han seguit les indicacions del fabricant referents a la instal·lació d'un condensador ceràmic de 2,2 μF amb tensió màxima suportable de 35 VDC a l'entrada del dispositiu per filtrar les variacions brusques de tensió. També s'ha dimensionat un condensador electrolític de 330 μF amb tensió màxima admissible de 16 VDC a la sortida del regulador per atenuar el ripple de la tensió de sortida.

A partir dels 6 VDC obtinguts amb el component anterior s'han aconseguit 5 VDC amb el regulador de tensió LF50CDT. Aquest admet una tensió màxima d'entrada de 16 VDC i pot proporcionar fins a 1 A a la seva sortida.

A prop de l'alimentació de cada un dels integrats s'ha connectat un condensador ceràmic de 100 nF, amb el fi de desacoblar les interferències generades per altres components del circuit o per ells mateixos. D'aquesta manera s'ha assegurat una tensió d'alimentació estable en tots els components electrònics sensibles a variacions sobtades de tensió.

4. PROTOCOL DE COMUNICACIÓ

La comunicació entre la placa electrònica i l'ordinador central de l'SPARUS II no és possible si les dues interfícies no estan connectades per un medi físic i si no comparteixen un llenguatge. Considerant la primera condició resolta al afegir la placa electrònica en el bus RS-485 on hi ha connectats els tres motors del robot, per a solucionar la segona clàusula, s'ha definit un protocol de comunicació.

4.1. Bases del protocol de comunicació

Un protocol de comunicació és un conjunt de normes i regles que defineixen les condicions sintàctiques, semàntiques, de sincronització i de detecció d'errors que han de seguir els missatges enviats a través d'una xarxa.

Atenent les necessitats que ha de cobrir el protocol de comunicació del present projecte, aquest s'ha dividit en dues parts. Per una banda, les comandes de preguntar per l'estat del sensor d'aigua i activar, desactivar i posicionar els servomotors, estan implementades en binari perquè la comunicació entre les dues interfícies sigui el més ràpida possible. Per altra banda, les comandes destinades a ser executades per un usuari, com les de configurar els actuadors i la d'interrogar per a la versió del firmware, estan implementades en ASCII (American Standard Code for Information Interchange) amb la finalitat de ser fàcils d'utilitzar.

En el moment de definir les comandes s'ha tingut especial atenció en evitar interferències amb el protocol de comunicació utilitzat pels motors, el qual es troba a continuació. En totes les comandes presentades en aquest capítol, els guions baixos únicament fan més entenedora la presentació de les comandes; a la hora d'utilitzar-les van sense i tot seguit.

Comanda (HEX)	Utilitat
FD_adreça_RPML_RPMH_BCC	Modificar la consigna d'un motor (termes RPML i RPMH), incloent poder-lo parar
FE_adreça_pregunta_pregunta	Preguntar per la magnitud del motor definida pel terme pregunta
FE_adreça_pregunta_resposta_ resposta_BCC	Respondre la informació sol·licitada per la comanda anterior

Taula 4. Comandes del protocol de comunicació dels motors

Cal destacar que cada motor disposa d'una adreça identificadora única a la xarxa, pel que a menys que s'utilitzin les mateixes comandes indicades anteriorment amb l'adreça d'un dels motors, no s'interferirà en el funcionament normal d'aquests actuadors. També cal explicar que el terme BCC (Block Check Character) és un valor afegit al final dels missatges que serveix per a la detecció d'errors en la transmissió, el qual pren el valor resultant d'aplicar l'operació lògica XOR en tots els termes de la comanda exceptuant el primer.

A més, com que una de les normes del bus RS-485 és que tots els elements connectats a la mateixa xarxa han de treballar amb el mateix baudrate, l'utilitzat pels motors ha marcat que el protocol de comunicació a implementar ha de configurar-se a una velocitat de 57.600 Bd.

4.1.1. Comandes d'execució

Al moment de definir la part del protocol de comunicació a ser utilitzada per l'ordinador central del robot es va optar perquè aquest fos semblant a l'utilitzat pels motors. D'aquesta manera s'evita que una combinació de bytes en el bus formin inesperadament una comanda interpretable per a un altre dispositiu, fent el protocol més robust i segur.

Per això, es va definir una adreça a la placa electrònica dels timons de profunditat que no utilitzés cap motor i que diferís en més d'un bit de la resta. Així, davant la possible inducció de soroll al bus, difícilment cap placa electrònica confondria el receptor d'una comanda. Després d'aquest anàlisis, es va escollir l'adreça 116, l'equivalent a 74 en hexadecimal.

Per tant, les comandes habilitades per preguntar l'estat del sensor d'aigua i engegar, parar i posicionar els servomotors, són les indicades tot seguit.

Comanda (HEX)	Utilitat
FD_74_LF_RF_BCC	Modificar la consigna dels dos servomotors. No inclou ni la engegada ni la parada
FE_74_00_00	Inhabilitar el posicionament dels dos servomotors, deixant-los sense parell
FE_74_01_01	Habilitar el posicionament dels dos servomotors, inicialment posant-los al centre
FE_74_02_02	Preguntar sobre l'estat del sensor d'aigua. A continuació s'explica la resposta

Taula 5. Comandes utilitzades per l'ordinador central del robot

Cal especificar que el terme LF i RF fan referència al servomotor esquerre i al dret respectivament, els quals indiquen el doble de la consigna desitjada; s'ha cregut adient aplicar la conversió anterior per tenir mig grau de resolució en el protocol de comunicació tot i perdre 8 º de rang, permetent un marge de treball màxim d'aproximadament 127 º.

Pel que fa la comanda que respon a la sol·licitud de l'estat del sensor d'aigua, aquesta també ha seguit el format genèric utilitzat pel protocol de comunicació dels motors, ja que en la mateixa trama de bytes s'indica la resposta i a quina pregunta pertany la informació.

Comanda (HEX)	Utilitat
FD_74_pregunta_resposta_ resposta_BCC	Respondre la informació sol·licitada per qualsevol comanda de pregunta

Taula 6. Comanda utilitzada per contestar a l'ordinador central del robot

Tot i que en el protocol actual únicament s'ha habilitat una pregunta, s'ha vist convenient conservar el terme que indica a quina qüestió s'està responent. Com que en un futur pot ser d'interès implementar el sensor angular proposat en aquest projecte, pel que es necessitaria una comanda per preguntar el posicionament angular dels eixos. Amb el format exposat, no caldria fer grans canvis en el protocol de comunicació.

Respecte el terme BCC, únicament és utilitzat en les comandes on els termes avaluats poden ser diferents, ja que contràriament el valor que assoleix aquest terme és el de l'adreça. L'Equació 8 indica com es calcula el seu valor per a les comanda de posicionament dels servomotors, i l'Equació 9 per la comanda de resposta de l'estat del sensor d'aigua.

$$BCC = 74 \oplus LF \oplus RF \quad \text{(Eq. 8)}$$

$$BCC = 74 \oplus pregunta \oplus pregunta \oplus resposta \quad \text{(Eq. 9)}$$

on,

BCC = bit de detecció d'errors, en hexadecimal

LF = doble de la consigna desitjada pel servomotor esquerre, en hexadecimal

RF = doble de la consigna desitjada pel servomotor esquerre, en hexadecimal

pregunta = codi identificador de la pregunta realitzada, en hexadecimal

resposta = valor de la resposta de la pregunta realitzada, en hexadecimal

Si el valor indicat en el terme BCC coincideix amb el calculat en el destí segons l'equació anterior corresponent, s'apliquen les consignes indicades per cada aleta del timó de profunditat o es creu la resposta rebuda. Contràriament, el firmware ha de descartar l'acció requerida per la comanda.

4.1.2. Comandes d'usuari

Les comandes d'usuari estan pensades perquè es pugui interactuar d'una forma fàcil amb la placa electrònica dels timons de profunditat i configurar els paràmetres més importants del seu funcionament. Com que aquest mode d'operació ha de ser esporàdic, s'ha cregut oportú evitar que durant el funcionament normal del robot es pugui donar una combinació de bytes iguals a una comanda d'usuari.

Per això s'ha definit una comanda en binari amb el mateix format que les anteriors, que permet habilitar i inhabilitar el processament de les comandes rebudes en ASCII. D'aquesta manera s'evita la possibilitat del problema plantejat anteriorment.

Comanda (HEX)	Utilitat
FE_74_0A_0A	Habilitar i inhabilitar el processament de les comandes rebudes en ASCII

Taula 7. Comanda d'habilitació i inhabilitació del processament de les comandes ASCII

Al ser una sola comanda per dues accions, és important que el firmware indiqui l'estat en què l'usuari està treballant cada cop que el microcontrolador rep aquesta comanda.

Les comandes indicades a la Taula 8 són les habilitades dins el protocol de comunicació perquè l'usuari pugui configurar els paràmetres dels servomotors. En aquestes, el terme XXXX indica l'espai màxim reservat per a posar el valor del paràmetre que es vol configurar, podent ser únicament d'un dígit o de quatre.

Comanda (ASCII)	Utilitat
MinLXXXX	Configuració del temps de pols equivalent a l'angle mínim del servomotor esquerre
ZerLXXXX	Configuració del temps de pols equivalent a l'angle central del servomotor esquerre
MaxLXXXX	Configuració del temps de pols equivalent a l'angle màxim del servomotor esquerre
RanLXXXX	Configuració del rang en graus del servomotor esquerre
MinRXXXX	Configuració del temps de pols equivalent a l'angle mínim del servomotor dret
ZerRXXXX	Configuració del temps de pols equivalent a l'angle central del servomotor dret
MaxRXXXX	Configuració del temps de pols equivalent a l'angle màxim del servomotor dret
RanRXXXX	Configuració del rang en graus del servomotor dret

Taula 8. Comandes de configuració dels servomotors (continuació)

També s'ha configurat una comanda que permet saber ràpidament la versió de firmware que hi ha gravada en el microcontrolador de la placa. Aquesta comanda és d'especial interès per a saber si el firmware de la placa electrònica està actualitzat.

Comanda (ASCII)	Utilitat
v	Informació de la versió de firmware existent en el microcontrolador

Taula 9. Comanda per saber la versió de firmware

Per a qualsevol d'aquestes comandes d'usuari, al seu final cal afegir-hi l'equivalent en ASCII d'un retorn si el terminal utilitzat no ho fa automàticament al prémer enviar.

4.2. Gestió del bus de comunicació

Gestionar adequadament l'ús d'un bus de comunicació és tant important com tenir un bon port sèrie a nivell de hardware i un protocol de comunicacions concís i segur; si diferents

dispositius emeten al mateix moment hi haurà col·lisió de missatges i, conseqüentment, es perdrà informació.

Pel present projecte s'ha hagut de gestionar el bus RS-485 amb cinc dispositius connectats a ell: el PC de l'SPARUS II, l'electrònica dels tres motors i la placa dissenyada pels timons de profunditat. Aprofitant que tots aquests dispositius es controlen a partir de l'ordinador central del robot, la gestió de les comandes s'ha realitzat a través del mateix. Exactament, s'ha efectuat a partir d'un programa escrit en C++ dins l'arquitectura COLA[2] i integrat dins el ROS, tècnicament anomenat driver. La funció d'aquest programa és convertir les consignes pels diferents actuadors provinents del control a unes comandes que segueixin el format especificat segons el protocol de comunicació de cada dispositiu. Tanmateix, el programa envia les comandes a través del port sèrie del PC i, en cas de rebre una comanda de resposta, la desxifra i la publica en un tòpic del ROS.

L'esmentat driver, abans de la realització d'aquest projecte, ja estava totalment operatiu, tot i que únicament pels tres motors. A més, les tasques les desenvolupava a una freqüència de 10 Hz, la qual no es podia ampliar degut a la gran quantitat d'informació que circulava a través del port sèrie. Dit d'una altra manera, l'ocupació del bus de comunicació no permetia augmentar la freqüència de control del robot, restricció que es va veure totalment necessària d'eliminar per futures millores en el control de l'SPARUS II.

Per això, abans d'incorporar en el driver la part de port sèrie i de ROS corresponent als timons, es va optar per a gestionar la informació que s'intercanviava amb l'electrònica dels motors. Per cada un d'aquests i a cada cicle, a més d'enviar-los-hi la consigna, se'ls hi demanava el seu voltatge d'alimentació, la intensitat que consumien, la temperatura de l'electrònica, les revolucions per minut a la que girava l'eix del motor i l'estat general i errors del dispositiu. Tota aquesta informació ocupava el bus uns 60 ms, percentatge força significatiu tenint en compte que el control iterava cada 100 ms.

La gestió de la informació intercanviada amb l'electrònica dels motors s'ha basat en adequar la periodicitat d'enviament de les comandes segons la importància i la dinàmica de la magnitud a la que s'està preguntant o modificant. D'acord amb aquests fonaments, les comandes per modificar les consignes i les de preguntar sobre la intensitat i les revolucions per minut s'han mantingut a una freqüència de 10 Hz. Per contra, la freqüència de preguntar sobre el voltatge, la temperatura de l'electrònica, l'estat general del dispositiu i els errors s'ha reduït fins a 0,66 Hz, és a dir, una dada cada 1,5 segons enlloc de 10 dades per segon.

La figura que segueix representa un cicle complet de la gestió del bus RS-485, el qual dura exactament 1,4 segons. En aquesta, cada línia vertical és l'inici d'un nou període, pels quals s'indica quines comandes s'envien i a quins dispositius: el vermell indica que s'envia als tres motors, mentre que els altres colors indiquen que únicament s'emet a un motor en concret.

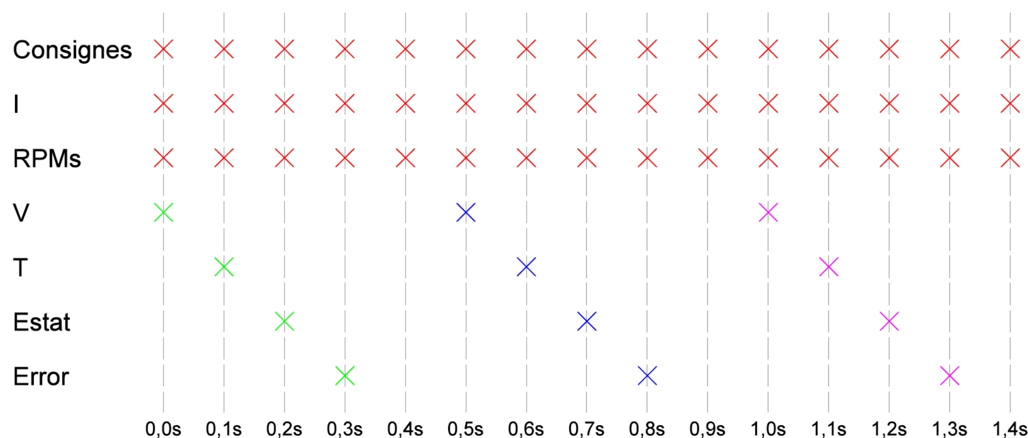

Figura 20. Diagrama temporal d'un cicle complet de la gestió del bus RS-485. En vermell, comandes enviades a tots els motors, en verd al motor vertical, en blau al motor esquerre i en magenta al motor dret

Un cop gestionada la informació corresponent als tres motors de l'SPARUS II, s'ha ampliat el driver per incorporar la part de ROS i la d'utilització del port sèrie corresponent als dos timons de profunditat, tal i com es pot veure a l'annex de programació del driver. Seguint el raonament anterior, les consignes per aquests actuadors s'han enviat a la mateixa freqüència que les dels motors, a 10 Hz, mentre que s'ha considerat suficient preguntar per l'estat del sensor d'aigua cada 3 segons. Tenir disponible el driver amb tots els actuadors integrats un cop definit el protocol de comunicació ha sigut imprescindible per a poder testejar degudament el firmware de la placa dels timons i la seva integració en el robot.

Tot i que en aquestes alçades del projecte el firmware de la placa electrònica dels timons no estava programat, sabent la quantitat de bytes addicionals que s'envien i es reben amb la incorporació d'aquesta placa, s'ha pogut estimar que la ocupació del bus continuarà sent d'uns 30 ms; per enviar la consigna als timons únicament s'ocupa el bus durant 694,4 us, i per enviar i rebre l'estat del sensor d'aigua uns 1,74 ms. Aquest càlcul permet afirmar que, en cas de ser necessari, per la part de comunicacions és possible ascendir la freqüència del sistema de control fins als 30 Hz.

5. FIRMWARE

Tot el hardware dissenyat pels timons de profunditat no serveix de res si en el microcontrolador de la placa electrònica no se l'hi proporciona un seguit d'instruccions adients a executar. El firmware és el conjunt d'instruccions de programa gravades en una memòria no volàtil que estableixen la lògica de funcionament dels equips de més baix nivell, l'electrònica. Per tant, es pot dir que el firmware fa d'interfície entre el software i el hardware.

Específicament en aquest projecte, el firmware és l'encarregat d'entendre les comandes rebudes pel port sèrie i executar la tasca que li encomanen. Aquestes poden ser configurar els servomotors, activar-los, desactivar-los, posicionar-los, informar de la versió de firmware i de l'estat del sensor d'aigua, tal i com s'ha vist en el capítol anterior.

El codi programat en el microcontrolador es basa en tres grans fases, de les quals, les dos primeres s'executen automàticament al proporcionar tensió a l'integrat i la tercera s'executa de forma dependent al bus de comunicació.

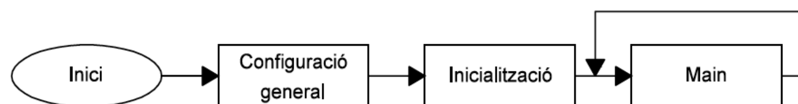

Figura 21. Diagrama de flux general del firmware

A l'annex de programació del firmware es pot veure tot el codi desenvolupat per a la implementació de les idees representades en aquest capítol.

5.1. Fase de configuració general

Quan s'alimenta el microcontrolador, el primer que realitza el codi és redirigir el punter de lectura de l'integrat cap a l'arxiu de configuració. Aquest fitxer caracteritza tots els serveis disponibles al microcontrolador d'acord amb les necessitats del present projecte.

De les diferents accions realitzades en aquesta fase cal destacar-ne algunes, com per exemple, la inclusió de la llibreria del propi PIC16F1829 i la configuració de l'ús d'un senyal de rellotge extern de 11,0592 MHz, dels registres de la memòria EEPROM i els diferents terminals de l'integrat. També s'aprofita aquesta fase per definir les característiques del port

de comunicacions amb els paràmetres adients i per definir un nom als terminals del dispositiu, així facilitant el seu ús al llarg del firmware.

Un cop llegit tot l'arxiu de configuració, el punter del microcontrolador retorna al programa principal, on es segueix adjuntant un servei de buffer per la comunicació sèrie i una llibreria imprescindible per realitzar la conversió de format de les cadenes de caràcters.

5.2. Fase d'inicialització

Un cop configurat el microcontrolador, cal inicialitzar els serveis que s'utilitzaran de l'integrat i totes les variables que s'hagin de fer servir en el programa. Per això, fora del bloc pertanyent al bucle principal, s'han definit les tasques a executar pels tres serveis utilitzats: interrupcions provinents de dos temporitzadors i del port sèrie. Un temporitzador s'ha inicialitzat per generar una interrupció cada 3,3 ms, mentre que l'altre s'ha habilitat per ser reconfigurable en tot moment oferint una resolució de 11,5625 us. Pel que fa a la interrupció del port sèrie, aquesta s'ha programat perquè sempre que es rebi quelcom es guardi la informació en el buffer adjuntat anteriorment a la fase de configuració general.

Per annexar les variables utilitzades pels serveis anteriors amb la el bucle principal, ha calgut declarar i inicialitzar variables globals. D'aquesta manera s'ha aconseguit disposar del seu valor en qualsevol part del programa. Per contra, totes aquelles variables que només s'han utilitzat dins el bucle principal únicament ha calgut declarar-les dins la mateixa funció. Per a totes les inicialitzacions de les variables s'ha definit el tipus adequat segons les característiques de la informació destinades a contenir.

Per aquelles variables que han de contenir els paràmetres de configuració de cada servomotor, després de ser declarades se les hi ha carregat els valors guardats a la memòria EEPROM del microcontrolador. Cal destacar que sempre que es gravi un codi nou en el dispositiu, la memòria EEPROM s'esborra, pel que caldrà utilitzar les comandes de configuració per guardar de nou els paràmetres desitjats.

5.3. Fase principal

Després de les configuracions i inicialitzacions necessàries, s'executa la fase principal del programa, la qual està activa indefinidament sempre que no s'interrompi l'alimentació del

microcontrolador. Durant aquesta fase l'integrat realitza tres grans funcions. Per una banda, gestiona tota la part de comunicació del port sèrie, atenent i responent les comandes rebudes des de l'ordinador central del robot. Independentment a aquesta tasca, si el timó de profunditat està habilitat, el microcontrolador genera el senyal de control corresponent pels dos servomotors. Finalment, el microcontrolador llegeix constantment el senyal procedent del sensor d'aigua per garantir la seguretat del vehicle.

5.3.1. Gestió del bus

La gestió del port sèrie comença amb la interrupció originada al rebre un byte a través del port de comunicació. Cada byte és analitzat per detectar si pertany al protocol de comunicació estipulat, i en cas negatiu tornar a l'estat d'inici. Aquest anàlisi es realitza byte per byte, pel que per determinar la pertinença al protocol de comunicació es consideren els bytes rebuts anteriorment. Cal destacar que segons el primer byte de cada trama el programa detecta si s'ha rebut una comanda d'execució o una d'usuari. A partir d'aquesta diferenciació s'executa una part de programa o altra atenent que els dos tipus de comandes són totalment diferents. La major part del codi únicament serveix per validar les comandes, mentre que l'altra part serveix per actuar com correspongui.

Pel que fa a les comandes d'execució, les quals s'analitzen segons la branca esquerra del diagrama de flux de la Figura 22, un cop validat l'inici de la trama es mira si la seva finalitat és modificar el posicionament dels timons o una altra. Pel primer cas, s'espera un byte addicional, el BCC, el qual es comprova que acordi amb el calculat dins el propi integrat. En cas afirmatiu, s'aplica la transformació a les consignes que permet disposar d'una resolució de mig grau. A partir d'aquest últim es calcula el valor que han de tenir els temporitzadors, tal i com s'adjunta a l'annex de càlculs.

En cas de tractar-se d'una altra comanda d'execució, es valida que la seva estructura compleixi amb el protocol de comunicació; si és correcte, es procedeix a actuar segons correspongui. Per a habilitar els timons, s'estipula que la seva consigna és de zero graus, es calculen els valors pels temporitzadors i finalment s'activen. Per a inhabilitar-los, simplement es desactiven els mateixos temporitzadors. En cas d'haver rebut una pregunta per l'estat del sensor d'aigua, s'envia l'inici de la trama, els dos bytes de resposta compresos en la variable aigua i finalment, es reseteja el filtre del sensor d'aigua. Si la comanda rebuda és per habilitar o inhabilitar les comandes d'usuari, s'actualitza una variable i s'envia l'estat actual de la mateixa, permetent a l'usuari saber si pot utilitzar les esmentades comandes.

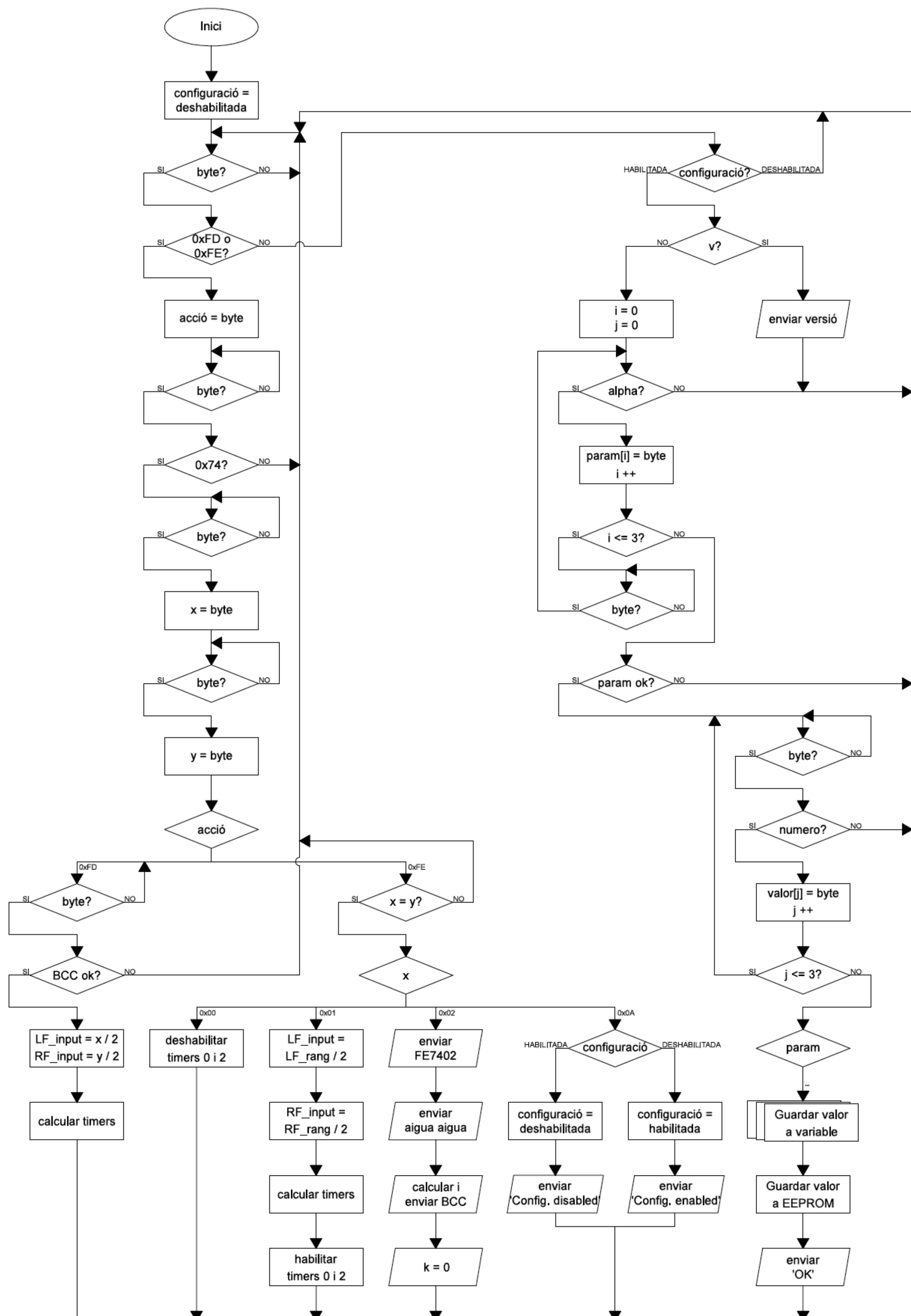

Figura 22. Diagrama de flux de la gestió del port sèrie

Respecte les comandes d'usuari, aquestes s'analitzen segons la branca dreta del diagrama de flux de la Figura 22. Primer de tot es verifica que la seva funcionalitat estigui activada, i en cas contrari, es descarten els seus bytes. Si les esmentades comandes estan habilitades, es comprova si s'està demanant informació sobre la versió del firmware, conseqüentment enviant-la. Contràriament, s'espera una de les trames per a configurar els servomotors.

En aquest últim cas s'analitza que l'inici de comanda sigui tota alfabètica i de la longitud marcada, pel que si s'assoleixen aquestes especificacions es verifica que el seguit de dígits alfabètics corresponguin a un paràmetre dels descrits en el protocol de comunicació. Tot seguit es procedeix a analitzar el valor numèric pel paràmetre, tot verificant que sigui un número i de longitud correcte. Finalment, en cas que totes les comprovacions anteriors hagin sigut satisfactòries, es guarda el valor numèric a la variable corresponent i a la memòria EEPROM, deixant actualitzat el paràmetre tant per l'ús immediat dels timons com per usos posteriors a l'apagada del robot. Per a informar que tot el procediment s'ha executat correctament, s'envia un missatge de confirmació a l'usuari.

Tot i que el diagrama de flux anterior que descriu la gestió del port sèrie detalla acuradament el firmware programat, hi ha accions rutinàries o mesures de seguretat que no s'hi han pogut representar breument. Per una banda, per a enviar quelcom pel port sèrie cal posar prèviament la sortida del microcontrolador connectada al transceptor de línia RS-485 en estat d'emissió, i un cop enviada tota la trama, s'ha de retornar en estat de recepció.

Pel que fa a les mesures de seguretat, abans de guardar i processar les consignes pels timons es comprova que aquestes estiguin compreses pel rang determinat pels paràmetres de configuració; en cas de no complir-se, s'aproxima al límit del rang més proper per evitar que l'eix del servomotor no ofereixi parell. A més a més, quan es troba un cas indefinit en una estructura de selecció s'ha programat que el punter de lectura de l'integrat retorni a l'inici de la part cíclica del codi de gestió del port sèrie, assegurant que el programa no es pot corrompre.

5.3.2. Generació dels senyals de control

La generació del senyal de control per a cada un dels servomotors és totalment dependent a la part de codi que gestiona les dades rebudes pel bus RS-485; si aquest no ha habilitat els temporitzadors i no els hi ha configurat un valor, el senyal PWM no es podrà generar. Per

això, tot i executar-se independentment, podem dir que les ordres rebudes pel bus predominen en tot moment sobre el sistema de posicionament dels timons.

Per a generar els senyals PWM s'utilitzen un total de dos temporitzadors, els quals no són independents per a cada senyal, sinó que amb les dues interrupcions s'aconsegueix cada un dels dos senyals. Aquest mode de funcionament ha sigut el resultat de considerar les propietats dels temporitzadors existents en l'integrat: únicament un entre els cinc temporitzadors disponibles és reprogramable, pel que la resta no es poden utilitzar per generar un PWM amb longitud de pols variable.

Davant aquesta limitació, s'ha enginyat una estratègia capaç de generar cada un dels dos senyals de control a una freqüència aproximada de 300 Hz a partir de dos temporitzadors de 8 bits; un no reconfigurable, el número dos pel PIC, i el reconfigurable, el número zero. El primer, el qual està indicat en vermell a la Figura 23, d'acord amb la configuració realitzada prèviament, genera una interrupció cada 3,3 ms per marcar l'inici del període del senyal a 300 Hz. En aquest moment es posen les dues sortides en estat alt i es configura el temporitzador zero, indicat en verd a la Figura 23, segons unes condicions dependents a la durada desitjada pels dos pols.

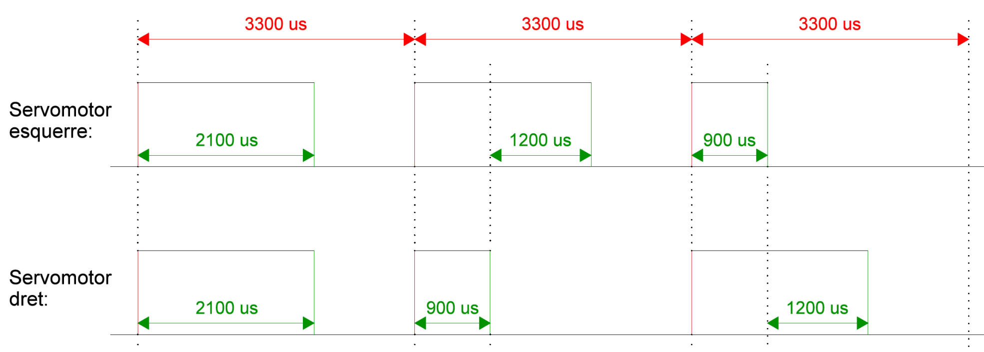

Figura 23. Estratègia seguida per a la generació dels senyals de control PWM

Si els dos pols han de ser igual de llargs, el temporitzador zero es configura amb aquest valor, pel que quan es doni la interrupció, les dues sortides es posaran en estat baix. Per contra, si els dos pols han de ser diferents, el temporitzador zero es programarà amb l'equivalent al temps menor. Quan es generi la interrupció, es posarà en estat baix la sortida corresponent i es configurarà el temporitzador zero amb el valor corresponent a la diferència entre el pols llarg i curt. Finalment, quan de nou torni a haver-hi aquesta interrupció, es posarà en estat baix l'altra sortida.

Un cop finalitzada aquesta seqüència, el microcontrolador manté les dues sortides en estat baix fins que es torna a generar la interrupció associada al temporitzador número dos, amb la qual es repeteix tot el procediment explicat anteriorment, el qual és descriptible segons el diagrama de flux que segueix.

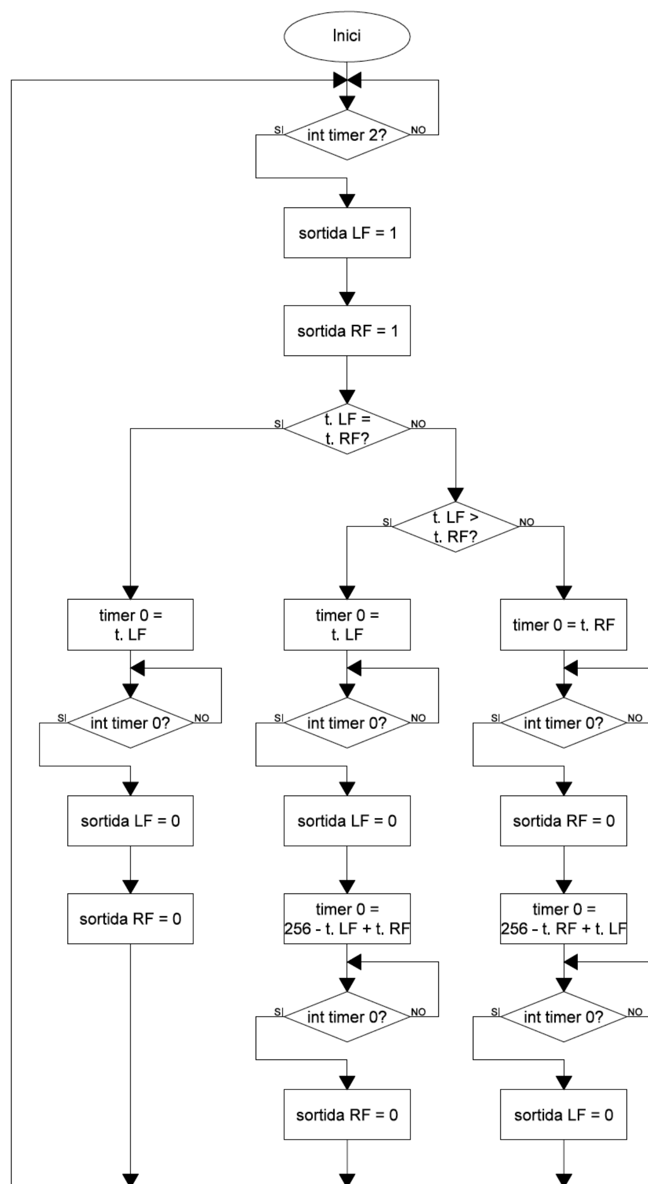

Figura 24. Diagrama de flux de la generació del senyal de control

5.3.3. Lectura del sensor d'aigua

Per a motius de seguretat del vehicle s'ha optat per realitzar una lectura del senyal provinent del circuit que tracta el sensor d'aigua no dependent a cap agent extern, és a dir, mesurar contínuament el seu estat. Tanmateix, s'ha vist oportú aplicar un filtre a la lectura de

l'esmentat senyal amb el fi d'evitar fer emprendre al robot una situació d'emergència per culpa de dades sorolloses.

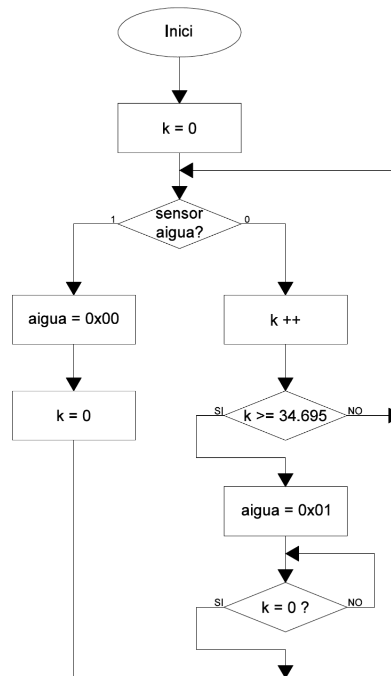

Figura 25. Diagrama de flux del filtre aplicat al sensor d'aigua

L'esmentat filtre no valida l'existència d'aigua dins del compartiment dels timons sinó ha llegit l'entrada del microcontrolador en estat baix 34.695 vegades seguides, el que equival a uns 450 ms segons els ajustos experimentals realitzats. En aquest cas, s'assigna el valor hexadecimal 0x01 a la variable aigua, la qual és utilitzada per confeccionar la deguda comanda de resposta a la part de codi que gestiona el port sèrie. A més, per assegurar que aquesta informació serà enviada a través del port sèrie, es manté el seu valor fins que el comptador del filtre on és resetejat pel propi codi gestor del port sèrie.

De nou s'ha vist que tot i tenir parts del firmware que s'executen parcialment de forma independent a la resta del codi, mantenen certa interactivitat amb el gestor del port sèrie, tot mantenint la total funcionalitat de la placa electrònica i garantint la seguretat del vehicle.

6. MODELITZACIÓ

Per a poder desenvolupar i ajustar satisfactòriament un sistema de control per a l'SPARUS II és imprescindible quantificar prèviament com els seus actuadors interaccionen amb el sistema que l'envolta, l'aigua. Tot i que en el capítol de l'estudi previ dels timons de profunditat s'han exposat els models teòrics de tots els actuadors del robot, tal i com s'ha vist hi ha una sèrie de coeficients que cal determinar.

Respecte el model dels motors horitzontals, aquest s'ha refet per a contemplar les característiques hidrodinàmiques de les noves hèlixs instal·lades en el robot i per a considerar el terme dinàmic en el model d'aquests actuadors. Tal i com s'ha comentat anteriorment, aquest últim terme pren gran importància amb les velocitats en surge de 2 m/s que el robot assolirà amb la realització d'aquest projecte.

Per altra banda, s'ha hagut de modelitzar l'efecte dels timons de profunditat integrats en el robot per a conèixer el seu comportament. Tot i el complex estudi previ realitzat, s'han apreciat diferències significatives entre les equacions teòriques exposades anteriorment i el comportament real de les pales un cop integrades en el robot. Per això, s'ha decidit modelar experimentalment aquests nous actuadors.

6.1. Modelització dels motors

La modelització del conjunt motor i hèlix s'ha realitzat a partir de l'Equació 4 del present document. Aquesta expressa que la força d'empenyiment transmesa al fluid per una hèlix en moviment es veu condicionada per la rotació d'aquesta i la velocitat del fluid entrant en el motor. Per això, al moment de parlar d'aquesta equació es poden diferenciar dues parts, l'estàtica i la dinàmica.

Per a trobar el valor del coeficient estàtic C_1 s'han realitzat un seguit d'assaigs a la piscina del CIRS on el terme dinàmic no influenciava, ja que el motor es mantenia quiet. Això ha sigut possible a partir d'una estructura de ferro galvanitzat articulada pel la meitat, on en un extrem s'hi col·loca un motor amb la hèlix a analitzar i a l'altre s'hi instal·la un dinamòmetre. Amb aquests experiments es prenen dades de la força d'empenyiment realitzada pel motor, el seu consum i la velocitat de gir del seu eix per a diferents consignes de motor, les quals estan compreses entre -1 i 1 d'acord amb el seu protocol.

De les dades obtingudes s'ha pogut extreure el model estàtic dels motors, el qual és diferent segons si la força d'empenyiment realitzada o les consignes enviades són positives o negatives. Per tant, la dependència de la força realitzada pels motors en funció de la velocitat de gir en Hz al quadrat és, per consignes positives, com es mostra a continuació.

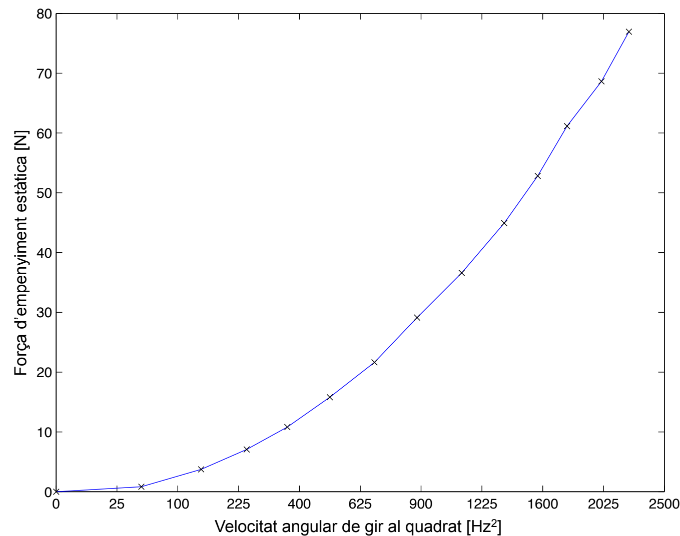

Figura 26. Característica estàtica del motor per a consignes positives

Aquesta dinàmica s'ha ajustat amb un coeficient de determinació R^2 de 0,9955 a una recta de pendent 0,03465.

Per contra, si les consignes són negatives, la relació entre la força d'empenyiment i la velocitat de rotació de l'eix en Hz al quadrat és com segueix.

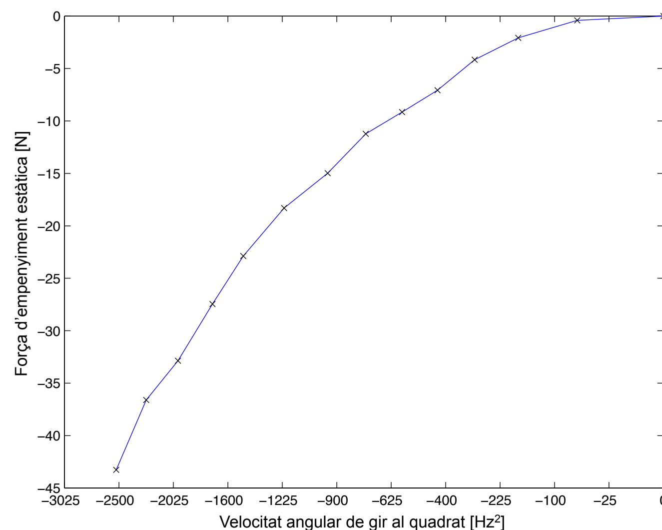

Figura 27. Característica estàtica del motor per a consignes negatives

La dinàmica observada a la Figura 27 s'ha ajustat amb un coeficient de determinació R^2 de 0,9936 a una recta de pendent 0,01659. Aquests dos pendents trobats són equiparables a tots els termes que multipliquen a N^2 de l'Equació 4, pel que es pot dir que el coeficient C_1 per a consignes positives val 0,4438 i per a negatives val 0,2124. Tanmateix, a partir d'aquests experiments també s'ha pogut determinar la força màxima que es pot realitzar amb les noves hèlixs, sent de 77 N per a consignes positives i de -43 N per a negatives.

Per altra banda, ha calgut determinar el coeficient que caracteritza la part dinàmica del model, és a dir, el valor de tots els termes que multipliquen U_{0T} de l'Equació 4. Únicament considerant el model estàtic trobat pels actuadors, s'han fet experiments a diferents velocitats de surge, els quals han consistit en assignar una força igual pels dos motors i llegir la velocitat que assolia el vehicle. Amb aquesta, tot calculant el model invers de fregament en surge del robot s'ha pogut saber la força real que s'estava transmetent al robot, el que ha permès definir l'error comés pel model estàtic. Analitzant totes les dades, s'ha ajustat el coeficient C_2 amb els valors de 0,71 per a consignes positives i de 0,34 per a negatives, el que ha eliminat considerablement l'esmentat error en tots els punts de treball.

Un cop definit completament el model dinàmic dels nous motors horitzontals de l'SPARUS II, s'ha hagut de trobar el model dinàmic invers per adaptar-lo a la manera que el control que es dissenyarà l'utilitzarà. Aquest calcularà la velocitat de gir del motor en rpms en funció de la força que es demani realitzar als actuadors i la velocitat d'entrada de l'aigua als mateixos, la qual pot ser aproximada a la velocitat en surge del vehicle.

