ICTINEU^{AUV} Takes the Challenge

D. Ribas, N. Palomeras., X. Ribas, G. García de Marina, E. Hernàndez, F.Chung, N. Hurtós,

J. Massich, A. Almohaya and J. Vila.

Institute of Informatics and Applications. University of Girona,
Campus de Montilivi
Girona, CP:17071, Spain
dribas@eia.udg.es

Abstract - *Within 1857-1868 Narcis Monturiol developed the first Spanish submarine. Almost one hundred and fifty years later, a pioneer team of students of the University of Girona decided to design and develop an Autonomous Underwater Vehicle (AUV) to face the Student Autonomous Underwater Challenge – Europe (SAUC-E). Our robot, called ICTINEU^{AUV}, is the result of a multidisciplinary project involving undergraduate, graduate and PhD students of industrial engineering and computer science. The prototype has evolved from the initial Computer Aided Design (CAD) model to become an operative Autonomous Underwater Vehicle (AUV) in the short period of seven months. The open frame and modular design principles, together with the compatibility with other robots previously developed at the lab, have been the main design philosophy. Hence, at the robot core, two networked computers give access to a wide set of sensors and actuators. The Gentoo/Linux distribution, with a 2.6 pre-emptive kernel has been chosen as the onboard operating system. The real-time POSIX, together with the ACE/TAO CORBA-RT ORB, have been extensively used to develop the control architecture as a set of distributed objects with soft real time capabilities. Common software engineering practises have been applied to ensure software reliability including Unified Modelling Language (UML) design, extensive documentation using DoxyGen, and the use of a Code Version Server (CVS) to handle the sharing of multiple code versions. Finally, in order to reduce the development time, concurrent engineering techniques based on Hardware In the Loop (HIL) simulation have been applied to overlap the hardware and software design and development.*

I. INTRODUCTION

From 1990, the Association for Unmanned Vehicle System International (AUVSI) promotes the design and development skills about Autonomous Underwater Vehicles by means of an annual competition for the USA students. Inspired in this competition, the Defence Science and Technology Lab (DSTL), the Heriot Watt University and the National Oceanographic Centre of Southampton have organized the first Student Autonomous Underwater Challenge Europe (SAUC-E). SAUC-E is a competition for students Europe-wide to foster the research and development in underwater technology. In this first edition it is expected that ten teams will take the challenge, four of them from abroad the UK (the place where the competition will take place). Last January, a team of students collaborating with the Underwater Robotics Lab of the University of Girona, decided to form the VICOROB-UdG^{TEAM} to face the challenge. Our team has designed a new AUV, which aims to pay homage to Narcis Monturiol, the developer of ICTINEU, the first Spanish submarine, from which our robot takes its name. Given the short period of time to invest in the project, our team has decided to overlap the hardware and the software development (concurrent engineering) taking profit of a hardware in the loop (HIL) simulator. This paper describes the ICTINEU^{AUV} as an entry to the SAUC-E competition. At the time of writing this paper, the robot is totally set-up and a first version of the software is running on board. Nevertheless there still a lot of work to do to achieve the mission tasks. For this reason, most of the reported results are based on HIL simulation.

The paper is organized as follows. The mechanical, the hardware and the software design are explained in sections II to IV. Section V explains the map-based navigation system and section VI the image processing algorithms used for target detection and tracking. Section VII, presents the control architecture and section VIII presents the HIL simulator. Finally, section IX and X present the mission and the results respectively before concluding in section XI.

Figure. 1 ICTINEU^{AUV} during a pool test.

II. MECHANICAL DESIGN

While survey missions covering a wide search area advocate for fly-type vehicles propelled by thrusters and steered by rudders and fins, the SAUC-E mission takes place in a bounded small area where a high manoeuvrability is required. In this situation a hover-type vehicle propelled and steered by thrusters is the most desirable configuration. The classical open frame design, commonly adopted by commercial ROVs, together with a modular design of the components, conveniently housed in pressure vessels, is probably the most simple and reliable approach for the design. Although hydrodynamics of open frame vehicles is known to be worst than the hydrodynamics of close hull type vehicles, it is simpler and cheaper, being very easy to upgrade and to maintain. Moreover, for slow moving robots hydrodynamics of the open frame design does not pose any problem at all. With the aim of designing a very manoeuvrable vehicle, easy to maintain and to modify during the project, ICTINEU^{AUV} has adopted the open frame design (fig. 1). Our robot is propelled by four thrusters. It can move in the heave and sway directions depending on the composition of forces generated by the vertical thrusters (see fig. 2). On the other hand, horizontal thrusters are used to move forward (surge DOF) as well as to change the heading (yaw DOF). Hence, the prototype is a full actuated vehicle in four DOF (surge, sway, heave and yaw, see fig. 2), while being passively stable in Roll and Pitch (its meta-centre is above the centre of gravity).

The robot chassis is made of Delrin material. This is an engineering resin which, due to its excellent mechanical properties can compete with metals in many applications. Its very low water absorption and the small effect of aqueous solutions on its properties make it an excellent candidate for our project.

Three pressure vessels are used for holding the electronics. The two bigger cylinders are made with aluminium while the smaller one is made with Delrin. All of them have a cover with all the connectors and use a conventional O-ring rubber for water sealing. One of the cylinders houses the computers, other the thrusters' controllers and the batteries, and the last one encapsulates the MRU.

The thrusters are built using MAXON DC motors of 250 Watts of power, using planetary gears and contained in stainless steel housings. Three blade propellers, made of brass, are linked to the motor through a stainless steel shaft mechanically sealed, providing around 14.7/14.2 Newtons of forward/backward thrust.

Buoyancy of the robot is provided by a cover of technical foam, with 10.5 litres of volume, and a weight of 0.6 Kg. This foam can withstand pressures up to eleven bars, which corresponds to a depth of one hundred meters.

Table 1 gives the main characteristics of ICTINEU^{AUV}.

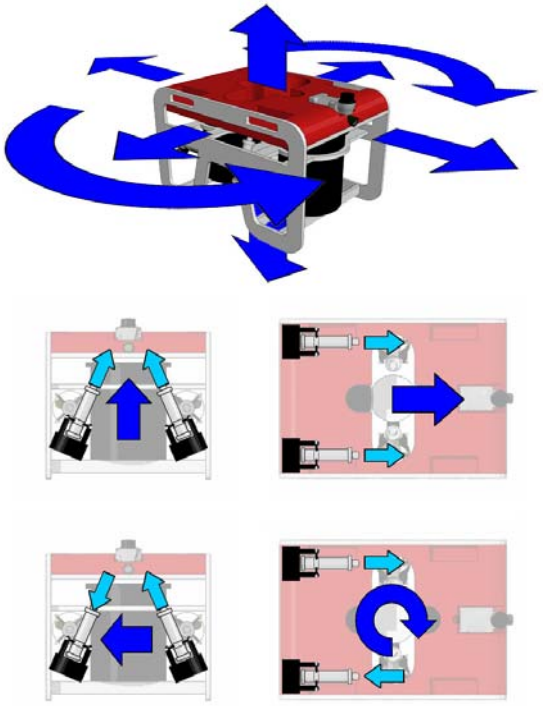


Figure. 2 Composition Forces and DOF.

III. HARDWARE DESIGN

The robot hardware (Apendix I) is composed by a Computers module and a Power module.

A. Computers Module

Two PCs, one for control and one for image and sonar processing, connected through a 100 MBs switch form the core of the robot hardware. The control PC is an AMD GEODE-300MHz, powered with 50 W power supply module. The PC104 stack also incorporates an A/D and digital I/O card with 8 analogue input channels, 4 analogue output channels and 24 digital I/O.

ICTINEU^{AUV}	
Dimensions	74 x 47 x 53 cm
Dry Weight	≅ 48 Kg
Umbilical	*Ethernet*, power supply
DOFs	4 (*Surge, Sway, Heave, Yaw*)
Hardware	PC-104 / Via C3 @ 1Gz
Software	GNU/Linux + RTAI; CORBA-RT ACE-TAO
Sensors	*Imaging Sonar; DVL;* MRU *MTi* from *XSens Technologies; 2* underwater cameras, depth sensor, set of hydrophones, echo sounder, water detectors and temperature sensors

Table 1. ICTINEU^{AUV} characteristics.