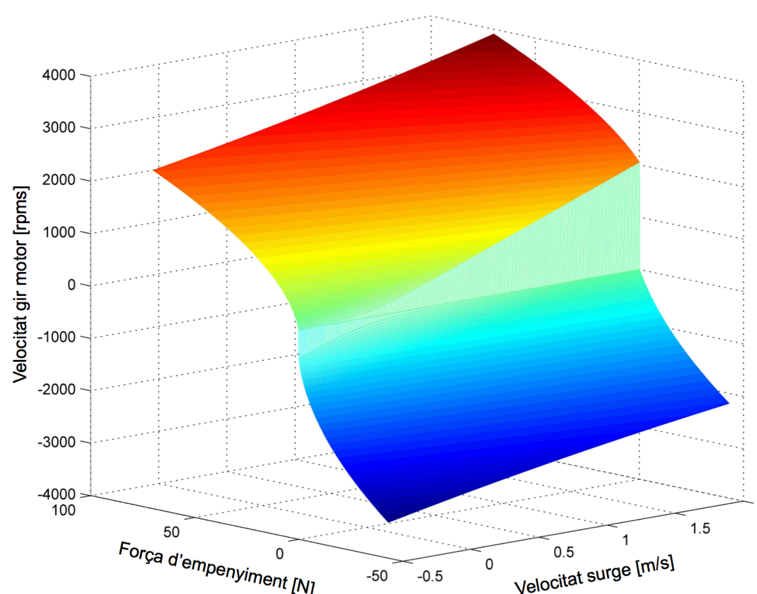

Figura 28. Model dinàmic invers dels motors horitzontals de l'SPARUS II

Gràcies a la representació del model dinàmic invers dels motors processada amb el MATLAB, la qual s'observa a la Figura 28, s'ha tingut una idea més tangible de la variació de la velocitat angular de l'eix del motor en rpms en funció de la velocitat del fluid a la tubera del mateix i la consigna de força d'empenyiment. Tanmateix, aquesta representació ha servit per validar que el model invers conserva totes les característiques del model directe.

Al calcular el model invers representat amb el gràfic anterior, però, no s'obté la consigna que entén l'electrònica dels motors, ja que el seu protocol de comunicació indica que han de ser valors compresos entre -1 i 1. Aquestes representen la proporció de la tensió d'alimentació de l'electrònica dels motors que és proporcionada en els motors en tant per u. Per això, s'han modelat els actuadors en qüestió d'acord amb el seu circuit elèctric aproximat, fent possible passar de la velocitat de rotació del motor obtinguda en rpms al voltatge necessari en borns del motor.

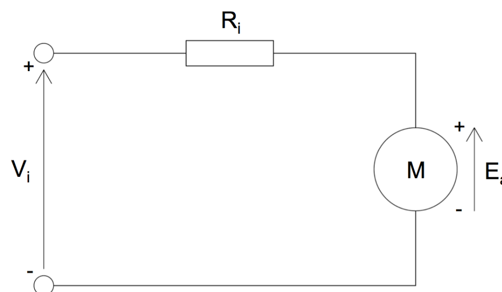

Figura 29. Circuit elèctric simplificat del motor a modelitzar de l'SPARUS II

Primer de tot s'estima la tensió necessària a l'armadura perquè el rotor giri a la velocitat angular desitjada. La relació existent entre aquests dos paràmetres es troba caracteritzada pel flux magnètic, la qual s'ha tractat com a una constant a trobar experimentalment.

$$E_a = C_F \cdot n \qquad \text{(Eq. 10)}$$

on,

E_a = tensió desitjada en l'armadura del motor, en volts

C_F = coeficient característic del flux magnètic del motor, en volts per segon partir per radians

n = revolucions desitjades per l'eix del motor, en revolucions per minut

Seguidament, sabent que qualsevol motor té una resistència interna i que per tant, la tensió d'armadura no és igual a l'existent en els borns de l'actuador, aplicant la llei d'Ohm s'ha pogut trobar la tensió a la que s'ha d'alimentar el motor perquè l'eix d'aquest roti a una velocitat angular desitjada. De nou, la resistència interna del motor s'ha tractat com una constant, la qual també s'ha trobat experimentalment.

$$V_i = E_a + C_R \cdot I \qquad \text{(Eq. 11)}$$

on,

V_i = tensió desitjada en borns del motor, en volts

E_a = tensió desitjada en l'armadura del motor, en volts

C_R = coeficient que caracteritza la resistència interna del motor, en ohms

I = corrent que consumeix el motor, en ampers

Finalment, per a obtenir la consigna a enviar als actuadors, s'ha aplicat un ratio entre el resultant anterior i la tensió d'alimentació de l'electrònica de cada actuador, així fent el càlcul invers que realitzarà l'electrònica del motor. D'aquesta manera s'aconsegueix una consigna compresa entre -1 i 1 al mateix temps que s'assegura que la tensió en borns del motor serà la calculada anteriorment. Si els motors es controlessin amb revolucions per minut, aquest últim model no seria necessari.

Abans de poder-se implementar, però, ha calgut trobar el valor dels coeficients C_F i C_R. Per a obtenir el primer, s'han fet experiments que involucressin un consum baix de corrent, conseqüentment podent assumir que la tensió a l'armadura és igual a la dels borns. Per això, aquest coeficient s'ha ajustat manualment per tal que la velocitat de gir angular real de l'eix del motor coincidís amb la consigna obtinguda del model dinàmic invers. Després d'un quants assaigs, s'ha obtingut un valor pel coeficient C_F de 0,0053 V·s/rad Per a l'altre coeficient, s'han realitzat una altra sèrie d'experiments que comportessin consums d'intensitat elevats. D'aquesta manera s'ha pogut ajustar manualment el coeficient C_R a un valor de 0,2 Ω, el qual ha comportat que en diferents punts de treball la consigna de velocitat angular coincidís amb les revolucions per minut reals.

6.2. Modelització dels timons de profunditat

A diferència del model anterior, per a la modelització dels timons de profunditat no s'han utilitzat estrictament les equacions teòriques esmentades en l'estudi teòric del mateix document. Tal i com s'ha explicat, el control a dissenyar per a l'SPARUS II utilitzarà els models inversos dels actuadors, pel que si aquest es processés a partir del model teòric hauria de contemplar tres variables d'entrada: la força sol·licitada per a una pala, la velocitat del fluid incident en el perfil i la força realitzada pel motor situat davant la pala en qüestió. Amb aquesta informació el model determinaria quin angle de pala generaria la força necessària segons el context actual del robot

Tot i això, davant la complexitat per a realitzar els experiments que permetessin ajustar els coeficients de les esmentades equacions, es va optar per fer un model aproximat de la força de sustentació generada per un timó de profunditat. Analitzant les equacions teòriques dels timons de profunditat, exactament el contingut reflectit en l'Equació 3, s'ha observat que la velocitat de l'aigua a la sortida d'un motor es veu molt més influenciada per la força d'empenyiment realitzada per l'actuador que per la velocitat d'entrada del propi fluid a la tubera. Per això s'ha decidit que en el model que es realitzés no es tindria en compte la velocitat del vehicle, reduint la complexitat del model.

Tanmateix, considerant que en el control de força dels motors s'apliquen una sèrie de saturacions, mescles de consigna i complexos càlculs que poden modificar la força que se'ls hi demana, s'ha vist oportú considerar la consigna enviada als motors enlloc de la força que teòricament fan. D'aquesta manera es parteix d'una variable d'entrada sense cap possible error, que representa tant el model hidrodinàmic com el del circuit elèctric del motor.

Per a tot això, el model invers que s'executaria periòdicament en el control per a obtenir l'angle adequat per a cada pala es va veure reduït a dues variables d'entrada, les forces a ser generades per a cada timó i les consignes dels corresponents motors.

L'obtenció del model es va basar en dues fases. La primera va ser experimental, en la qual s'observava l'angle de pitch al que s'estabilitzava el robot per a diferents angles de pala i consignes de motor positives. Aquesta metodologia es va executar per a les consignes de motor de 0,1 a 0,5 amb increments de 0,1 per cada posicionament de la pala comprès entre -10 º i -60 º amb increments de -10 º, és a dir, un total de 30 assaigs. Les característiques de

la piscina del CIRS no van permetre experimentar de forma segura amb consignes de motor majors, ja que el robot assolia velocitats realment altes. Tot i això, les dades restants s'han pogut extrapolar a partir de la informació obtinguda.

Tot i que únicament es van prendre dades amb angles de pala negatius, teòricament no hi ha cap indici que l'efecte absolut d'aquests actuador sigui diferent segons el posicionament del robot. Per això, es va creure totalment vàlid extreure un únic model, el resultat del qual s'adaptés segons el signe de la força d'entrada. Tanmateix, el model realitzat únicament es computarà si les consignes de motor són positives; per a negatives l'angle de les pales des forçarà a zero graus.

La segona part de la modelització va consistir en obtenir les forces generades pels timons a partir d'entendre la seva influència en el DOF pitch, pel que va ser de gran ajudar realitzar el diagrama del cos lliure representant les principals forces que afecten en aquest DOF.

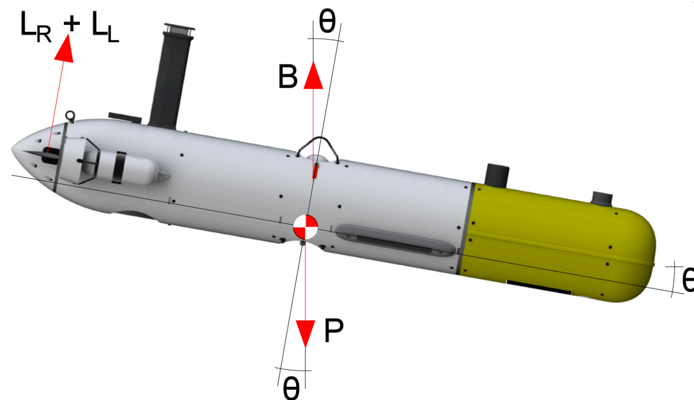

Figura 30. Diagrama del cos lliure de les forces influents en el DOF pitch

De la descomposició de forces respecte els eixos del robot es va poder formular un model simplificat que caracteritza el pitch del robot, el qual es basa en un sumatori de moments respecte l'eix Y. Tot i que altres components tals com l'acceleració de Coriolis o els termes de massa afegida es podrien haver considerat, per a la realització d'aquest model no es van considerar molt influents.

El model del DOF pitch que es presenta a l'Equació 12 no considera l'acceleració angular en l'eix Y ni la inèrcia característica del cos en el mateix eix, ja que el moviment en pitch en tots els experiments acabava sent nul. Conseqüentment, el moment generat pels dos timons de profunditat ha de ser igual al moment generat per la flotació del vehicle.

$$B \cdot \sin(\theta) \cdot a_z = (L_L + L_R) \cdot d_p \qquad \text{(Eq. 12)}$$

on,

B = 580 N. Força de flotació del vehicle, en newtons

θ = angle de pitch assolit pel vehicle, en radians

a_z = 0,02 m. Distància en z de la força de flotació respecte el centre de gravetat, en metres

L_L = força de sustentació realitzada pel timó esquerre, en newtons

L_R = força de sustentació realitzada pel timó dret, en newtons

d_p = 0,65 m. Distància en X entre el centre del robot i el centre d'esforços dels timons

Del model anterior es va extreure la força generada per a un timó, el que va ser possible al realitzar els experiments amb consignes iguals pels mateixos actuadors, és a dir, assegurant que la força de sustentació realitzada per a cada pala seria igual. La dinàmica d'aquests resultants en funció dels angles de pala i de les consignes de motor positives, és a dir, el model directe, es va representar amb l'ajuda del MATLAB per tal de comprovar la seva lògica i veracitat. També va donar una idea visual de la interacció i influència de les dues variables d'entrada en la generació de forces sustentadores.

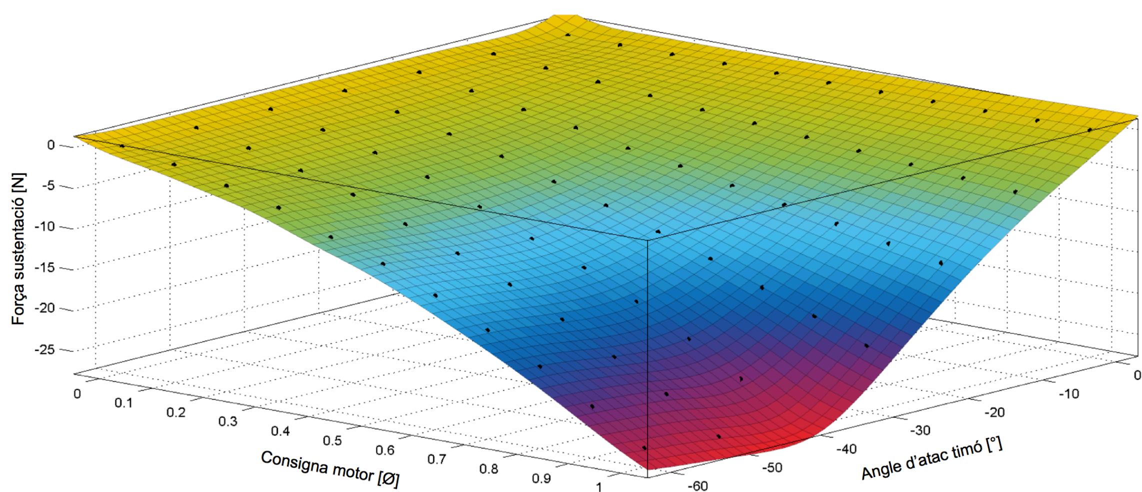

Figura 31. Model directe dels timons de profunditat de l'SPARUS II

Amb aquest gràfic es va poder determinar que l'angle òptim d'atac dels timons de profunditat de l'SPARUS II és de ±42 º, ja que amb aquest posicionament es pot arribar a generar la força màxima de ±26,15 N. A nivell de software l'angle d'atac òptim s'ha arrodonit a ±40 º.

Tot i haver representat el model directe realitzat pels timons de profunditat al ser més entenedor, el control del robot necessita el model invers. Per això, a partir de les dades obtingudes es va aproximar la seva dependència a un polinomi de tercer ordre, aconseguint un ajustament caracteritzat per a un coeficient de determinació R^2 de 0,9897.

$$\alpha = \sqrt{268.140{,}86 \cdot s^2 - 21.481{,}84 \cdot s + 1.544{,}68 \cdot L + 832{,}42} - 517{,}82 \cdot s + 191{,}51 \quad \text{(Eq. 13)}$$

on,

α = angle d'atac calculat pel timó, en graus

s = consigna real del motor, adimensional

L = força de sustentació desitjada, en newtons

Cal recordar que el model realitzat és únicament vàlid per a una entrada de consignes de motor positives i consignes de força de sustentació negatives; en cas que la primera condició no es compleixi, per codi es fixa que l'angle d'atac pel timó de profunditat sigui zero graus. En cas de ser la segona que no es compleix, es canvia el signe per a poder computar el model i al resultat se li adequa el signe a positiu.

Tanmateix, hi ha combinacions de les variables d'entrada que donen nombres imaginaris, el qual succeeix amb consignes de motor molt petites. Per això, considerant l'efecte menyspreable que tindrien les forces de sustentació generades, la consigna de les pales es fixa a zero graus. Per altra banda, si la força demanda és major a la màxima que pot ser feta pel timó, directament es força la seva posició a ±40 º segons correspongui.

Finalment, al llarg de l'experimentació amb el robot s'ha observat un comportament destacable que no s'havia considerat en el model anterior; quan el vehicle es mou únicament en heave i per tant, les consignes dels motors horitzontals són zero, el model

amb el seu seguit de condicions estableixen que el posicionament dels timons ha de ser de zero graus. Conseqüentment, aquests actuadors generen dues forces de resistència orientades en l'eix Z del robot que, al estar 0,65 m separades del centre de gravetat del vehicle, provoquen un gran moment en pitch que comporta que el vehicle es posicioni amb un angle de pitch no desitjat.

Per a solucionar-ho s'ha forçat per codi que davant aquestes combinacions la consigna dels timons fos de ±60 º, ja que l'angle d'atac del fluid en els perfils sustentadors seria mínim, pel que la força de resistència originada també ho seria. Per això, quan el vehicle es mou positivament en Z o, el que és el mateix, es submergeix, els timons es situen a 60 º, aconseguint un angle d'atac mínim de 30 º. Per contra, quan el vehicle es mou negativament en Z, els timons es situen a -60 º, aconseguint el mateix efecte. En el cas de no respectar-se aquesta lògica, l'angle d'atac resultant seria el doble en els dos casos, fet que cal evitar.

7. CONTROL

El sistema de control és un element imprescindible en un robot, i més si aquest és autònom; les capacitats de moviment en els diferents DOFs, la velocitat de reacció, la fiabilitat del vehicle i en general, les limitacions del robot, poden dependre en gran part del seu sistema de control. A tall d'exemple, no serviria de res disposar de tota la mecànica i electrònica del timons de profunditat implementada fins al moment si no es proporcionés a l'SPARUS II un controlador capaç d'interaccionar amb tota la seva arquitectura.

Abans de la realització d'aquest projecte, però, el robot ja disposava d'un complex sistema de control, el qual s'ha mantingut com a part bàsica de l'arquitectura COLA[2]. Principalment, aquest es divideix en dues grans parts: un control d'alt nivell i un de baix nivell.

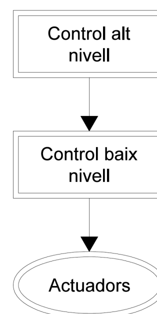

Figura 32. Sistema de control de l'SPARUS II

El control d'alt nivell genera consignes específiques per a cada DOF d'acord amb la tasca que s'encomani complir al robot, com per exemple, anar a unes coordenades globals del món, seguir trajectòries, mantenir una posició o desplaçar-se a una alçada constant del fons marí. Les consignes resultants d'aquest control són tractades pel control de baix nivell.

En canvi, el control de baix nivell s'encarrega de gestionar els graus de llibertat de manera desacoblada, és a dir, d'assolir individualment per a cada DOF les posicions, velocitats i forces desitjades pel control d'alt nivell o altres nodes de l'arquitectura, com el de seguretat o el de teleoperació. Això és possible gràcies a la realimentació proporcionada per la navegació del vehicle, la qual estima contínuament les posicions i velocitats tot aplicant un EKF (Extended Kalman Filter) a totes les dades provinents dels sensors de navegació.

Concretament, el control de baix nivell està concebut per tres controladors concatenats, un de posició, un de velocitat i un de força. La sortida de tot aquest control són les consignes

pels actuadors, les quals són enviades a través d'un topic de ROS al driver anteriorment programat. Aquesta tipologia de control també s'ha mantingut, ja que experiències anteriors al CIRS han determinat que dóna bons resultats pel tipus de vehicle a controlar.

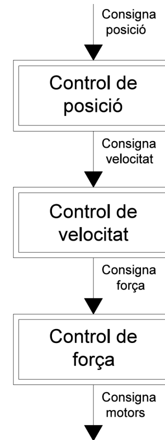

Figura 33. Estructura del control de baix nivell de l'SPARUS II

Així doncs, abans de la realització d'aquest projecte, l'SPARUS II ja disposava d'una estructura de control que s'executava a 10 Hz totalment vàlida per tres DOFs, tres motors i velocitats en surge de fins a 0,5 m/s. Considerant els objectius descrits en el present document, s'ha vist la necessitat de modificar individualment cada una de les parts que constitueixen el control de baix nivell amb el fi d'incorporar els timons de profunditat i poder assolir la velocitat de 2 m/s en surge pel qual el robot va ser dissenyat. Tanmateix, s'ha aprofitat la reforma per a optimitzar l'esmentat control de baix nivell, passant d'un codi escrit en Python a C++. Tot el codi programat es pot veure a l'annex de programació del control.

7.1. Control de posició

El concepte de posició en el camp de la robòtica es pot dividir en dues parts: la ubicació i la orientació. La primera significa la distància existent entre els centres dels sistemes de coordenades del robot i d'un punt de referència, sent expressada vectorialment respecte els eixos d'aquest últim. Per altra banda, la orientació és la diferència de rotació del sistema de coordenades del robot envers el del sistema de referència.

Tot i que la ubicació del vehicle pot ser determinada a partir d'un posicionament georeferenciat respecte la terra, és a dir, amb la longitud i la latitud, amb robòtica submarina es sol treballar amb un sistema de coordenades anomenat NED (North-East-Down) (Fossen,

2011). Aquest és un punt de referència relatiu pel robot, el qual es sol ubicar a prop de la seva zona de treball i sempre orientat en Nord, en Est i cap al centre de la terra, tal i com es pot observar a continuació.

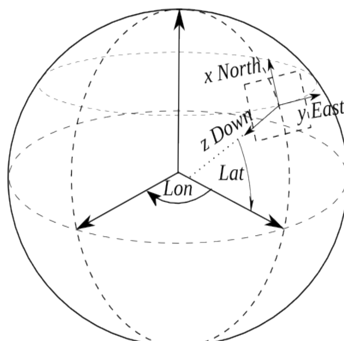

Figura 34. Coordenades geogràfiques i NED (Fossen, 2011)

El NED no únicament serveix per determinar la ubicació del robot, sinó també la seva orientació; per exemple, quan es parla del yaw es fa referència a la rotació necessària en l'eix Z del robot per tal d'igualar el sentit del seu eix X amb el Nord. Per a tot això la posició de l'SPARUS II ve marcada per un vector de sis termes. Els tres primers indiquen la posició, o el que és el mateix, el Nord, l'Est i la profunditat, mentre que els tres restants indiquen la rotació respecte l'eix X, Y i Z, és a dir, la orientació en roll, pitch i yaw respectivament.

Aquest format de vector és l'utilitzat per a l'arquitectura COLA[2] per enviar una consigna de posició al sistema de control. A més, com pot ser d'interès influenciar únicament en unes certes ubicacions o orientacions, l'anterior missatge s'acompanya amb un vector booleà d'igual longitud, el qual indica al control quins termes ha de considerar i quins no.

Tot i això, abans de realitzar-se cap acció i tenint en compte que més d'un node de la pròpia arquitectura pot publicar una consigna a la mateixa iteració, el primer que es realitza dins el propi control és filtrar i combinar totes les dades rebudes. A grans trets, el filtre escull per cada eix les consignes amb més prioritat, aquestes sent les provinents dels nodes de seguretat, seguides per les de teleoperació i finalment, les resultants del control d'alt nivell o altres nodes. D'aquesta manera s'assegura que les accions de gran importància s'executen dins el sistema de control, garantint la seguretat i integritat del vehicle.

Un cop escollits els termes de posició prioritaris, el mateix control transforma la ubicació desitjada resultant, la qual està referenciada respecte el NED, al sistema de coordenades

XYZ del robot, ja que el control treballa respecte aquest últim. Per això s'ha programat una transformació inversa considerant les tres rotacions amb l'ordre ZYX, tal i com es pot apreciar amb les següents equacions.

$$R_{ZYX} = \begin{pmatrix} c(\psi)\cdot c(\theta) & c(\psi)\cdot s(\theta)\cdot s(\phi) - s(\psi)\cdot c(\phi) & c(\psi)\cdot s(\theta)\cdot c(\phi) + s(\psi)\cdot s(\phi) \\ s(\psi)\cdot c(\theta) & s(\psi)\cdot s(\theta)\cdot s(\phi) + c(\psi)\cdot c(\phi) & s(\psi)\cdot s(\theta)\cdot c(\phi) - c(\psi)\cdot s(\phi) \\ -s(\theta) & c(\theta)\cdot s(\phi) & c(\theta)\cdot c(\phi) \end{pmatrix} \quad \text{(Eq. 14)}$$

$$T = \begin{pmatrix} R_{ZYX} & & & p_a \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{(Eq. 15)}$$

$$d = T^{-1}\cdot p_d \quad \text{(Eq. 16)}$$

on,

R_{ZYX} = matriu de rotació ZYX del robot respecte el NED, adimensional

Ψ = rotació en yaw del robot respecte el NED, en radians

θ = rotació en pitch del robot respecte el NED, en radians

Φ = rotació en roll del robot respecte el NED, en radians

T = matriu de transformació homogènia del robot respecte el NED, adimensional

p_a = vector de la ubicació actual del robot respecte el NED, en metres

d = vector de la distància fins al destí desitjat respecte el robot, en metres

T^{-1} = matriu homogènia de rotació ZYX inversa del robot respecte el NED, adimensional

p_d = vector de la ubicació desitjada del robot respecte el NED, en metres

Amb els càlculs anteriors s'obté la distància entre el destí i la posició actual del robot, o dit d'una altra manera, l'error que ha de rectificar el control. Per a obtenir l'error d'orientació del robot s'aplica una resta per a cada DOF entre l'orientació desitjada i l'actual i es normalitza el resultat, és a dir, s'adequa el valor anterior al menor gir necessari per orientar-se a una consigna; per assolir una orientació de 20 º partint dels 340 º, la resta directa dóna un error de -320 º. Tot i això, el més lògic és girar en sentit contrari o corregir únicament un error de 40 º. Aquesta adequació de l'error d'orientació és el que realitza la normalització.

Tot i això, considerant que l'SPARUS II no disposa del grau de llibertat sway, aquest mètode de posicionament XYZ no resulta gaire útil, ja que únicament s'aconseguiria la posició orientada en el surge i en el heave del vehicle. D'aquest fet sorgeix la necessitat de disposar d'un control d'alt nivell capaç de proporcionar solucions a qualsevol tasca que es vulgui executar tot combinant la funcionalitat dels diferents DOFs del vehicle.

Un cop obtinguts els errors de posicionament del vehicle s'aplica un control en llaç tancat del tipus PID (Proportional-Integral-Derivative) amb la part derivativa a la realimentació, el qual s'expressa en temps discret com es mostra a continuació.

$$u(n) = K_p \cdot e(n) + \frac{K_p}{T_i} \cdot \sum_n \left(e(n)_n \cdot T\right) - K_p \cdot T_d \cdot \frac{r(n) - r(n-1)}{T} \qquad \text{(Eq. 17)}$$

on,

$u(n)$ = sortida del controlador, adimensional

K_p = paràmetre referent al guany proporcional, adimensional

$e(n)$ = error existent entre la consigna i la realimentació, unitats del SI

T_i = paràmetre referent al temps integratiu, adimensional

T = diferència de temps entre la iteració actual i l'anterior, en segons

T_d = paràmetre referent al temps derivatiu, adimensional

r(n) = senyal de realimentació, unitats del SI

r(n-1) = senyal de realimentació a la iteració anterior, unitats del SI

Tanmateix, tot i no haver-se expressant a l'Equació 17, el propi controlador conté un terme anti windup, el qual serveix per evitar que el terme integral del PID es carregui indefinidament i conseqüentment introdueixi retards en el control. El seu valor de saturació ve marcat per un paràmetre anomenat i_limit.

La consigna pels controladors de posició és l'error calculat anteriorment, mentre que el senyal de realimentació és un zero. El PID es calcula a partir dels errors per simplicitat i efectivitat en el codi, ja que per calcular-lo de la manera clàssica caldria fer alguna operació addicional a les anteriors.

El resultat de tots els controladors es satura entre -1 i 1, de tal manera que únicament multiplicant el resultat per una constant s'aconsegueix escalar tota la dinàmica del controlador a una magnitud física nova. D'acord amb l'estructura de control ja existent, els DOFs de surge, heave i yaw s'escalen a la velocitat lineal o angular màxima característica del robot.

Pels dos nous DOFs, el pitch i el roll, no es va veure coherent realitzar un control de velocitat d'aquests; per la maniobralitat del vehicle no aportaria cap funcionalitat útil, ja que en les tasques a realitzar no és d'interès comandar aquests graus de llibertat a partir de velocitats. Per això, al moment d'escalar el resultat dels seus PIDs de posició, es va fer d'acord amb els moments màxims en pitch i en roll al que es pot veure sotmès el robot com a conseqüència de la influència dels timons de profunditat.

Finalment, la consigna resultant del pas anterior es satura d'acord amb uns valors modificables fàcilment en un fitxer de configuració. Aquest saturador, per exemple, és de gran interès en tasques executades dins d'espais confinats, on és de vital importància assegurar que el robot no assolirà grans velocitats ni aplicarà certes forces en els DOFs.

En resum, el control de posició de l'SPARUS II es pot esquematitzar amb el diagrama que s'exposa a la Figura 35, en el qual s'indica de color verd tot el que s'ha modificat del control anterior i de color vermell tot el que s'ha incorporat amb la realització del present projecte.

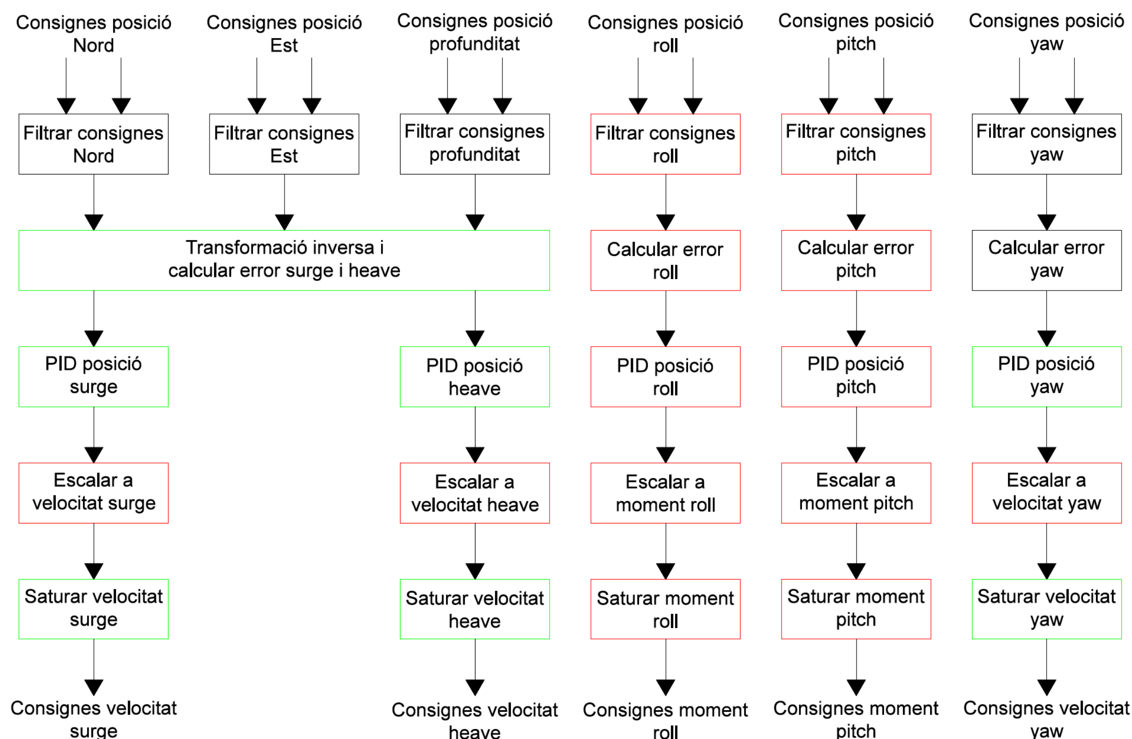

Figura 35. Diagrama del control de posició de l'SPARUS II

Tal i com es pot observar, el canvi més significatiu en el control de posició és la incorporació dels dos nous DOFs, el pitch i el roll. També cal destacar la transformació completa ZYX enlloc de la Y que es realitzava fins al moment. Tanmateix, abans la variable d'escalar i saturar els PIDs era la mateixa, el que comportava haver de sintonitzar de nou els controladors si es canviava el saturador; havent diferenciat les dues funcions, cada DOF es pot saturar lliurament tot mantenint la dinàmica prèviament ajustada del controlador. Pel que fa als PIDs ja existents, aquests únicament s'han sintonitzat de nou segons les noves característiques de tot el control de baix nivell.

7.1.1. Control de profunditat amb pitch

Tot i que la idea del control de baix nivell de l'SPARUS II és controlar els diferents DOFs de manera desacoblada, a nivell d'investigació s'ha estudiat la possibilitat de fer un control de profunditat del vehicle tot modificant el pitch. Per aproximar-se a una bona estratègia de control, ha calgut tenir molt clar els següents conceptes.

Per una banda, tal i com es pot veure a la Figura 36, quan el vehicle es troba submergit experimenta una força P originada per la massa del robot i una de flotació B, la qual és major a l'anterior. Tot i que aquest fet és contraproduent per l'estalvi d'energia al obligar al

motor vertical a generar contínuament una força T per a mantenir una profunditat, està fet a consciència per motius de seguretat; en cas que el robot es quedi sense bateries, emergirà.

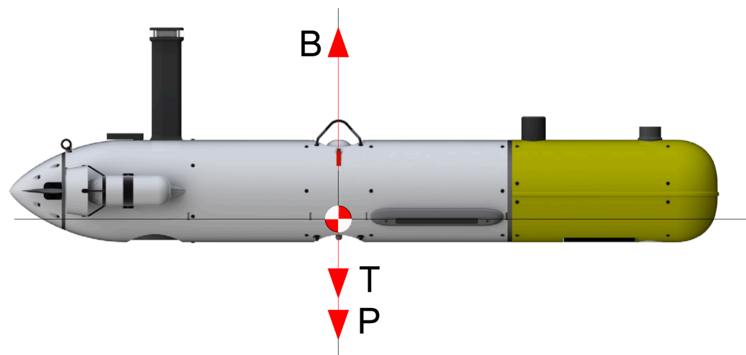

Figura 36. Ús del motor vertical per a mantenir una profunditat amb heave

Contràriament, en un control de profunditat amb pitch el motor vertical no és l'encarregat de compensar l'excés de força de flotació B. En el diagrama del cos lliure representat a la Figura 37, de color blau es pot veure la contribució resultant de les forces originades pels quatre actuadors actius, L_R i L_L pels timons de profunditat i T_R i T_L pels motors horitzontals.

Aquesta força resultant es pot descompondre en els mateixos eixos que la força de flotació i el pes, tot obtenint una component vertical i una horitzontal. Concretament, la vertical és la que contraresta l'excés de flotació, és a dir, la que manté el robot en Z tot i no tenir el motor vertical activat.

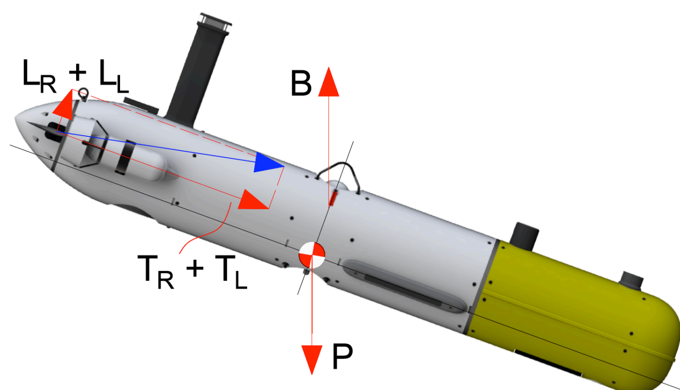

Figura 37. Ús dels timons per a mantenir una profunditat amb pitch

El segon concepte que cal tenir clar és que quan el fluid transcorre per l'entrada del motor vertical a gran velocitat, aquest no és capaç d'absorbir el líquid i transmetre-li una força d'empenyiment, pel que a partir d'una certa velocitat de surge és totalment inútil tenir aquest

actuador actiu. Aquest fet és el que ha impossibilitat al robot assolir satisfactòriament la velocitat de 2 m/s sense els timons de profunditat.

Per últim, però no menys important, tot i la primera premissa per a mantenir una profunditat a través del DOF pitch, no és coherent moure's amb un angle molt gran per aconseguir-ho, ja que és totalment no hidrodinàmic. Aquest posicionament seria necessari amb velocitats de surge reduïdes, ja que tal i com s'ha vist en l'estudi teòric dels timons de profunditat, quan el fluid incident als perfils té poca velocitat, l'efecte dels timons és mínim. Per això davant aquest cas caldrà mantenir el DOF heave activat.

De totes les consideracions anteriors han sorgit les condicions imprescindibles a implementar en aquest controlador que, acompanyades d'altres conceptes provinents de l'estudi teòric dels timons de profunditat i de les idees adquirides amb l'experimentació d'aquest mode de funcionament, han permès fer un control de profunditat amb pitch.

El control en qüestió s'ha constituït de dues parts: per una banda d'un PID, la funció del qual és proporcionar una consigna de pitch a partir de la diferència existent entre la consigna de profunditat i la ubicació actual del vehicle. Per l'altra banda, amb la finalitat de tenir en compte totes les consideracions esmentades anteriorment, s'ha dimensionat una estructura de lògica difusa o FLS (Fuzzy Logic System).

Un FLS és, en grans termes, un mapeig no lineal d'unes variables d'entrada en unes variables escalars de sortida (Mendel, 1995). Aquests sistemes es componen de tres etapes, tal i com es mostra a continuació.

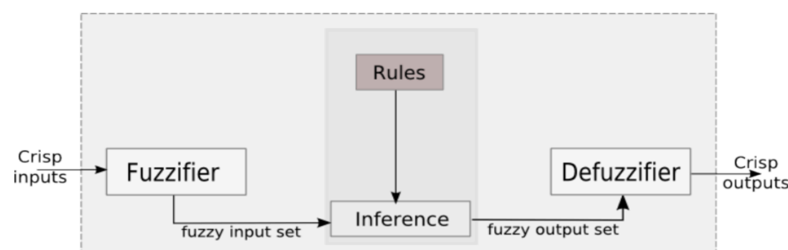

Figura 38. Constitució d'un FLS (Mendel, 1995)

La primera etapa s'anomena fuzzification i avalua totes les entrades per a determinar a quina variable lingüística o etiqueta pertanyen. Tot seguit, d'acord amb les etiquetes obtingudes i una sèrie de normes preestablertes, es determina una solució difusa a través

d'una etiqueta. Finalment, l'etapa de defuzzification, converteix la variable lingüística anterior a uns valors concrets per a cada una de les sortides del sistema. En resum, aquest control obté unes sortides escalars a partir d'unes entrades del mateix tipus tot passant per unes variables lingüístiques difuses o no exactes matemàticament.

Concretament, el FLS realitzat per aquest projecte considera la velocitat en surge i la profunditat del vehicle per determinar les accions que s'han de realitzar en els DOFs heave i pitch. En aquest cas, però, considerant que tota l'estructura de control ja estava constituïda, s'ha optat per a integrar aquest controlador de la manera més subtil possible.

Com que no es volien fer grans modificacions a la resta del control, les sortides del sistema s'han configurat com a tres coeficients d'escalat, un pel control de posició en heave, l'altre per la força que compensa l'efecte de la flotació i l'últim per escalar el PID de profunditat amb pitch. Cada un d'aquests s'ha aplicat en el lloc oportú dins del control de baix nivell.

L'etapa de fuzzification classifica en tres grups o classes la velocitat de surge, als quals els hi dóna les etiquetes de "lent", "normal" o "ràpid" segons correspongui. La teoria de la lògica difusa diu que hi ha varies opcions a la hora de constituir la seva distribució; triangular, trapezoïdal o gaussiana, entre d'altres. Davant la impossibilitat d'establir teòricament els llindars de cada grup, s'ha escrit el codi de tal manera que la seva distribució pugui ser tant triangular com trapezoïdal.

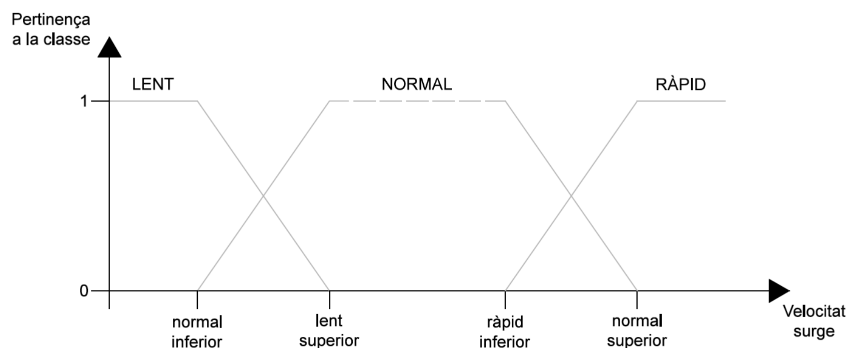

Figura 39. Funció de pertinença a les classes de l'entrada velocitat en surge

Pel que fa a l'altra variable d'entrada, la qual està representada a la Figura 40, el controlador classifica en dos grups la profunditat on es troba el vehicle, etiquetant-los com a "fora" i "dins" de l'aigua. La finalitat de considerar aquest paràmetre en el propi FLS és evitar que la part posterior del vehicle surti parcialment fora de l'aigua com a conseqüència d'un gir en

pitch. Al succeir això, el robot prendria un comportament no estable, ja que tant l'efecte dels timons de profunditat com el dels motors quedaria inutilitzat fins que el vehicle aconseguís submergir-se totalment.

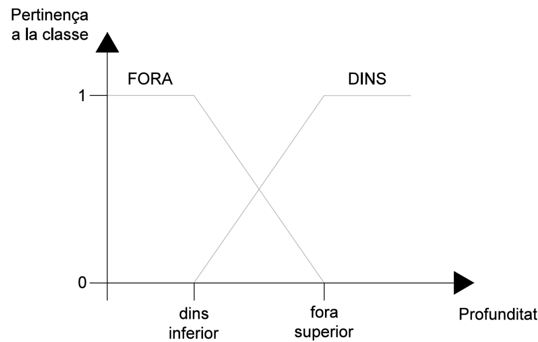

Figura 40. Funció de pertinença a les classes de l'entrada profunditat

Així doncs, la sortida d'aquesta primera etapa és el conjunt d'etiquetes que identifiquen els grups als que pertanyen les dues variables d'entrada i el seu grau de pertinença en tant per u. En aquest punt ja es treballa amb variables difuses.

En la segona etapa el FLS aplica un seguit de regles heurístiques per a determinar quant de possible és assolir o mantenir una profunditat únicament actuant amb el DOF pitch. La resposta a aquesta pregunta s'indica amb tres etiquetes, "impossible", "possible" o "segur", de les quals el FLS n'escull una i li assigna un percentatge de veracitat a través d'un procés d'inferència en AND, o el que és el mateix, un producte. La següent matriu reflecteix els resultats de les regles aplicades i el càlcul de la inferència.

	FORA == 1	DINS == D	DINS == 1
LENT == 1	IMPOSSIBLE = 1	IMPOSSIBLE = 1	IMPOSSIBLE = 1
NORMAL == N	IMPOSSIBLE = 1	POSSIBLE = D · N	POSSIBLE = N
NORMAL == 1	IMPOSSIBLE = 1	POSSIBLE = D	POSSIBLE = 1
RÀPID == R	IMPOSSIBLE = 1	POSSIBLE = D · R	SEGUR = R
RÀPID == 1	IMPOSSIBLE = 1	SEGUR = D	SEGUR = 1

Taula 10. Matriu de les regles heurístiques i càlcul de la inferència

A la fila superior de la matriu anterior es representa les possibles pertinences a la variable d'entrada de profunditat: totalment a fora, parcialment a dins o totalment a dins. La pertinença parcial a la classe "dins" en un D en tant per u significa que també pertany un (1

– D) al grup “fora”. Per altra banda, a la columna de l’esquerra estan representades les possibles pertinences a la variable d’entrada de velocitat de surge, on N i R representen la pertinença a les classes de “normal” i “ràpid” en tant per u. De nou, pertànyer N a “normal” comporta ser (1 – N) “lent”, mentre que pertànyer R a “ràpid” comporta ser (1 – R) al grup normal.

Tanmateix, com el FLS realitzat controla tres sortides, les etiquetes anteriors s’han de diversificar per a cada funció de sortida. Així doncs, cada funció de sortida s’ha caracteritzat en dos grups: “parat” i “actiu” per l’escalat del PID de posició en heave, “inhabilitat” i “habilitat” per l’escalat la força que compensa l’efecte de la flotació i “zero” i “PID” per l’escalat del control PID de profunditat amb pitch.

	Coef. escalat PID posició heave	Coef. escalat força contrària flotació	Coef. escalat PID profunditat amb pitch
IMPOSSIBLE == I	PARAT = 0 ACTIU = 1	INHABIL. = 0 HABIL. = 1	ZERO = 1 PID = 0
POSSIBLE == P	PARAT = P ACTIU = (1 – P)	INHABIL. = 0 HABIL. = 1	ZERO = (1 – P) PID = P
SEGUR == S	PARAT = 1 ACTIU = 0	INHABIL. = S HABIL. = (1 – S)	ZERO = 0 PID = 1

Taula 11. Desacoblament de les etiquetes

Finalment, d’acord amb les etiquetes desacoblades amb la seva corresponent veracitat obtinguda de l’aplicació de les regles heurístiques i del procés d’inferència, s’aplica l’etapa de defuzzification. Aquesta estableix exactament el valor de les tres sortides del sistema, és a dir, es passa d’una variable difusa a un valor escalar. Per això, la teoria de la lògia difusa presenta diferents mètodes, tal com valors ponderats, càlcul del centre de masses o computació de màxims o mínim sectorials, entre d’altres.

Concretament s’ha escollit el mètode de valors ponderats. En un primer pas, s’obtenen els valors de l’eix de les abscisses que corresponen a la pertinença donada a cada una de les sis classes. Després, s’aplica la mitjana d’aquests valors, el que proporciona el resultat final de la de sortida en qüestió. En el FLS de profunditat amb pitch, però, al haver definit uns grups amb pendents d’igual magnitud no cal fer aquest últim pas, ja que el valor corresponent en l’eix de les abscisses de la classe amb pendent positiu és directament la mitjana, tal i com s’ha representat a la Figura 41.

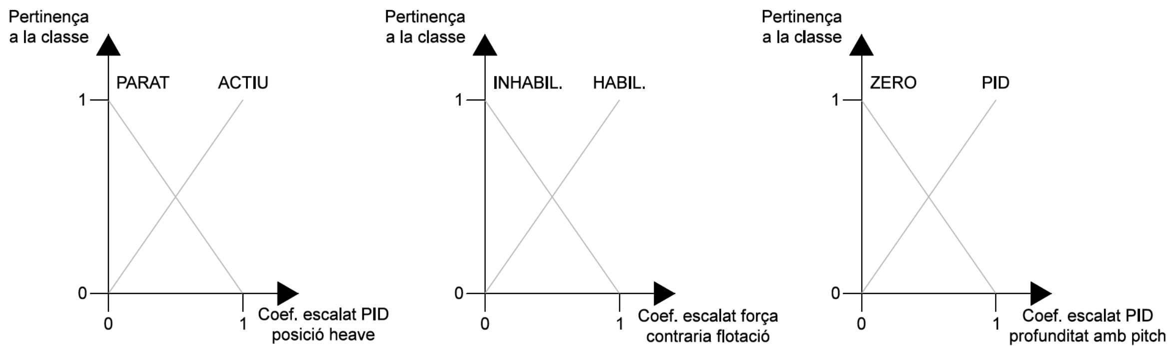

Figura 41. Funcions de les variables de sortida. A l'esquerre, coeficient del PID de posició en heave. Al mig, coeficient de la força compensadora de l'efecte de flotació. A la dreta, coeficient del PID de profunditat amb pitch

Així doncs, s'obtenen tres coeficients d'escalat compresos entre zero i u, dels quals dos d'aquests serveixen per a modificar el DOF heave, mentre que el tercer el pitch. A partir de la seva aplicació s'aconsegueix complir amb totes les premisses contemplades abans de la realització d'aquest FLS.

7.2. Control de velocitat

El control de velocitat de l'SPARUS II és l'encarregat de calcular la força necessària en cada DOF per tal de complir amb les sol·licituds de velocitat que li arriben, les quals poden ser demanades tant per l'anterior control de posició com per altres nodes de la pròpia arquitectura. Aquesta estipula que el missatge per enviar les consignes de velocitat és semblant a l'explicat anteriorment per les consignes de posició, és a dir, un vector de sis termes, en el qual s'hi indiquen les consignes, i un de booleà que determina quins DOFs s'han de controlar.

Concretament, els primers tres termes del vector de consignes corresponen a les velocitats lineals desitjades pels DOFs de surge, sway i heave, mentre que els últims tres corresponen a les velocitats angulars desitjades pel roll, pitch i yaw. Tot i que el control de velocitat únicament s'aplica als graus de llibertat de surge, heave i yaw, els vectors reflecteixen els sis DOFs perquè l'arquitectura utilitza missatges estàndards, els quals estan constituïts prèviament d'aquesta manera.

Així doncs, igual que en el control de posició, el primer que es realitza considerant que més d'un node de la pròpia arquitectura pot publicar una consigna pel control de velocitat, és filtrar totes les dades rebudes per a cada DOF. La priorització de les consignes segueix el mateix ordre que en el control de posició.

En un AUV és molt important que els seus moviments siguin suaus, pel que a les consignes resultants de surge i de yaw se les hi aplica una rampa no simètrica. Aquesta rampa es caracteritza per limitar la velocitat de creixement o decreixement de les consignes partint de zero, és a dir, per passar d'una consigna en surge de 0 m/s a 1 m/s, el control aplicarà una rampa, però per passar de la última consigna a 0,5 m/s no n'aplicarà cap. Tanmateix, per canviar dels últims 0,5 m/s a -0,5 m/s, el control únicament aplicarà la rampa des de 0 m/s a -0,5 m/s. D'aquesta manera s'eviten moviments bruscs del robot sense produir grans retards en el control. Pel heave aquesta rampa no s'aplica, ja que és un DOF que necessita molta rapidesa en el seu control i que difícilment assolirà acceleracions elevades.

A partir d'aquest punt, el control de cada grau de llibertat varia lleugerament, tot i que la seva estructura general rep el nom de model following o feedforward analyzation. Aquest sistema es composa de dues parts, un llaç obert que prediu la força necessària que cal aplicar per assolir la velocitat sol·licitada, i una altra en llaç tancat que ajusta la resposta del sistema a la consigna. El principal avantatge d'aquesta tipologia és la seva validesa en un ampli rang de consignes, ja que el PID únicament ha de cobrir les petites inexactituds del model.

Cal dir que el llaç obert s'ha constituït per un model extret d'estudis realitzats anteriorment al CIRS, els quals modelaven el fregament sofert en cada un d'aquests DOFs segons la seva velocitat a través d'un polinomi de segon grau (Vidal, 2014). Tot i haver-hi models més complexes, aquesta aproximació és totalment vàlida pel control desenvolupat.

$$F = A + B \cdot v + C \cdot v^2 \qquad \text{(Eq. 18)}$$

on,

F = força de fregament soferta pel DOF, en newtons o newtons per metre

A = terme independent del polinomi quadràtic, adimensional

B = terme lineal del polinomi quadràtic, adimensional

C = terme quadràtic del polinomi quadràtic, adimensional

V = velocitat actual del DOF, en metres per segon

Els coeficients característics del model de cada DOF s'indiquen a continuació, dels quals ja es pot anticipar que el terme independent és zero; a velocitat nul·la no hi ha fregament.

	A [Ø]	B [Ø]	C [Ø]
Model fregament surge	0,00	9,92	10,17
Model fregament heave	0,00	0,00	259,84
Model fregament yaw	0,00	0,00	53,28

Taula 12. Paràmetres dels models de velocitat pel control en llaç obert

A part del model existent en cada un dels llaços oberts, cal esmentar algunes peculiaritats del control de cada DOF. Pel surge i el heave s'ha previst la contemplació dels corrents de l'aigua dins la branca de llaç obert, ja que l'SPARUS II incorpora un sensor anomenat DVL (Doppler Velocity Log) que permet mesurar la velocitat de l'aigua respecte el terra. Tot i això, a nivell de software no està implementat, pel que no s'ha pogut comprovar el sistema de control dissenyat amb aquesta funcionalitat.

Una de les altres peculiaritats és en el control del heave. Tal i com s'ha explicat anteriorment, com a conseqüència de les mesures de seguretat preses perquè el vehicle emergeixi en cas de quedar-se sense bateries, és necessari aplicar una força per a mantenir el robot submergit. Aquesta es contempla a través d'una constant, la qual, al modelar una pertorbació permanent es pot parlar d'un sistema feedforward.

Respecte el llaç tancat, igual que en el control de posició, tots els PIDs s'han saturat entre -1 i 1, ja que d'aquesta manera els seus resultats són fàcilment escalables a les forces i moments màxims produïbles pels motors en cada un d'aquests tres DOFs del robot. A més, un cop sumada la contribució del llaç obert amb el tancat, es satura la sortida per motius de seguretat. Les tres sortides d'aquesta última funció esdevindran les consignes finals de força i moments que es sol·licitaran als motors, ja que els moments de roll i de pitch que es sol·licitaran als timons de profunditat ja s'han calculat en el control de posició.

En resum, el control de velocitat de l'SPARUS II es pot esquematitzar amb el diagrama que s'exposa a la Figura 42, en el qual s'indica de color verd tot el que s'ha modificat del control anterior i de color vermell tot el que s'ha incorporat amb la realització del present projecte.

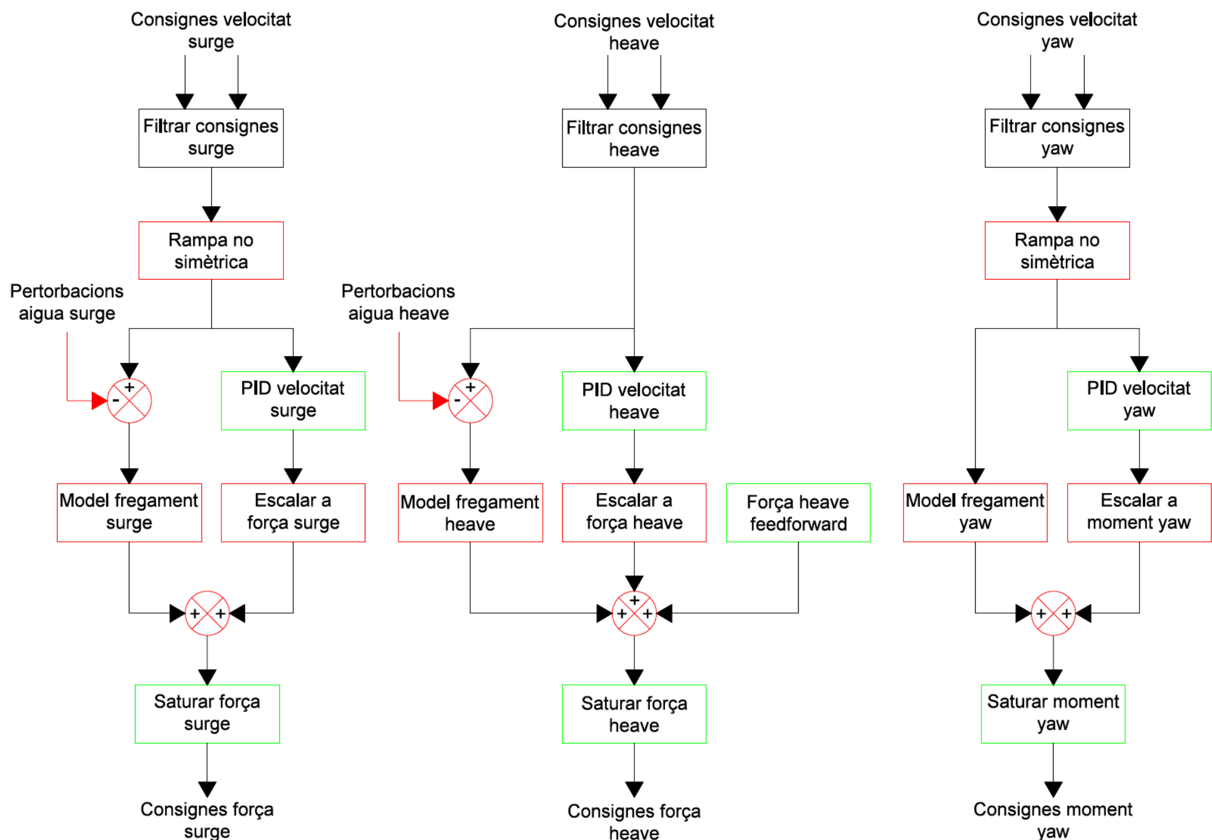

Figura 42. Diagrama del control de velocitat de l'SPARUS II

Tal i com es pot observar, el control ha sofert un canvi important envers la proposta anterior, refent-lo pràcticament de nou, ja que les especificacions que havia de complir eren totalment diferents. Respecte les modificacions realitzades, aquestes resideixen en l'ajustament dels diferents PIDs i del model feedforward de la força que compensa l'efecte de la força de flotació. A més, s'ha replicat l'estratègia utilitzada en el control de posició de separar la variable d'escalar el PID i la de saturar el resultat del control de cada DOF, ja que principalment pel control de velocitat és important poder saturar el resultat sense modificar la dinàmica de tot el sistema.

7.3. Control de força

La última etapa de l'estructura de control de baix nivell de l'SPARUS II consisteix en un control de força, és a dir, a partir d'unes consignes de força o moment aquest calcula les consignes que el driver ha d'adaptar als protocols de comunicacions i enviar als actuadors.

Per això, les comandes de força arriben en un format semblant a l'esmentat en les etapes de control anteriors; un vector de sis termes, on s'indica la consigna de força o moment per a

cada DOF, i un vector booleà de la mateixa longitud que indica quines consignes cal considerar. De nou, tenint en compte que més d'un node de la mateixa arquitectura pot publicar al mateix moment, s'aplica un filtre a totes les consignes d'entrada de cada DOF utilitzant la mateixa priorització que en els casos anteriors.

Com que l'SPARUS II no té cap sistema per a mesurar la força real que estan realitzant els seus actuadors, el control de força es basa en un model o llaç obert per a cada actuador. Així doncs, a partir d'aquest punt cal subdividir el control de força en dos parts: el dels tres motors que executaran les forces en surge, heave i yaw, i el dels timons de profunditat que duran a terme els moments de roll i pitch.

7.3.1. Control de força dels motors

L'entrada del control de força dels motors són les consignes de forces i parells requerides en els DOFs surge, heave i yaw. La ubicació dels motors respecte el robot marca que qualsevol força en surge i parell en yaw han de ser fets pels motors horitzontals, mentre que l'efecte en heave ha de ser dut a terme pel motor vertical.

Concretament en el control de força en heave, no s'ha fet una gran modificació respecte el control anterior. Al no canviar la hèlix del motor vertical i al considerar que les velocitats màximes assolibles en heave no són majors als ±0,3 m/s, s'ha cregut oportú mantenir el model estàtic del motor que governa el control de força en aquest DOF.

Abans d'aplicar el model, però, s'ha programat un coeficient corrector de la consigna, la qual depèn linealment de la tensió de les bateries, d'acord amb les observacions realitzades després de varis assaigs. Per això, es va determinar experimentalment una funció lineal que resulta 1,0 per tensions majors a 33 V, 1,34 per tensions inferiors a 28,5 V i un valor escalat per tensions compreses dins l'esmentat rang.

El resultat d'aquesta correcció es satura entre ±30 N, el qual és la força màxima que suporten les hèlixs sense perill de ruptura. Tot seguit, es computa inversament el model estàtic del motor vertical, obtenint consignes de motor d'acord amb la força demanada. Tot i això, en aquest document es presenta el model directe d'aquest actuador, el qual està constituït per dues equacions de segon grau com la de l'Equació 19. Una, és per a les consignes negatives o compreses entre -1 i 0, i l'altra per a les positives o entre 0 i 1.

$$T = A + B \cdot s + C \cdot s^2 \quad \text{(Eq. 19)}$$

on,

T = força produïda pel motor vertical, en newtons

A = terme independent del polinomi quadràtic, adimensional

B = terme lineal del polinomi quadràtic, adimensional

C = terme quadràtic del polinomi quadràtic, adimensional

s = consigna del motor, adimensional

A la següent taula s'indiquen els coeficients característics del model del motor vertical, dels quals és d'esperar que el terme independent sigui nul.

	A [Ø]	B [Ø]	C [Ø]
Model consignes > 0	0,00	37,66	45,69
Model consignes < 0	0,00	0,85	-44,78

Taula 13. Paràmetres del model del motor vertical

Un cop calculat el model invers d'aquest motor, el resultat és saturat entre -1 i 1 i finalment enviat al driver dels actuadors.

Respecte els DOFs surge i yaw, la seva aplicació en els motors horitzontals no és tant directe com en el cas anterior. Considerant que amb la realització del present projecte l'SPARUS II assolirà satisfactòriament una velocitat màxima en surge de 2 m/s, és de gran importància modelar acuradament aquests actuadors.

El primer que fa el control de força dels motors, però, és barrejar adequadament les consignes dels DOFs surge i yaw. Com que la força que pot realitzar cada motor horitzontal és limitada, s'ha prioritzat la consigna de yaw davant la de surge en cas que la seva barreja

resultés major a l'esmentat límit. Per tant, la força a realitzar per a cada motor és la meitat del moment demanat en yaw, tot conservant el sentit corresponent, més la meitat, si es pot, de la consigna en surge. La contribució en yaw de cada motor es satura perquè sigui simètrica, evitant moviments en surge no desitjats. Tanmateix, la consigna en surge es satura a dues vegades la força màxima realitzable per un motor horitzontal.

Un cop obtingudes les forces a realitzar per a cada motor, les quals no seran iguals a menys que la consigna de yaw sigui nul·la, es calcula el model invers per a cada motor tal i com s'ha explicat en el capítol de modelització, el resultat del qual es satura entre -1 i 1.

En resum, el control de força dels tres motors de l'SPARUS II es pot esquematitzar amb el diagrama que segueix, en el qual s'indica de color verd tot el que s'ha modificat del control anterior i de color vermell tot el que s'ha incorporat amb la realització del present projecte.

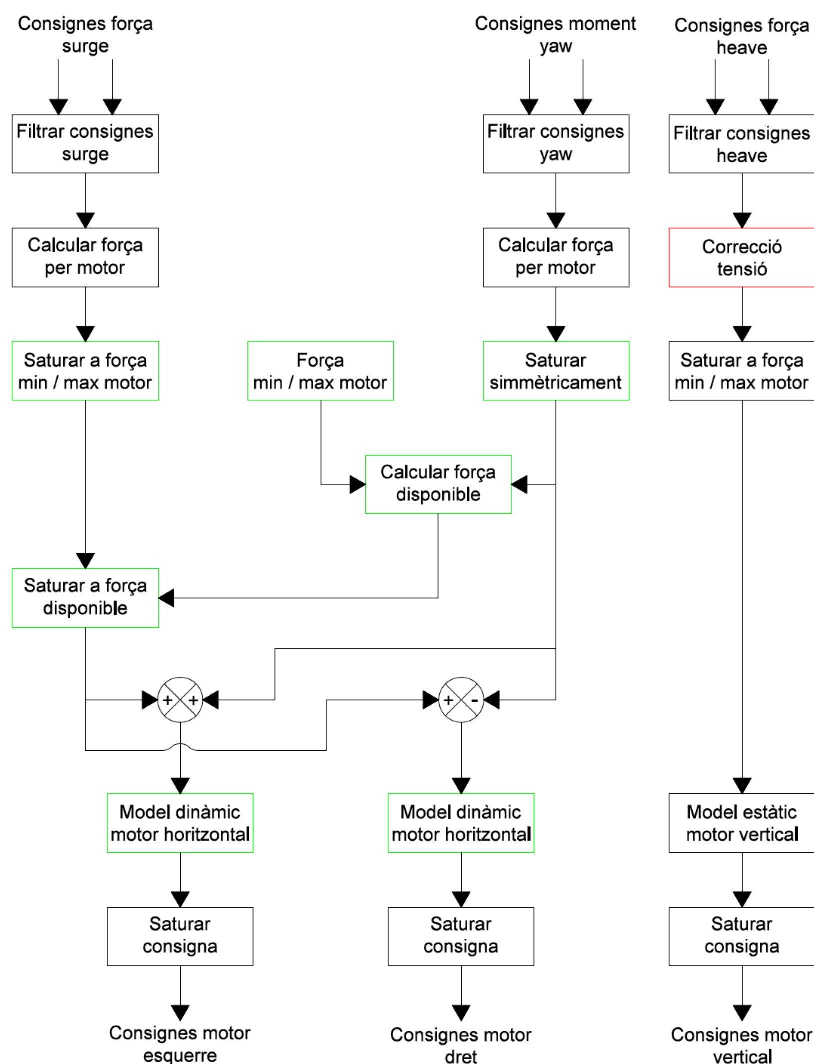

Figura 43. Diagrama del control de força dels motors de l'SPARUS II

Tal i com es pot observar a la Figura 43, el control de força dels motors no ha patit un canvi significatiu en la seva estructura, però si en el contingut. El canvi més important realitzat amb el fi d'assolir l'objectiu principal d'aquest projecte és canviar el model estàtic dels motors horitzontals per un de dinàmic, tal i com s'ha explicat anteriorment. A més, tot el control de força dels motors s'ha fet linealment dependent a l'estat de les bateries dels robot, tot evitant qualsevol diferència de comportament del vehicle al llarg del seu ús.

7.3.2. Control de força del timons de profunditat

L'entrada del control de força dels timons de profunditat són les consignes del parell requerit en els DOFs roll i pitch, ja que la seva disposició respecte la resta del robot estableix que són les dues úniques accions que poden realitzar. Per aquests actuadors s'ha cregut oportú seguir la mateixa estructura del control de força explicat anteriorment.

Per això, el primer que realitza el control és calcular la força que ha de fer cada timó en roll i en pitch per tal de complir amb les consignes de parell rebudes. Tot seguit, considerant que la força que pot realitzar cada timó és limitada, s'ha prioritzat la consigna de roll davant la de pitch en cas que la seva barreja resultés major a l'esmentat límit. Per tant, la força a realitzar per cada pala és la meitat del moment demanat en roll, tot conservant el sentit corresponent, més la meitat, si es pot, de la consigna en pitch.

Un cop obtingudes les forces de sustentació a realitzar per a cada timó de profunditat, les quals no seran iguals a menys que la consigna de roll sigui nul·la, es calcula el model invers per cada pala tal i com s'ha explicat en el capítol de modelització. En aquest cas, al parlar del càlcul del model es fa referència no únicament al model invers dels timons de profunditat, sinó també a totes les condicions explicades anteriorment.

Finalment, la consigna per a cada timó de profunditat obtinguda del seu model es satura a ±60 º si el robot es desplaça en heave i les consignes del motor són nul·les o entre ±40 º per a tots els altres casos. Aquesta consigna és la que s'envia al driver, el qual les adapta al protocol de comunicació per a ser finalment enviat a través del port sèrie.

En resum, el control de força dels timons de profunditat de l'SPARUS II es pot esquematitzar amb el diagrama que s'exposa a la Figura 44, en el qual, seguint el codi de colors establert en la resta de diagrames de control presentats en aquest document, s'indica tot de color

vermell, ja que al integrar per primer cop els timons de profunditat en el robot aquest control s'ha hagut de dissenyar i programar des de zero.

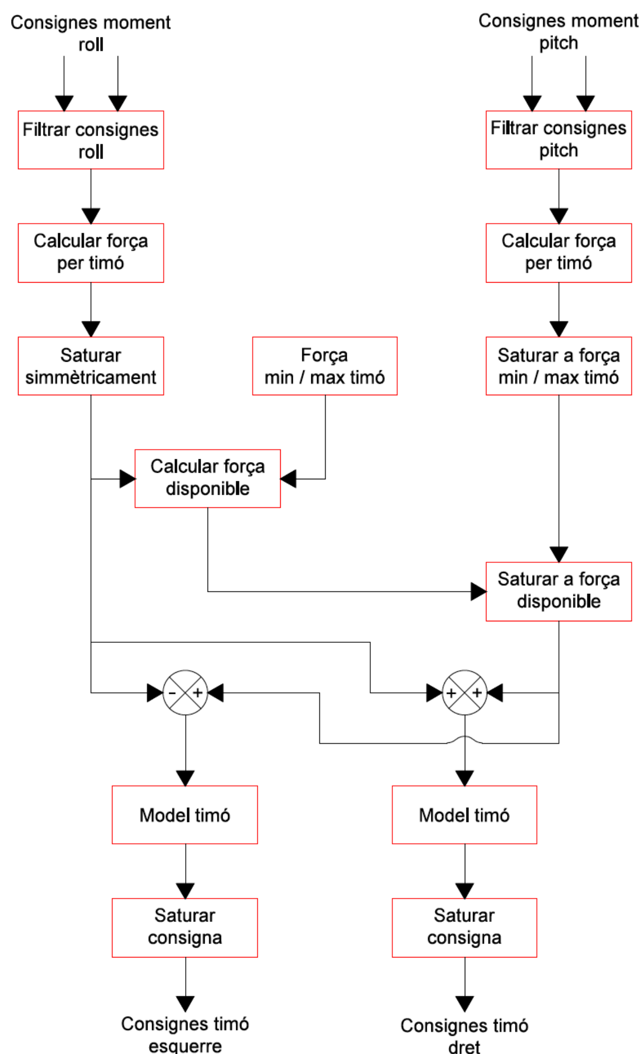

Figura 44. Diagrama del control de força dels timons de l'SPARUS II

Tot i que aquest control de força pels timons de l'SPARUS II s'ha realitzat i implementat per primer cop amb la realització d'aquest projecte, s'ha constituït d'una manera que s'ha integrat totalment amb l'arquitectura COLA[2] del robot.

7.4. Sintonització del control

Una part fonamental a la hora de fer el disseny de qualsevol estructura de control és enginyar una metodologia per a la sintonització de tot el sistema. Per això, davant la multitud de models i controladors que s'han dimensionat, concretament un total de deu models

aplicats, nou PIDs amb els seus corresponents quaranta-cinc paràmetres i una infinitat de coeficients imprescindibles per al bon funcionament del control, pot arribar a ser molt complicat ajustar tot el sistema sense seguir els passos que s'exposen a continuació.

Abans, però, cal remarcar que tot i la possibilitat d'implementar els models teòrics i de poder sintonitzar el control en llaç tancat teòricament, per exemple a través d'un mapa de pols, no s'ha cregut encertat. Tot i que els models existents del robot són bastant acurats, l'SPARUS II ha sigut concebut per a realitzar una gran varietat de missions, pel que el seu equipament a la zona del payload canvia constantment, i conseqüentment, la seva massa, flotació i hidrodinàmica. Tots aquests factors s'haurien de determinar per a cada equipament, el que suposaria una feina extra en la sintonització del control.

A més, els models existents del robot tenen un nivell de complexitat bastant elevat per a ser tractats en el pla s, ja que depenen de nombroses variables d'entrada. Per contra, si es fes una simplificació d'aquests models per a poder sintonitzar teòricament el control de l'SPARUS II, els resultats que s'obtindrien s'haurien d'acabar ajustant manualment, ja que segurament no oferirien el comportament desitjat pel robot.

Per a tot això, tal i com s'ha anat observant, pel control en llaç obert o, dit d'una altra manera, els models, s'ha tingut una clara tendència a basar-se amb equacions teòriques condicionades per paràmetres experimentals, així assegurant que el model implementat s'ajusta totalment al vehicle real. Respecte el control en llaç tancat, aquest s'ha sintonitzat manualment tot considerant i tenint molt clara l'aportació de cada paràmetre en el tipus de PID utilitzat, el qual té la part derivativa a la realimentació.

Així doncs, independentment a l'ordre que s'hagi utilitzat per explicar tot el realitzat en aquest projecte, al fer el procés de sintonització tant del llaç obert com el del tancat manualment, és molt important començar pel més baix nivell. En un primer pas, caldria modelar el motor vertical, el model del circuit elèctric dels motors horitzontals i, tot seguit, el model dinàmic d'aquests últims actuadors. Amb aquests estudis s'obtenen les forces màximes realitzables pels actuadors, a més dels coeficients d'escalat dels PIDs de velocitat pels DOFs de surge, heave i yaw.

Com que el model dels timons de profunditat és dependent al dels motors, un cop realitzat el pas anterior s'és capaç de modelitzar aquests actuadors. De nou, amb els resultats anteriors

es pot saber la força màxima que pot generar un perfil sustentador i els parell màxims que aquests podran generar en el robot tant en el DOF roll com pitch, pel que aquests valors es podran utilitzar per a escalar els PIDs de posició dels corresponents graus de llibertat.

En una tercera etapa caldria modelitzar en surge, heave i yaw el fregament al que s'oposa el vehicle al moure's en diferents velocitats a través d'un medi aquós. Tanmateix, s'ha de modelitzar el control feedforward que contraresta l'efecte de la força de flotació en heave. Un cop implementats aquests models, cal ajustar la rampa no simètrica per tal que el robot no tingui un comportament brusc però pensant amb la petita inducció de retard que s'origina en el control. Arribats en aquest punt, únicament falta sintonitzar el llaç tancat d'aquests DOFs per tal que cobreixi els errors de velocitat restants.

Pel mode de control de profunditat amb pitch hi ha una gran varietat de paràmetres a ajustar, incloent un controlador PID. Per entendre el seu procés de sintonització és recomanable repassar el seu mode de funcionament, ja que hi ha moltes possibles combinacions.

Finalment queda ajustar el control de posició. Per això cal determinar la velocitat màxima dels DOFs surge, heave i yaw, les quals són les velocitats que assolirà el vehicle quan les consignes dels motors estiguin saturades a -1 o 1. Amb els coeficients d'escalat ajustats, únicament queda sintonitzar el control de posició dels cinc DOFs del robot.

Realitzant la sintonització de l'estructura de control en aquest ordre s'evita ajustar el control iterativament i per tant, una gran inversió de temps. Començant pel més baix nivell i assegurant que aquest compleix correctament amb la seva funcionalitat, quan es sintonitza el següent nivell es té la garantia que la base de control és ferma. Aquest fet assegura que únicament modificant el nivell en qüestió és possible assolir pràcticament tots els objectius de control que es proposin.

8. PROVES REALITZADES I RESULTATS

Per a comprovar el correcte funcionament de cada una de les parts funcionals del present projecte s'han realitzat nombroses proves. Aquests assaigs parcials han servit per a eliminar petits errors de fabricació, debugar els diferents codis escrits, sintonitzar el control i, conseqüentment, han permès avançar al llarg de la integració dels timons de profunditat amb una base totalment ferma.

Tanmateix, per a verificar la funcionalitat dels timons de profunditat i demostrar el compliment del principal objectiu d'aquest projecte, s'han fet experiments que involucren al mateix temps tot el que s'ha implementat i corroboren la seva integració en el robot.

8.1. Proves parcials

Durant la execució de tot el descrit en aquest document s'han fet algunes proves per a garantir el bon funcionament de cada part, de les quals s'esmenta breument la metodologia de testeig i alguns dels resultants obtinguts.

8.1.1. Proves de hardware

A nivell de hardware s'ha testejat la placa al complert. Aquesta s'ha alimentat amb una font de tensió variable per a comprovar que el voltatge de sortida del regulador instal·lat proporciona una alimentació fixa de 6 VDC independentment de la tensió d'entrada, la qual s'ha variat dins del mateix rang que les bateries del robot, de 24 a 34 VDC. Amb la mateixa prova s'ha verificat que la sortida del segon regulador també és manté constant a aproximadament 5 VDC i que tots els integrats i connectors els hi arriba la tensió desitjada.

De la placa electrònica també s'ha pogut verificar el bon funcionament del circuit de tractament del senyal del sensor d'aigua. Per això, estant la placa alimentada, s'ha mesurat la tensió a l'entrada del microcontrolador al mateix temps que es simulava l'existència d'aigua dins del compartiment dels timons de profunditat. Tanmateix, s'ha buscat a partir de quin llindar de tensió el circuit en qüestió interpreta l'existència d'aigua, el qual ha resultat ser d'uns 1,8 VDC. Tot i ser un valor coherent d'acord amb el dimensionament dels components passius del circuit, aquest fet ha reafirmat la necessitat d'un filtre al firmware de la placa per tal d'evitar interpretar el possible soroll induït com una veritable emergència.

Per a garantir que amb el microcontrolador escollit es podia generar un PWM de qualitat i de les característiques requerides pels servomotors, s'ha escrit un codi senzill que just alimentar l'integrat generés un senyal de control. Es va realitzar una primera prova amb un senyal de freqüència igual a 300 Hz i amb una longitud de pols de 1.200 µs.

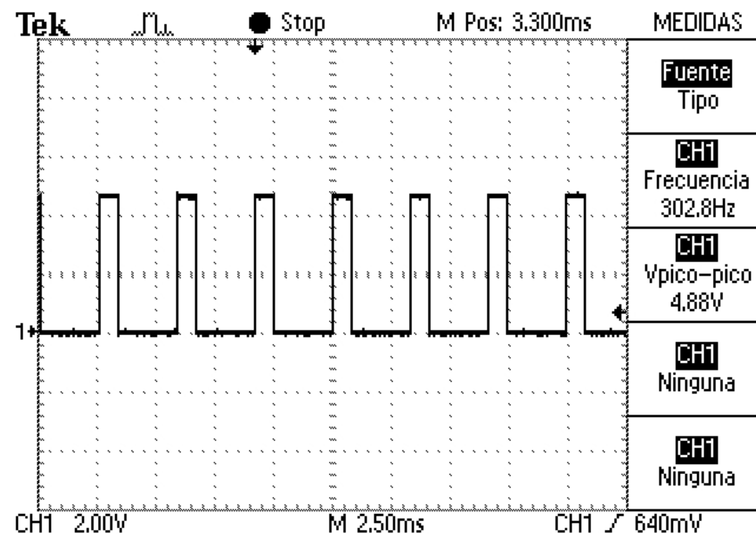

Figura 45. Senyal de control PWM a 300 Hz i amb longitud de pols de 1.200 µs

Una segona prova va consistir en un senyal de control de freqüència igual a 300 Hz i longitud de pols de 2.250 µs.

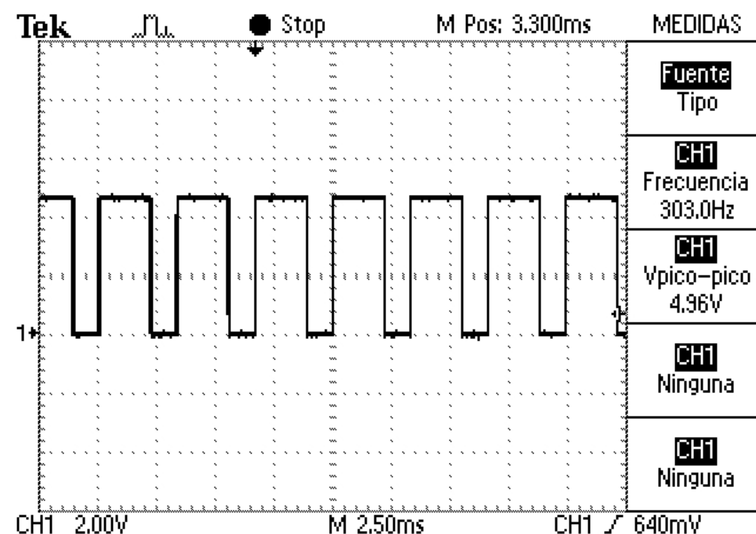

Figura 46. Senyal de control PWM a 300 Hz i amb longitud de pols de 2.250 µs

A partir de les visualitzacions del senyal amb l'oscil·loscopi es va afirmar que les característiques del hardware eren suficients per a produir un PWM estable i precís.

De la pròpia placa també s'ha visualitzat el senyal de rellotge del cristall de quars per tal de comprovar que oscil·lava a la freqüència de 11,0592 MHz. D'aquesta manera, a més de comprovar el seu bon muntatge en la placa s'ha verificat el codi del microcontrolador que permet que oscil·li.

8.1.2. Proves de protocol i firmware

Abans de provar el funcionament de tota la placa electrònica acoblada al bus RS-485 del robot, aquesta s'ha volgut testejar individualment. Per això s'ha connectat a un adaptador ADAM 4520, el qual converteix els nivells de tensió del protocol RS-485 a RS-232 i viceversa. D'aquesta manera ha sigut possible comunicar-se amb la placa a través d'un terminal sèrie d'un PC.

Un cop realitzat tot el connexionat per tal que l'ADAM 4520 funcionés, el primer que s'ha testejat és la comunicació entre el PC i la placa. Per això s'ha utilitzat la única comanda d'execució que no requereix d'una configuració prèvia, la de preguntar per l'estat d'aigua. Amb aquesta s'ha verificat nombroses vegades la resposta que proporcionava segons l'assaig que s'estigués fent en els pins del connector, inclús provant si el filtre de soroll programat funcionava degudament.

Tot seguit, es van introduir els vuit paràmetres de configuració dels dos servomotors per a poder utilitzar les comandes principals del sistema, precedides per la comanda que habilita les comandes d'usuari. Un cop rebut el missatge de confirmació per cada una de les configuracions, es va inhabilitar les comandes d'usuari per a poder testejar sense cap interferència la resta del protocol de comunicació i firmware.

Al mateix temps que es visualitzava el senyal PWM de les dues sortides a l'oscil·loscopi i aquestes estan inicialment parades, es van utilitzar les comandes d'habilitació, modificació i inhabilitació del posicionament dels servomotors. Durant tot aquest procés es va verificar que les consignes en graus enviades en hexadecimal es traduïssin correctament al senyal de PWM adequat.

També es va verificar que les configuracions es guardessin a la memòria EEPROM. Per això es va treure l'alimentació de la placa, es va esperar breument a que els condensadors existents en la mateixa es descarreguessin i es va tornar a alimentar. Al generar-se PWMs

coherents al posicionar els timons de profunditat al centre, en angles positius i negatius, es va poder garantir que tota la configuració es guarda satisfactòriament a la EEPROM.

Finalment, per a testejar que la placa podia gestionar totes les comandes a una freqüència mínima de 10 Hz, a través d'un PC amb interfície de Linux i amb ROS instal·lat es va testejar la part del driver escrit referent als timons de profunditat. Amb aquest assaig es va afirmar que no únicament la placa electrònica treballava sense cap problema enviant-li comandes a 10 Hz, sinó que també gestionava totes les comandes correctament a la freqüència màxima de 30 Hz que permet la gestió del bus de comunicació.

8.1.3. Proves d'integració

Un cop es va assegurar que la placa electrònica realitzada funcionava correctament de forma individual, abans d'integrar-la en el bus RS-485 del robot es va voler fer una altra prova. Per això es va fer una rèplica del bus del robot, en el qual s'hi van connectar els mateixos dispositius: un PC amb ROS, les tres plaques electròniques corresponents als motors i la placa a integrar amb aquest projecte. El primer que es va fer va ser executar l'arquitectura del robot amb el driver vell, per tal de recordar l'ocupació del bus abans de la gestió de les comandes dels motors.

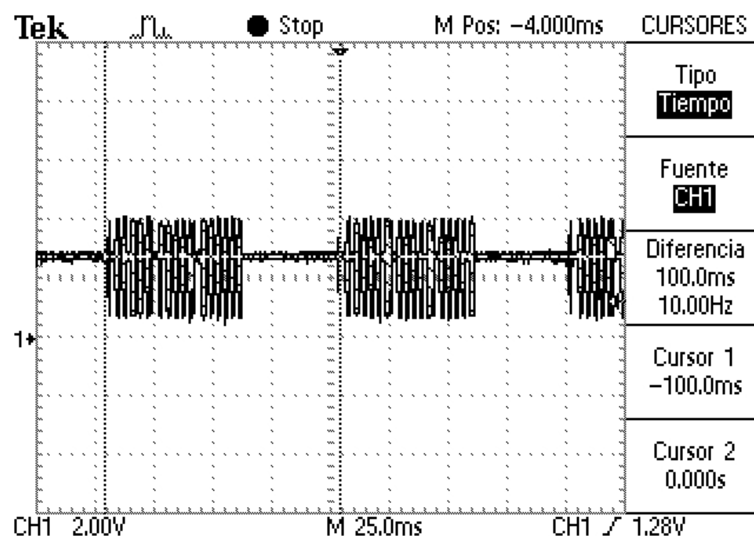

Figura 47. Ocupació del bus RS-485 sense gestió de les comandes dels motors

Tal i com s'ha esmentat en el capítol corresponent, totes les comandes que s'intercanviaven amb l'electrònica dels motors ocupava el bus durant uns 60 ms, deixant-lo lliure únicament

40 ms. Per contra, es va tornar a executar l'arquitectura però amb el driver nou, és a dir, gestionant les comandes intercanviades amb les motors.

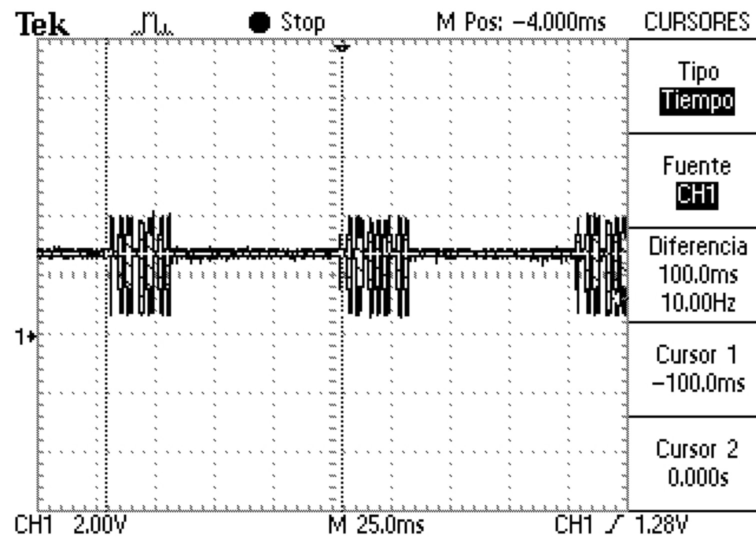

Figura 48. Ocupació del bus RS-485 un cop gestionades les comandes dels motors

Mantenint la freqüència d'enviament a 10 Hz, s'ha aconseguit tenir el bus ocupat únicament uns 30 ms. Tot i això, en aquest punt únicament s'estaven enviant les comandes dels motors, pel que finalment es va incorporar les dels timons de profunditat. Amb aquest últim pas es va assegurar la correcta programació i total funcionalitat del driver en ROS.

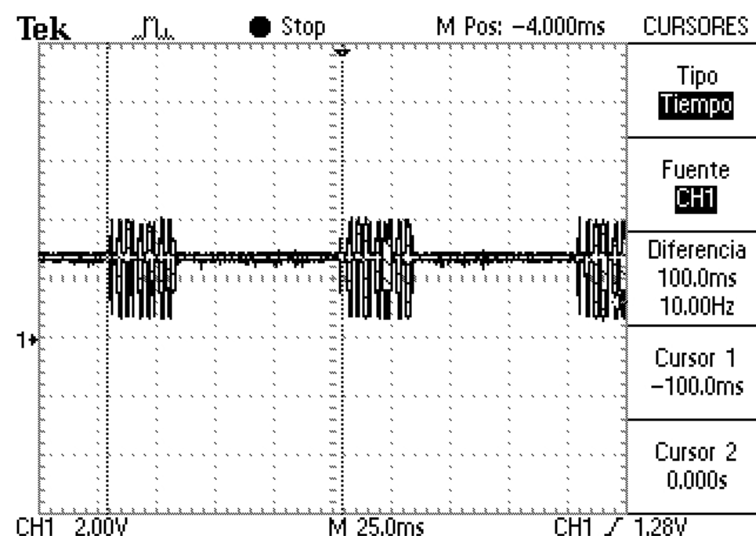

Figura 49. Ocupació del bus RS-485 un cop gestionades les comandes dels motors

D'acord amb el que s'havia esmentat anteriorment, la incorporació de les comandes que posicionen els timons de profunditat suposa una diferència mínima en la ocupació del bus,

finalment essent d'uns 32 ms. Per això, es pot afirmar que la gestió de totes les comandes enviades a través del bus RS-485 permet ampliar la freqüència d'enviament fins als 30 Hz.

Un cop assegurat el bon funcionament del sistema, es va integrar tant la part mecànica com electrònica a l'SPARUS II. Amb aquesta es van repetir experiments ja realitzats tant a nivell de hardware com de la comunicació sèrie, per assegurar la bona integració amb el robot.