The 4 analogue outputs are used to send the velocity set-points to the thrusters' drivers while the 4 analogue inputs feedback the velocity of the propellers. Two more analogue inputs are used to monitor the temperature and the pressure (for safety purposes) within the pressure vessel. With respect to the digital signals, 4 outputs are used to enable/disable the thrusters' drivers, 1 input is used to read the mission switch which signals the start of the mission, 4 more inputs are connected to the water leakage detectors and 1 digital output is used to signal when to throw the marker. Two serial lines provide access to the Argonaut Doppler Velocity Log (DVL) and the MTi Motion Reference Unit (MRU) sensors.
The mini-ITX computer is a Via C3 1 GHz Pentium clone connected to the imaging sonar through a high speed serial line. A cheap PCTV110 from Pinnacle is used for image processing. Since our design uses two cameras (one looking forward and one looking down), both input channels, the composite video and the S-video, are used.

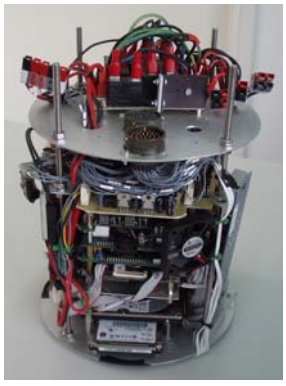


Figure. 3. Computer module.

B. Power module.

The power module contains the four power drivers for the thrusters as well as a pack of 2 cheap and sealed lead-acid batteries. A DC-DC converter is included to provide a stabilized voltage to the rest of components. There is also a simple relay circuit which commutes between the internal and the external power. External power, supplied through an optional umbilical, is very useful for running long term experiments before the competition. It is worth noting that the batteries have been dimensioned for the short time experiments to be done during the competition days. Moreover, when the robot works with external power it can recharge the internal batteries.

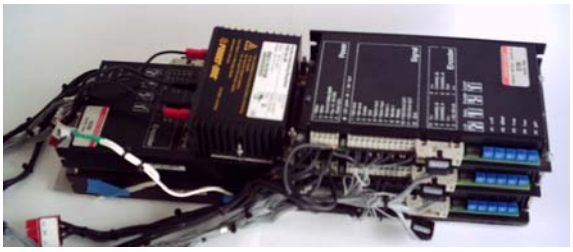


Figure 4. Detail of the power drivers included in the power module.

C. Umbilical Cable

As mentioned above, the robot also incorporates an umbilical cable providing power and Ethernet signals with the aim of aiding the system development in the lab.

IV. SOFTWARE DESIGN

The software architecture (Apendix II) has the task of guaranteeing the AUV functionality. It is built with a set of objects distributed among the two onboard PCs and the external PC. The last one is only used during the experiments in the lab, being connected to the robot through the umbilical cable for monitoring purposes. The architecture [Hernàndez 2005] is composed by a base system and a set of objects customized for the desired robot (see fig. 5). There are classes providing soft real-time capabilities, this is guaranteeing the period of execution of the periodic tasks like the controllers or the sensors. Whenever a thread is not able to guarantee the assigned period (overrun), a signal is raised allowing accounting for the total number of overruns. Consulting this number it is possible to know if the time constrains of the different tasks are satisfied or not, allowing the programmer to introduce the needed modifications to solve the problem. Another important part of the base systems are the loggers. The logger system is used to log data from sensors, actuators or any other object component. Loggers do not execute in real time. They are background processes which receive

the data from real time objects. Their role consists on packing the data and saving them into files. It is worth noting that although loggers do not run in real time, the data has a time-stamp corresponding to the gather time. Moreover all the computers in the network are synchronized by means of the NTP (Network Time Protocol) and hence all the data coming from different sensors can be time related.
The software architecture is divided in three modules (fig. 5): (1) Robot interface module, (2) Perception module and (3) Control module. In the following subsections these modules are briefly introduced.

A. Robot interface module
This is the unique module that contains software objects that dialog with the hardware. There are basically to types of objects: (1) sensor objects responsible for reading data from sensors and (2) actuator objects responsible for sending commands to the actuators.
Sensor objects for ICTINEU^{AUV} include the DVL, the imaging sonar, the MRU, both cameras, the depth sensor, and the echo sounder. There are also objects for the safety sensors like the water leakage detectors and internal temperature and pressure sensors that allow for the monitoring of the conditions within the pressure vessels.
Actuator objects for the ICTINEU^{AUV} include the thrusters, and the marker thrower.