L'etapa d'integració va concloure en calibrar els timons de profunditat degudament, tot determinant manualment la longitud de pols que posiciona els timons als angles de -60 º, 0 º i 60 º. Per saber el seu posicionament angular respecte el robot primerament es van aplicar raons trigonomètriques a un seguit de mesures preses amb aquest fi. Els resultats obtinguts es contrarestaven amb el giroscopi del mòbil, el qual coincidia únicament amb errors absoluts d'un grau. Per això, finalment tot el calibratge es a fer amb el giroscopi del mòbil, ja que agilitzava molt tot el procés.

	t angle mínim [µs]	t angle central [µs]	t angle màxim [µs]	Rang [º]
Servomotor esquerre	925	1.550	2.170	120
Servomotor dret	805	1.425	2.000	120

Taula 14. Configuració dels servomotors calibrats respecte el robot

Cal recordar que aquesta configuració sol ser característica per a cada actuador, pel que en cas d'haver-se de canviar un servomotor caldrà determinar de nou els paràmetres referents al temps d'angle mínim, central i màxim.

8.1.4. Proves de control

Les proves de control s'han realitzat tant amb simulació, com a la piscina del CIRS com a mar amb la embarcació SEXTANT i han servit per a depurar el codi i per a sintonitzar els diferents DOFs del vehicle.

En el CIRS es disposa d'un simulador anomenat UWSim (UnderWater Simulator), el qual està desenvolupat pel propi centre conjuntament amb la Universitat Jaume I de València. Aquest està totalment integrat dins ROS i l'arquitectura COLA[2], pel que permet simular la

dinàmica de l'SPARUS II en un entorn marítim. A més, aquest simulador permet testejar una gran varietat de codis, missions i trajectòries únicament adaptant mínimament l'entorn a les necessitats de l'aplicació.

Figura 50. Simulador UWSIM amb l'SPARUS II

Tot i que aquesta eina és molt útil per la majoria d'aplicacions, tal i com s'ha explicat anteriorment, el model teòric existent del robot divergeix notablement del real, pel que aquest simulador únicament ha servit per a depurar el codi programat pel control de baix nivell.

Abans, però, ha calgut afegir al codi que descriu la dinàmica del robot una aproximació de les equacions que descriuen els efectes derivats de les forces generades pels timons de profunditat. D'aquesta manera també s'ha pogut depurar mínimament el codi del control referent als DOFs de roll i pitch. Tot i introduir les equacions en el simulador, no s'hi han dibuixat les aletes, ja que no estava contemplat dins l'abast del present projecte.

La següent fase de proves de control va ser realitzada a la piscina, on es van modelar les noves hèlixs instal·lades al vehicle i els timons de profunditat, obtenint els models mostrats en el capítol de modelització. Alguns dels experiments esmentats en el mateix capítol, concretament la determinació del coeficient C_2 del model dinàmic dels motors, ha calgut fer-los a mar amb l'embarcació SEXTANT, ja que les dimensions de la piscina de proves del CIRS no permeten que el robot assoleixi velocitats majors a 0,5 m/s de forma segura. Un cop implementats els models es va notar una gran millora en el control, maniobralitat i estabilitat del vehicle, ja que aquests suposen una computació bastant real al que realment experimenten els actuadors del robot.

De l'ajustament dels models dels actuadors es va extreure la seva força màxima i mínima, la qual era de ±30 N pel motor vertical, de 77 N i -43 N per cada motor horitzontal i ±26,15 N per cada timó de profunditat. A partir d'aquests valors i amb l'experimentació adequada s'han pogut determinar els factors d'escalat dels PIDs dels cinc DOFs del robot.

	surge	heave	roll	pitch	yaw
Factor velocitats [m/s] o [rad/s]	2,00	0,00	--	--	0,60
Factor forces [N] o [N·m]	154,00	30,00	3,64	17,15	14,43

Taula 15. Coeficients d'escalat dels PIDs de l'SPARUS II

El factor d'escalat del PID de profunditat amb pitch es va establir a 1,57 radians. Aquest valor, al no dependre directament de les característiques del robot, es va definir segons les dues orientacions extremes en pitch que tindria lògica que assolís el robot amb el fi de complir amb les diferents consignes de profunditat. La resta de paràmetres d'aquest control, exactament els del FLS, es van determinar experimentalment d'acord amb la conseqüent influència amb les variables de sortida. Els valors trobats per a la parametrització de les classes de surge van ser de 0,7 m/s, 0,9 m/s, 1,2 m/s i 1,4 m/s, mentre que per les classes de profunditat van ser de 0,2 m i 1 m respectivament.

Amb aquests paràmetres definits es va procedir a la sintonització del control. Tot i les característiques de la piscina del CIRS, s'hi va poder sintonitzar correctament el control de heave i de yaw, ja que per fer-ho no es necessita de molt d'espai. Pel pitch i el roll també es va poder fer una primera sintonització. Finalment, amb les diferents estades a mar per a fer experiments de control es va extreure la sintonització del nou control de baix nivell de l'SPARUS II, la qual s'ha reflectit a la Taula 16 i a la Taula 17.

	K_p [Ø]	T_i [Ø]	T_d [Ø]	i_limit [Ø]	fff [Ø]
PID prof. amb pitch	0,15	12,00	0,00	0,20	0,00
PID posició surge	0,05	15,00	0,00	0,20	0,00
PID posició heave	0,90	0,00	0,50	0,00	0,00

Taula 16. Paràmetres dels PIDs del control de baix nivell de l'SPARUS II

	K_p [Ø]	T_i [Ø]	T_d [Ø]	i_limit [Ø]	fff [Ø]
PID posició roll	0,40	15,00	0,00	0,25	0,00
PID posició pitch	0,80	15,00	8,00	0,10	0,00
PID posició yaw	0,50	0,00	2,00	0,00	0,00
PID velocitat surge	0,25	20,00	0,00	0,25	0,00
PID velocitat heave	3,00	8,00	0,00	0,25	0,10
PID velocitat yaw	1,50	8,00	0,00	0,15	0,00

Taula 17. Paràmetres dels PIDs del control de baix nivell de l'SPARUS II (continuació)

Apart dels paràmetres clàssics d'un PID, els quals són K_p, T_i i T_d, a la taula anterior el terme i_limit fa referència a un anti windup, el qual serveix per evitar que el terme integral del PID es carregui indefinidament, tal i com s'ha explicat anteriorment. Per altra banda, el terme feedforward force parlat en el capítol de control s'ha abreviat com fff.

Aquesta configuració, conjuntament amb tot l'esquema de control i models explicats anteriorment, ha permès aconseguir que la resposta de control dels diferents DOFs desacoblats assolís la consigna desitjada, amb un temps d'establiment reduït i evitant que el robot adquirís un comportament molt agressiu.

8.2. Proves finals

Finalment, tant per comprovar que tota la sintonització anterior del control de baix nivell com per veure la correcte integració amb el control d'alt nivell, s'han fet diferents experiments basats amb l'execució de missions, les quals es basen amb un seguit de punts XYZ per on el robot ha de passar. Per això, aquestes missions són interpretades pel control d'alt nivell.

Davant les diferents estratègies disponibles en el control d'alt nivell per a assolir punts s'ha escollit l'anomenada LOS (Line Of Sight). Aquest algorisme es caracteritza, en grans termes, per no únicament assolir el punt XYZ desitjat, sinó per vetllar que el vehicle no es desviï de la línia recta que uneix un punt amb el següent. Com a resultant de de totes les consideracions preses pel LOS s'obté una consigna de velocitat de surge, una de profunditat i una d'orientació de yaw, les quals són tractades pel control de baix nivell.

Els controls involucrats amb les esmentades consignes resultants del LOS, conjuntament amb els de posició de pitch i de roll, són els ha sigut interessant visualitzar per tal de verificar la correcte operativitat del sistema dissenyat. Únicament comprovant aquests cinc es pot concloure si els controls i models concatenats de més baix nivell actuen correctament.

El LOS es configura a partir de varis paràmetres, dels quals ha sigut d'interès modificar el de velocitat lenta o desitjada per quan el vehicle estigui a prop d'un punt i el de velocitat ràpida, la qual serà la sol·licitada per la resta de casos.

8.2.1. Primer assaig general del control

Un primer experiment ha consistit en seguir un conjunt de punts a 3 m de profunditat, la ubicació dels quals és assimilable a una forma de H d'acord amb la Figura 51. La velocitat lenta del LOS s'ha estipulat a 0,25 m/s, mentre que la ràpida a 0,5 m/s. De la Figura 52 a la Figura 56 es mostra el comportament de l'SPARUS II a través dels seus cinc DOFs.

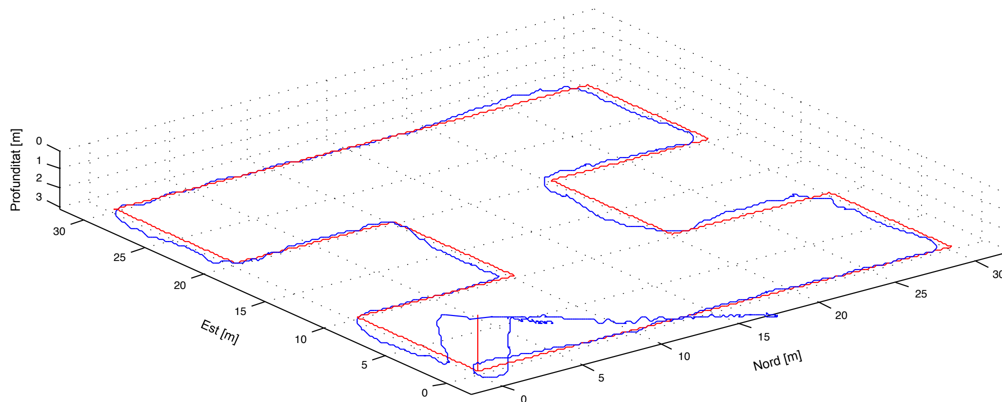

Figura 51. Trajectòria XYZ realitzada pel robot. Timons habilitats i LOS 0,25 m/s – 0,5 m/s

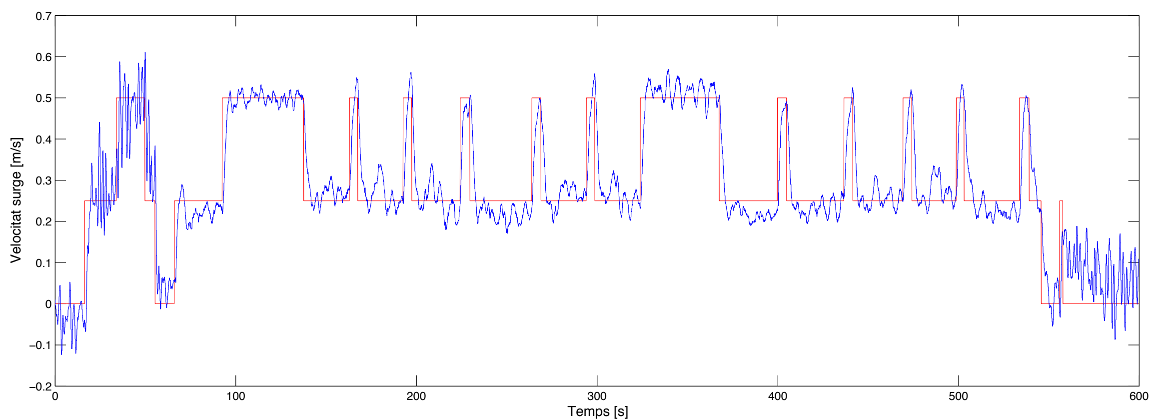

Figura 52. Dinàmica temporal de la velocitat en surge. Timons habilitats i LOS 0,25 m/s – 0,5 m/s

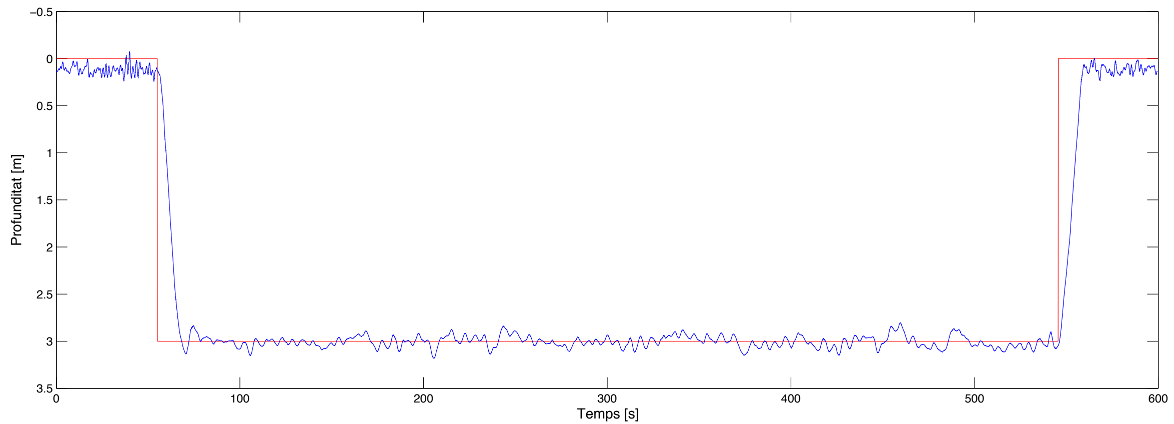

Figura 53. Dinàmica temporal de la profunditat. Timons habilitats i LOS 0,25 m/s – 0,5 m/s

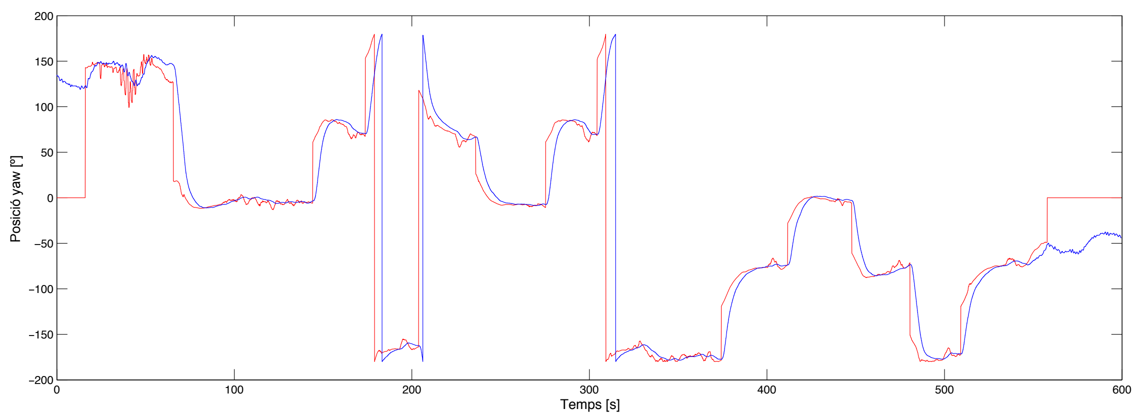

Figura 54. Dinàmica temporal de la posició en yaw. Timons habilitats i LOS 0,25 m/s – 0,5 m/s

Es pot veure que la velocitat de surge del robot presenta un petit rissat al voltant de la consigna. Aquest fet és inevitable considerant els retards existents en la navegació i sense la implementació de l'estimador de corrents. Pel que fa a la profunditat i la posició en yaw, el seu comportament és totalment satisfactori.

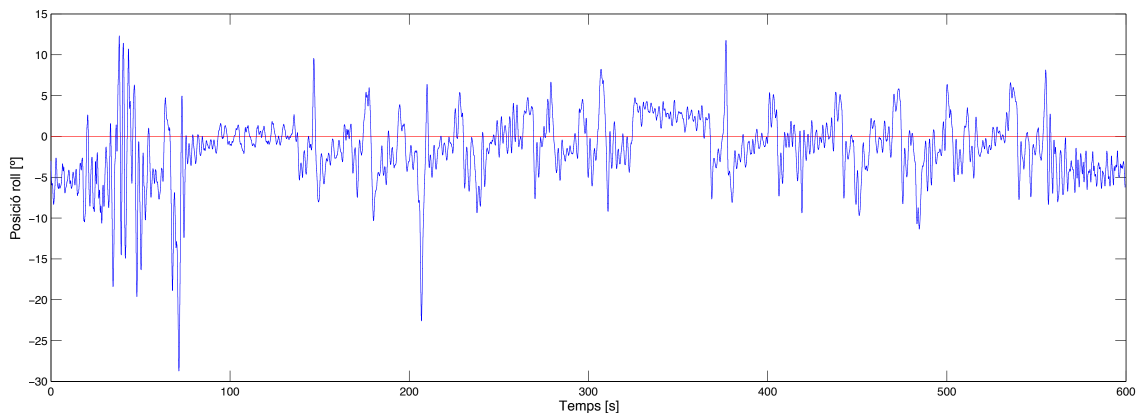

Figura 55. Dinàmica temporal de la posició en roll. Timons habilitats i LOS 0,25 m/s – 0,5 m/s

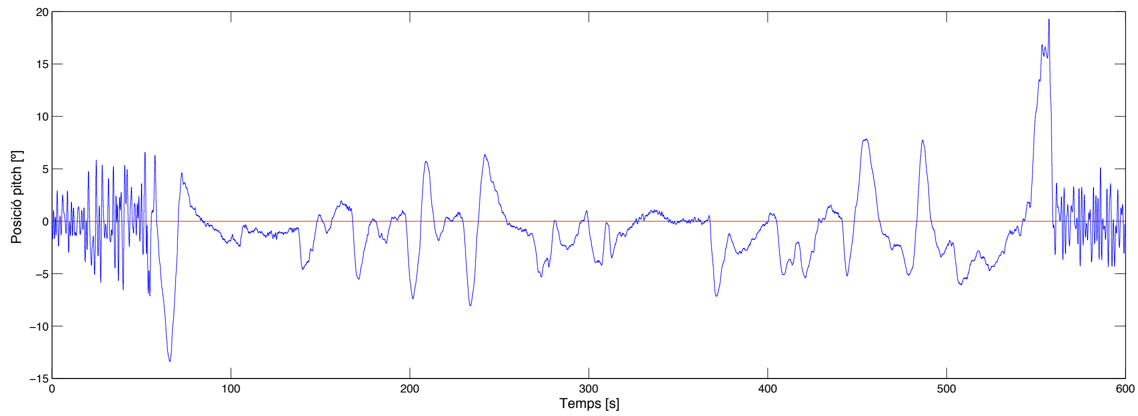

Figura 56. Dinàmica temporal de la posició en pitch. Timons habilitats i LOS 0,25 m/s – 0,5 m/s

Com que és difícil determinar si les accions dels timons de profunditat han estabilitzat els DOFs roll i pitch, es va repetir la mateixa missió però amb els timons fixats a un angle d'atac de zero graus. Com que el comportament del robot en surge i en heave no es veu alterat pels timons, aquests no s'han estudiat comparativament.

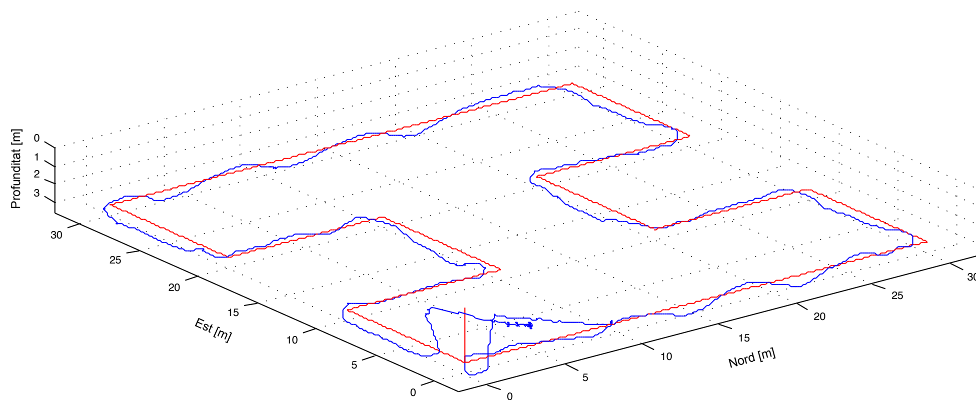

Figura 57. Trajectòria XYZ realitzada pel robot. Timons inhabilitats i LOS 0,25 m/s – 0,5 m/s

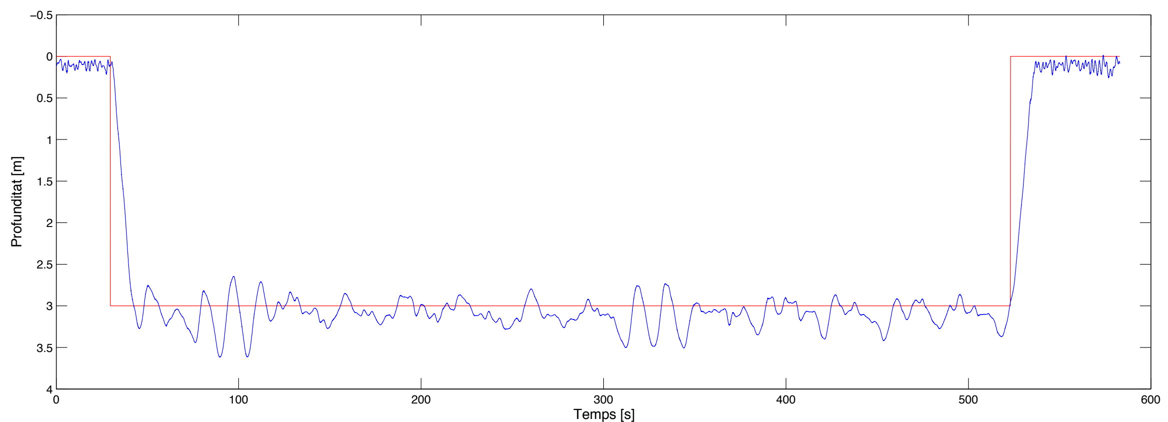

Figura 58. Dinàmica temporal de la profunditat. Timons inhabilitats i LOS 0,25 m/s – 0,5 m/s

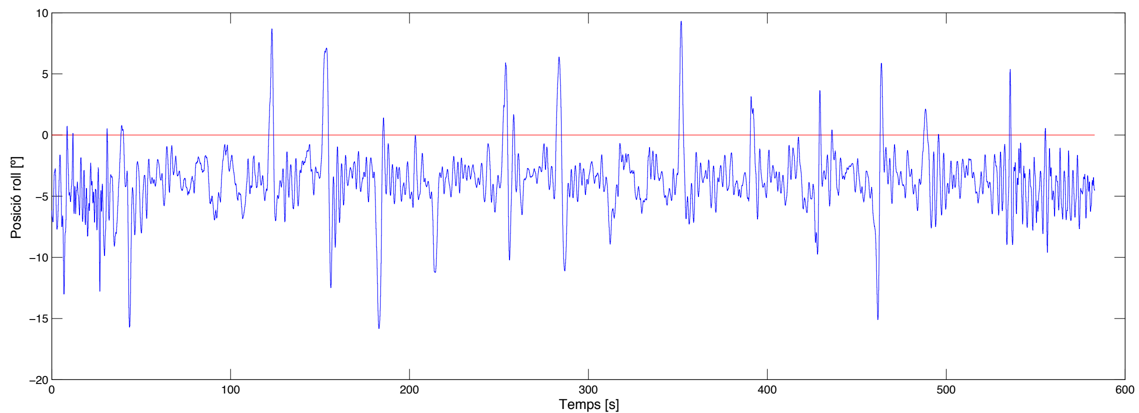

Figura 59. Dinàmica temporal de la posició en roll. Timons inhabilitats i LOS 0,25 m/s – 0,5 m/s

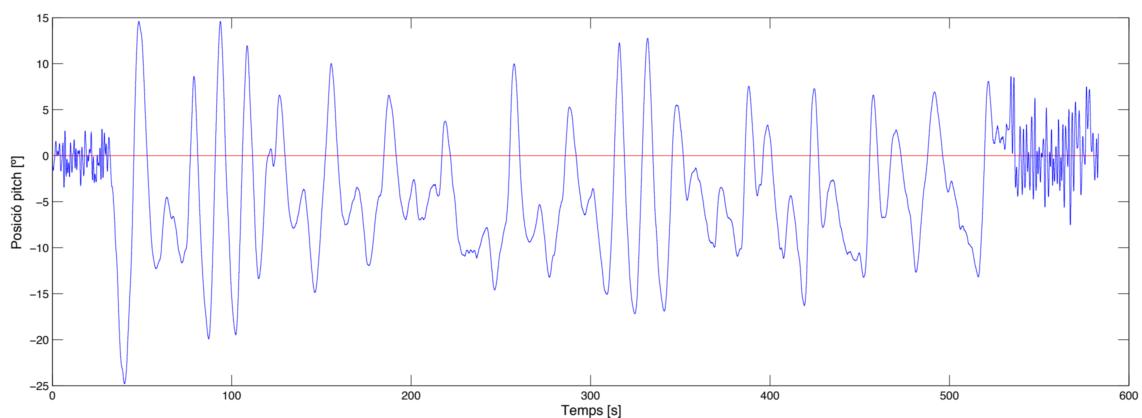

Figura 60. Dinàmica temporal de la posició en pitch. Timons inhabilitats i LOS 0,25 m/s – 0,5 m/s

Comparant les dades anteriors amb els resultats expressats des de la Figura 57 fins a la Figura 60, ambdós incloses, es pot apreciar una notable diferència, el que numèricament s'ha reflectit tot calculant la mitjana dels errors absoluts de profunditat, de roll i de pitch.

	Profunditat [mm]	Roll [º]	Pitch [º]
Timons hab.	47,82	2,50	2,11
Timons inhab.	146,49	3,91	7,01

Taula 18. Mitjana dels errors absoluts dels experiments realitzats amb LOS 0,25 m/s – 0,5 m/s

Per altra banda, la incorporació dels timons de profunditat a l'SPARUS II ha comportat que aquest fos més eficient energèticament, ja que el motor vertical no ha de compensar tantes oscil·lacions en el DOF heave. Per això, s'han analitzat tant les dades de consum provinents del propi motor com les proporcionades pel BMS (Battery Management System), el qual és

un sensor que indica la potència consumida pel robot. De l'estudi s'ha extret que amb els timons habilitats el motor vertical ha gastat una mitjana de 11,60 W, uns 4,55 W menys, el que correspon a una reducció del seu consum en un 20,53 %. Tanmateix, l'ús dels timons de profunditat envers la seva inhabilitació comporta una reducció del consum total del robot d'un 1,48 %. Per tant, amb velocitats de 0,5 m/s o inferiors, l'habilitació dels timons de profunditat pren sobretot importància per la millora del control de profunditat, roll i pitch.

8.2.2. Segon assaig general del control

El segon experiment ha consistit en seguir una sèrie de punts a 2 m de profunditat situats estratègicament per a cobrir una zona de 900 m^2 a partir de deu rectes de 30 m, tal i com apreciar lleugerament a la Figura 61. De la Figura 62 a la Figura 66 es mostra el comportament de l'SPARUS II al llarg de la missió amb una estratègia LOS configurada amb 0,3 m/s per la velocitat lenta i 1,0 m/s per la ràpida.

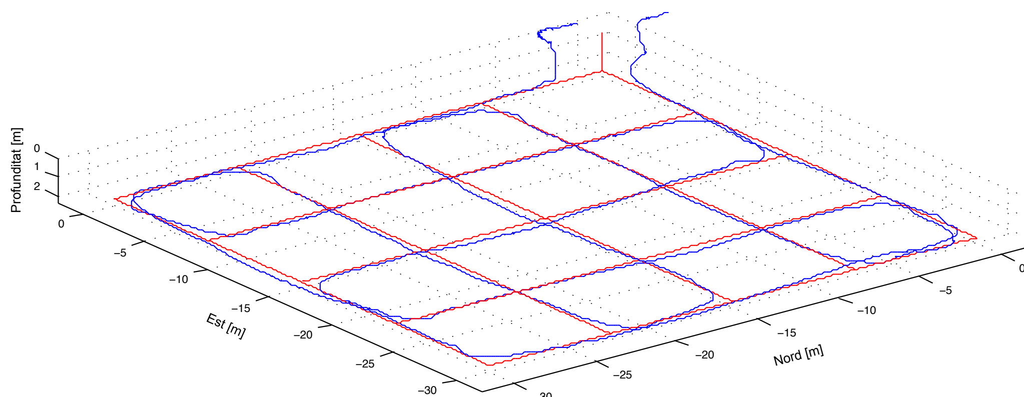

Figura 61. Trajectòria XYZ realitzada pel robot. Timons habilitats i LOS 0,3 m/s – 1,0 m/s

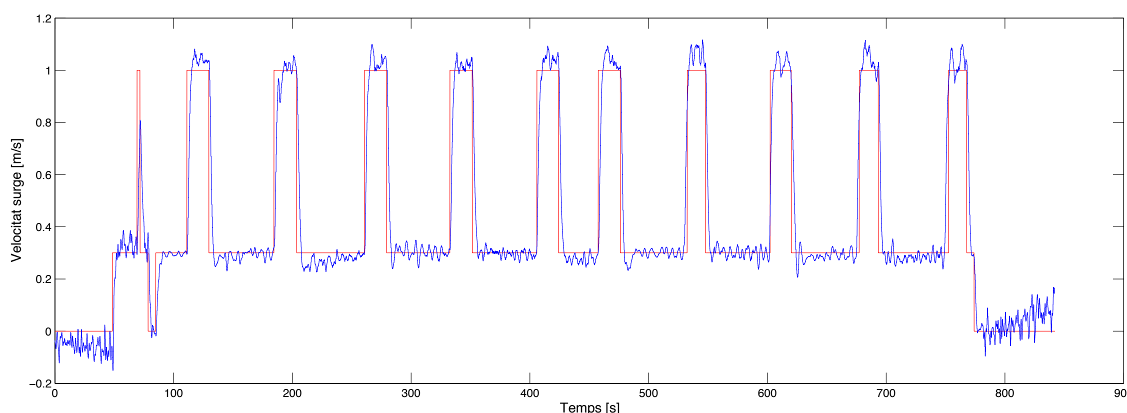

Figura 62. Dinàmica temporal de la velocitat en surge. Timons habilitats i LOS 0,3 m/s – 1,0 m/s

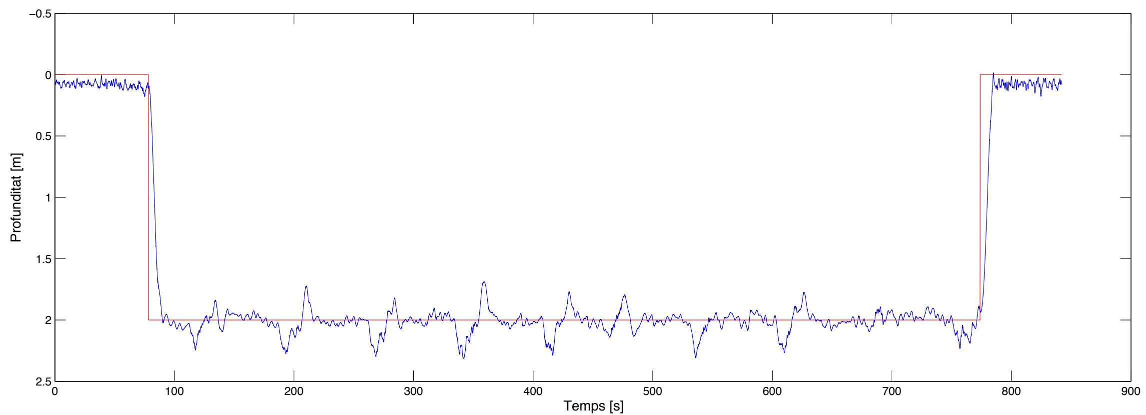

Figura 63. Dinàmica temporal de la profunditat. Timons habilitats i LOS 0,3 m/s – 1,0 m/s

Amb la figura anterior es pot apreciar que el control de profunditat presenta certes desviacions respecte la consigna. Aquest comportament s'ha associat al gir del robot en yaw, el que comporta consignes negatives dels motors horitzontals i conseqüentment, que el control de pitch quedi inhabilitat durant el període de temps que dura el gir.

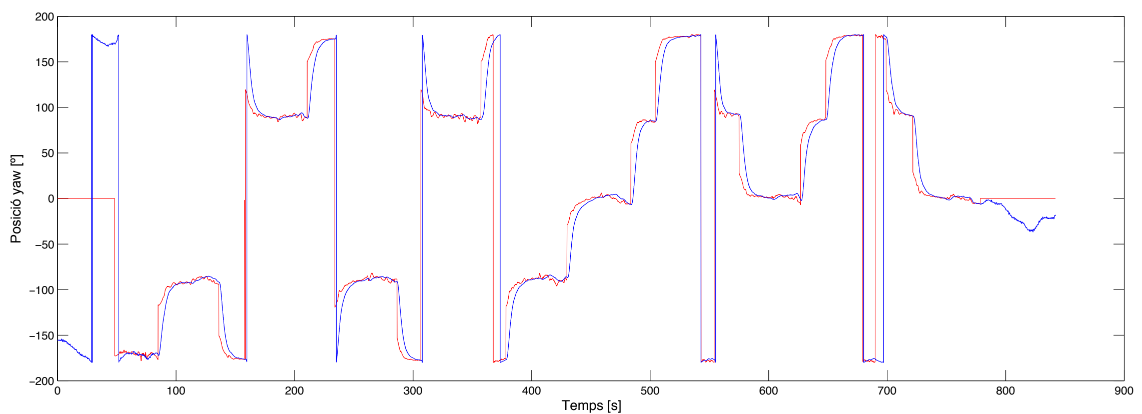

Figura 64. Dinàmica temporal de la posició en yaw. Timons habilitats i LOS 0,3 m/s – 1,0 m/s

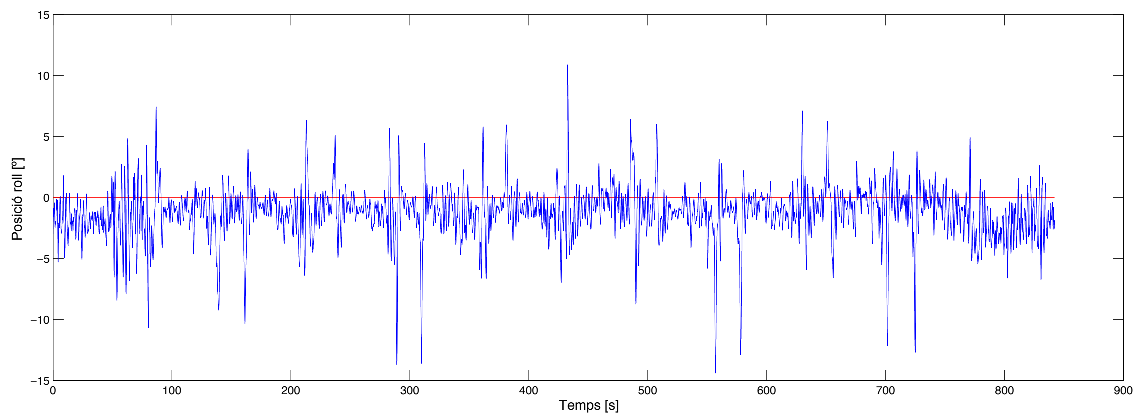

Figura 65. Dinàmica temporal de la posició en roll. Timons habilitats i LOS 0,3 m/s – 1,0 m/s

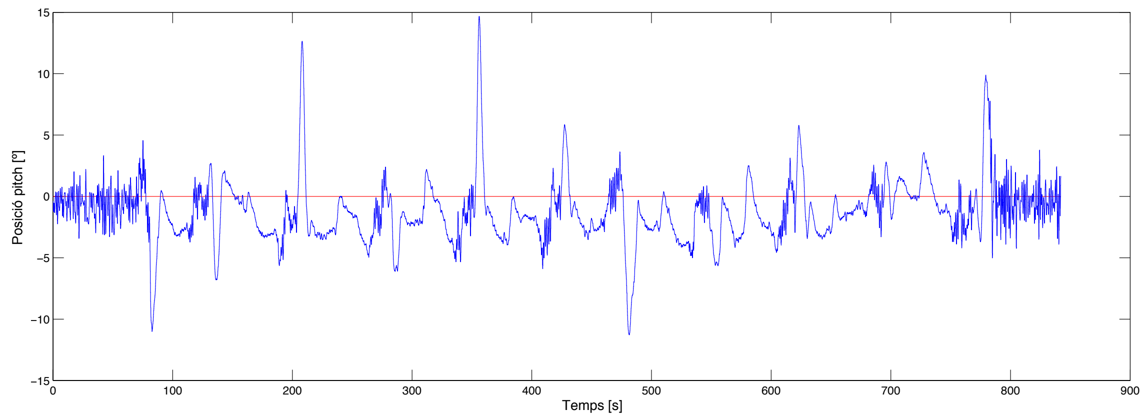

Figura 66. Dinàmica temporal de la posició en pitch. Timons habilitats i LOS 0,3 m/s – 1,0 m/s

S'ha repetit la trajectòria explicada anteriorment amb els timons inhabilitats, obtenint el resultat de la Figura 67 i la posició en profunditat i els DOFs de roll i de pitch representats amb la Figura 68 fins a la Figura 70. Amb aquestes es pot concloure el perquè anteriorment a la realització d'aquest projecte el robot s'utilitzava a una velocitat màxima de 0,5 m/s.

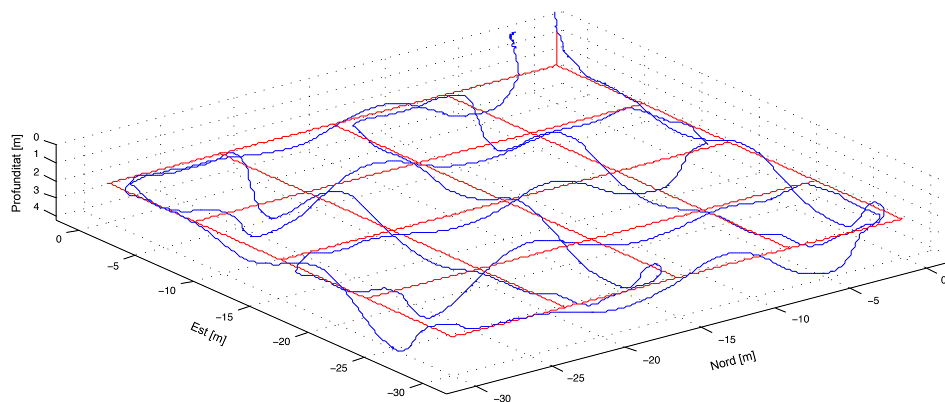

Figura 67. Trajectòria XYZ realitzada pel robot. Timons inhabilitats i LOS 0,3 m/s – 1,0 m/s

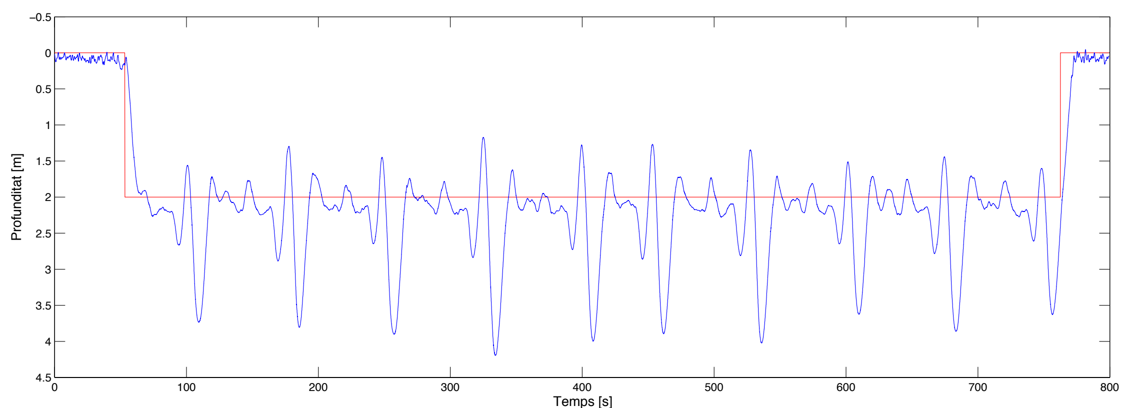

Figura 68. Dinàmica temporal de la profunditat. Timons inhabilitats i LOS 0,3 m/s – 1,0 m/s

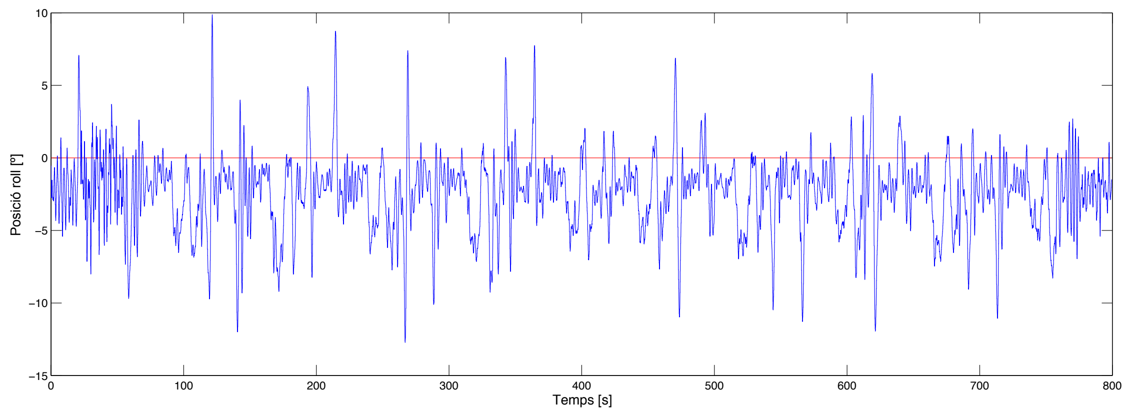

Figura 69. Dinàmica temporal de la posició en roll. Timons inhabilitats i LOS 0,3 m/s – 1,0 m/s

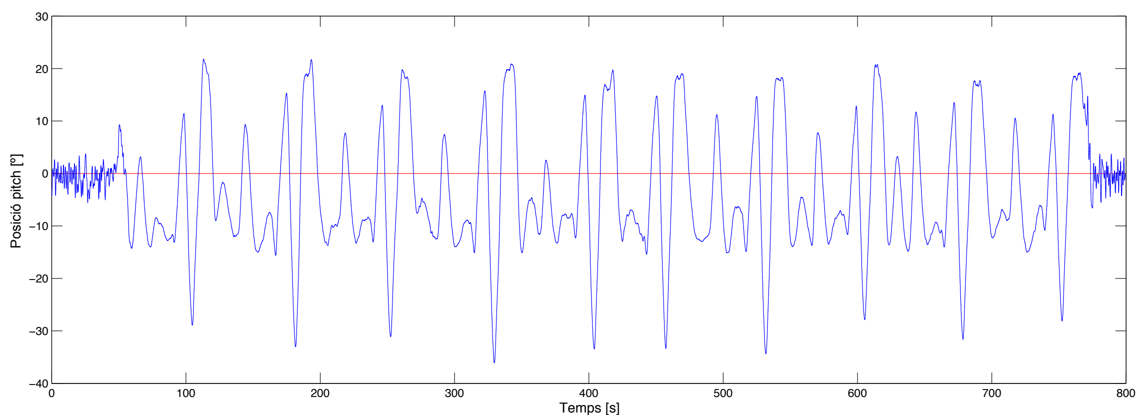

Figura 70. Dinàmica temporal de la posició en pitch. Timons inhabilitats i LOS 0,3 m/s – 1,0 m/s

Addicionalment als gràfics temporals, s'ha calculat la mitjana dels errors absoluts de profunditat, de roll i de pitch durant el temps que el robot ha estat submergit per a realitzar la missió. D'aquesta manera s'ha pogut avaluar el canvi numèricament.

	Profunditat [mm]	Roll [º]	Pitch [º]
Timons hab.	60,76	1,62	2,25
Timons inhab.	655,42	2,84	8,69

Taula 19. Mitjana dels errors absoluts dels experiments realitzats amb LOS 0,3 m/s – 1,0 m/s

Tant important com comparar la mitjana dels errors entre experiments pot ser analitzar els valors màxims respecte la consigna. Exactament es pot observar que amb els timons inhabilitats el robot s'ha submergit fins als 4 m de forma incontrolada, fet totalment no desitjat per a la seguretat del vehicle.

Finalment, s'ha avaluat la millora energètica aportada amb la integració dels timons de profunditat. El resultat d'aquest anàlisi ha sigut que amb els timons habilitats el motor vertical ha gastat una mitjana de 14,28 W, uns 28,54 W menys, el que correspon a una reducció del seu consum en un 66,64 %. Respecte la millora energètica global, la utilització dels timons de profunditat redueix un 9,81 % el consum total del robot.

8.2.3. Tercer assaig general del control

Seguint amb els experiments amb diferents configuracions de LOS per tal de comprovar el funcionament del control de baix nivell de l'SPARUS II, s'ha repetit la trajectòria anterior, el resultat de la qual està representat a la Figura 71. En aquest cas s'ha establert una configuració de LOS de 0,4 m/s per la velocitat lenta i de 1,5 m/s per la ràpida. Des de la Figura 72 fins a la Figura 76 es mostra la resposta temporal dels cinc DOFs del robot. En aquest cas també es poden apreciar les oscil·lacions de la profunditat obtingudes cada cop que el vehicle realitza un gir en yaw.

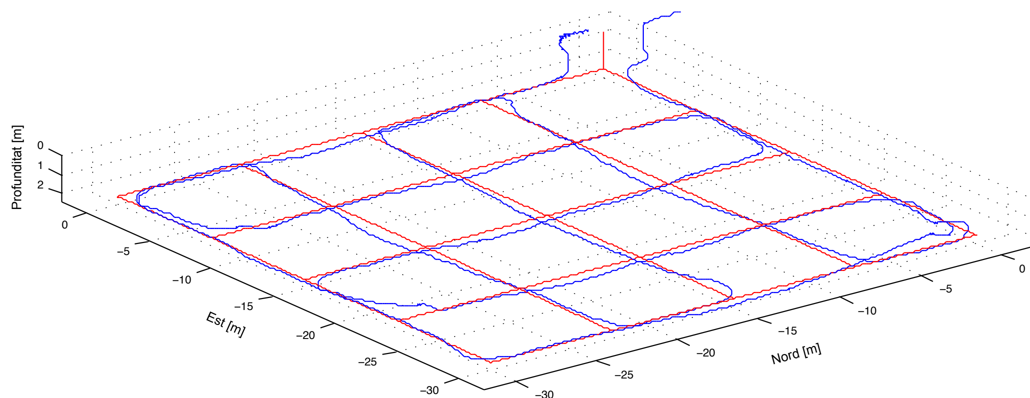

Figura 71. Trajectòria XYZ realitzada pel robot. Timons habilitats i LOS 0,4 m/s – 1,5 m/s

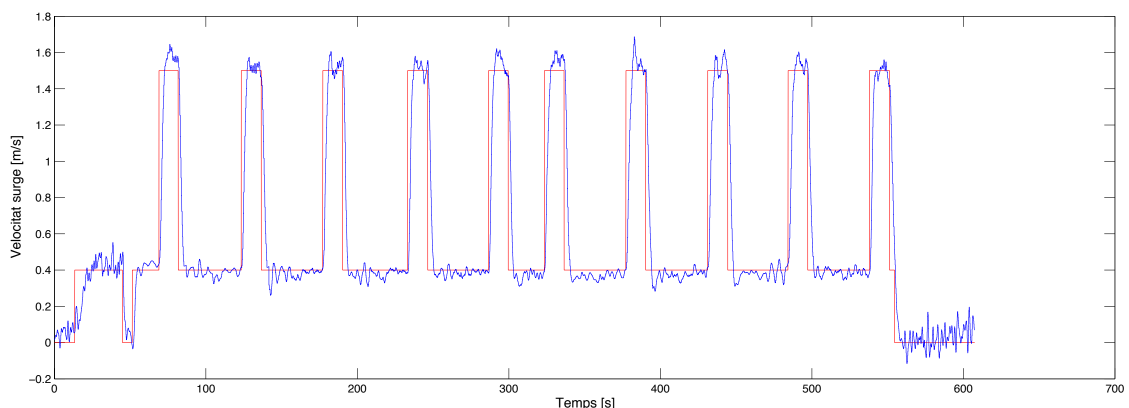

Figura 72. Dinàmica temporal de la velocitat en surge. Timons habilitats i LOS 0,4 m/s – 1,5 m/s

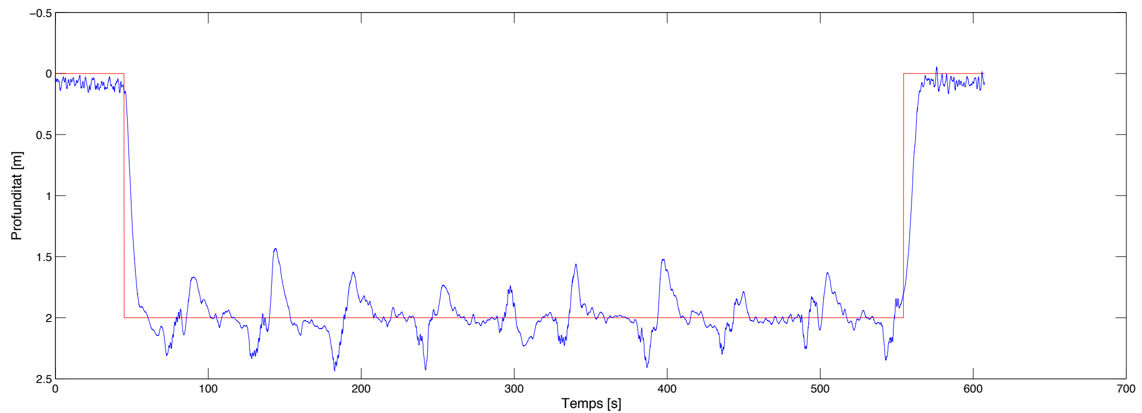

Figura 73. Dinàmica temporal de la profunditat. Timons habilitats i LOS 0,4 m/s – 1,5 m/s

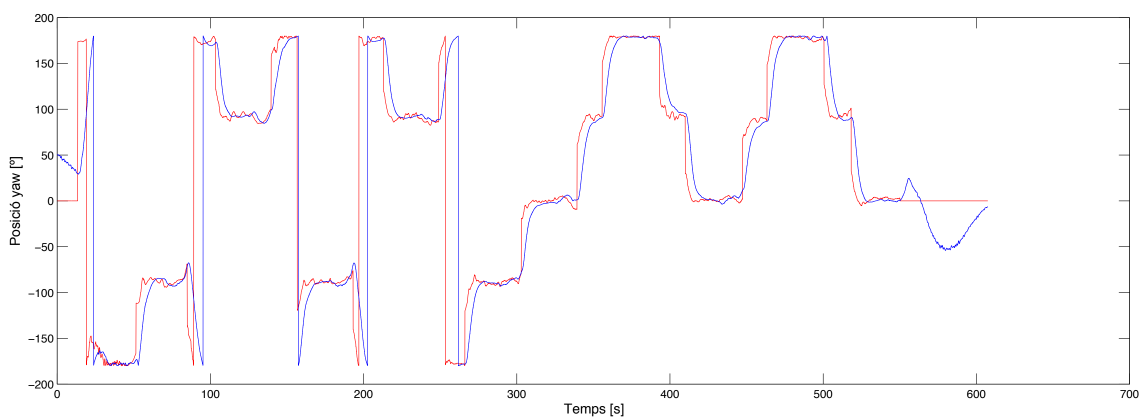

Figura 74. Dinàmica temporal de la posició en yaw. Timons habilitats i LOS 0,4 m/s – 1,5 m/s

Per a motius de seguretat aquest experiment no s'ha pogut replicar amb els timons de profunditat inhabilitats, ja que la resposta del vehicle seria totalment impredictible. Tot i això, es representa el comportament del robot en els DOFs roll i pitch, ja que en aquests es pot apreciar la tendència de complir amb la consigna prefixada.

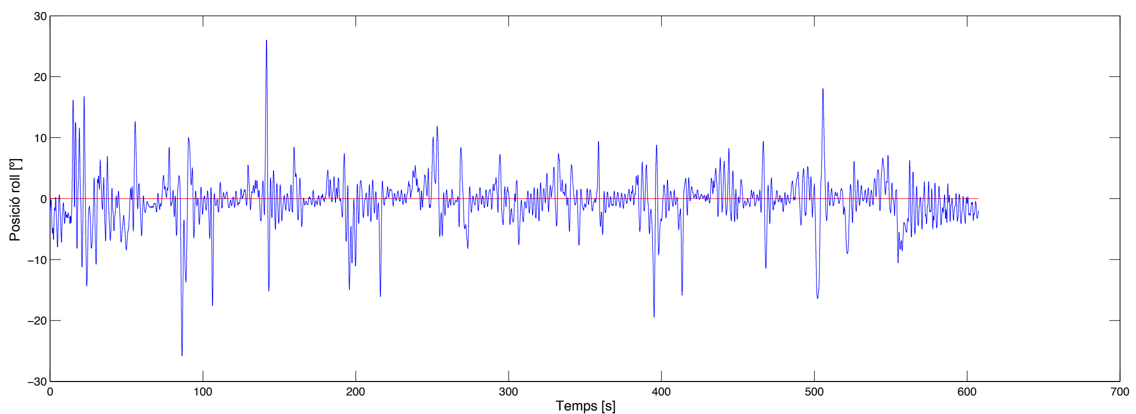

Figura 75. Dinàmica temporal de la posició en roll. Timons habilitats i LOS 0,4 m/s – 1,5 m/s

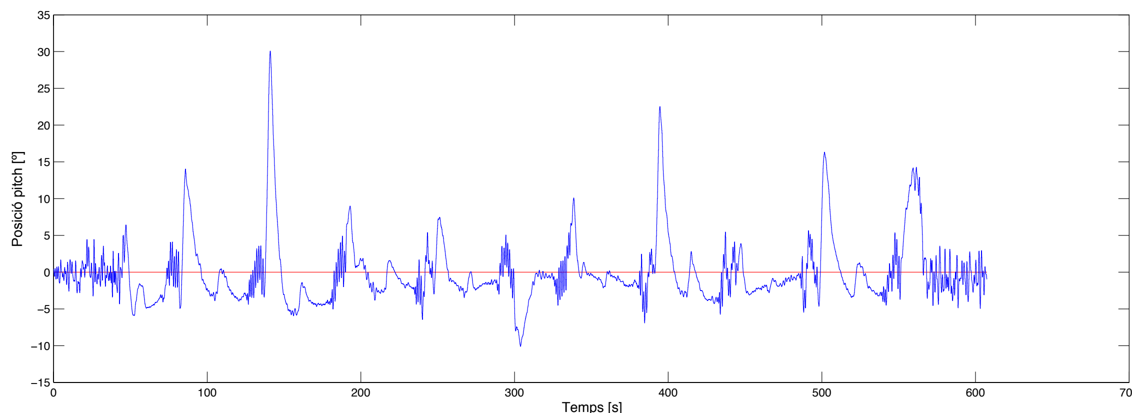

Figura 76. Dinàmica temporal de la posició en pitch. Timons habilitats i LOS 0,4 m/s – 1,5 m/s

També s'han calculat les mitjanes dels errors absoluts de la profunditat i els DOFs roll i pitch per veure la seva evolució al augmentar la velocitat de surge. Per a la profunditat s'ha obtingut una mitjana de 105,70 mm, pel roll de 2,36 º i pel pitch de 2,97 º, resultats totalment satisfactoris. Com que en aquest cas no s'ha pogut determinar l'estalvi energètic resultant de la integració dels timons de profunditat, únicament s'ha valoritzat el consum mitjà del motor vertical per un valor de 20,20 W. Aquestes dades serviran per a comparacions posteriors.

8.2.4. Quart assaig general del control

Finalment s'ha realitzat una nova trajectòria en forma de H, tal i com es pot apreciar a la Figura 77, amb una configuració de LOS de 0,25 m/s de velocitat lenta i de 2,0 m/s de velocitat ràpida. Amb aquesta parametrització s'ha volgut validar el curt temps d'establiment en la velocitat de surge i que el robot continuava podent controlar la posició en yaw tot i anar a la seva velocitat màxima. Tots els resultats obtinguts es troben reflectits des de la Figura 78 fins a la Figura 82.

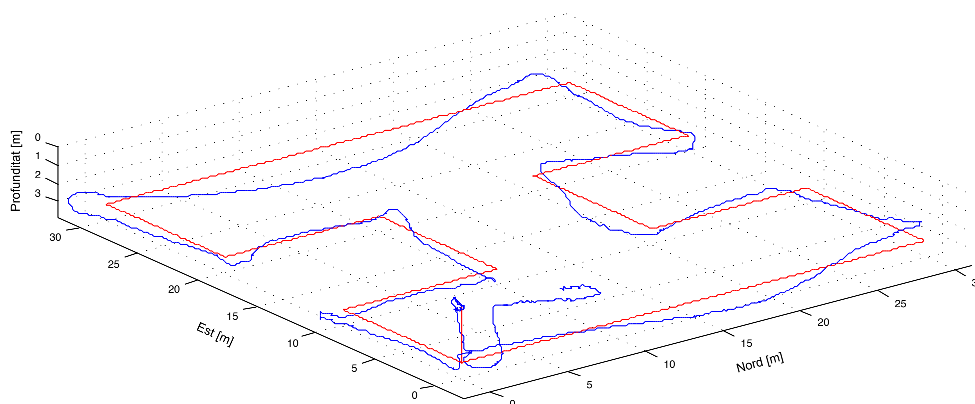

Figura 77. Trajectòria XYZ realitzada pel robot. Timons habilitats i LOS 0,25 m/s – 2,0 m/s

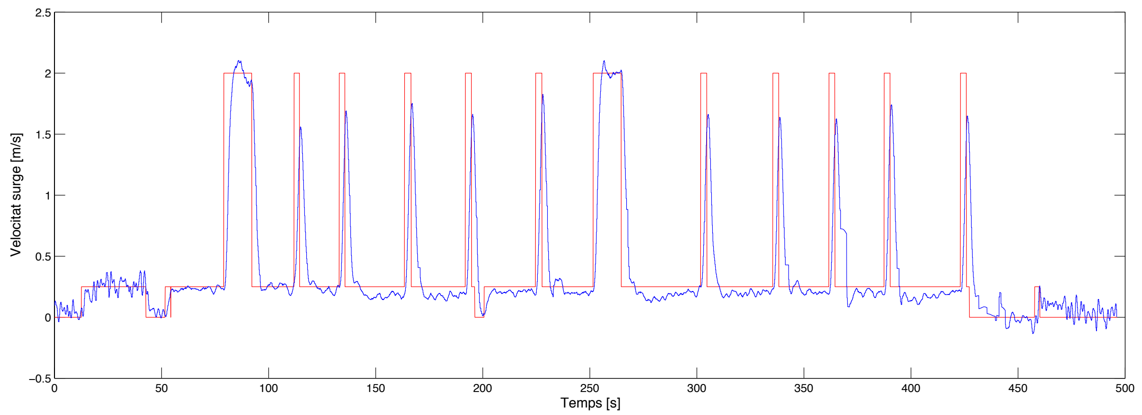

Figura 78. Dinàmica temporal de la velocitat en surge. Timons habilitats i LOS 0,25 m/s – 2,0 m/s

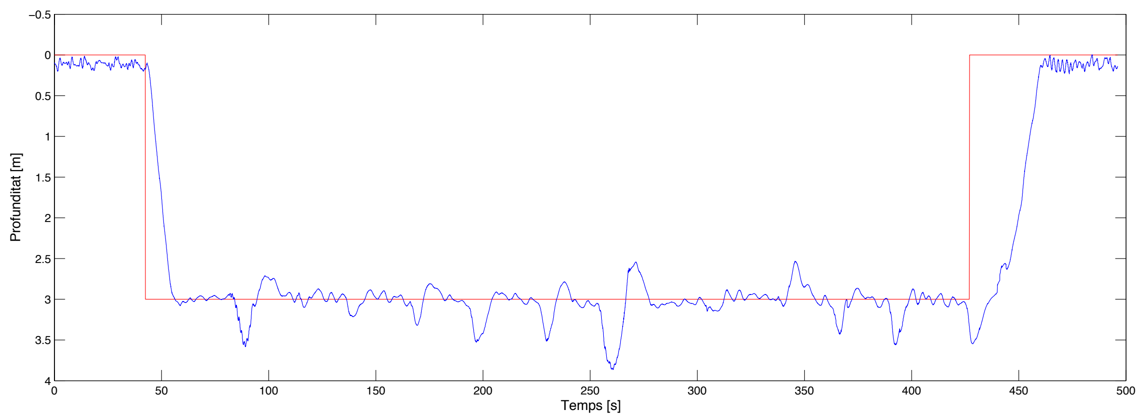

Figura 79. Dinàmica temporal de la profunditat. Timons habilitats i LOS 0,25 m/s – 2,0 m/s

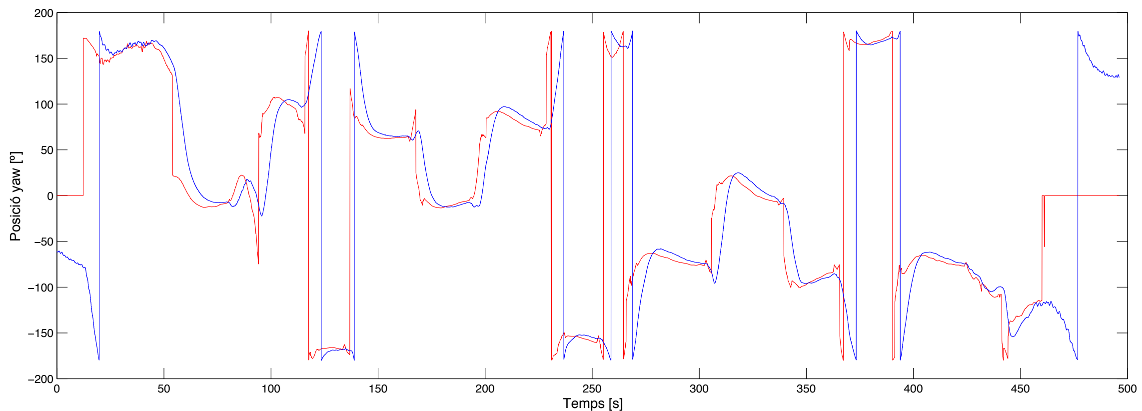

Figura 80. Dinàmica temporal de la posició en yaw. Timons habilitats i LOS 0,25 m/s – 2,0 m/s

A partir de la Figura 78 i de la Figura 80 es pot afirmar que, tot i que el vehicle anava a 2 m/s de velocitat en surge, la dinàmica de la posició en yaw no es veia influenciada. Respecte el control de profunditat, es torna a apreciar que la major desviació respecte la consigna es comet quan el robot acaba de realitzar un gir en yaw.

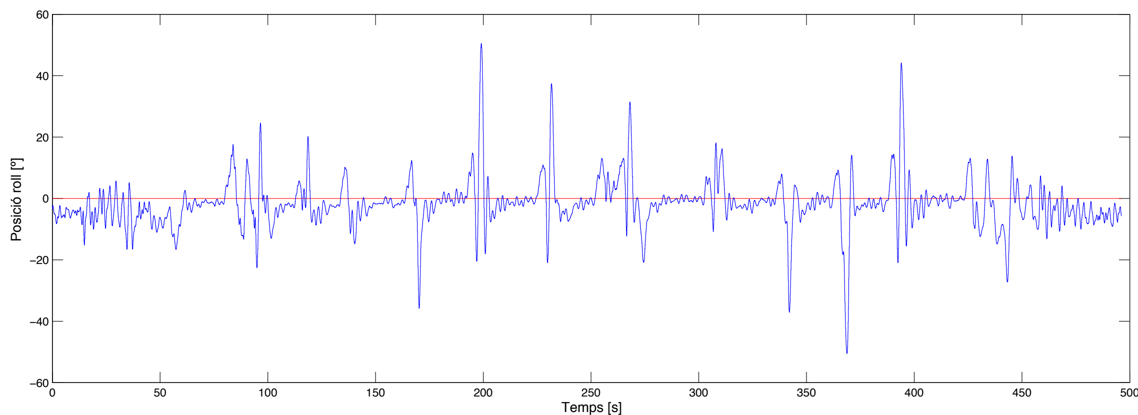

Figura 81. Dinàmica temporal de la posició en roll. Timons habilitats i LOS 0,25 m/s – 2,0 m/s

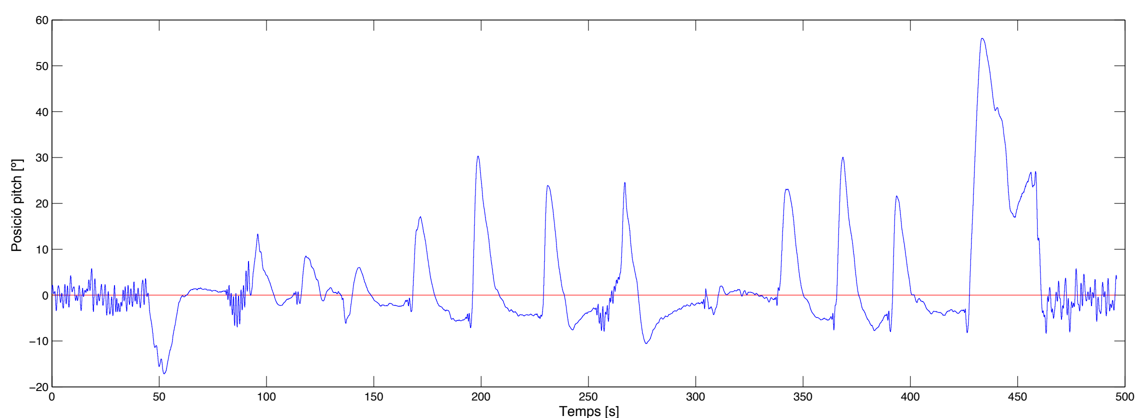

Figura 82. Dinàmica temporal de la posició en pitch. Timons habilitats i LOS 0,25 m/s – 2,0 m/s

A partir de les dades obtingudes s'ha quantificat la mitjana dels errors absoluts, així podent avaluar numèricament l'estabilitat de les diferents variables. El resultat obtingut per a la profunditat ha sigut de 122,98 mm, mentre que pel roll de 4,86 º i pel pitch de 5,47 º. Tot i que els DOFs de roll i pitch es mantenen relativament estables durant el transcurs del temps, en certs girs el robot assoleix unes posicions molt extremes que matemàticament són dades atípiques; aquestes fan augmentar considerablement la mitjana dels errors.

Per altra banda, la mitjana de consum del motor vertical durant la realització de la missió ha sigut de 35,51 W, dada que s'utilitzarà per a comparacions posteriors.

8.2.5. Assaig del control de profunditat amb pitch

Un cop realitzades totes les experimentacions necessàries per a validar el funcionament del control de baix nivell i la seva total integració amb el control d'alt nivell de l'arquitectura

COLA[2] del robot, s'ha assajat el control de profunditat amb pitch basat amb el FLS explicat anteriorment en el capítol de control.

A diferència dels assaigs anteriors, en aquests experiments l'anàlisi s'ha centrat en l'assoliment de la consigna de profunditat, tot revisant les sortides del FLS, és a dir, l'estat dels actuadors d'acord amb la velocitat de surge i la profunditat del vehicle. Per això, ha sigut de gran importància dissenyar una trajectòria que comportés tant la submersió com la emersió del vehicle i el manteniment d'una profunditat constant. A més, al ser una trajectòria formada per varies consignes XYZ o punts, el FLS ha hagut de gestionar els diferents canvis de velocitat en surge comandats pel LOS.

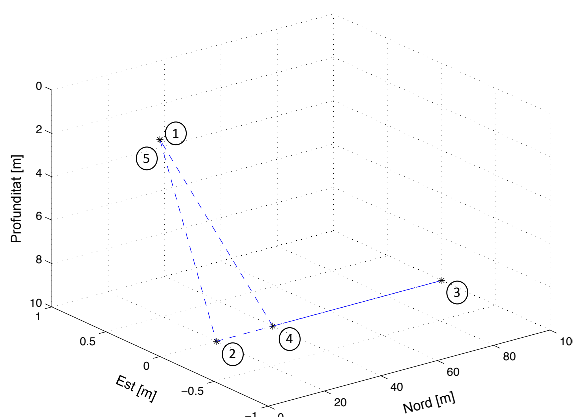

Figura 83. Trajectòria programada per l'assaig del control de profunditat amb pitch

Exactament, la primera acció era submergir-se a 10 m de profunditat amb una distància de 20 m en direcció Nord. Tot seguit, el robot havia de mantenir el rumb i la profunditat fins als 100 m respecte el NED. Un cop realitzat un gir en yaw de 180 º, el vehicle havia d'anar en direcció Sud 60 m mantenint la profunditat. A partir d'aquí i a falta de 40 m fins al punt de partida se li programava que emergís, tenint tot aquest marge per a realitzar-ho. Aquesta trajectòria s'ha realitzat quatre cops amb diferents configuracions del LOS amb el fi d'assajar el control de profunditat amb pitch per a tots els punts de treball de l'SPARUS II.

Així doncs, la parametrització per al primer experiment, el resultat del qual correspon a la Figura 84, ha sigut de 0,2 m/s per a la velocitat lenta i 0,5 m/s per a la ràpida. Per això, s'espera que el FLS determini que el control de profunditat s'ha de realitzar totalment amb el DOF heave i generi una consigna pel control de posició en pitch de 0 º. Conseqüentment, la dinàmica temporal de la profunditat serà pràcticament igual a l'obtinguda en els experiments realitzats anteriorment amb la mateixa configuració de LOS.

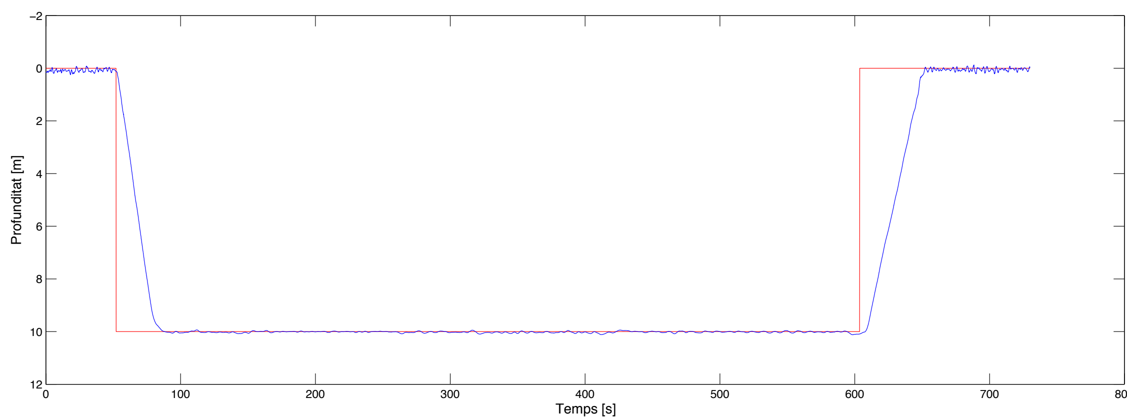

Figura 84. Dinàmica temporal de la profunditat. Control de profunditat amb pitch i LOS 0,2 m/s – 0,5 m/s

Per la següent missió s'ha configurat el LOS amb 0,3 m/s i 1,0 m/s. D'acord amb el FLS implementant, quan el robot vagi en velocitat ràpida la sortida del control de posició en heave es veurà reduït aproximadament un 30 %, mentre que s'assignarà una consigna en el control de pitch que posicioni el robot amb un angle d'atac lleugerament negatiu. Cada cop que el robot s'aproximi a un dels punts XYZ programats, la velocitat en surge reduirà fins als 0,3 m/s, pel que el FLS adequarà com correspongui les seves tres sortides.

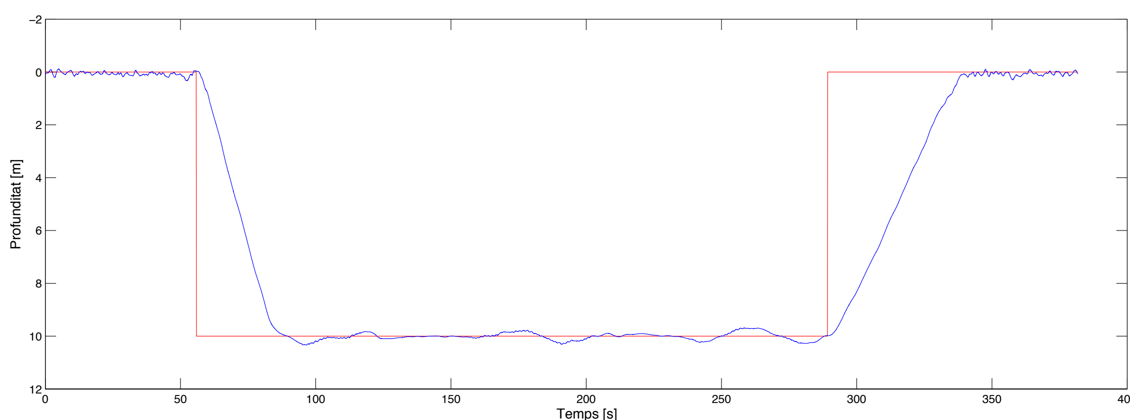

Figura 85. Dinàmica temporal de la profunditat. Control de profunditat amb pitch i LOS 0,3 m/s – 1,0 m/s

Els paràmetres del tercer experiment, el resultat del control de profunditat del qual està representat per la Figura 86, s'han fixat a 0,4 m/s i 1,5 m/s respectivament. Amb aquesta configuració, quan el vehicle vagi a la velocitat ràpida tot el control de heave es veurà anul·lat, és a dir, tant la força feedforward que compensa la força de flotació com el control de posició de heave es multiplicaran per un coeficient zero. Dit d'una altra manera, en aquest cas el robot controlarà la profunditat únicament amb els timons de profunditat. Tot i això, cada cop que el robot s'acosti a un punt XYZ, el FLS haurà d'adaptar les seves sortides al llarg de la transició.

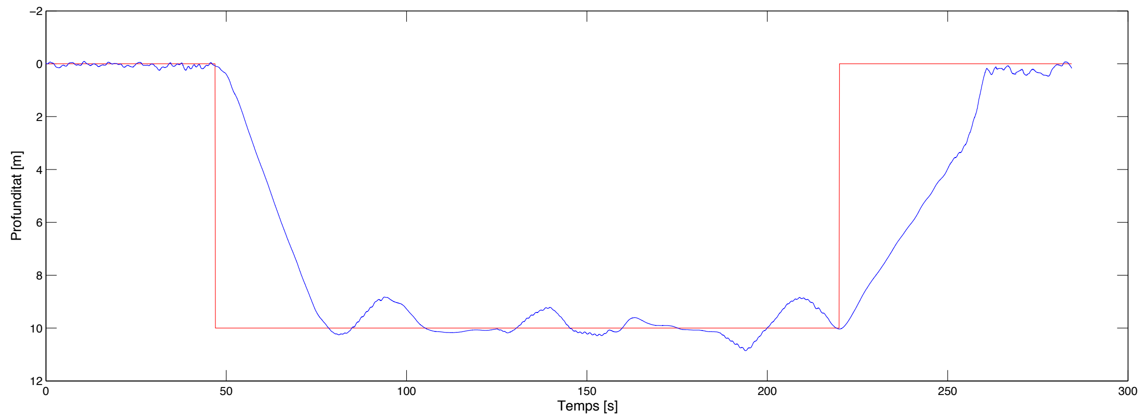

Figura 86. Dinàmica temporal de la profunditat. Control de profunditat amb pitch i LOS 0,4 m/s – 1,5 m/s

Finalment, s'ha repetit l'experiment amb configuració del LOS amb 0,5 m/s de velocitat lenta i 2,0 m/s de ràpida. Igual que en el cas anterior, s'espera que el vehicle controli la profunditat únicament amb els timons de profunditat quan vagi a la seva velocitat alta. La finalitat d'aquest experiment, per diferenciar-lo de l'anterior, era veure com reacciona el sistema quan se'l fa variar bruscament de velocitats en surge.

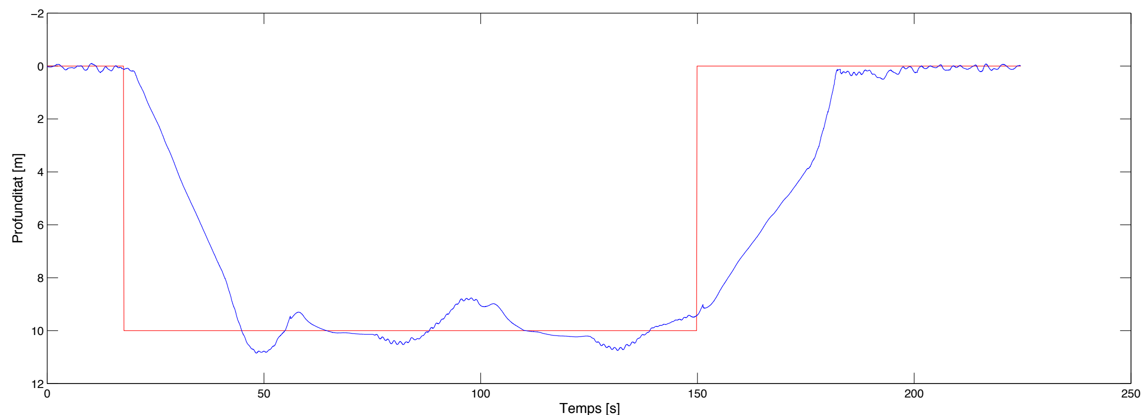

Figura 87. Dinàmica temporal de la profunditat. Control de profunditat amb pitch i LOS 0,5 m/s – 2,0 m/s

Al llarg dels diferents experiments s'ha pogut afirmar que el FLS funciona correctament, ja que el valor de cada una de les seves tres sortides s'adequava a les idees exposades en aquest document. Tanmateix, cal recordar que aquest control s'ha realitzat a nivell d'investigació, pel que els resultats obtinguts són totalment satisfactoris per a ser la primera iteració i assaigs que es realitzen amb aquest mode de funcionament.

Els errors de profunditat obtinguts durant els tests estan contribuïts al mateix efecte explicat anteriorment; quan el robot frena o realitza un gir i conseqüentment s'envien als motors consignes negatives, els timons queden inhabilitats al posar-se a zero graus. Tot i això, s'ha

analitzat la mitjana dels errors en absolut de profunditat per tal de poder comparar aquest mode de control amb el convencional.

	v = 0,5 m/s	v = 1,0 m/s	v = 1,5 m/s	v = 2,0 m/s
Control convencional	47,82 mm	60,76 mm	105,70 mm	122,98 mm
Control profunditat amb pitch	43,29 mm	108,20 mm	352,47 mm	402,21 mm

Taula 20. Comparativa de la mitjana dels errors absoluts de la profunditat entre modes de control

Dels resultats mostrats a la taula anterior un cop descartades les diferències menyspreables entre les trajectòries realitzades a 0,5 m/s, es pot observar que el control de profunditat amb pitch proporciona una solució no tant ajustada com el control convencional, però satisfactòria per ser un mode de control nou en la robòtica submarina. Fins a 1 m/s la diferència entre els dos modes és similar, però a velocitats majors les dades atípiques obtingudes en els girs fan augmentar significativament la diferència de la mitjana dels errors.

També s'ha calculat el consum mitjà del motor vertical per tal de tenir una idea de l'estalvi energètic que suposa aquest mode de funcionament respecte el convencional. A diferència de les comparacions anteriors, aquest valor únicament es orientatiu, ja que els sensors actius en cada experiment eren diferents i per tant, les dades provinents del BMS no s'han pogut comparar directament. Tot i això, l'aproximació realitzada és prou precisa, ja que en els dos casos els servomotors estaven actius.

	v = 0,5 m/s	v = 1,0 m/s	v = 1,5 m/s	v = 2,0 m/s
Control convencional	11,60 W	14,28 W	20,20 W	35,51 W
Control de profunditat amb pitch	11,93 W	8,38 W	7,76 W	10,21 W
Estalvi energètic aproximat	-2,84 %	41,32 %	61,58 %	71,25 %

Taula 21. Comparativa del consum mitjà del motor vertical entre modes de control

L'estalvi energètic creix considerablement al augmentar la velocitat de surge. Tot i això, el consum del motor vertical no redueix com li correspondria; al arribar a un punt XYZ i frenar per assolir la consigna de velocitat lenta en surge, els timons queden inhabilitats. En aquest moment el posicionament en pitch sol ser significatiu, el que fa moure el vehicle en Z al

mateix temps que el motor vertical intenta compensar l'error de profunditat. Aquesta combinació fa que el motor vertical generi pics de consum elevats.

Finalment cal dir que per a la selecció de la configuració del LOS tot utilitzant el control de profunditat amb pitch caldrà buscar un punt entremig al compromís plantejat; si es vol tenir una bona estabilitat en la profunditat caldrà escollir velocitats lentes, mentre que si el que es vol és assolir ràpidament una posició XYZ amb un rendiment del vehicle òptim, s'haurà de triar una de les velocitats altes.

9. RESUM DEL PRESSUPOST

El pressupost realitzat per a la valoració del present projecte inclou el disseny, fabricació, programació, implementació i assaig de tot el que s'ha exposat en aquest document, és a dir, d'una placa electrònica, d'un protocol de comunicacions i d'un sistema de control.

El cost associat de tot el descrit és de vint-i-tres mil tres-cents quaranta euros amb seixanta-un cèntims, sense IVA.

10. CONCLUSIONS

El plantejament d'aquest projecte ha partit dels objectius exposats en el present document, els quals eren clars i concisos. L'objectiu general era integrar dos timons de profunditat a l'SPARUS II per tal de controlar-lo en cinc graus de llibertat, pel que s'ha dissenyat, implementat i programat tot el necessari per assolir les prestacions pel qual va ser dissenyat.

Per aconseguir l'objectiu principal s'ha seguit l'estratègia d'assolir fites parcials en cada una de les cinc grans temàtiques del projecte: el hardware, la comunicació sèrie, el firmware, la modelització i el control. Completant-les en aquest ordre s'ha aconseguit treballar fluidament sobre una base ferma.

Els resultats obtinguts amb el present projecte afirmen que l'SPARUS II ha assolit les seves característiques de disseny de manera totalment satisfactòria, el que al mateix temps ha verificat que tota l'electrònica dissenyada és robusta i fiable. Tanmateix, part de la feina realitzada ha marcat l'inici d'un nou camí d'investigació en la robòtica submarina, concretament en com fer un control acurat de la profunditat a través del DOF pitch.

Com a futurs treballs relacionats amb el hardware seria interessant redissenyar el compartiment dels timons de profunditat amb el fi de poder-hi allotjar els sensors angulars presentats en aquest document, oferint la possibilitat de detectar problemes mecànics i de fer un autocalibratge dels timons.

A nivell de control caldria modelar els timons de profunditat per a qualsevol direcció del flux incident a les pales, així aconseguint un control de pitch i roll constant per a qualsevol moviment del robot i conseqüentment una major estabilitat dels seus cinc graus de llibertat.

Èric Pairet Artau
Graduat en enginyeria electrònica industrial i automàtica

Banyoles, 1 de setembre de 2015

11. RELACIÓ DE DOCUMENTS

El present projecte es troba desenvolupat al llarg de diferents documents, els quals són la memòria, els plànols, el plec de condicions, l'estat d'amidaments i el pressupost.

12. BIBLIOGRAFIA

ADVANTECH. ADAM 4520. Isolated RS-232 to RS-422/485 converter. Taipei. (http://newt.phys.unsw.edu.au/~jl/GATTINO/Manual/manuals/ADAM4520.pdf, 11 de novembre de 2014)

CARLTON, J. Marine propellers and propulsion. Elsevier. Oxford. 2012.

CARRERAS, M., CANDELA, C., RIBAS, D., MALLIOS, A., MAGÍ, LL., VIDAL, E., PALOMERAS, RIDAO, P. SPARUS II, design of a lightweight hovering AUV. International Workshop on Marine Technology. 2013.

FOLGER, L., FEHLNER, L. Free-stream characteristics of a family of low-aspect-ratio, all-movable control surfaces for application to ship design. Hydromechanics laboratory research and development report. 1958.

FOSSEN, T. Handbook of marine craft hydrodynamics and motion control. Wiley. 2011.

MENDEL, J. Fuzzy logic systems for engineering: a tutorial. Procedings of the IEEE, 83(3):345–377. 1995.

MOLLAND, A., TURNOCK, S. Marine rudders and control surfaces. Elsevier. 2007.

MGM COMPRO. HSBC – SERIES TMM V7. República Xeca. 2011.

Microchip. PIC16F/LF1825/1829 Data Sheet. West Chandler Blvd. (http://www.mouser.com/ds/2/268/41440A-50115.pdf, 24 de novembre de 2014)

CARNEROS, B., RAYÓN, P. Reglamento Electrotécnico para Baja Tensión. Mc Graw Hill. Aravaca. 2003.

OGATA, K. Ingeniería de control moderna. Pearson. Cinquena edició. Madrid. 2010.

PALOMERAS, N., EL-FAKDI, M., CARRERAS, M., RIDAO, P. COLA2: A control architecture for AUVs. IEEE Journal of Oceanic Engineering. 2012.

PEREZ, T. Ship motion control. Course Keeping and Roll Stabilisation using Rudder and Fins. Springer. London. 2005.

PIHER. MTS-360 Through shaft contactless sensor. Espanya. (http://piher.net/pdf/mts360_datasheet.pdf, 15 de setembre de 2014)

PRESTERO, T. Verification of a Six-Degree of Freedom Simulation Model for the REMUS Autonomous Underwater Vehicle. Master Tesis. Master of Science in Ocean Engineering and Master of Science in Mechanical Engineering. Massachusetts Institute of Technology. Setembre 2001.

RIBAS, D., PALOMERAS, N., RIDAO, P., CARRERAS, M., MALLIOS,. A. Girona 500 AUV: From survey to intervention. Mechatronics, IEEE/ASME Transaction. 2012.

ROS. ROS documentation. (http://www.ros.org/wiki/, 21 de decembre de 2014)

SAVÖX. SC-1251MG Low Profile High Speed Metal Gear Digital Servo. SAVÖX. (http://www.savoxusa.com/Savox_SC1251MG_Digital_Servo_p/savsc1251mg.htm, 23 de juny de 2015)

SHELDAHL, R., KLIMAS, P. Aerodinamic characteristics of seven symmetrical airfoil sections through 180-degree angle of attack for use in aerodynamic analysis of vertical axis wind turbins. Sandia National Laboratories report. 1981.

SHOTTS, W. The Linux Command Line. Creative Commons. California. 2009.

STMicroelectronics. Very low drop voltage regulator with inhibit function. Geneva. (http://www.st.com/web/en/resource/technical/document/datasheet/CD00000546.pdf, 10 de setembre de 2014)

Texas Instruments. DS75176B/DS75176BT Multipoint RS-485/RS-422 Transceivers. Texas. (http://www.ti.com/lit/ds/symlink/ds75176b.pdf, 11 de setembre de 2014)

Texas Instruments. RS-485: Passive failsafe for an idle bus. Texas. (http://www.ti.com/lit/an/slyt324/slyt324.pdf, 10 de setembre de 2014)

Texas Instruments. 6-A, Wide-Input Adjustable Switching Regulator. Texas. (http://www.ti.com/lit/ds/slts228c/slts228c.pdf, 26 de setembre de 2014)

VIDAL, E. Navegació, control i modelització del robot submarí Sparus II. Projecte/Treball Fi de Carrera. Enginyeria Industrial. Escola Politècnica Superior. Universitat de Girona. Setembre 2014.

13. GLOSSARI

ASCII	American Standard Code for Information Interchange
AUV	Autonomous Underwater Vehicle
BCC	Block Check Character
BMS	Battery Management System
CIRS	Centre d'Investigació en Robòtica Submarina
COLA^2	Component Oriented Layered-base Architecture for Autonomy
DOF	Degree Of Freedom
DVL	Doppler Velocity Log
EEPROM	Electrically-Erasable Programmable Read-Only Memory
EKF	Extended Kalman Filter
FLS	Fuzzy Logic System
GIRONA 500	Generic Intelligent Robot Operated and Navigated Autonomously
LOS	Line Of Sight
NACA	National Advisory Council for Aeronautics
NED	North-East-Down
PC	Personal Computer

PCB	Printed Circuit Board
PID	Proportional-Integral-Derivative
PWM	Pulse Width Modulation
REBT	Reglement Electrotècnic de Baixa Tensió
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
UWSim	UnderWater Simulator
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
UdG	Universitat de Girona
VDC	Voltatge of Direct Current
VICOROB	Visió per Computador i Robòtica

A. MANUAL DE LA PLACA ELECTRÒNICA

Aquest annex adjunta el manual d'instal·lació de la placa electrònica implementada pel posicionament dels timons de profunditat de l'SPARUS II. En el CIRS tot el hardware que s'incorpora al robot es documenta per estar a la disposició de l'encarregat de manteniment del vehicle.

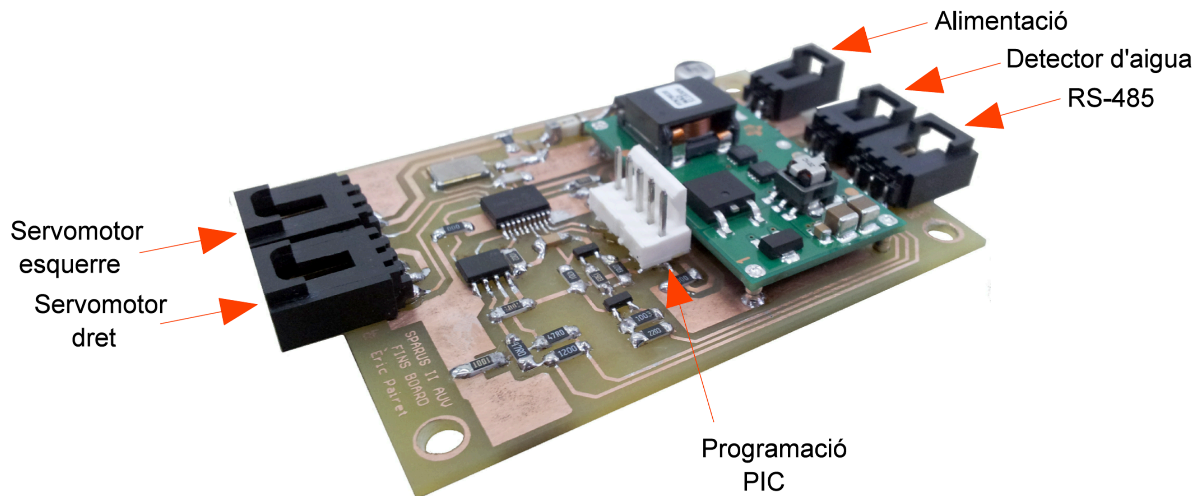

Figura 88. Placa electrònica dels timons de profunditat de l'SPARUS II

A continuació es detalla el pinout de cada connector de placa electrònica dels timons de profunditat.

Connector de programació PIC.

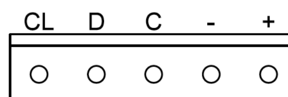

Figura 89. Detall connector programació PIC

Connector d'alimentació.

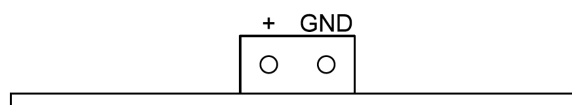

Figura 90. Detall connector d'alimentació

Connector bus RS-485.

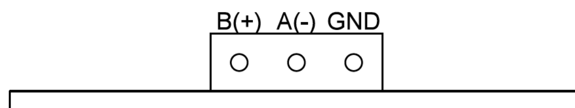

Figura 91. Detall connector bus RS-485

Connector del detector d'aigua.

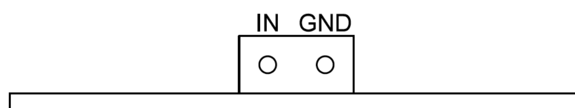

Figura 92. Detall connector del detector d'aigua

Connector del servomotor esquerre.

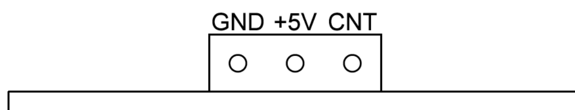

Figura 93. Detall connector servomotor esquerre

Connector del servomotor dret.

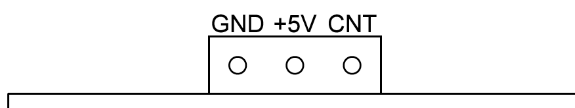

Figura 94. Detall connector servomotor dret

B. CÀLCULS

Per a obtenir la durada dels pols del senyal de control a partir de les consignes en graus enviades per l'ordinador central del robot, s'han hagut d'incorporar varis càlculs dins el firmware. L'estratègia utilitzada ha sigut escalar les consignes de cada servomotor a partir dels seus paràmetres preconfigurats. Com que un d'aquests és el temps equivalent per la posició central, s'ha requerit de l'Equació 20 per escalar les consignes que són iguals o inferiors a la meitat del rang del propi actuador i de l'Equació 21 per les que són superiors.

$$t_{pols} = t_{mínim} + \frac{t_{centre} - t_{mínim}}{\frac{rang}{2}} \cdot consigna \qquad \text{(Eq. 20)}$$

$$t_{pols} = t_{centre} + \frac{t_{màxim} - t_{centre}}{\frac{rang}{2}} \cdot \left(consigna - \frac{rang}{2} \right) \qquad \text{(Eq. 21)}$$

on,

t_{pols} = temps de durada del pols del senyal de control, en microsegons

$t_{mínim}$ = temps configurat equivalent a l'angle mínim, en microsegons

t_{centre} = temps configurat equivalent a l'angle central, en microsegons

$t_{màxim}$ = temps configurat equivalent a l'angle màxim, en microsegons

rang = recorregut d'obertura configurat, en graus

consigna = posició desitjada del servomotor, en graus

Els resultats obtinguts de les equacions anteriors per a cada servomotor es comproven que estiguin dins del rang de temps compres entre els seus paràmetres $t_{màxim}$ i $t_{mínim}$. En cas de no ser així, t_{pols} es satura al valor més proper dels límits del rang, amb el fi d'evitar que l'eix de sortida del servomotor quedi sense parell.

Particularment pel servomotor dret i com a conseqüència del seu muntatge dins del compartiment dels timons de profunditat, abans d'aplicar les equacions anteriors s'ha de condicionar la consigna de la manera que s'indica a continuació.

$$\text{consigna} = \text{rang} - \text{consigna_rebuda} \qquad \text{(Eq. 22)}$$

on,

consigna = posició desitjada del servomotor, en graus

rang = recorregut d'obertura configurat, en graus

consigna_rebuda = posició desitjada per l'aleta, en graus

Finalment, com que el temporitzador reconfigurable utilitzat per a adequar el senyal de control al cicle de treball desitjat és de 8 bits i incremental, abans de programar-li el temps de pols calculat amb les equacions anteriors cal adequar el valor.

$$\text{temporitzador} = 256 - \frac{t_{pols}}{11{,}5625} \qquad \text{(Eq. 23)}$$

on,

temporitzador = valor enter a programar al temporitzador, adimensional

t_{pols} = temps de durada del pols del senyal de control, en microsegons

Amb l'equació anterior queda justificada la raó pel qual el temporitzador zero s'ha configurat amb una resolució de 11,5625 µs a la fase d'inicialització del firmware. Aquesta parametrització ha sigut el resultat de considerar el compromís entre cobrir les possibles necessitats del senyal de control, les quals són inferiors als 2.960 ms màxims programables en el temporitzador, i disposar d'una bona resolució angular en els timons de profunditat.

C. PROGRAMACIÓ DEL FIRMWARE

C.1. Fitxer de configuració

```
////////////////////////////////////////////////////////////////////////////
////                          fins_program.h                            ////
////                           version: 1.5                             ////
////                          date: 04/03/15                            ////
////                        author: Èric Pairet                         ////
////                                                                    ////
//// This program defines and configures the used hardware and serial  ////
//// port. Also it gives names to the different pins and values to     ////
//// some variables                                                     ////
////////////////////////////////////////////////////////////////////////////

#include <16F1829.h>
#device adc=16

#define FIRMWARE_VERSION "1.5"
#define ADDRESS_ID        116


#FUSES NOWDT  //No Watch Dog Timer
#FUSES HS  //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES WDT_SW  //No Watch Dog Timer, enabled in Software
#FUSES PROTECT  //Code protected from reads
#FUSES NOBROWNOUT  //No brownout reset
#FUSES WRT  //Program Memory Write Protected
#FUSES PLL_SW  //4X HW PLL disabled, 4X PLL enabled/disabled in software
#FUSES NOSTVREN  //Stack full/underflow will not cause reset
#FUSES NOLVP  //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O

#use delay(clock=11059200)

#ROM 0x2100 = { '1','2','3','4' }

#use FIXED_IO( B_outputs=PIN_B6 )
#use FIXED_IO( C_outputs=PIN_C7,PIN_C6 )
#define DIRECTION   PIN_B6
#define WATER_SENSOR   PIN_C0
#define SERVO_L   PIN_C6
#define SERVO_R   PIN_C7

//RS-485 flow control
#define RECEIVE              0
#define TRANSMIT             1

//Main state machine options
#define COMMAND         0
#define ADDRESS         1
#define PARAMETER1      2
#define PARAMETER2      3
#define CONTROL_BIT     4
#define CONFIRMATION    5
#define CONFIGURATION   6

//Configuration state machine options
```

```
#define MINL   0
#define ZERL   2
#define MAXL   4
#define RANL   6
#define MINR   8
#define ZERR   10
#define MAXR   12
#define RANR   14

#use  rs232(  baud=57600,parity=N,xmit=PIN_B7,rcv=PIN_B5,bits=8,stream=PORT1
)
```

C.2. Programa principal

```
///////////////////////////////////////////////////////////////////////////
////                          fins_program.c                          ////
////                           version: 1.5                           ////
////                          date: 04/03/15                          ////
////                        author: Èric Pairet                       ////
////                                                                  ////
////  This program is able to control up to 2 servomotors and detect  ////
////  the existence of not desired water in the fins compartment      ////
////                                                                  ////
////  - The recognised ASCII commands for this program are:           ////
////                                                                  ////
////  v\r            to ask for the firmware version                  ////
////                                                                  ////
////  MinLXXXX\r     to configure the equivalent time to the minimum  ////
////                 angle for the left fin                           ////
////  ZerLXXXX\r     to configure the equivalent time to the          ////
////                 horizontal    position for the left fin          ////
////  MaxLXXXX\r     to configure the equivalent time to the màximum  ////
////                 angle for the left fin                           ////
////  RanLXXXX\r     to configure the left fin range in degrees       ////
////                                                                  ////
////  MinRXXXX\r     to configure the equivalent time to the mínimum  ////
////                 angle for the right fin                          ////
////  ZerRXXXX\r     to configure the equivalent time to the          ////
////                 horitzontal position for the right fin           ////
////  MaxRXXXX\r     to configure the equivalent time to the màximum  ////
////                 angle for the right fin                          ////
////  RanRXXXX\r     to configure the right fin range in degrees      ////
////                                                                  ////
////                                                                  ////
////  Where XXXX is a maximum 4 digits number which contents the value ////
////  of the configured parameter                                     ////
////                                                                  ////
////  - The recognised binary commands for this program are:          ////
////                                                                  ////
////  253 + ADDRESS + LF + RF + BCC    to control the fins position   ////
////                                                                  ////
////  Where:                                                          ////
////                                                                  ////
////     ADDRESS is the specified address in the 'fins_program.h' file ////
////     LF is the desired angle in degrees for the left fin          ////
////     RF is the desired angle in degrees for the right fin         ////
////     BCC = ADDRESS xor LF xor RF                                  ////
////                                                                  ////
```

```
////  254 + ADDRESS + 0 + 0         to disable the fins control        ////
////  254 + ADDRESS + 1 + 1         to enable the fins control         ////
////  254 + ADDRESS + 2 + 2         to ask for water existence         ////
////  254 + ADDRESS + 10 + 10       to enable the ASCII commands       ////
////                                                                   ////
////  Where:                                                           ////
////                                                                   ////
////     ADDRESS is the specified address in the 'fins_program.h' file ////
///////////////////////////////////////////////////////////////////////////

//Attach of required libraries
#include <fins_program.h>
#include <inputRaw.h>
#include <stdlib.h>

//Global variables declaration
unsigned int8 delta_time;
unsigned int8 servomotor = 0;
unsigned int8 times_array[2];
float LF_input_angle;
float LF_max;
float LF_min;
float LF_range;
float LF_time_angle;
float LF_zero;
float RF_input_angle;
float RF_max;
float RF_min;
float RF_range;
float RF_time_angle;
float RF_zero;

//TIMER0 interruption -- outputs to low
#int_TIMER0
void  TIMER0_isr(void)
{
   disable_interrupts(INT_TIMER0);
   clear_interrupt(INT_TIMER0);

   if ( servomotor == 1 )
   {
      output_low(SERVO_L);
      set_timer0(delta_time);

      enable_interrupts(INT_TIMER0);
   }
   else if ( servomotor == 2 )
   {
      output_low(SERVO_R);
      set_timer0(delta_time);

      enable_interrupts(INT_TIMER0);
   }
   else if ( servomotor == 0 )
   {
      output_low(SERVO_L);
      output_low(SERVO_R);
   }
   servomotor = 0;
}
```

```
//TIMER2 interruption -- outputs to high
#int_TIMER2
void  TIMER2_isr(void)
{
   clear_interrupt(INT_TIMER0);

   if ( times_array[0] > ( times_array[1] + 1 ) )
   {
      set_timer0(times_array[0]);
      delta_time = 256 - ( times_array[0] - times_array[1] ) + 1;
      servomotor = 1;
   }
   else if ( ( times_array[0] + 1 ) < times_array[1] )
   {
      set_timer0(times_array[1]);
      delta_time = 256 - ( times_array[1] - times_array[0] ) + 1;
      servomotor = 2;
   }
   else
   {
      set_timer0(times_array[0]);
      servomotor = 0;
   }

   output_high(SERVO_L);
   output_high(SERVO_R);
   enable_interrupts(INT_TIMER0);
}

//Serial port interruption
#int_RDA
void  RDA_isr(void)
{
   putInputRaw(getc());
}

//Function to compute the setpoints
void computeSetpoints()
{
   //Calculating the equivalent time to the desired angle for the left fin
   if ( LF_input_angle <= ( LF_range / 2 ) )
   {
      LF_time_angle = ( ( ( ( LF_input_angle * ( LF_zero - LF_min ) ) /
LF_range ) * 2 ) + LF_min );

      //Avoiding invalid time values
      if ( LF_time_angle < LF_min )
      {
         LF_time_angle = LF_min;
      }
   }
   else // LF_input_angle > ( LF_range / 2 )
   {
      LF_input_angle -= ( LF_range / 2 );
      LF_time_angle = ( ( ( ( LF_input_angle * ( LF_max - LF_zero ) ) /
LF_range ) * 2 ) + LF_zero );

      //Avoiding invalid time values
      if ( LF_time_angle > LF_max )
      {
         LF_time_angle = LF_max;
```

```
      }
   }

   //Horizontal mirror of the desired angle for the right fin according to
the mechanical disposition
   if ( RF_input_angle < RF_range )
   {
      RF_input_angle = RF_range - RF_input_angle;
   }
   else
   {
      RF_input_angle = 0;
   }

   //Calculating the equivalent time to the desired angle for the right fin
   if ( RF_input_angle <= ( RF_range / 2 ) )
   {
      RF_time_angle = ( ( ( ( RF_input_angle * ( RF_zero - RF_min ) ) /
RF_range ) * 2 ) + RF_min );

      //Avoiding invalid time values
      if ( RF_time_angle < RF_min )
      {
         RF_time_angle = RF_min;
      }
   }

   else // RF_input_angle > ( RF_range / 2 )
   {
      RF_input_angle -= ( RF_range / 2 );
      RF_time_angle = ( ( ( ( RF_input_angle * ( RF_max - RF_zero ) ) /
RF_range ) * 2 ) + RF_zero );

      //Avoiding invalid time values
      if ( RF_time_angle > RF_max )
      {
         RF_time_angle = RF_max;
      }
   }
}

//Main function
void main()
{
   //PIC initialising
   //2.96 ms overflow, 11.5625us resolution
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32|RTCC_8_bit);
   //357 us overflow, 3.3 ms interrupt
   setup_timer_2(T2_DIV_BY_4,253,9);
   setup_timer_4(T4_DISABLED,0,1);
   setup_timer_6(T6_DISABLED,0,1);
   setup_comparator(NC_NC_NC_NC);
   disable_interrupts(INT_TIMER0);
   disable_interrupts(INT_TIMER2);
   disable_interrupts(INT_TIMER4);
   disable_interrupts(INT_TIMER6);
   enable_interrupts(INT_RDA);
   enable_interrupts(GLOBAL);

   //Local variables declaration
   int1 enable_ascii = 0;
```

```
int1 water_detected = 0;

unsigned int8 alpha_counter = 0;
unsigned int8 bcc;
unsigned int8 configure_paramater[4];
unsigned int8 high_part;
unsigned int8 low_part;
unsigned int8 new_data;
unsigned int8 number_counter = 0;
unsigned int8 request;
unsigned int8 selected_paramater;
unsigned int8 sent_command;
unsigned int8 state = COMMAND;

unsigned int16 times_water_detected = 0;
unsigned int16 value;

char paramater_value[4];

//External functions call
inputRawInit();
RCSTA = 0x90;

//Recuperation of essential parameters from EEPROM
LF_min = make16(read_eeprom(0),read_eeprom(1));
LF_zero = make16(read_eeprom(2),read_eeprom(3));
LF_max = make16(read_eeprom(4),read_eeprom(5));
LF_range = make16(read_eeprom(6),read_eeprom(7));
RF_min = make16(read_eeprom(8),read_eeprom(9));
RF_zero = make16(read_eeprom(10),read_eeprom(11));
RF_max = make16(read_eeprom(12),read_eeprom(13));
RF_range = make16(read_eeprom(14),read_eeprom(15));

//Configuration of serial port
output_bit(DIRECTION, RECEIVE);

//Main loop
while(TRUE)
{
   //Check water sensor
   if ( times_water_detected < 34695 )
   {
      if ( input(WATER_SENSOR) == 0 )
      {
         times_water_detected += 1;
      }
      else
      {
         times_water_detected = 0;
      }
   }

   //Check serial port
   if (RCSTA & 0x02) // OVERRUN ERROR
   {
      RCSTA = 0x80;  // CLEAR CREN BIT
      RCSTA = 0x90;
   }

   //PC data protocol
   if ( !inputRawEmpty() )
```

```
      {
         new_data = getFromInputRaw();

         switch ( state )
         {
            case COMMAND:
               if ( ( ( new_data == 253 ) || ( new_data == 254 ) ) && (
alpha_counter == 0 ) )
               {
                  bcc = 0;
                  sent_command = new_data;
                  state = ADDRESS;
               }
               else if ( enable_ascii )
               {
                  if ( ( new_data == 'v' ) && ( alpha_counter == 0 ) )
                  {
                     state = CONFIRMATION;
                     break;
                  }
                  else if ( !( isalpha(new_data) ) && ( alpha_counter >= 1
) )
                  {
                     //ERROR
                     alpha_counter = 0;
                     state = COMMAND;
                     break;
                  }
                  else if ( isalpha(new_data) )
                  {
                     configure_paramater[alpha_counter] = new_data;
                     alpha_counter++;

                     if ( alpha_counter == 4 )
                     {
                        if ( ( configure_paramater[0] == 'M' ) && (
configure_paramater[1] == 'i' ) && ( configure_paramater[2] == 'n' ) && (
configure_paramater[3] == 'L' ) )
                        {
                           selected_paramater = 0;
                        }
                        else if ( ( configure_paramater[0] == 'Z' ) && (
configure_paramater[1] == 'e' ) && ( configure_paramater[2] == 'r' ) && (
configure_paramater[3] == 'L' ) )
                        {
                           selected_paramater = 2;
                        }
                        else if ( ( configure_paramater[0] == 'M' ) && (
configure_paramater[1] == 'a' ) && ( configure_paramater[2] == 'x' ) && (
configure_paramater[3] == 'L' ) )
                        {
                           selected_paramater = 4;
                        }
                        else if ( ( configure_paramater[0] == 'R' ) && (
configure_paramater[1] == 'a' ) && ( configure_paramater[2] == 'n' ) && (
configure_paramater[3] == 'L' ) )
                        {
                           selected_paramater = 6;
                        }
```

```
                        else if ( ( configure_paramater[0] == 'M' ) && ( configure_paramater[1] == 'i' ) && ( configure_paramater[2] == 'n' ) && ( configure_paramater[3] == 'R' ) )
                        {
                           selected_paramater = 8;
                        }
                        else if ( ( configure_paramater[0] == 'Z' ) && ( configure_paramater[1] == 'e' ) && ( configure_paramater[2] == 'r' ) && ( configure_paramater[3] == 'R' ) )
                        {
                           selected_paramater = 10;
                        }
                        else if ( ( configure_paramater[0] == 'M' ) && ( configure_paramater[1] == 'a' ) && ( configure_paramater[2] == 'x' ) && ( configure_paramater[3] == 'R' ) )
                        {
                           selected_paramater = 12;
                        }
                        else if ( ( configure_paramater[0] == 'R' ) && ( configure_paramater[1] == 'a' ) && ( configure_paramater[2] == 'n' ) && ( configure_paramater[3] == 'R' ) )
                        {
                           selected_paramater = 14;
                        }
                        else
                        {
                           //ERROR
                           alpha_counter = 0;
                           state = COMMAND;
                           break;
                        }
                        alpha_counter = 0;
                        number_counter = 0;
                        state = CONFIGURATION;
                     }
                  }
                  else
                  {
                     state = COMMAND;
                  }
               }
               else
               {
                  state = COMMAND;
               }
            break;

            case ADDRESS:
               if ( new_data == ADDRESS_ID )
               {
                  bcc = ADDRESS_ID;
                  state = PARAMETER1;
               }
               else
               {
                  state = COMMAND;
               }
            break;

            case PARAMETER1:
               if ( sent_command == 253 )
```

```
                {
                   LF_input_angle  =  (  new_data  /  2  );        //  Double
resolution
                   bcc ^= new_data;
                   state = PARAMETER2;
                }
                else if ( sent_command == 254 )
                {
                   request = new_data;
                   state = PARAMETER2;
                }
                else
                {
                   state = COMMAND;
                }
             break;

             case PARAMETER2:
                if ( sent_command == 253 )
                {
                   RF_input_angle = ( new_data / 2 );   // Double resolution
                   bcc ^= new_data;
                   state = CONTROL_BIT;
                }
                else if ( ( sent_command == 254 ) && ( request == new_data )
)
                {
                   switch ( request )
                   {
                      case 0:
                         disable_interrupts(INT_TIMER0);
                         disable_interrupts(INT_TIMER2);
                      break;

                      case 1:
                         //Arranging the table with the desired position
                         times_array[0]  =  (int8)(  256  -  (int16)(  LF_zero  /
11.5625 ) );
                         times_array[1]  =  (int8)(  256  -  (int16)(  RF_zero  /
11.5625 ) );

                         enable_interrupts(INT_TIMER2);

                      break;

                      case 2:
                         if  (  times_water_detected  ==  34695  )  //  34695
iteration is aprox 450 ms
                         {
                            water_detected = 1;

                            // Once answered the request -> reset
                            times_water_detected = 0;
                         }
                         else
                         {
                            water_detected = 0;
                         }

                         output_bit(DIRECTION, TRANSMIT);
                         printf("%c",254);
```

```
                        printf("%c",ADDRESS_ID);
                        printf("%c",2);
                        printf("%c",water_detected);
                        printf("%c",water_detected);
                        printf("%c",ADDRESS_ID ^ 2);
                        delay_us(348);
                        output_bit(DIRECTION, RECEIVE);
                     break;

                     case 10:
                        if ( enable_ascii == 1 )
                        {
                           enable_ascii = 0;

                           output_bit(DIRECTION, TRANSMIT);
                           printf("Config. disabled\n");
                           delay_us(348);
                           output_bit(DIRECTION, RECEIVE);
                        }
                        else if ( enable_ascii == 0 )
                        {
                           enable_ascii = 1;

                           output_bit(DIRECTION, TRANSMIT);
                           printf("Config. enabled\n");
                           delay_us(348);
                           output_bit(DIRECTION, RECEIVE);
                        }
                     break;

                     default:
                        state = COMMAND;
                     break;
                  }
                  state = COMMAND;
               }
               else
               {
                  state = COMMAND;
               }
            break;

            case CONTROL_BIT:
               if ( ( bcc == new_data ) && ( sent_command == 253 ) )
               {
                  //
                  computeSetpoints();

                  //Arranging the table with the desired position
                  times_array[0] = (int8)( 256 - (int16)( LF_time_angle /
11.5625 ) );
                  times_array[1] = (int8)( 256 - (int16)( RF_time_angle /
11.5625 ) );
               }
               state = COMMAND;
            break;

            case CONFIRMATION:
               if ( new_data == '\r' )
               {
                  output_bit(DIRECTION, TRANSMIT);
```

```
            printf("%s\n",FIRMWARE_VERSION);
            delay_us(348);
            output_bit(DIRECTION, RECEIVE);
            state = COMMAND;
         }
         else
         {
            //ERROR
            output_bit(DIRECTION, TRANSMIT);
            printf("E\n");
            delay_us(348);
            output_bit(DIRECTION, RECEIVE);
            state = COMMAND;
         }
      break;

      case CONFIGURATION:
         if ( new_data != '\r' )
         {
            if ( isdigit(new_data) && (number_counter < 4) )
            {
               paramater_value[number_counter] = new_data;
               number_counter++;
            }
            else
            {
               //ERROR
               output_bit(DIRECTION, TRANSMIT);
               printf("E\n");
               delay_us(348);
               output_bit(DIRECTION, RECEIVE);
               number_counter = 0;
               state = COMMAND;
            }
            if ( ( state == CONFIGURATION ) && ( number_counter < 4 )
)
            {
               paramater_value[number_counter+1] = '\0';
            }
         }
         else
         {
            if ( number_counter != 0 )
            {
               value = atol(paramater_value);
               high_part = make8(value,1);
               low_part = make8(value,0);

               switch ( selected_paramater )
               {
                  case MINL:
                     LF_min = value;
                  break;

                  case ZERL:
                     LF_zero = value;
                  break;

                  case MAXL:
                     LF_max = value;
                  break;
```

```
                    case RANL:
                       LF_range = value;
                    break;

                    case MINR:
                       RF_min = value;
                    break;

                    case ZERR:
                       RF_zero = value;
                    break;

                    case MAXR:
                       RF_max = value;
                    break;

                    case RANR:
                       RF_range = value;
                    break;
                 }
                 write_eeprom(selected_paramater,high_part);
                 write_eeprom(selected_paramater+1,low_part);

                 output_bit(DIRECTION, TRANSMIT);
                 printf("OK\n");
                 delay_us(348);
                 output_bit(DIRECTION, RECEIVE);

                 state = COMMAND;
              }
              else
              {
                 //ERROR
                 output_bit(DIRECTION, TRANSMIT);
                 printf("E\n");
                 delay_us(348);
                 output_bit(DIRECTION, RECEIVE);
                 state = COMMAND;
              }
              number_counter = 0;
           }
        break;

        default:
           state = COMMAND;
        break;
     }
   }
  }
}
```

D. PROGRAMACIÓ DEL DRIVER

D.1. Fitxers de configuració

D.1.1. Configuració port sèrie

```
actuators_485/sp_path: "/dev/ttyS7"
actuators_485/sp_baud_rate: 57600
actuators_485/sp_char_size: 8
actuators_485/sp_stop_bits: 1
actuators_485/sp_parity: "NONE"
actuators_485/sp_flow_control: "NONE"
```

D.1.2. Configuració actuadors

```
# Common parameters
thrusters/request_info: True
thrusters/max_step: 0.1
thrusters/symmetric: True

# vertical, right (estribord), left (babord)
thrusters/addresses: [164, 10, 80]
thrusters/signs: [1, -1, 1]
thrusters/limiter: [0.6, 1.0, 1.0]

fins/address: 116
```

D.2. Programa principal

```
#include <ros/ros.h>
#include "cola2_lib/cola2_rosutils/DiagnosticHelper.h"
#include <cola2_lib/cola2_io/SerialPort.h>
#include <cola2_control/Setpoints.h>
#include <cola2_control/ThrustersInfo.h>
#include <std_srvs/Empty.h>

#include <iostream>
#include <stdint.h>
#include <string>
#include <signal.h>
#include <cmath>
#include <sstream>
#include <iomanip>
#include <boost/bind.hpp>
#include <boost/date_time.hpp>
#include <boost/thread.hpp>

// Serial port
cola2::io::SerialPort serial_port ;

class FINS_485
```

```
{
private:
    // Var. used to store the name of the node
    std::string _node_name ;

    // Serial port mutex
    boost::mutex serial_mutex ;

public:
    FINS_485( std::string node_name )
    {
        _node_name = node_name ;
    } ;

    ~FINS_485()
    {
    } ;

    void disableFins(unsigned char address)
    {
        std::vector<unsigned char> fins_command ;
        fins_command.push_back(0xfe) ;
        fins_command.push_back(address) ;
        fins_command.push_back(0x00) ;
        fins_command.push_back(0x00) ;

        // Send
        try
        {
            // Mutex
            boost::mutex::scoped_lock serial_lock(serial_mutex) ;

            serial_port.write(fins_command, _node_name, 100.0) ;
        }
        catch (std::exception& e)
        {
            std::cerr << _node_name << ": unable to send command: " <<
e.what() << std::endl ;
        }
    }

    void enableFins(unsigned char address)
    {
        std::vector<unsigned char> fins_command ;
        fins_command.push_back(0xfe) ;
        fins_command.push_back(address) ;
        fins_command.push_back(0x01) ;
        fins_command.push_back(0x01) ;

        // Send
        try
        {
            // Mutex
            boost::mutex::scoped_lock serial_lock(serial_mutex) ;

            serial_port.write(fins_command, _node_name, 100.0) ;
        }
        catch (std::exception& e)
        {
            std::cerr << _node_name << ": unable to send command: " <<
e.what() << std::endl ;
```

```
        }
    }

    bool computeFinsSetpoint(unsigned char address, unsigned char left,
unsigned char right)
    {
        std::vector<unsigned char> fins_command ;
        fins_command.push_back(0xfd) ;
        fins_command.push_back(address) ;
        fins_command.push_back(left) ;
        fins_command.push_back(right) ;
        fins_command.push_back(address xor left xor right) ;

        // Send
        try
        {
            // Mutex
            boost::mutex::scoped_lock serial_lock(serial_mutex) ;

            serial_port.write(fins_command, _node_name, 100.0) ;
        }
        catch (std::exception& e)
        {
            std::cerr << _node_name << ": unable to send command: " <<
e.what() << std::endl ;
        }
    }

    //
    bool waterRequest(unsigned char address)
    {
        std::vector<unsigned char> fins_command ;
        fins_command.push_back(0xfe) ;
        fins_command.push_back(address) ;
        fins_command.push_back(0x02) ;
        fins_command.push_back(0x02) ;

        bool answer = false ;

        try
        {
            // Mutex
            boost::mutex::scoped_lock serial_lock(serial_mutex) ;

            serial_port.write(fins_command, _node_name, 100.0) ;

            std::vector<unsigned char> line ;

            // Compute request
            line.resize(0) ;
            unsigned char nextChar = 0 ;
            int iter ;
            for (iter = 0 ; iter < 4 ; iter++)
            {
                nextChar = serial_port.readByte(100) ;
                line.push_back(nextChar) ;
            }

            // Check protocol
            if (line.size() == 6 and line[1] == address and line[2] == 0x02
and line[3] == line[4] and line[3] == 1 and line[5] == address xor 0x02)
```

```
            {
                answer = true ;
            }
        }
        catch (std::exception& e)
        {
            std::cerr << _node_name << ": unable to ask for water: " <<
e.what() << std::endl ;
        }
        return answer ;
    }
} ;

class THRUSTERS_485
{
private:
    // Var. used to store the name of the node
    std::string _node_name ;

    // Serial port mutex
    boost::mutex serial_mutex ;

    // Var. used to store old setpoints
    std::vector<double> old_setpoints ;

    // Var. used to alternate requests
    unsigned int request ;

    // Var. used to alternate thrusters
    unsigned int alternate ;

public:

    THRUSTERS_485( std::string node_name )
    {
        _node_name = node_name ;

        request = 0 ;

        alternate = 0 ;
    } ;
    ~THRUSTERS_485()
    {
    } ;

    struct ThrusterInfo
    {
        std::vector<double> thruster_voltages ;
        std::vector<double> thruster_currents ;
        std::vector<double> thruster_rpms ;
        std::vector<double> thruster_controller_temperatures ;
        std::vector<double> thruster_output_powers ;
        std::vector<std::string> thruster_warnings ;
        std::vector<std::string> thruster_errors ;
        double total_ms_in_driver ;
        std::vector<double> setpoints ;
        std::vector<double> processed_setpoints ;
    } ;
    ThrusterInfo thrusters_info;

    void init(std::vector<unsigned char> thrusters_addresses)
```

```
    {
        // Sending something to init thrusters
        for (int i = 0 ; i < thrusters_addresses.size() ; i++)
        {
            computeSetpoint(thrusters_addresses[i], 0.01) ;
        }

        // Vars used to check thrusters
        std::vector<int> zero_setpoint_counter ;
        std::vector<int> bad_current_counter ;
        int i ;
        for (i = 0 ; i < thrusters_addresses.size() ; i++)
        {
            zero_setpoint_counter.push_back(0) ;
            bad_current_counter.push_back(0) ;
            old_setpoints.push_back(0.0) ;

            thrusters_info.thruster_voltages.push_back(0.0) ;
            thrusters_info.thruster_currents.push_back(0.0) ;
            thrusters_info.thruster_rpms.push_back(0.0) ;
            thrusters_info.thruster_controller_temperatures.push_back(0.0)
;
            thrusters_info.thruster_output_powers.push_back(0.0) ;
            thrusters_info.thruster_warnings.push_back("Not  received  yet")
;
            thrusters_info.thruster_errors.push_back("Not received yet") ;
        }
    }

    void sendZeros(std::vector<unsigned char> thrusters_addresses)
    {
        std::cout << _node_name << ": sending zero" << std::endl ;

        for (int i = 0 ; i < thrusters_addresses.size() ; i++)
        {
            computeSetpoint(thrusters_addresses[i], 0.0) ;
        }
    }

    bool computeSetpoint(unsigned char address, double double_setpoint)
    {
        try
        {
            std::vector<unsigned char> thruster_command ;
            unsigned int int_setpoint ;
            uint16_t converted_setpoint ;
            uint8_t low ;
            uint8_t high ;

            if (double_setpoint < -1.0)
            {
                double_setpoint = -1.0 ;
            }

            if (double_setpoint > 1.0)
            {
                double_setpoint = 1.0 ;
            }

            int_setpoint = round(1024 * double_setpoint * 0.65) ;
```

```
            if (int_setpoint < 0)
            {
                converted_setpoint = ~((uint16_t)(-int_setpoint)) + 0b1 ;
            }
            else
            {
                converted_setpoint = (uint16_t)int_setpoint ;
            }

            low = (uint8_t)((converted_setpoint & 0b0000000011111111) >> 0)
;
            high = (uint8_t)((converted_setpoint & 0b1111111100000000) >>
8) ;

            thruster_command.push_back(0xfd) ;
            thruster_command.push_back(address) ;
            thruster_command.push_back(low) ;
            thruster_command.push_back(high) ;
            thruster_command.push_back(address xor low xor high) ;

            // Scope for the mutex
            {
                // Mutex
                boost::mutex::scoped_lock serial_lock(serial_mutex) ;

                // Send
                try
                {
                    serial_port.write(thruster_command, _node_name, 100.0)
;
                }
                catch (std::exception& e)
                {
                    std::cerr << _node_name << ": unable to send command: "
<< e.what() << std::endl ;
                }

                // Sleep some time

boost::this_thread::sleep(boost::posix_time::microseconds(1000));

                // Empty serial port buffer, since some drivers return ff
byte after receiving command
                bool empty = false ;
                int number_of_bytes_after_setpoint = 0 ;

                while (!empty)
                {
                    try
                    {
                        serial_port.readByte(1) ;
                        number_of_bytes_after_setpoint += 1 ;
                    }
                    catch (std::exception&)
                    {
                        empty = true ;
                    }
                }
            }
            return true ;
        }
```

```
        catch (std::exception& e)
        {
            std::cout << _node_name << ": exception while computing
setpoint: " << e.what() << std::endl ;
            return false ;
        }
    }

    bool serialComputeRequest(std::vector<unsigned char>& line, unsigned
char address, unsigned char request)
    {
        try
        {
            boost::mutex::scoped_lock serial_lock(serial_mutex) ;

            // Empty serial port buffer
            bool empty = false ;
            while (!empty)
            {
                try
                {
                    serial_port.readByte(1) ;
                }
                catch (std::exception&)
                {
                    empty = true ;
                }
            }

            // Send request
            std::vector<unsigned char> thruster_command ;
            thruster_command.push_back(0xfe) ;
            thruster_command.push_back(address) ;
            thruster_command.push_back(request) ;
            thruster_command.push_back(request) ;

            // Send
            try
            {
                serial_port.write(thruster_command, _node_name, 100.0) ;
            }
            catch (std::exception& e)
            {
                std::cerr << _node_name << ": unable to send command: " <<
e.what() << std::endl ;
            }

            // Compute request
            bool line_ok = false ;
            line.resize(0) ;
            unsigned char nextChar = 0 ;
            int iter ;
            for (iter = 0 ; iter < 6 ; iter++)
            {
                nextChar = serial_port.readByte(100) ;
                line.push_back(nextChar) ;
            }

            if (line.size() == 6)
            {
                // Check checksum, request and address
```

```
                if ((line[5] == (line[1] xor line[2] xor line[3] xor
line[4])) && (line[2] == request) && (line[1] == address))
                {
                    line_ok = true ;
                }
            }

            return line_ok ;
        }
        catch (std::exception& e)
        {
            std::cout << _node_name << ": exception while computing
request: " << e.what() << std::endl ;
            return false ;
        }
    }

    bool computeVoltageRequest(unsigned char address, double& parameter)
    {
        try
        {
            std::vector<unsigned char> line ;

            if (serialComputeRequest(line, address, 0x01))
            {
                parameter = 0.1 * (double)(line[3] + 256 * line[4]) ;
                return true ;
            }
            else
            {
                return false ;
            }
        }
        catch (std::exception&)
        {
            return false ;
        }
    }

    bool computeCurrentRequest(unsigned char address, double& parameter)
    {
        try
        {
            std::vector<unsigned char> line ;

            if (serialComputeRequest(line, address, 0x02))
            {
                parameter = 0.1 * (double)checkTwosComplement(line[3],
line[4]) ;
                return true ;
            }
            else
            {
                return false ;
            }
        }
        catch (std::exception&)
        {
            return false ;
        }
    }
```

```
    bool computeRpmRequest(unsigned char address, double& parameter)
    {
        try
        {
            std::vector<unsigned char> line ;

            if (serialComputeRequest(line, address, 0x03))
            {
                parameter = 10.0 * (double)(line[3] + 256 * line[4]) ;
                return true ;
            }
            else
            {
                return false ;
            }
        }
        catch (std::exception&)
        {
            return false ;
        }
    }

    bool    computeTemperatureRequest(unsigned    char    address,    double&
parameter)
    {
        try {
            std::vector<unsigned char> line ;

            if (serialComputeRequest(line, address, 0x04))
            {
                parameter = (double)checkTwosComplement(line[3], line[4]) ;
                return true ;
            }
            else
            {
                return false ;
            }
        }
        catch (std::exception&)
        {
            return false ;
        }
    }

    bool    computeOutputPowerRequest(unsigned    char    address,    double&
parameter)
    {
        try
        {
            std::vector<unsigned char> line ;

            if (serialComputeRequest(line, address, 0x09))
            {
                parameter = (double)checkTwosComplement(line[3], line[4]) ;
                return true ;
            }
            else
            {
                return false ;
            }
```

```
        }
        catch (std::exception&)
        {
            return false ;
        }
    }

    bool   computeWarningRequest(unsigned   char   address,   std::string&
parameter)
    {
        try
        {
            std::vector<unsigned char> line ;
            bool valid = false ;

            if (serialComputeRequest(line, address, 0x0a))
            {
                std::stringstream message ;

                if (line[3] == 0)
                {
                    message<<"All OK. " ;
                    valid = true ;
                }
                if ((line[3] & 0x01) == 0x01)
                {
                    message<<"Low voltage. " ;
                    std::cerr << _node_name << ": warning request -> Low
voltage" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x02) == 0x02)
                {
                    message<<"High current. " ;
                    std::cerr << _node_name << ": warning request -> High
current" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x04) == 0x04)
                {
                    message<<"High controller temperature. " ;
                    std::cerr << _node_name << ": warning request -> High
controller temperature" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x08) == 0x08)
                {
                    message<<"High battery temperature. " ;
                    std::cerr << _node_name << ": warning request -> High
battery temperature" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x10) == 0x10)
                {
                    message<<"High motor temperature. " ;
                    std::cerr << _node_name << ": warning request -> High
motor temperature" << std::endl ;
                    valid = true ;
                }

                if (!valid)
```

```
                {
                    message<<"??? " ;
                    std::cerr << _node_name << ": warning request returns
invalid response" << std::endl ;
                }

                parameter = message.str().erase(message.str().size() - 1) ;
                return true ;
            }
            else
            {
                return false ;
            }
        }
        catch (std::exception&)
        {
            return false ;
        }
    }


    bool computeErrorRequest(unsigned char address, std::string& parameter)
    {
        try
        {
            std::vector<unsigned char> line ;
            bool valid = false ;

            if (serialComputeRequest(line, address, 0x0b))
            {
                std::stringstream message ;

                if (line[3] == 0)
                {
                    message<<"All OK. " ;
                    valid = true ;
                }
                if ((line[3] & 0x01) == 0x01)
                {
                    message<<"Signal lost. " ;
                    std::cerr << _node_name << ": error request -> Signal
lost" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x02) == 0x02)
                {
                    message<<"Waiting for neutral. " ;
                    std::cerr << _node_name << ": error request -> Waiting
for neutral" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x04) == 0x04)
                {
                    message<<"Settings changed (PC). " ;
                    std::cerr << _node_name << ": error request -> Settings
changed (PC)" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x08) == 0x08)
                {
                    message<<"Memory failed. " ;
```

```
                    std::cerr << _node_name << ": error request -> Memory
failed" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x10) == 0x10)
                {
                    message<<"Settings changed (ESC). " ;
                    std::cerr << _node_name << ": error request -> Settings
changed (ESC)" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x20) == 0x20)
                {
                    message<<"Motor hall sensor error. " ;
                    std::cerr << _node_name << ": error request -> Motor
hall sensor error" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x40) == 0x40)
                {
                    message<<"Internal voltage error. " ;
                    std::cerr << _node_name << ": error request -> Internal
voltage error" << std::endl ;
                    valid = true ;
                }
                if ((line[3] & 0x80) == 0x80)
                {
                    message<<"Motor hall learning OK. " ;
                    std::cerr << _node_name << ": error request -> Motor
hall learning OK" << std::endl ;
                    valid = true ;
                }
                if ((line[4] & 0x01) == 0x01)
                {
                    message<<"Battery temperature sensor. " ;
                    std::cerr << _node_name << ": error request -> Battery
temperature sensor" << std::endl ;
                    valid = true ;
                }
                if ((line[4] & 0x02) == 0x02)
                {
                    message<<"Motor temperature sensor. " ;
                    std::cerr << _node_name << ": error request -> Motor
temperature sensor" << std::endl ;
                    valid = true ;
                }

                if (!valid)
                {
                    message<<"??? " ;
                    std::cerr << _node_name << ": error request returns
invalid response" << std::endl ;
                }

                // Remove last space
                parameter = message.str().erase(message.str().size() - 1) ;
                return true ;
            }
            else
            {
                return false ;
```

```
            }
        }
        catch (std::exception&)
        {
            return false ;
        }
    }

    int checkTwosComplement(const unsigned char& low, const unsigned char&
high)
    {
        uint16_t both = 0 ;
        if ((high>>7) == 1)
        {
            both = both | (uint16_t)high ;
            both = both<<8 ;
            both = both | (uint16_t)low ;
            return -(int)(both xor 0xffff) - 1 ;
        }
        return low + high * 256 ;
    }

    void thrusterSetpointCallback(std::vector<double> thrusters_setpoints,
std::vector<unsigned char> thrusters_addresses, std::vector<double>
thrusters_signs, std::vector<double> thrusters_limiter, bool
thrusters_symmetric, double thrusters_max_step, bool
thrusters_request_info)
    {
        // Get the initial time
        boost::posix_time::ptime previousTime =
boost::posix_time::microsec_clock::universal_time() ;

        // Compute thrusters
        if (thrusters_addresses.size() == thrusters_setpoints.size())
        {
            std::vector<double> setpoints ;
            setpoints.resize(thrusters_addresses.size()) ;

            // Compute each thruster
            int i ;
            for (i = 0 ; i < thrusters_addresses.size() ; i++)
            {
                // Compute signs
                setpoints[i] = (double)thrusters_setpoints[i] *
thrusters_signs[i] ;

                // Limiter
                if (setpoints[i] > thrusters_limiter[i])
                {
                    setpoints[i] = thrusters_limiter[i] ;
                    std::cout << _node_name << ": limiting thruster " << i
<< std::endl ;
                }
                else if (setpoints[i] < -thrusters_limiter[i])
                {
                    setpoints[i] = -thrusters_limiter[i] ;
                    std::cout << _node_name << ": limiting thruster " << i
<< std::endl ;
                }

                // Derivative filter
```

```
                if (thrusters_symmetric)
                {
                    // Filter in both directions
                    if      (setpoints[i]      >      old_setpoints[i]      +
thrusters_max_step)
                    {
                        setpoints[i]       =       old_setpoints[i]       +
thrusters_max_step ;
                    }
                    else    if    (setpoints[i]    <    old_setpoints[i]    -
thrusters_max_step)
                    {
                        setpoints[i]       =       old_setpoints[i]       -
thrusters_max_step ;
                    }
                }
                else
                {
                    // Non symmetric. Filter only when increasing setpoint
                    if      ((setpoints[i]      >      old_setpoints[i]      +
thrusters_max_step) && (setpoints[i] > thrusters_max_step))
                    {
                        if   ((old_setpoints[i]   +   thrusters_max_step)   >
thrusters_max_step)
                        {
                            setpoints[i]      =      old_setpoints[i]      +
thrusters_max_step ;
                        }
                        else
                        {
                            setpoints[i] = thrusters_max_step ;
                        }
                    }
                    else    if    ((setpoints[i]    <    old_setpoints[i]    -
thrusters_max_step) && (setpoints[i] < -thrusters_max_step))
                    {
                        if  ((old_setpoints[i]  -  thrusters_max_step)  <  -
thrusters_max_step)
                        {
                            setpoints[i]      =      old_setpoints[i]      -
thrusters_max_step ;
                        }
                        else
                        {
                            setpoints[i] = -thrusters_max_step ;
                        }
                    }
                }

                // Store old setpoints
                old_setpoints[i] = setpoints[i] ;

                // Send setpoints
                if (!computeSetpoint(thrusters_addresses[i], setpoints[i]))
                {
                    std::cerr << _node_name << ": unable to send setpoint,
thruster " << i << std::endl ;
                }

                // Request info
                if (thrusters_request_info)
```

```
                {
                    // Current request at 10 Hz
                    if      (!computeCurrentRequest(thrusters_addresses[i],
thrusters_info.thruster_currents[i]))
                    {
                        std::cout  <<  _node_name  <<  ":  unable  to  compute
current request, thruster " << i << std::endl ;
                    }

                    // RPM request at 10 Hz
                    if          (!computeRpmRequest(thrusters_addresses[i],
thrusters_info.thruster_rpms[i]))
                    {
                        std::cout << _node_name << ": unable to compute rpm
request, thruster " << i << std::endl ;
                    }

                    // Request each 1.5 seconds
                    if (alternate == i)
                    {
                        switch (request)
                        {
                            case 0:
                                if
(!computeVoltageRequest(thrusters_addresses[i],
thrusters_info.thruster_voltages[i]))
                                {
                                    std::cout << _node_name << ": unable to
compute voltage request, thruster " << i << std::endl ;
                                }
                            break;

                            case 1:
                                // Save some time here...
                                thrusters_info.thruster_output_powers[i]   =
std::abs(thrusters_info.thruster_voltages[i]                              *
thrusters_info.thruster_currents[i]) ;
                                //if
(!computeOutputPowerRequest(thrusters_addresses[i],
thrusters_info.thruster_output_powers[i]))
                                //{
                                //     std::cout << _node_name << ": unable
to compute output power request, thruster " << i << std::endl ;
                                //}
                            break;

                            case 2:
                                if
(!computeWarningRequest(thrusters_addresses[i],
thrusters_info.thruster_warnings[i]))
                                {
                                    std::cout << _node_name << ": unable to
compute warning request, thruster " << i << std::endl ;
                                }
                            break;

                            case 3:
                                if
(!computeErrorRequest(thrusters_addresses[i],
thrusters_info.thruster_errors[i]))
                                {
```

```
                                    std::cout << _node_name << ": unable to
compute error request, thruster " << i << std::endl ;
                                }
                            break;

                            case 4:
                                if
(!computeTemperatureRequest(thrusters_addresses[i],
thrusters_info.thruster_controller_temperatures[i]))
                                {
                                    std::cout << _node_name << ": unable to
compute controller temperature request, controller " << i << std::endl ;
                                }
                            break;

                            default:
                                request = 4 ;
                            break;
                        }

                        if (request != 4 )
                        {
                            request += 1 ;
                        }
                        else
                        {
                            request = 0 ;

                            // Change alternate var.
                            alternate += 1 ;
                            if (alternate == thrusters_addresses.size())
                            {
                                alternate = 0 ;
                            }
                        }
                    }
                }
            }

            // Compute time
            double dt = 0.001 * boost::posix_time::time_period(
                        previousTime,
boost::posix_time::microsec_clock::universal_time()).length().total_microse
conds() ;
            if (dt > 100.0)
            {
                std::cout << _node_name << ": slow serial port waiting for
requests. Total elapsed time: " << dt << "ms" << std::endl ;
            }

            thrusters_info.total_ms_in_driver = dt ;
            thrusters_info.setpoints = thrusters_setpoints ;
            thrusters_info.processed_setpoints = setpoints ;
        }
        else
        {
            std::cerr << _node_name << ": invalid setpoints lenght" <<
std::endl ;
        }
    }
} ;
```

```
class ACTUATORS_ROS
{
private:
    // Node handle
    ros::NodeHandle n ;

    // Timer
    ros::Timer timer ;
    ros::Timer timer_water ;

    // Diagnostics
    cola2::rosutils::DiagnosticHelper diagnostic ;

    // Fins subscriber
    ros::Subscriber fins_sub ;

    // Thruster subscriber
    ros::Subscriber thruster_sub ;

    // Thruster publisher
    ros::Publisher thruster_info_pub ;

    // Message type to publish
    cola2_control::ThrustersInfo thrusters_info ;

    // Pointer to FINS_485 class
    FINS_485 * _fins_ptr ;

    // Pointer to THRUSTERS_485 class
    THRUSTERS_485 * _thrusters_ptr ;

    // Var. used to store the name of the node
    std::string _node_name ;

    // Var. used to store times
    boost::posix_time::ptime lastSetpointCallback ;

    // Var. used to store the state of the fins
    int _fins_enabled ;

    // Var. used to store water detection stat
    bool _water ;

    // Var. used to avoid false water detections
    int _water_offset ;

    // Var. used to store fins_address
    unsigned char _fins_address ;

    // Var. used to store old fins setpoints
    std::vector<double> _old_fins_setpoints ;

    // Device configuration
    struct DevConfig
    {
        std::string sp_path ;
        int sp_baud_rate ;
        int sp_char_size ;
        int sp_stop_bits ;
        std::string sp_parity ;
```

```
        std::string sp_flow_control ;

        bool request_info ;
        double max_step ;
        bool symmetric ;
        std::vector<unsigned char> addresses ;
        std::vector<double> signs ;
        std::vector<double> limiter ;
    } ;
    DevConfig device_config ;

public:

    ACTUATORS_ROS(   std::string   node_name,   FINS_485   *fins_485_ptr,
THRUSTERS_485 *thrusters_485_ptr ):
        diagnostic(n, node_name, "none"),
        _water(false),
        _water_offset(0),
        _old_fins_setpoints(2)
    {
        _node_name = node_name ;

        _fins_ptr = fins_485_ptr ;
        _thrusters_ptr = thrusters_485_ptr ;
    } ;

    ~ACTUATORS_ROS()
    {
        delete _fins_ptr ;
        delete _thrusters_ptr ;
    } ;

    void init()
    {
        ACTUATORS_ROS::getConfig() ;
        ACTUATORS_ROS::openSerialPort() ;

        // Enable fins with actuators driver start up
        _fins_ptr->enableFins(_fins_address) ;

        // Config to thrusters
        _thrusters_ptr->init(device_config.addresses) ;

        // Fins subscriber
        fins_sub     =     n.subscribe("/cola2_control/fins_data",     3,
&ACTUATORS_ROS::finsSetpointCallback, this) ;

        // Thruster subscriber
        thruster_sub   =   n.subscribe("/cola2_control/thrusters_data",   3,
&ACTUATORS_ROS::thrustersSetpointCallback, this) ;

        // Thruster publisher
        thruster_info_pub                                                  =
n.advertise<cola2_control::ThrustersInfo>("/cola2_control/thrusters_info" ,
3) ;

        // Timer
        lastSetpointCallback                                               =
boost::posix_time::microsec_clock::universal_time() ;
        timer              =              n.createTimer(ros::Duration(0.1),
&ACTUATORS_ROS::sendZeroTimerCallback, this) ;
```

```
        // Water timer
        timer_water           =           n.createTimer(ros::Duration(3),
&ACTUATORS_ROS::sendWaterRequestTimerCallback, this) ;

        unsigned int i = 0 ;
    }

    void getConfig()
    {
        bool valid_config = true ;

        // Get params
        if                 (!ros::param::getCached("actuators_485/sp_path",
device_config.sp_path))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
actuators_485/sp_path in param server!") ;
            valid_config = false ;
        }
        if            (!ros::param::getCached("actuators_485/sp_baud_rate",
device_config.sp_baud_rate))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
actuators_485/sp_baud_rate in param server!") ;
            valid_config = false ;
        }
        if            (!ros::param::getCached("actuators_485/sp_char_size",
device_config.sp_char_size))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
actuators_485/sp_char_size in param server!") ;
            valid_config = false ;
        }
        if            (!ros::param::getCached("actuators_485/sp_stop_bits",
device_config.sp_stop_bits))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
actuators_485/sp_stop_bits in param server!") ;
            valid_config = false ;
        }
        if               (!ros::param::getCached("actuators_485/sp_parity",
device_config.sp_parity))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
actuators_485/sp_parity in param server!") ;
            valid_config = false ;
        }
        if         (!ros::param::getCached("actuators_485/sp_flow_control",
device_config.sp_flow_control))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
actuators_485/sp_flow_control in param server!") ;
            valid_config = false ;
        }
        if                (!ros::param::getCached("thrusters/request_info",
device_config.request_info))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
thrusters/request_info in param server!") ;
            valid_config = false ;
```

```
        }
        if                    (!ros::param::getCached("thrusters/max_step",
device_config.max_step))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
thrusters/max_step in param server!") ;
            valid_config = false ;
        }
        if                   (!ros::param::getCached("thrusters/symmetric",
device_config.symmetric))
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  invalid  parameters  for
thrusters/symmetric in param server!") ;
            valid_config = false ;
        }

        // Addresses list
        try
        {
            XmlRpc::XmlRpcValue my_list_a ;
            if(ros::param::getCached("thrusters/addresses", my_list_a))
            {
                ROS_ASSERT(my_list_a.getType()                           ==
XmlRpc::XmlRpcValue::TypeArray) ;

                for (int32_t i = 0; i < my_list_a.size(); ++i)
                {
                    ROS_ASSERT(my_list_a[i].getType()                    ==
XmlRpc::XmlRpcValue::TypeInt) ;
                    device_config.addresses.push_back(static_cast<unsigned
char>(static_cast<int>(my_list_a[i]))) ;
                }
            }
            else
            {
                ROS_FATAL_STREAM(_node_name  <<  ": invalid parameters  for
thrusters/addresses in param server!") ;
                valid_config = false ;
            }
        }
        catch (std::exception& e)
        {
            ROS_FATAL_STREAM(_node_name  <<  ":  exception  when  loading
thrusters/addresses: " << e.what()) ;
            valid_config = false ;
        }

        // Signs list
        try
        {
            XmlRpc::XmlRpcValue my_list_b ;
            if(ros::param::getCached("thrusters/signs", my_list_b))
            {
                ROS_ASSERT(my_list_b.getType()                           ==
XmlRpc::XmlRpcValue::TypeArray) ;

                for (int32_t i = 0; i < my_list_b.size(); ++i)
                {
                    ROS_ASSERT(my_list_b[i].getType()                    ==
XmlRpc::XmlRpcValue::TypeInt) ;
```

```
device_config.signs.push_back(static_cast<double>(static_cast<int>(my_list_
b[i]))) ;
                }
            }
            else
            {
                ROS_FATAL_STREAM(_node_name << ": invalid parameters for
thrusters/signs in param server!") ;
                valid_config = false ;
            }
        }
        catch (std::exception& e)
        {
            ROS_FATAL_STREAM(_node_name << ": exception when loading
thrusters/signs: " << e.what()) ;
            valid_config = false ;
        }

        // Limiter list
        try
        {
            XmlRpc::XmlRpcValue my_list_c ;
            if(ros::param::getCached("thrusters/limiter", my_list_c))
            {
                ROS_ASSERT(my_list_c.getType() ==
XmlRpc::XmlRpcValue::TypeArray) ;

                for (int32_t i = 0; i < my_list_c.size(); ++i)
                {
                    ROS_ASSERT(my_list_c[i].getType() ==
XmlRpc::XmlRpcValue::TypeDouble) ;

device_config.limiter.push_back(static_cast<double>(my_list_c[i])) ;
                }
            }
            else
            {
                ROS_FATAL_STREAM(_node_name << ": invalid parameters for
thrusters/limiter in param server!") ;
                valid_config = false ;
            }
        }
        catch (std::exception& e)
        {
            ROS_FATAL_STREAM(_node_name << ": exception when loading
thrusters/limiter: " << e.what()) ;
            valid_config = false ;
        }

        // Fins address
        int fins_address ;
        if (!ros::param::getCached("fins/address", fins_address))
        {
            ROS_FATAL_STREAM(_node_name << ": invalid parameters for
fins/address in param server!") ;
            valid_config = false ;
        }
        else
        {
            _fins_address = (unsigned char)fins_address ;
```

```
        }

        // Shutdown if not valid
        if (!valid_config)
        {
            ROS_FATAL_STREAM(_node_name << ": shutdown due to invalid
config parameters!") ;
            ros::shutdown() ;
        }
    }

    bool openSerialPort()
    {
        if (!serial_port.isOpen())
        {
            try
            {
                serial_port.open(device_config.sp_path) ;

serial_port.setBaudRate(cola2::io::SerialPort::baudRateFromInteger(device_c
onfig.sp_baud_rate)) ;

serial_port.setCharSize(cola2::io::SerialPort::charSizeFromInteger(device_c
onfig.sp_char_size)) ;

serial_port.setNumOfStopBits(cola2::io::SerialPort::numOfStopBitsFromIntege
r(device_config.sp_stop_bits)) ;

serial_port.setParity(cola2::io::SerialPort::parityFromString(device_config
.sp_parity)) ;

serial_port.setFlowControl(cola2::io::SerialPort::flowControlFromString(dev
ice_config.sp_flow_control)) ;
            }
            catch (std::exception& e)
            {
                ROS_ERROR_STREAM(_node_name << ": error setting up the
serial port: " << e.what()) ;
            }
        }
        else
        {
            ROS_ERROR_STREAM(_node_name << ": serial port already open!") ;
        }

        return serial_port.isOpen() ;
    }

    // Last setpoint executable for the main
    void finsSetToZero()
    {
        // Using the enable command to set fins to zero
        _fins_ptr->enableFins(_fins_address) ;
    }

    // Disable fins executable for the main
    void disableFinsAux()
    {
        _fins_ptr->disableFins(_fins_address) ;
    }
```

```
    void finsSetpointCallback( const cola2_control::Setpoints& fins_msg)
    {
        std::vector<double> fins_setpoint = fins_msg.setpoints ;

        // Apply a symmetric ramp to each fin
        int i ;
        for (i = 0 ; i < fins_setpoint.size() ; i++)
        {
            if (fins_setpoint[i] > _old_fins_setpoints[i] + 8)
            {
                fins_setpoint[i] = _old_fins_setpoints[i] + 8 ;
            }
            else if (fins_setpoint[i] < _old_fins_setpoints[i] - 8)
            {
                fins_setpoint[i] = _old_fins_setpoints[i] - 8 ;
            }

            _old_fins_setpoints[i] = fins_setpoint[i] ;
        }

        // Compute fins setpoints
        _fins_ptr->computeFinsSetpoint(_fins_address,  (  fins_setpoint[0]  +
60 ) * 2, ( fins_setpoint[1] + 60 ) * 2 ) ;
    }

    void sendWaterRequestTimerCallback(const ros::TimerEvent& event)
    {
            if (not _water)
            {
                _water = _fins_ptr->waterRequest(_fins_address) ;

                if (_water)
                {
                    ROS_FATAL_STREAM(_node_name  <<  ": Water  inside  fins  "
<< _water_offset) ;

                    if (_water_offset < 5)
                    {
                        _water_offset++ ;
                        _water = false ;
                    }
                    else
                    {
                        _water = true ;
                    }
                }
                else
                {
                    _water_offset = 0 ;
                }
            }

            if (_water)
            {
                diagnostic.add("water_inside_fins", "True") ;

diagnostic.setLevel(diagnostic_msgs::DiagnosticStatus::ERROR, "Water inside
fins") ;
                ROS_FATAL_STREAM(_node_name    <<    ":    WATER    IN    FINS
COMPARTMENT! WATER IN FINS COMPARTMENT!") ;
            }
```

```
            else
            {
                diagnostic.add("water_inside_fins", "False") ;
                diagnostic.setLevel(diagnostic_msgs::DiagnosticStatus::OK,
"No water") ;
            }
    }

    void     thrustersSetpointCallback(const     cola2_control::Setpoints&
thrusters_msg)
    {
        _thrusters_ptr->thrusterSetpointCallback(thrusters_msg.setpoints,
device_config.addresses,    device_config.signs,    device_config.limiter,
device_config.symmetric,                          device_config.max_step,
device_config.request_info) ;

        if (device_config.request_info)
        {
            thrusters_info.thruster_voltages        =        _thrusters_ptr-
>thrusters_info.thruster_voltages ;
            thrusters_info.thruster_currents        =        _thrusters_ptr-
>thrusters_info.thruster_currents ;
            thrusters_info.thruster_rpms          =          _thrusters_ptr-
>thrusters_info.thruster_rpms ;
            thrusters_info.thruster_controller_temperatures              =
_thrusters_ptr->thrusters_info.thruster_controller_temperatures ;
            thrusters_info.thruster_output_powers      =     _thrusters_ptr-
>thrusters_info.thruster_output_powers ;
            thrusters_info.thruster_warnings        =        _thrusters_ptr-
>thrusters_info.thruster_warnings ;
            thrusters_info.thruster_errors         =         _thrusters_ptr-
>thrusters_info.thruster_errors ;
            thrusters_info.total_ms_in_driver       =        _thrusters_ptr-
>thrusters_info.total_ms_in_driver ;
            thrusters_info.setpoints            =            _thrusters_ptr-
>thrusters_info.setpoints ;
            thrusters_info.processed_setpoints       =       _thrusters_ptr-
>thrusters_info.processed_setpoints ;

            thruster_info_pub.publish(thrusters_info) ;
        }
    }

    void sendZeroTimerCallback(const ros::TimerEvent& event)
    {
        if            (boost::posix_time::time_period(lastSetpointCallback,
boost::posix_time::microsec_clock::universal_time()).length().total_microse
conds() > 200000)
        {
            _thrusters_ptr->sendZeros(device_config.addresses) ;
        }
        lastSetpointCallback                                                =
boost::posix_time::microsec_clock::universal_time() ;
    }

    // Send zeros executable for the main
    void sendZeroAux()
    {
        _thrusters_ptr->sendZeros(device_config.addresses) ;
    }
} ;
```

```
sig_atomic_t volatile running = 1 ;
void stopHandler(int sig)
{
    running = 0 ;
}

int main(int argc, char **argv)
{
    ros::init(argc,                  argv,                  "actuators_485",
ros::init_options::NoSigintHandler) ;
    signal(SIGINT, stopHandler) ;

    // Node name
    std::string node_name ;
    node_name = ros::this_node::getName() ;

    // Init. FINS_485 object
    FINS_485 *fins_ptr ;
    fins_ptr = new FINS_485(node_name) ;

    // Init. THRUSTERS_485 object
    THRUSTERS_485 *thrusters_ptr ;
    thrusters_ptr = new THRUSTERS_485(node_name) ;

    // Init. ACTUATORS_485 object
    ACTUATORS_ROS actuators(node_name, fins_ptr, thrusters_ptr) ;

    // Get parameters, enable fins and init thrusters
    actuators.init() ;

    // Show message
    ROS_INFO_STREAM(node_name << " :initialized") ;

    // Iterate while not shutdown requested
    while (running)
    {
        ros::spinOnce() ;
        ros::Duration(0.01).sleep() ;
    }

    // Pose fins to zero
    actuators.finsSetToZero() ;

    // Let the fins go to zero
    ros::Duration(0.01).sleep() ;

    // Disable fins
    actuators.disableFinsAux() ;

    // Send zeros to thrusters
    actuators.sendZeroAux() ;

    // Shutdown
    ros::shutdown() ;
    return 0 ;
}
```

E. PROGRAMACIÓ DEL CONTROL

E.1. Fitxer de configuració

```
# DEPTH TO PITCH PID CONTROLLER
controller/d_p_kp: 0.15
controller/d_p_ti: 12.0
controller/d_p_td: 0.0
controller/d_p_limit: 0.2
controller/d_p_fff: 0.0



# ROLL PID CONTROLLER
controller/f_roll_kp: 0.4
controller/f_roll_ti: 15.0
controller/f_roll_td: 0.0
controller/f_roll_i_limit: 0.25
controller/f_roll_fff: 0.0



# PITCH PID CONTROLLER
controller/f_pitch_kp: 0.8
controller/f_pitch_ti: 15.0
controller/f_pitch_td: 8.0
controller/f_pitch_i_limit: 0.1
controller/f_pitch_fff: 0.0



# POSE PID CONTROLLER
controller/p_surge_kp: 0.0
controller/p_surge_ti: 0.0
controller/p_surge_td: 0.0
controller/p_surge_i_limit: 0.0
controller/p_surge_fff: 0.0

controller/p_sway_kp: 0.0
controller/p_sway_ti: 0.0
controller/p_sway_td: 0.0
controller/p_sway_i_limit: 0.0
controller/p_sway_fff: 0.0

controller/p_heave_kp: 0.9
controller/p_heave_ti: 0.0
controller/p_heave_td: 0.5
controller/p_heave_i_limit: 0.0
controller/p_heave_fff: 0.0

controller/p_roll_kp: 0.0
controller/p_roll_ti: 0.0
controller/p_roll_td: 0.0
controller/p_roll_i_limit: 0.0
controller/p_roll_fff: 0.0

controller/p_pitch_kp: 0.0
```

```
controller/p_pitch_ti: 0.0
controller/p_pitch_td: 0.0
controller/p_pitch_i_limit: 0.0
controller/p_pitch_fff: 0.0

controller/p_yaw_kp: 0.5
controller/p_yaw_ti: 0.0
controller/p_yaw_td: 2.0
controller/p_yaw_i_limit: 0.0
controller/p_yaw_fff: 0.0



# MAX VELOCITY
/controller/max_velocity_x: 2.0
/controller/max_velocity_y: 0.0
/controller/max_velocity_z: 0.30
/controller/max_velocity_roll: 0.0
/controller/max_velocity_pitch: 0.0
/controller/max_velocity_yaw: 0.30



# TWIST PID CONTROLLER
controller/t_surge_kp: 0.25
controller/t_surge_ti: 20.0
controller/t_surge_td: 0.0
controller/t_surge_i_limit: 0.25
controller/t_surge_fff: 0.0

controller/t_sway_kp: 0.0
controller/t_sway_ti: 0.0
controller/t_sway_td: 0.0
controller/t_sway_i_limit: 1.0
controller/t_sway_fff: 0.0

controller/t_heave_kp: 3.0
controller/t_heave_ti: 8.0
controller/t_heave_td: 0.0
controller/t_heave_i_limit: 0.25
controller/t_heave_fff: 0.0 #0.26

controller/t_roll_kp: 0.0
controller/t_roll_ti: 0.0
controller/t_roll_td: 0.0
controller/t_roll_i_limit: 1.0
controller/t_roll_fff: 0.0

controller/t_pitch_kp: 0.0
controller/t_pitch_ti: 0.0
controller/t_pitch_td: 0.0
controller/t_pitch_i_limit: 1.0
controller/t_pitch_fff: 0.0

controller/t_yaw_kp: 1.5
controller/t_yaw_ti: 8.0
controller/t_yaw_td: 0.0
controller/t_yaw_i_limit: 0.15
controller/t_yaw_fff: 0.0
```

```
# TWIST POLY CONTROLLER ( A + Bx + Cx^2 )
controller/poly_surge_A: 0.0
controller/poly_surge_B: 9.9239
controller/poly_surge_C: 10.1728

controller/poly_sway_A: 0.0
controller/poly_sway_B: 0.0
controller/poly_sway_C: 0.0

controller/poly_heave_A: 0.0
controller/poly_heave_B: 0.0
controller/poly_heave_C: 259.8366

controller/poly_roll_A: 0.0
controller/poly_roll_B: 0.0
controller/poly_roll_C: 0.0

controller/poly_pitch_A: 0.0
controller/poly_pitch_B: 0.0
controller/poly_pitch_C: 0.0

controller/poly_yaw_A: 0.0
controller/poly_yaw_B: 0.0
controller/poly_yaw_C: 53.2837



# SIGNIFICANCE OF THE POLYNOMIAL RESULT ( must be between 0 and 1 )
controller/poly_surge_percentatge: 1.0
controller/poly_sway_percentatge: 0.0
controller/poly_heave_percentatge: 1.0
controller/poly_roll_percentatge: 0.0
controller/poly_pitch_percentatge: 0.0
controller/poly_yaw_percentatge: 1.0



# MAX WRENCH
/controller/max_wrench_X: 154.0
/controller/max_wrench_Y: 0.0
/controller/max_wrench_Z: 30.0
/controller/max_wrench_Roll: 3.66
/controller/max_wrench_Pitch: 17.15
/controller/max_wrench_Yaw: 14.43



# THRUSTER ALLOCATOR
/controller/max_force_thruster_forward: 77.0
/controller/max_force_thruster_backward: 43.0
/controller/thruster_distance_yaw: 0.16783

/controller/TCM: [0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, -0.16783, 0.16783]


# FIN ALLOCATOR
/controller/max_fin_force: 26.15
/controller/fin_distance_surge: 0.65
/controller/fin_distance_yaw: 0.14
```

```
/controller/max_fin_angle: 40.0
/controller/min_fin_angle: -40.0

/controller/force_to_fins_ratio: 13.0

/controller/FCM: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.14, 0.14, 0.65, 0.65,
0.0, 0.0]
/controller/fin_poly: [0.0, 1.0, 0.0, 0.0, 1.0, 0.0]
```

E.2. Programes principals

E.2.1. Programa del PID

```
#include "IController.hpp"

#ifndef __PID_CLASS__
#define __PID_CLASS__

class Pid: public IController {
public:
    Pid(std::string name):
        IController( name ),
        _kp( 0 ),
        _ti( 0 ),
        _td( 0 ),
        _i_limit( 0 ),
        _fff( 0 ),
        _time_old( 0 ),
        _feedback_old( 0 ),
        _error_old( 0 ),
        _eik_old( 0 ),
        _derivative_term_from_feedback( true ),
        _edotk_old(0)
    {}

    ~Pid() {}

    void reset()
    {
        _eik_old = 0.0;
        _time_old = 0;
    }

    double
    compute(double time_in_sec, double setpoint, double feedback)
    {
        // std::cout << "Compute " << _name << ", time: " << time_in_sec <<
", setpoint: " << setpoint << ", feedback: " << feedback << std::endl;

        // Check time
        if ( _time_old == 0 ){
            // Fist time or just reset
            _time_old = time_in_sec;
            _feedback_old = feedback;
            _error_old = setpoint - feedback;
            _edotk_old = 0.0;
```

```
            return 0.0;
        }

        double dt = time_in_sec - _time_old;
        if ( dt > 0.2 ) {
            // To much time without controlling this DoF, reset it!
            std::cout << "ERROR wit PID " << _name << " dt = " << dt <<
"\n";
            reset();
            return 0.0;
        }

        if ( dt < 0.01 ) dt = 0.01;
        _time_old = time_in_sec;
        // std::cout << "PID " << _name << " dt: " << dt << std::endl;

        // Compute error, derivative of feedback and integral part
        double ek = setpoint - feedback;
        // std::cout << "PID "  << _name << " ek: " << ek << std::endl;

        double edotk = 0.0;
        if( _derivative_term_from_feedback ) {
            edotk = ( feedback - _feedback_old ) / dt;
        }
        else {
            edotk = ( ek - _error_old ) / dt;
        }

        // std::cout << "edotk: " << edotk << "\n";
        // std::cout << "tD: " << _td << "\n";
        // std::cout << "part derivativa: " << _kp * _td * edotk << "\n";

        // edotk = (edotk + _edotk_old) / 2.0;
        // std::cout << "edotk filtered: " << edotk << "\n";

        double eik = _eik_old + ( ek * dt );

        // Compute the integral part if ti > 0
        double integral_part = 0.0;
        double tau = 0;
        if ( _ti > 0.0 ) {
            // Integral part
            integral_part = ( _kp / _ti ) * eik;

            // Saturate integral part
            //  (anti-windup  condition,  integral  part  not  higher  than  a
value)
            integral_part = _saturateValue(integral_part, _i_limit);

            // Restore eik
            if ( _kp > 0.0 ) {
                // Avoid division by zero
                eik = integral_part * _ti / _kp;
            }
            else {
                eik = 0.0;
            }
            // Compute tau
            tau = _kp * ( ek - _td * edotk) + integral_part + _fff;
        }
        else {
```

```
            // Compute tau without integral part
            tau = _kp * ( ek - _td * edotk) + _fff;
        }
        // std::cout << "PID " << _name << " tau: " << tau << std::endl;

        // Store for the next time
        _feedback_old = feedback;
        _error_old = ek;
        _eik_old = eik;
        _edotk_old = edotk;


        // Return saturate tau
        return _saturateValue(tau, 1.0);
    }


    bool
    setParameters( std::map< std::string, double > params )
    {
        std::cout << "Set params for " << _name << " as: " << params["kp"]
<<  ",  "  <<  params["ti"]  <<  ",  "  <<  params["td"]  <<  ",  "  <<
params["derivative_term_from_feedback"] << "\n";

        try {
            _kp = params["kp"];
            _ti = params["ti"];
            _td = params["td"];
            _i_limit = params["i_limit"];
            _fff = params["fff"];
            _derivative_term_from_feedback                                 =
bool(params["derivative_term_from_feedback"]);
        }
        catch (...) {
            std::cout  <<  "PID  "  <<  _name  <<  "  setting  parameters  ERROR!
\n";
            return false;
        }
        return true;
    }


private:
    double _kp;
    double _ti;
    double _td;
    double _i_limit;
    double _fff;
    double _time_old;
    double _feedback_old;
    double _error_old;
    double _eik_old;
    bool _derivative_term_from_feedback;

    // for filtering purposes
    double _edotk_old;
};

#endif //__PID_CLASS__
```

E.2.2. Programa de l'estructura de control

```
#ifndef __SPARUS2_CONTROLLER__
#define __SPARUS2_CONTROLLER__

#include "Merge.hpp"
#include "NDofController.hpp"
#include "Pid.hpp"
#include "Poly.hpp"
#include "IAUVController.hpp"
#include "SPARUS2ThrusterAllocator.hpp"
#include "SPARUS2FinAllocator.hpp"
#include <algorithm>
#include <Eigen/Dense>


class SPARUS2Controller: public IAUVController {

public:

    SPARUS2Controller( double period,
                       unsigned int n_dof,
                       unsigned int n_thrusters,
                       unsigned int n_fins = 0 ):
        IAUVController( period, n_dof, n_thrusters, n_fins ),
        _thruster_allocator( n_thrusters ),
        _fin_allocator( n_fins ),
        _n_thrusters( n_thrusters ),
        _pose_controller( n_dof ),
        _twist_factor( n_dof ),
        _old_twist( n_dof ),
        _twist_controller( n_dof ),
        _force_factor( n_dof ),
        _twist_poly_controller( n_dof ),
        _poly_percentatge( n_dof )
    {
        // Init controllers
        initFinsControllers();
        initPoseController();
        initTwistController();
        initTwistPolyController();
    }


    void
    initFinsControllers()
    {
        _d_to_p_controller = new Pid( "depth_to_pitch" );

        _f_roll = new Pid( "roll_to_force" );
        _f_pitch = new Pid( "pitch_to_force" );
    }

    void
    initPoseController()
    {
        _p_surge = new Pid( "pose_surge" );
        _pose_controller.addController( _p_surge );
```

```
        _p_sway = new Pid( "pose_sway" );
        _pose_controller.addController( _p_sway );

        _p_heave = new Pid( "pose_heave" );
        _pose_controller.addController( _p_heave );

        _p_roll = new Pid( "pose_roll" );
        _pose_controller.addController( _p_roll );

        _p_pitch = new Pid( "pose_pitch" );
        _pose_controller.addController( _p_pitch );

        _p_yaw = new Pid( "pose_yaw" );
        _pose_controller.addController( _p_yaw );
    }

    void
    initTwistController()
    {
        _t_surge = new Pid( "twist_surge" );
        _twist_controller.addController( _t_surge );

        _t_sway = new Pid( "twist_sway" );
        _twist_controller.addController( _t_sway );

        _t_heave = new Pid( "twist_heave" );
        _twist_controller.addController( _t_heave );

        _t_roll = new Pid( "twist_roll" );
        _twist_controller.addController( _t_roll );

        _t_pitch = new Pid( "twist_pitch" );
        _twist_controller.addController( _t_pitch );

        _t_yaw = new Pid( "twist_yaw" );
        _twist_controller.addController( _t_yaw );
    }


    void
    initTwistPolyController()
    {
        _tp_surge = new Poly( "twist_poly_surge" );
        _twist_poly_controller.addController( _tp_surge );

        _tp_sway = new Poly( "twist_poly_sway" );
        _twist_poly_controller.addController( _tp_sway );

        _tp_heave = new Poly( "twist_poly_heave" );
        _twist_poly_controller.addController( _tp_heave );

        _tp_roll = new Poly( "twist_poly_roll" );
        _twist_poly_controller.addController( _tp_roll );

        _tp_pitch = new Poly( "twist_poly_pitch" );
        _twist_poly_controller.addController( _tp_pitch );

        _tp_yaw = new Poly( "twist_poly_yaw" );
        _twist_poly_controller.addController( _tp_yaw );
    }
```

```
    void
    setControllerParams( const std::vector< std::map< std::string, double >
> d_p_params,
                         const std::vector< std::map< std::string, double >
> f_params,
                         const std::vector< std::map< std::string, double >
> p_params,
                         const std::vector< std::map< std::string, double >
> t_params,
                         const std::vector< std::map< std::string, double >
> poly_params,
                         std::vector< double > poly_percentatge)
    {
        _d_to_p_controller->setParameters( d_p_params.at(0) );
        _f_roll->setParameters( f_params.at(0) );
        _f_pitch->setParameters( f_params.at(1) );

        _pose_controller.setControllerParams( p_params );
        _twist_controller.setControllerParams( t_params );
        _twist_poly_controller.setControllerParams( poly_params );

        _twist_factor = { 2.0, 0.0, 0.3, 0.0, 0.0, 0.6 };
        _force_factor = { 154.0, 0, 30.0, 3.64, 17.15, 14.43 };
        _old_twist = { 0, 0, 0, 0, 0, 0 };
        _poly_percentatge = poly_percentatge;

        _d_to_p_controller->reset();
        _f_roll->reset();
        _f_pitch->reset();
        _pose_controller.reset();
        _twist_controller.reset();
        _twist_poly_controller.reset();
    }

    void
    reset( )
    {
        _pose_controller.reset();
        _twist_controller.reset();
        _twist_poly_controller.reset();
    }

    void
    iteration( double current_time )
    {
        _mtx.lock();

        // Merge pose requests
        Request desired_pose = _pose_merge.merge( current_time );
        _merged_pose = desired_pose;

        float th0_coef = 1;
        float boy_coef = 1;
        bool apply_boy_coef = false;

        // SPARUS II fins
        //
***************************************************************************
********************************************************

        // Fins wrench requests
```

```
        Request fins_wrench ( desired_pose );

        std::vector< double > pose_values = desired_pose.getValues();
        std::vector<        bool        >        pose_axis_values        =
desired_pose.getDisabledAxis();

        std::vector< double > fins_values = fins_wrench.getValues();
        std::vector< bool > fins_axis = fins_wrench.getDisabledAxis();

        if ( ( _is_fin_allocator_enable ) and ( _twist_feedback[0] > 0.0 )
) {

            // Takes values between -pi and pi
            double desired_pitch;

            // If pitch pose axis is disabled -> 'automatic' pitch mode (2
cases: operating with velocity (PITCH MODE) or with 'automatic' pose (DEPTH
MODE))
            if ( pose_axis_values[4] ) {

                // When operating the vehicle with velocity and there is
not any WWR, do not compute the PID depth to pitch
                if ( pose_axis_values[2] ) {

                    // With velocity control the desired pitch is 0.0
                    desired_pitch = 0.0;

                    std::cout << "XXXXXXXXXXXXXXXXX HEAVE VELOCITY CONTROL
ENABLED XXXXXXXXXXXXXXXXX \n";
                    std::cout << "XXXXXXXXXXXXXXXXXX SETTING DESIRED PITCH
TO 0.0 XXXXXXXXXXXXXXXXXX \n\n";
                }
                // Modify desired depth pose to reach it with pitch
                else {

                    // Membership for depth
                    //float s; //shallow
                    float d; //deep
                    float low_d = 0.2;
                    float high_d = 1.0;

                    // Memberships for surge velocity
                    float s1; // slow
                    float s2; // normal
                    float s3; // fast
                    float low_s = 0.7; //0.1
                    float high_s = 0.9; //0.25
                    float low_f = 1.2; //0.35
                    float high_f = 1.4; //0.5

                    // Compute memership functions
                    d = ( 1 / ( high_d - low_d ) ) * _pose_feedback[2] + (
1 - ( 1 / ( high_d - low_d ) ) * high_d );
                    if( d > 1 ) d = 1;
                    if( d < 0 ) d = 0;

                    s1 = ( -1 / ( high_s - low_s ) ) * _twist_feedback[0] +
( high_s / ( high_s - low_s ) );
                    if( s1 > 1 ) s1 = 1;
                    if( s1 < 0 ) s1 = 0;
```

```
                    s3 = ( 1 / ( high_f - low_f ) ) * _twist_feedback[0] +
( -low_f / ( high_f - low_f ) );
                    if( s3 > 1 ) s3 = 1;
                    if( s3 < 0 ) s3 = 0;

                    if ( _twist_feedback[0] < high_s ) {
                        s2 = ( 1 - s1 );
                    }
                    else if ( _twist_feedback[0] > low_f ) {
                        s2 = ( 1 - s3 );
                    }
                    else {
                        s2 = 1;
                    }

                    // PID depth -> pitch ( taking in account the depth )
                    // Applying an offset to correct the buoyancy
                    float pitch_offset = 0.0;
                    /*if ( ( ( ( pose_values[2] + 1.5 ) > _pose_feedback[2]
) and ( ( pose_values[2] - 1.5 ) < _pose_feedback[2] ) ) or (
pose_values[2] > _pose_feedback[2] ) ) {
                        pitch_offset = - atanf( fabs( _twist_feedback[2] )
/ _twist_feedback[0] );
                    }*/
                    desired_pitch = d * (3.14159265359 / 2) * ( -
_d_to_p_controller->compute( current_time, pose_values[2],
_pose_feedback[2] ) + pitch_offset );

                    // Compute vertical thruster correction coeficients
                    th0_coef = 1 - d + s1 * d;
                    boy_coef = 1 - s3 * d;

                    // std::cout << "XXXXXXXXXXXXXXXXXXXX DEPTH TO PITCH
ENABLED XXXXXXXXXXXXXXXXXXXX \n";
                    // std::cout << "XXXXXXXXXXXXXXXXXXXX REACH DEPTH WITH
PITCH XXXXXXXXXXXXXXXXXXXX \n\n";

                    // Rules
                    if ( ( d == 0 ) or ( s1 == 1 ) ) {
                        // depth control by th0 at 100%
                        // keeping 'bouyancy' at 100%
                        // desired pitch is 0 degrees
                        // std::cout << "XXXXXXXXXXXXXXXXXX IMPOSSIBLE
XXXXXXXXXXXXXXXXXX \n\n";
                        pose_axis_values[2] = false;
                        th0_coef = 1;
                        boy_coef = 1;
                        desired_pitch = 0.0;
                    }
                    else if ( ( s1 > 0 ) and ( s2 > 0 ) and ( s3 == 0 ) ) {
                        // scaling depth control by th0
                        // keeping 'bouyancy' at 100%
                        // scaling desired pitch
                        // std::cout << "XXXXXXXXXXXXXXXXXXXX STATE 2
XXXXXXXXXXXXXXXXXXX \n\n";
                        pose_axis_values[2] = false;
                        desired_pitch = s2 * desired_pitch;
                    }
                    else if ( ( s1 == 0 ) and ( s2 > 0 ) and ( s3 == 0 ) )
{
                        // depth control by th0 at 0%
```

```
                        // keeping 'bouyancy' at 100%
                        // desired pitch is the DtoP PID controller result
                        //  std::cout  <<  "XXXXXXXXXXXXXXXXXXXX  STATE  3
XXXXXXXXXXXXXXXXXXX \n\n";
                        if ( d == 1 ) {
                            pose_axis_values[2] = true;
                        }
                        apply_boy_coef = true;
                    }
                    else if ( ( s1 == 0 ) and ( s2 > 0 ) and ( s3 > 0 ) ) {
                        // depth control by th0 at 0%
                        // scaling 'bouyancy'
                        // desired pitch is the DtoP PID controller result
                        //  std::cout  <<  "XXXXXXXXXXXXXXXXXXXX  STATE  4
XXXXXXXXXXXXXXXXXXX \n\n";
                        if ( d == 1 ) {
                            pose_axis_values[2] = true;
                        }
                        apply_boy_coef = true;
                    }
                    else if ( ( s1 == 0 ) and ( s2 == 0 ) and ( s3 > 0 ) )
{
                        // depth control by th0 at 0%
                        // 'buoyancy' at 0%
                        // desired pitch is the DtoP PID controller result
                        //  std::cout  <<  "XXXXXXXXXXXXXXXXXX  JUST  FINS
XXXXXXXXXXXXXXXXXX \n\n";
                        if ( d == 1 ) {
                            pose_axis_values[2] = true;
                        }
                    }

                    // Saturate desired pitch
                    desired_pitch = _saturateValue( desired_pitch, 0.785 );

                    // std::cout << "---------- V.TH at " << th0_coef << "
% \n";
                    // std::cout << "---------- BUO. at " << boy_coef << "
% \n\n";
                    // std::cout << "---- PITCH OFFSET: " << pitch_offset *
180 / 3.1415 << "\n";
                    // std::cout << "------ DES. PITCH: " << desired_pitch
* 180 / 3.1415 << "\n\n";
                }
            }
            // If pitch pose axis is enabled -> direct pitch mode (1 case:
operating with 'manual' pose (PITCH MODE))
            else  {
                // The desired pitch is directly the input pitch
                desired_pitch = pose_values[4];

                //   std::cout   <<   "XXXXXXXXXXXXXXXXXXXX   DIRECT   PITCH
XXXXXXXXXXXXXXXXXXXX \n\n";
            }

            // PIDs: ( roll -> force ) and ( pitch -> force )
            fins_values[3] = _saturateValue( _f_roll->compute(current_time,
pose_values[3], _pose_feedback[3]) * _force_factor.at(3), _max_wrench.at(3)
);
```

```
            fins_values[4]        =        _saturateValue(        _f_pitch-
>compute(current_time,      desired_pitch,      _pose_feedback[4])      *
_force_factor.at(4), _max_wrench.at(4) );
        }
        else {
            // Set roll and pitch desired torque to 0 due to fins are
disabled
            fins_values[3] = 0.0;
            fins_values[4] = 0.0;
        }
        // Disable the roll and pitch axis for the twist controller (is not
able to control this DOFs)
        pose_axis_values[3] = true;
        pose_axis_values[4] = true;

        // Set the modified axis and values to the pose requester
        desired_pose.setValues(pose_values);
        desired_pose.setDisabledAxis(pose_axis_values);

        // Enable the roll and pitch axis in the fins requester
        fins_axis[3] = false;
        fins_axis[4] = false;

        // Set the computed axis and values to the fins requester
        fins_wrench.setValues(fins_values);
        fins_wrench.setDisabledAxis(fins_axis);

        std::cout  <<  "Fins  controller  result:  \n"  <<  fins_wrench  <<
std::endl;
        //
***************************************************************************
********************************************************



        if( _is_pose_controller_enable ) {

            // Initialize the twist req. that will be generated by the pose
controller
            Request velocity_req( desired_pose );

            // Compute pose error
            std::vector<   double   >      pose_error   =   computeError(
desired_pose.getValues(), _pose_feedback );

            // Compute PID between pose error and zero
            desired_pose.setValues( pose_error );
            std::vector< double > zero(6, 0.0);
            std::vector< double > pose_ctrl_tau = _pose_controller.compute(
current_time, desired_pose, zero );

            // Normalize output to max velocity
            assert( pose_ctrl_tau.size() == _max_velocity.size() );
            for( unsigned int i = 0; i < pose_ctrl_tau.size(); i++ ) {
                if ( i == 2 ) {
                    // Scale desired velocity according to fins controller
                    pose_ctrl_tau.at(i)   =   th0_coef   *   _saturateValue(
pose_ctrl_tau.at(i) * _twist_factor[i], _max_velocity.at(i) );
                }
                else {
```

```
                    pose_ctrl_tau.at(i)          =          _saturateValue(
pose_ctrl_tau.at(i) * _twist_factor[i], _max_velocity.at(i) );
                }
            }

            // Add Pose controller response to body velocity requests
            velocity_req.setValues( pose_ctrl_tau );
            std::cout  <<  "Pose  controller  result:  \n"  <<  velocity_req  <<
std::endl;
            _twist_merge.addRequest( velocity_req );
        }

        // Merge twist request
        Request desired_twist = _twist_merge.merge( current_time );
        _merged_twist = desired_twist;



        if( _is_velocity_controller_enable ) {
            // Initialize the wrench_req req. that will be generated by the
twist controller
            Request wrench_req( desired_twist );

            // Apply a non symmetric ramp to the twist input
            std::vector< double > twist_step = { 0.1, 0, 0.0, 0.0, 0.00,
0.02 };
            std::vector< bool > apply_step = {true, false, false, false,
false, true};
            std::vector< double > des_twist = desired_twist.getValues();
            for ( unsigned int i = 0; i < 6; i++ ) {
                if ( apply_step[i] ) {
                    if ( ( des_twist[i] > _old_twist[i] + twist_step[i] )
and ( des_twist[i] > twist_step[i] ) ) {
                        if   (   (   _old_twist[i]   +   twist_step[i]   )   >
twist_step[i] ) {
                            des_twist[i] = _old_twist[i] + twist_step[i] ;
                        }
                        else {
                            des_twist[i] = twist_step[i];
                        }
                    }
                    else   if   (   (   des_twist[i]   <   _old_twist[i]   -
twist_step[i] ) and ( des_twist[i] < -twist_step[i] ) ) {
                        if   (   (   _old_twist[i]   -   twist_step[i])   <   -
twist_step[i] ) {
                            des_twist[i] = _old_twist[i] - twist_step[i] ;
                        }
                        else {
                            des_twist[i] = -twist_step[i] ;
                        }
                    }
                    _old_twist[i] = des_twist[i];
                    desired_twist.setValues(des_twist);
                }
            }

            // Compute twist control (and set the resulting wrench req.)
            std::vector<  double  >  pid_twist  =  _twist_controller.compute(
current_time,

desired_twist,
```

```
_twist_feedback );

            // For the poly input take in account the currents (
desired_twist = desired_twist - _currents_feedback )

            std::vector< double > poly_twist =
_twist_poly_controller.compute( current_time,

desired_twist,

_twist_feedback);

            // Normalize output to max force
            assert( pid_twist.size() == poly_twist.size() );
            std::vector< double > total;
            for ( unsigned int i = 0; i < pid_twist.size(); i++ ) {
                if ( i == 2 ) {
                    // Add buoyancy
                    pid_twist.at(i) = pid_twist.at(i) + 0.10 * boy_coef;

                    if ( apply_boy_coef ) {
                        std::vector< bool > axis =
wrench_req.getDisabledAxis();
                        axis[2] = false;
                        wrench_req.setDisabledAxis( axis );
                    }
                }

                // Scale and saturate force
                total.push_back( _saturateValue( _saturateValue(
pid_twist.at(i), 1.0 ) * _force_factor[i] + poly_twist.at(i) *
_poly_percentatge.at(i), _max_wrench.at(i) ) );
                //std::cout << i << " Total: " << _saturateValue(
scaled[i], _max_wrench.at(i) ) << "\n";
                //std::cout << " PID: " << pid_twist.at(i) << "\n";
                //std::cout << " Poly: " << poly_twist.at(i) << "\n\n";
            }

            wrench_req.setValues( total );
            std::cout << "Twist controller result: \n" << wrench_req <<
std::endl;
            _wrench_merge.addRequest( wrench_req );
        }

        // Merge wrench req. and save them
        _merged_wrench = _wrench_merge.merge( current_time );

        // Compute thruster setpoints -> done in the interatin loop

        // Compute fin setpoints
        //_fin_setpoints = _fin_allocator.calibration( fins_setpoints );
        _fin_setpoints = _fin_allocator.compute( fins_wrench,
_thruster_setpoints, _twist_feedback );

        _mtx.unlock();
    }


    std::vector< double >
    computeError( const std::vector< double > setpoint,
```

```
                  const std::vector< double > feedback )
    {
        // WARNING: This function is intended for 6 DoF controllers only!
        assert( setpoint.size() == 6 );
        assert( feedback.size() == 6 );

        std::vector< double > error( 6, 0.0 );
        poseError( setpoint, feedback, error );
        error.at( 2 ) = setpoint.at( 2 ) - feedback.at( 2 );
        error.at( 3 ) = _normalizeAngle( setpoint.at( 3 ) - feedback.at( 3
) );
        error.at( 4 ) = _normalizeAngle( setpoint.at( 4 ) - feedback.at( 4
) );
        error.at( 5 ) = _normalizeAngle( setpoint.at( 5 ) - feedback.at( 5
) );

        return error;
    }


    void
    poseError( const std::vector< double > setpoint,
               const std::vector< double > feedback,
               std::vector< double >& error )
    {
        double yaw = feedback.at( 5 );
        Eigen::MatrixXd m( 3, 3 );
        m( 0, 0 ) = cos( yaw );     m( 0, 1 ) = -sin( yaw );    m( 0, 2 ) =
0.0;
        m( 1, 0 ) = sin( yaw );     m( 1, 1 ) = cos( yaw );     m( 1, 2 ) =
0.0;
        m( 2, 0 ) = 0.0;            m( 2, 1 ) = 0.0;            m( 2, 2 ) =
1.0;
        //std::cout << "m:\n" << m << "\n\n";

        double pitch = feedback.at( 4 );
        Eigen::MatrixXd n( 3, 3 );
        n( 0, 0 ) = cos( pitch );   n( 0, 1 ) = 0.0;            n( 0, 2 ) =
sin( pitch );
        n( 1, 0 ) = 0.0;            n( 1, 1 ) = 1.0;            n( 1, 2 ) =
0.0;
        n( 2, 0 ) = -sin( pitch );  n( 2, 1 ) = 0.0;            n( 2, 2 ) =
cos( pitch );
        //std::cout << "n:\n" << n << "\n\n";

        double roll = feedback.at( 3 );
        Eigen::MatrixXd o( 3, 3 );
        o( 0, 0 ) = 1.0;            o( 0, 1 ) = 0.0;            o( 0, 2 ) =
0.0;
        o( 1, 0 ) = 0.0;            o( 1, 1 ) = cos( roll );    o( 1, 2 ) =
-sin( roll );
        o( 2, 0 ) = 0.0;            o( 2, 1 ) = sin( roll );    o( 2, 2 ) =
cos( roll );
        //std::cout << "o:\n" << o << "\n\n";

        Eigen::MatrixXd p( 3, 1 );
        p( 0, 0 ) = feedback.at( 0 );
        p( 1, 0 ) = feedback.at( 1 );
        p( 2, 0 ) = 0.0;
        //std::cout << "p:\n" << p << "\n\n";
```

```
        // Rotation: yaw -> pitch -> roll
        Eigen::MatrixXd rotZYX = m * n * o;
        //std::cout << "rotation:\n" << rotZYX << "\n\n";

        Eigen::MatrixXd aux = -rotZYX.transpose() * p;
        //std::cout << "aux:\n" << aux << "\n\n";

        Eigen::MatrixXd T( 4, 4 );
        T( 0, 0 ) = rotZYX.transpose()( 0, 0 );    T( 0, 1 ) =
rotZYX.transpose()( 0, 1 );  T( 0, 2 ) = rotZYX.transpose()( 0, 2 );  T( 0,
3 ) = aux( 0, 0 );
        T( 1, 0 ) = rotZYX.transpose()( 1, 0 );    T( 1, 1 ) =
rotZYX.transpose()( 1, 1 );  T( 1, 2 ) = rotZYX.transpose()( 1, 2 );  T( 1,
3 ) = aux( 1, 0 );
        T( 2, 0 ) = rotZYX.transpose()( 2, 0 );    T( 2, 1 ) =
rotZYX.transpose()( 2, 1 );  T( 2, 2 ) = rotZYX.transpose()( 2, 2 );  T( 2,
3 ) = aux( 2, 0 );
        T( 3, 0 ) = 0.0;                           T( 3, 1 ) = 0.0;
T( 3, 2 ) = 0.0;                         T( 3, 3 ) = 1.0;
        //std::cout << "T:\n" << T << "\n\n";

        Eigen::MatrixXd p_req( 4, 1 );
        p_req( 0, 0 ) = setpoint.at( 0 );
        p_req( 1, 0 ) = setpoint.at( 1 );
        p_req( 2, 0 ) = 0.0;
        p_req( 3, 0 ) = 1.0;
        //std::cout << "p_req:\n" << p_req << "\n\n";

        Eigen::MatrixXd distance = T * p_req;
        error.at( 0 ) = distance( 0, 0 );
        error.at( 1 ) = distance( 1, 0 );
        //std::cout << "distance:\n" << distance << "\n\n";
    }


    void
    computeThrusterAllocator()
    {
        if ( _is_thruster_allocator_enable ) {
            _thruster_setpoints       =       _thruster_allocator.compute(
_merged_wrench, _twist_feedback );
        }
    }


    // Must be public to be visible from AUVROSBaseController
    ThrusterAllocatorSPARUS2 _thruster_allocator;
    FinAllocatorSPARUS2 _fin_allocator;

    unsigned int
    getNumberofThrusters() const
    {
        return _n_thrusters;
    }

private:
    unsigned int _n_thrusters;

    // Depth to pitch PID controller
    Pid *_d_to_p_controller;
```

```
    // Pitch PID controller
    Pid *_f_roll;
    Pid *_f_pitch;

    // Pose PID controller
    Pid *_p_surge, *_p_sway, *_p_heave, *_p_roll, *_p_pitch, *_p_yaw;
    NDofController _pose_controller;

    // Scale the output of the pose PID controller
    std::vector< double > _twist_factor;

    // Old computed twist value
    std::vector< double > _old_twist;

    // Twist PID controller
    Pid *_t_surge, *_t_sway, *_t_heave, *_t_roll, *_t_pitch, *_t_yaw;
    NDofController _twist_controller;

    // Scale the output of the twist PID controller
    std::vector< double > _force_factor;

    // Twist Poly controller
    Poly   *_tp_surge,   *_tp_sway,   *_tp_heave,   *_tp_roll,   *_tp_pitch,
*_tp_yaw;
    NDofController _twist_poly_controller;

    // Twist Poly percentatge
    std::vector< double > _poly_percentatge;
};

#endif //__SPARUS2_CONTROLLER__
```

E.2.3. Programa del model dels motors

```
#ifndef __SPARUS2THRUSTERALLOCATOR_CLASS__
#define __SPARUS2THRUSTERALLOCATOR_CLASS__
#include <algorithm>
#include <map>

#include <Eigen/Dense>
#include <Eigen/SVD>
#include <sstream>
#include "Request.hpp"
#include <math.h>
#include "ros/ros.h"
#include "cola2_control/ThrustersInfo.h"

bool simulation = true;

class ThrusterAllocatorSPARUS2 {
public:
    ThrusterAllocatorSPARUS2( unsigned int n_thrusters ):
        _n_thrusters( n_thrusters ),
        _th_voltages( n_thrusters ),
        _th_currents( n_thrusters ),
        _th_rpms( n_thrusters ),
        _is_init( false )
    {
```

```
        // Load params
        _max_force_thruster_forward = 77.0;
        _max_force_thruster_backward = 43.0;
        _thruster_distance_yaw = 0.16783;

        // Thrusters model coeficients
        _c1_f = 28.89;
        _c1_b = 60.12;
        _c2 = 8.5;
        _c3 = 1.2 * 0.00442747 * 1.54;
        _c4 = 2 * 0.1 * 1.54;

        Eigen::MatrixXd tcm( 6, _n_thrusters );
        tcm << 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, -0.16783, 0.16783;
        //std::cout << "tcm: \n" << tcm << "\n";

        // Compute pseudoinverse
        _tcm_inv = ( tcm.transpose() * tcm ).inverse() * tcm.transpose();

        // Subscribers
        _th_info   =   _n.subscribe(   "/cola2_control/thrusters_info",   2,
&ThrusterAllocatorSPARUS2::updateThInfo, this );
    }

    ~ThrusterAllocatorSPARUS2()
    {}

    void
    setParams( const double max_force_thruster_forward,
               const double max_force_thruster_backward,
               const double thruster_distance_yaw,
               const std::vector< double > tcm_values )
    {
        std::cout << "ThrusterAllocatorSPARUS2 set params\n";

        _max_force_thruster_forward = max_force_thruster_forward;
        _max_force_thruster_backward = max_force_thruster_backward;
        _thruster_distance_yaw = thruster_distance_yaw;

        // Init TCM inverse
        std::cout << "tcm_values.size(): " << tcm_values.size() << "\n";
        assert(tcm_values.size() == 6*_n_thrusters);
        Eigen::MatrixXd tcm( 6, _n_thrusters );
        for( unsigned int i = 0; i < 6; i++ ) {
            for( unsigned int j = 0; j < _n_thrusters; j++ ) {
                tcm(i, j) = tcm_values.at( i*_n_thrusters + j );
            }
        }

        //std::cout << "TCM:\n"<< tcm << "\n";
        _tcm_inv = ( tcm.transpose() * tcm ).inverse() * tcm.transpose();
        //std::cout << "TCM inv:\n" << _tcm_inv << "\n";

        std::cout << "ThrusterAllocatorSPARUS2 initialized!\n";
        _is_init = true;
    }

    void
    updateThInfo(const cola2_control::ThrustersInfo& msg) {
        _th_voltages = msg.thruster_voltages;
```

```
        _th_currents = msg.thruster_currents;
        _th_rpms = msg.thruster_rpms;
    }

    Eigen::VectorXd
    compute( Request& wrench, const std::vector< double >& twist )
    {
        if( !_is_init ) return Eigen::MatrixXd::Zero( _n_thrusters, 1 );

        // Take wrench request if disabled false, otherwise, get 0.0.
        //std::cout << "thruster allocator\n" << wrench << "\n";
        Eigen::VectorXd wrench_req( 6 );
        for( unsigned int i = 0; i < wrench.getDisabledAxis().size(); i++ )
{
            if( wrench.getDisabledAxis().at(i) ) {
                wrench_req[i] = 0.0;
            }
            else {
                wrench_req[i] = wrench.getValues().at(i);
            }
        }
        //std::cout << "Wrench:\n" << wrench_req << "\n";

        // Keep the surge velocity and the thruster feedback for the hole
iteration
        _AllocatorFeedback feedback_info;
        feedback_info.surge_velocity = twist[0];
        feedback_info.thrusters_voltage = _th_voltages;
        feedback_info.thrusters_current = _th_currents;
        feedback_info.thrusters_rpm = _th_rpms;

        // Merge Surge and Yaw
        mergeSurgeYaw( wrench_req[0], wrench_req[5], feedback_info );
        // std::cout << "wrench:\n" << wrench_req << "\n";

        // Multiply wrench by thruster allocation matrix
        Eigen::VectorXd force_per_thruster;
        force_per_thruster = _tcm_inv * wrench_req;
        //std::cout << "force_per_thruster: \n" << force_per_thruster <<
"\n";

        // Force to setpoint
        Eigen::VectorXd setpoint = forceToSetpoint( force_per_thruster,
feedback_info );
        //std::cout << "setpoint: \n" << setpoint << "\n";

        // Publish
        return setpoint;
    }


private:
    unsigned int _n_thrusters;
    double _max_force_thruster_forward;
    double _max_force_thruster_backward;
    double _thruster_distance_yaw;
    Eigen::MatrixXd _tcm_inv;
    std::vector< double > _th_voltages;
    std::vector< double > _th_currents;
    std::vector< double > _th_rpms;
    bool _is_init;
```

```
    double _c1_f;
    double _c1_b;
    double _c2;
    double _c3;
    double _c4;

    // Node handle
    ros::NodeHandle _n;

    // Subscriber
    ros::Subscriber _th_info;

    struct _AllocatorFeedback {
        double surge_velocity;
        std::vector< double > thrusters_voltage;
        std::vector< double > thrusters_current;
        std::vector< double > thrusters_rpm;
    };

    Eigen::VectorXd
    forceToSetpoint( Eigen::VectorXd& wrench,
                     const struct _AllocatorFeedback& feedback ) {

        Eigen::VectorXd ret( wrench.size() );
        // Compute newtons to setpoints for each thruster
        for( unsigned int i = 0; i < wrench.size(); i++ ) {
            //std::cout << "ret " << i << ": " << ret[i] << "\n";

            // Initialize setpoint
            ret[i] = 0.0;

            if (i == 0) { //vertical thruster

                if ( simulation == false ) {
                    // Adjust force according to voltage level
                    double   v_th_correction   =   3.4882   -   0.0754   *
feedback.thrusters_voltage[i];
                    saturate( v_th_correction, 1.34, 1 );
                    wrench[i] *= v_th_correction;
                }

                // Checking the wrench limit
                saturate( wrench[i], 30, -30 );

                // Applying the thruster model
                if  (  wrench[i]  >  0.381  )  {  //  Minimum  setpoint  of  0.11
justified by wrench request, not by the model offset
                    ret[i] = 0.1 + ( -37.66796 + pow( ( pow( 37.66796 , 2 )
- 4 * 45.69876 * -wrench[i] ) , 0.5 ) ) / ( 2.0 * 45.69876 );
                }
                else  if  (  wrench[i]  <  -0.013  )  {  //  Minimum  setpoint  of
0.11 justified by wrench request, not by the model offset
                    ret[i] = -0.1 + ( -0.85304 + pow( ( pow( 0.85304 , 2 )
- 4 * -44.78478 * -wrench[i] ) , 0.5 ) ) / ( 2.0 * -44.78478 );
                }
                else {
                    ret[i] = 0.0;
                }
            }
            else { //horitzontal thrusters
```

```
                // Checking the wrench limit
                saturate( wrench[i], 77, -43 );

                if ( simulation ) {
                    // Applying the thruster model
                    if ( wrench[i] > 0 ) {
                        ret[i] = ( 0.000003129 * pow( wrench[i] , 2 ) -
0.000468 * wrench[i] + 0.0305 ) * wrench[i];
                    }
                    else if ( wrench[i] < 0 ) {
                        ret[i] = ( 0.000013317 * pow( wrench[i] , 2 ) +
0.0013 * wrench[i] + 0.0528 ) * wrench[i];
                    }
                    else {
                        ret[i] = 0.0;
                    }
                }
                else {
                    double th_rpms = 0;
                    double voltage = 0;

                    if ( wrench[i] > 0 ) {
                        // Taking in account the actual surge velocity
                        if ( ( pow( _c2 * feedback.surge_velocity , 2 ) + 4
* _c1_f * wrench[i] ) > 0 ) {
                            th_rpms = ( ( _c2 * feedback.surge_velocity +
pow( fabs( pow( ( _c2 * feedback.surge_velocity ) , 2 ) + 4 * _c1_f *
wrench[i] ) , 0.5 ) ) / 2 ) * 60;
                        }
                        else if ( ( pow( _c2 * feedback.surge_velocity , 2
) + 4 * _c1_f * wrench[i] ) < 0 ) {
                            th_rpms = ( ( _c2 * feedback.surge_velocity -
pow( fabs( pow( ( _c2 * feedback.surge_velocity ) , 2 ) + 4 * _c1_f *
wrench[i] ) , 0.5 ) ) / 2 ) * 60;
                        }

                        // Applying the thruster model (1st part)
                        if (th_rpms > 0 ) {
                            voltage = _c3 * th_rpms + _c4 *
feedback.thrusters_current[i];
                        }
                    }
                    else if ( wrench[i] < 0 ) {
                        // Taking in account the actual surge velocity
                        if ( ( pow( _c2 * feedback.surge_velocity , 2 ) - 4
* _c1_b * wrench[i] ) > 0 ) {
                            th_rpms = ( ( _c2 * feedback.surge_velocity -
pow( fabs( pow( ( _c2 * feedback.surge_velocity ) , 2 ) - 4 * _c1_b *
wrench[i] ) , 0.5 ) ) / 2 ) * 60;
                        }
                        else if ( ( pow( _c2 * feedback.surge_velocity , 2
) + 4 * _c1_b * wrench[i] ) < 0 ) {
                            th_rpms = ( ( _c2 * feedback.surge_velocity +
pow( fabs( pow( ( _c2 * feedback.surge_velocity ) , 2 ) - 4 * _c1_b *
wrench[i] ) , 0.5 ) ) / 2 ) * 60;
                        }

                        // Applying the thruster model (1st part)
                        if ( th_rpms < 0 ) {
```

```
                            voltage    =    _c3    *    th_rpms    -    _c4    *
feedback.thrusters_current[i];
                        }
                    }
                    else {
                        ret[i] = 0.0;
                    }

                    if    (    (    voltage    !=    0    )    and    (
feedback.thrusters_voltage[i] > 0 ) ) {
                        // Applying the thruster model (2nd part)
                        ret[i] = voltage / feedback.thrusters_voltage[i];
                    }
                }
            }
            // Saturate between -1.0 and 1.0
            saturate( ret[i], 1.0, -1.0 );
            // std::cout << "ret saturate " << i << ": " << ret[i] << "\n";
        }
        return ret;
    }

    void
    mergeSurgeYaw( double& surge,
                   double& yaw,
                   const struct _AllocatorFeedback& feedback ) {
        /*  If the composition of Surge (v[0]) and Yaw (v[5]) overrides
            the maximum force per thruster,  Yaw is respected and Surge is
            reduced. */

        if ( simulation == false ) {

            std::vector< double > th_max_force( 3 );
            std::vector< double > th_min_force( 3 );

            for( unsigned int i = 1; i < feedback.thrusters_voltage.size();
i++ ) {
                // Maximum RPMs to get a setpoint of +1 considering the
voltage and the maximum current (15 A)
                double rpms = ( feedback.thrusters_voltage[i] - _c4 * 15.0
) / _c3;

                // Compute the maximum force that can be done according to
the velocity feedback
                th_max_force[i] = ( pow( ( rpms / 60 ), 2 ) - _c2 * ( rpms
/ 60 ) * feedback.surge_velocity ) / _c1_f;
                th_min_force[i] = ( pow( ( rpms / 60 ), 2 ) + _c2 * ( rpms
/ 60 ) * feedback.surge_velocity ) / _c1_b;
                //std::cout << i << " Max th " << " : " << th_max_force[i]
<< "\n";
                //std::cout << i << " Min th " << " : " << th_min_force[i]
<< "\n\n";

                // Saturate the computed max and min forces
                saturate(  th_max_force[i],  _max_force_thruster_forward,  0
);
                saturate(  th_min_force[i],  _max_force_thruster_backward,  0
);
                //std::cout << i << " Max th " << " : " << th_max_force[i]
<< "\n";
```

```
                //std::cout << i << " Min th " << " : " << th_min_force[i]
<< "\n\n";
            }

            // Compute the maximum symmetric force for each thruster taking
in account the surge force
            // The maximum force at yaw is not equal for different forces
of surge

            double average = 0; // surge force that lets do the maximum
torque at yaw
            double yaw_limit = 0;

            if ( yaw > 0 ) {

                yaw_limit = 2 * _thruster_distance_yaw * std::min(
th_min_force[1], th_max_force[2] );
                //std::cout << "AAAA " << yaw_limit << "\n";

                average = ( -th_min_force[1] + th_max_force[2] ) / 2;

                 if ( average > 0 ) {
                    if ( ( surge / 2 ) >= average ) {
                        yaw_limit = 2 * _thruster_distance_yaw * (
th_max_force[2] - average );
                        //std::cout << "A \n";
                    }
                    else if ( ( ( surge / 2 ) > 0 ) and ( ( surge / 2 ) <
average ) ) {
                        yaw_limit = 2 * _thruster_distance_yaw * ( ( surge
/ 2 ) + th_min_force[1] );
                        //std::cout << "B \n";
                    }
                    else {
                        yaw_limit = 2 * _thruster_distance_yaw *
th_min_force[1];
                        //std::cout << "C \n";
                    }
                }
                else if ( average < 0 ) {
                    if ( ( surge / 2 ) <= average ) {
                        yaw_limit = 2 * _thruster_distance_yaw * (
th_max_force[2] - average );
                        //std::cout << "D \n";
                    }
                    else if ( ( ( surge / 2 ) < 0 ) and ( ( surge / 2 ) >
average ) ) {
                        yaw_limit = 2 * _thruster_distance_yaw * (
th_max_force[2] - ( surge / 2 ) );
                        //std::cout << "E \n";
                    }
                    else {
                        yaw_limit = 2 * _thruster_distance_yaw *
th_max_force[2];
                        //std::cout << "F \n";
                    }
                }
            }
            else if ( yaw < 0 ) {
```

```
                yaw_limit  =  -  2  *  _thruster_distance_yaw  *  std::min(
th_max_force[1], th_min_force[2] );
                //std::cout << "BBBB " << yaw_limit << "\n";

                average = ( th_max_force[1] - th_min_force[2] ) / 2;

                if ( average > 0 ) {
                    if ( ( surge / 2 ) >= average ) {
                        yaw_limit  =  2  *  _thruster_distance_yaw  *  (
th_max_force[1] - average );
                        //std::cout << "A \n";
                    }
                    else if ( ( ( surge / 2 ) > 0 ) and ( ( surge / 2 ) <
average ) ) {
                        yaw_limit = 2 * _thruster_distance_yaw * ( ( surge
/ 2 ) + th_min_force[2] );
                        //std::cout << "B \n";
                    }
                    else {
                        yaw_limit  =  2  *  _thruster_distance_yaw  *
th_min_force[2];
                        //std::cout << "C \n";
                    }
                }
                else if ( average < 0 ) {
                    if ( ( surge / 2 ) <= average ) {
                        yaw_limit  =  2  *  _thruster_distance_yaw  *  (
th_max_force[1] - average );
                        //std::cout << "D \n";
                    }
                    else if ( ( ( surge / 2 ) < 0 ) and ( ( surge / 2 ) >
average ) ) {
                        yaw_limit  =  2  *  _thruster_distance_yaw  *  (
th_max_force[1] - ( surge / 2 ) );
                        //std::cout << "E \n";
                    }
                    else {
                        yaw_limit  =  2  *  _thruster_distance_yaw  *
th_max_force[1];
                        //std::cout << "F \n";
                    }
                }
            }

            // Saturate the maximum symmetric force at heave with surge
force zero
            saturate( yaw, yaw_limit, -yaw_limit );

            // Maximum yaw forces stored in thrusters variables
            double r_thruster = - ( yaw / _thruster_distance_yaw / 2.0 );
            double l_thruster = ( yaw / _thruster_distance_yaw / 2.0 );

            // Compute the maximum available force at surge and add it to
each thruster
            if ( surge > 0 ) {
                double  max_increase  =  std::min(  th_max_force[1]  -
r_thruster,
                                                   th_max_force[2]  -
l_thruster );
                //std::cout << "Max Inc " << max_increase << "\n";
```

```
                if ( ( surge / 2 ) > max_increase ) {
                    r_thruster += max_increase;
                    l_thruster += max_increase;
                    //std::cout << "W \n";
                }
                else {
                    r_thruster += surge / 2;
                    l_thruster += surge / 2;
                    //std::cout << "X \n";
                }
            }
            else if ( surge < 0 ) {
                double   max_decrease   =   std::max(   -th_min_force[1]   -
r_thruster,
                                                    -th_min_force[2]          -
l_thruster );
                //std::cout << "Max Dec " << max_decrease << "\n";

                if ( ( surge / 2 ) < max_decrease ) {
                    r_thruster += max_decrease;
                    l_thruster += max_decrease;
                    //std::cout << "Y \n";
                }
                else {
                    r_thruster += surge / 2;
                    l_thruster += surge / 2;
                    //std::cout << "Z \n";
                }
            }

            //std::cout << "Left th: " << l_thruster << "\n";
            //std::cout << "Right th: " << r_thruster << "\n";

            // Compose again surge force and yaw torque
            yaw = ( l_thruster - r_thruster ) * _thruster_distance_yaw;
            surge = l_thruster + r_thruster;
        }
        else {
            // Saturate surge
            saturate( surge,
                      2.0 * _max_force_thruster_forward,
                      -2.0 * _max_force_thruster_backward );

            // Saturate yaw
            double min_force = std::min( _max_force_thruster_forward,
                                         _max_force_thruster_backward );

            saturate( yaw,
                      2.0 * min_force * _thruster_distance_yaw,
                      -2.0 * min_force * _thruster_distance_yaw );

            // Compose left and rigth thruster forces
            double   l_thruster   =   (   surge   /   2.0   )   +   (   yaw   /
_thruster_distance_yaw / 2.0 );
            double   r_thruster   =   (   surge   /   2.0   )   -   (   yaw   /
_thruster_distance_yaw / 2.0 );

            // Check limits
            double diff = fabs( l_thruster - r_thruster );
```

```
            if       (diff       >       _max_force_thruster_forward       +
_max_force_thruster_backward) {
                if( l_thruster > r_thruster ) {
                    l_thruster = _max_force_thruster_forward;
                    r_thruster = -_max_force_thruster_backward;
                }
                else {
                    r_thruster = _max_force_thruster_forward;
                    l_thruster = -_max_force_thruster_backward;
                }
            }
            else {
                if( l_thruster > r_thruster ) {
                    if( l_thruster > _max_force_thruster_forward ) {
                        l_thruster = _max_force_thruster_forward;
                        r_thruster = _max_force_thruster_forward - diff;
                    }
                    if( r_thruster < -_max_force_thruster_backward ) {
                        r_thruster = -_max_force_thruster_backward;
                        l_thruster = -_max_force_thruster_backward + diff;
                    }
                }
                if( r_thruster > l_thruster ) {
                    if( r_thruster > _max_force_thruster_forward ) {
                        r_thruster = _max_force_thruster_forward;
                        l_thruster = _max_force_thruster_forward - diff;
                    }
                    if( l_thruster < -_max_force_thruster_backward ) {
                        l_thruster = -_max_force_thruster_backward;
                        r_thruster = -_max_force_thruster_backward + diff;
                    }
                }
            }
            // Compose again surge force and yaw torque
            yaw = ( l_thruster - r_thruster ) * _thruster_distance_yaw;
            surge = l_thruster + r_thruster;
        }
    }


    void
    saturate( double& value,
              const double max_value,
              const double min_value ) {
        if( value > max_value ) value = max_value;
        if( value < min_value ) value = min_value;
    }
};

#endif // __SPARUS2THRUSTERALLOCATOR_CLASS__
```

E.2.4. Programa del model dels timons de profunditat

```
#ifndef __SPARUS2FINALLOCATOR_CLASS__
#define __SPARUS2FINALLOCATOR_CLASS__
#include <algorithm>
#include <map>
```

```
#include <Eigen/Dense>
#include <Eigen/SVD>
#include <sstream>
#include "Request.hpp"

#include <math.h>
#include <complex>


class FinAllocatorSPARUS2 {
public:
    FinAllocatorSPARUS2( unsigned int n_fins ):
                         _n_fins( n_fins ),
                         _old_ret( n_fins ),
                         _is_init( false )
    {
        // Load params
        _max_fin_force = 26.15;
        _max_fin_angle = 40;
        _min_fin_angle = -40;
        _fin_distance_surge = 0.65;
        _fin_distance_yaw = 0.14;
        //_force_to_fins_ratio = 10;

        Eigen::MatrixXd fcm( 6, _n_fins );
        fcm << 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.14, 0.14, 0.65, 0.65, 0.0,
0.0;
        std::cout << "fcm: \n" << fcm << "\n";

        // Compute pseudoinverse
        _fcm_inv = ( fcm.transpose() * fcm ).inverse() * fcm.transpose();
    }

    ~FinAllocatorSPARUS2()
    {}

    void
    setParams( const double max_fin_force,
               const double fin_distance_surge,
               const double fin_distance_yaw,
               const double max_fin_angle,
               const double min_fin_angle,
               const              double              force_to_fins_ratio,
//*** remove it!
               const std::vector< double > fcm_values )

    {
        std::cout << "FinsAllocatorSPARUS2 set params\n";

        _max_fin_force = max_fin_force;
        _fin_distance_surge = fin_distance_surge;
        _fin_distance_yaw = fin_distance_yaw;
        _max_fin_angle = max_fin_angle;
        _min_fin_angle = min_fin_angle;
        //_force_to_fins_ratio = force_to_fins_ratio;

        // Init FCM inverse
        std::cout << "fcm_values.size(): " << fcm_values.size() << "\n";
        assert(fcm_values.size() == 6*_n_fins);
        Eigen::MatrixXd fcm( 6, _n_fins );
        for( unsigned int i = 0; i < 6; i++ ) {
```

```
            for( unsigned int j = 0; j < _n_fins; j++ ) {
                fcm(i, j) = fcm_values.at( i*_n_fins + j );
            }
        }

        std::cout << "FCM:\n"<< fcm << "\n";
        _fcm_inv = ( fcm.transpose() * fcm ).inverse() * fcm.transpose();
        std::cout << "FCM inv:\n" << _fcm_inv << "\n";

        std::cout << "FinsAllocatorSPARUS2 initialized!\n";
        _is_init = true;
    }

    Eigen::VectorXd
    calibration( double fins_setpoint )
    {
        Eigen::VectorXd setpoint( 2 );
        setpoint[0] = fins_setpoint * 5/0.0349;
        setpoint[1] = fins_setpoint * 5/0.0349;
        std::cout << "Fins setpoint: \n" << setpoint << "\n";

        // Publish
        return setpoint;
    }

    Eigen::VectorXd
    compute( const Request& wrench, const Eigen::VectorXd& th_set, const
std::vector<double>& twist_feedback )
    {
        if( !_is_init ) return Eigen::MatrixXd::Zero( _n_fins, 1 );

        // Take wrench request if disabled false, otherwise, get 0.0
        //std::cout << "fin allocator\n" << wrench << "\n";
        Eigen::VectorXd wrench_req( 6 );
        for( unsigned int i = 0; i < wrench.getDisabledAxis().size(); i++ )
{
            if( wrench.getDisabledAxis().at(i) ) {
                wrench_req[i] = 0.0;
            }
            else {
                wrench_req[i] = wrench.getValues().at(i);
            }
        }

        // Merge Roll and Pitch
        /*double roll = wrench_req[3];
        double pitch = wrench_req[4];
        mergeRollPitch( roll, pitch );
        wrench_req[3] = roll;
        wrench_req[4] = pitch;*/
        mergeRollPitch( wrench_req[3], wrench_req[4] );
        //std::cout << "wrench:\n" << wrench_req << "\n";

        // Multiply wrench by fin allocation matrix
        Eigen::VectorXd force_per_fin;
        force_per_fin = _fcm_inv * wrench_req;
        //std::cout << "force_per_fin: \n" << force_per_fin << "\n";

        // Get the setpoints of the horizontal thrusters
        Eigen::VectorXd thruster_setpoint( 3 );
        thruster_setpoint[1] = th_set[1]; //right_th
```

```
        thruster_setpoint[0] = th_set[2];  //left_th
        thruster_setpoint[2] = th_set[0];

        std::vector<double> twist ( 6 );
        twist[0] = twist_feedback[0];
        twist[2] = twist_feedback[2];
        //std::cout << "Left thruster: " << horizontal_thruster_setpoint[0]
<< "\n";
        //std::cout      <<      "Right      thruster:      "      <<
horizontal_thruster_setpoint[1] << "\n";

        // Force to setpoint
        Eigen::VectorXd   setpoint   =   forceToSetpoint(   force_per_fin,
thruster_setpoint, twist );
        //std::cout << "setpoint: \n" << setpoint << "\n";

        // Publish
        return setpoint;
    }


private:
    unsigned int _n_fins;
    double _max_fin_force;
    double _fin_distance_surge;
    double _fin_distance_yaw;
    double _max_fin_angle;
    double _min_fin_angle;
    //double _force_to_fins_ratio;
    Eigen::MatrixXd _fcm_inv;
    Eigen::VectorXd _old_ret;
    bool _is_init;



    Eigen::VectorXd
    forceToSetpoint(  Eigen::VectorXd&   wrench,   const   Eigen::VectorXd&
thruster_setpoints, const std::vector< double >& twist )
    {
        Eigen::VectorXd ret( wrench.size() );
        // Compute newtons to setpoints for each fin
        for( unsigned int i = 0; i < wrench.size(); i++ ) {
            //std::cout << "ret " << i << ": " << ret[i] << "\n";

            // Checking the wrench limit
            saturate( wrench[i], 26, -26 );

            // Initialize setpoint
            ret[i] = 0.0;

            if ( twist.at(0) < 0.05 and twist.at(0) > -0.1 and twist.at(2)
> 0.1 and thruster_setpoints[i] < 0.15 ) {

                // With + heave loate fins to +60 degrees
                ret[i] = 60.0;
                std::cout << "TO MAX !!!!!!!!!!!!!!!!!!!!!!!!!!!! \n";
            }
            else  if  (  twist.at(0)  <  0.05  and  twist.at(0)  >  -0.1  and
twist.at(2) < -0.1 and thruster_setpoints[i] < 0.15 ) {

                // With - heave loate fins to -60 degrees
```

```
                ret[i] = -60.0;
                std::cout << "TO MIN !!!!!!!!!!!!!!!!!!!!!!!!!!!!! \n";
            }
            else { // Apply fins model

                // Ensure the output angle if desired torque is 0 despite
model inacuracies
                if ( wrench[i] == 0 ) {
                    ret[i] = 0;
                }
                else {
                    bool saturate_setpoint = false;

                    // Ensure the inputs of the model inside the limits
                    if ( ( thruster_setpoints[i] > 0 ) and (
thruster_setpoints[i] <= 1 ) ) {
                        if ( fabs( wrench[i] ) < _max_fin_force ) {

                            // Fins model ->
angle(force_per_fin,th_setpoint): Output in degrees
                            std::complex<double> angle = pow( 268140.86 *
pow ( thruster_setpoints[i], 2 ) - 21481.84 * thruster_setpoints[i] +
1544.68 * -fabs(wrench[i]) + 832.42 , 0.5 ) - 517.82 *
thruster_setpoints[i] + 19.51;

                            // Avoiding bad values due to computations out
of the model
                            if ( ( std::isfinite( std::real( angle ) ) )
and ( std::imag( angle ) == 0 ) ) {
                                // Avoiding bad values ( the model has to
give negative angles )
                                if ( std::real( angle ) < 0.0 ) {
                                    // At this point the given angle value
for the model is ok, just update the angle sign depending on the wrench
request
                                    //std::cout << i << " Force: " << -
fabs(wrench[i]) << " Setpoints: " << thruster_setpoints[i] << " Angle: " <<
angle << "\n";
                                    if ( wrench[i] < 0 ) {
                                        ret[i] = std::real( angle );
                                    }
                                    else if ( wrench[i] > 0 ) {
                                        ret[i] = -std::real( angle );
                                    }
                                }
                                else {
                                    // Set to zero degrees if there is any
inaccuracy in the model
                                    ret[i] = 0;
                                }
                            }
                            else {
                                // If there is imaginary part the model has
computed too small thruster setpoint for the asked force, so
                                // the output angle is saturated
                                saturate_setpoint = true;
                            }
                        }
                        else {
                            //If more force is asked, saturate
                            saturate_setpoint = true;
```

```
                        }

                        // Saturate the value if it is required
                        if ( saturate_setpoint ) {
                            if ( wrench[i] < 0 ) {
                                ret[i] = _min_fin_angle;
                            }
                            else if ( wrench[i] > 0 ) {
                                ret[i] = _max_fin_angle;
                            }
                        }
                    }
                    else {
                        //  Set  to  zero  degrees  if  the  model  can  not
computed
                        ret[i] = 0;
                    }
                }
                 //std::cout << "ret normalized " << i << ": " << ret[i] <<
"\n";

                // Saturate between fins angle limits
                saturate ( ret[i], _max_fin_angle, _min_fin_angle );
                //std::cout << "saturated ret " << i << ": " << ret[i] <<
"\n";
            }
        }
        return ret;
    }

    void
    mergeRollPitch( double& roll, double& pitch ) {
        //  If  the  composition  of  Pitch  and  Roll  torques  override  the
maximum force
        // per fin, Roll torque is respected and Pitch torque is reduced

        // Saturate roll torque
        //std::cout << "Roll input: " << roll << "\n";
        saturate(  roll,  2  *  _max_fin_force  *  _fin_distance_yaw,  -  2  *
_max_fin_force * _fin_distance_yaw );
        //std::cout << "Saturated roll: " << roll << "\n";

        // Saturate pitch torque
        //std::cout << "Pitch input: " << pitch << "\n";
        saturate( pitch, 2 * _max_fin_force * _fin_distance_surge, - 2 *
_max_fin_force * _fin_distance_surge );
        //std::cout << "Saturated pitch: " << pitch << "\n";

        double l_fin = 0.0;
        double r_fin = 0.0;

        if ( roll != 0.0 ) {
            l_fin = l_fin - ( ( roll / _fin_distance_yaw ) / 2 );
            r_fin = r_fin + ( ( roll / _fin_distance_yaw ) / 2 );
        }

        if ( pitch != 0 ) {
            l_fin = l_fin + ( ( pitch / _fin_distance_surge ) / 2 );
            r_fin = r_fin + ( ( pitch / _fin_distance_surge ) / 2 );
        }
```

```
        // Check limits
        double diff = 0.0;

        if ( fabs( l_fin ) > _max_fin_force ) {
            diff = l_fin - ( fabs( l_fin ) / l_fin ) * _max_fin_force;
        }

        if ( fabs( r_fin ) > _max_fin_force ) {
            diff = r_fin - ( fabs( r_fin ) / r_fin ) * _max_fin_force;
        }

        if ( diff != 0.0 ) {
            l_fin = l_fin - diff;
            r_fin = r_fin - diff;
        }

        //std::cout << "Left fin force: " << l_fin << "\n";
        //std::cout << "Right fin force: " << r_fin << "\n";

        // Compose again pitch and roll torques
        roll = ( -l_fin + r_fin ) * _fin_distance_yaw;
        pitch = ( l_fin + r_fin ) * _fin_distance_surge;
    }

    void
    saturate( double& value,
              const double max_value,
              const double min_value )
    {
        if( value > max_value ) value = max_value;
        if( value < min_value ) value = min_value;
    }
};

#endif // __SPARUS2FINALLOCATOR_CLASS__
```