B. Perception module.
This module contains two basic components: (1) the Navigator and (2) the Obstacle Detector. The Navigator object has the goal of estimating the position of the robot. To accomplish this task, there exists an interface called NavigationSensor from which all the localization sensors (DVL, MRU, depth sensor) inherit. This interface provides to all these sensors a set of methods to return the position, velocity and acceleration in the six DOF together with an estimation of the quality of these measurements. The Navigator can be dynamically connected to any NavigationSensor and, using the quality factor fuses the data to obtain a more accurate position, velocity and acceleration. The Control module can use the navigation data provided by the Navigator keeping the behaviours independent of the physical sensors being used for the localization.
The Obstacle detector uses the same philosophy to provide the obstacles position in the world fixed frame. The Obstacle detector is also used to detect the distance between the vehicle and the bottom of the pool. Detecting horizontal obstacles is possible using the imaging sonar, and the pool bottom obstacle can be detected with the DVL sensor.

C. Control module.
The control module receives sensor inputs from the perception module and sends command outputs to the Actuators residing in the Robot Interface Module.
Since task and behaviours are words that are interpreted in different ways for different authors in the literature, hereafter we describe how they are interpreted within our project. A behaviour is function that maps the sensor input space (stimuli) into a velocity setpoint (behaviour response) for the robot low level controller. The behaviour response is chosen in a way that drives the robot towards its corresponding goal. In this way, the goal corresponding to the *keepDepth* behaviour is considered to be achieved when the robot is within an interval around the desired depth. A task is a set of behaviours that are enabled together to achieve a more complex goal. For instead, *KeepDetph* and *Motion2D* can work together to allow for planar navigation.
The control module follows the principles of the hybrid control architecture organized in three layers: (1) Mission Level, (2) Task Level and (3) Vehicle Level. The vehicle level is composed by

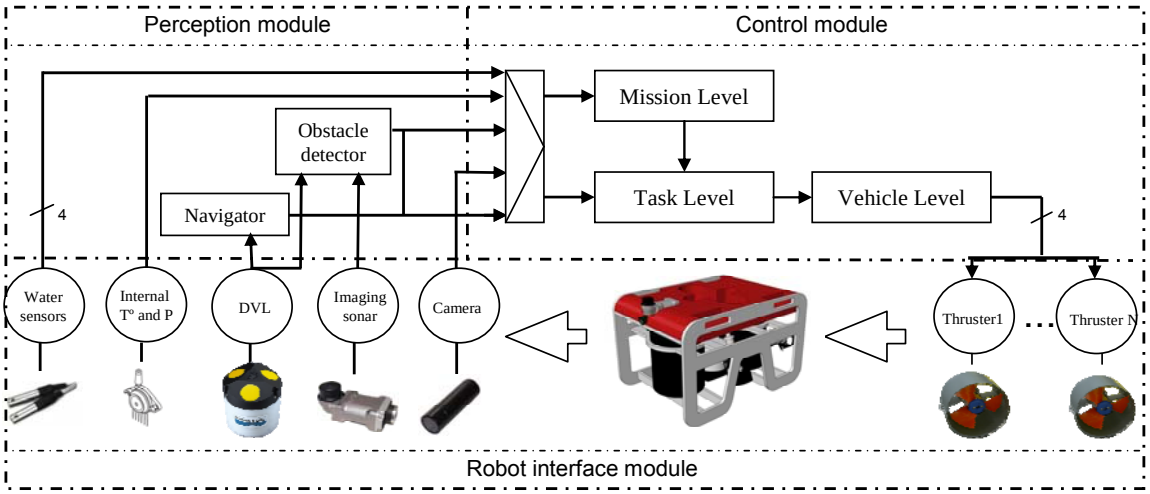


Figure. 5 Software Architecture

a MIMO PID velocity controller for each DOF. The Task level is a conventional behavioural layer including several behaviours running in parallel and a coordinator module which fuses all the behaviour responses into a unique coordinated response to be used as velocity set point for the low level controller. Finally, the upper layer (mission level) is responsible for the sequencing of the mission tasks, selecting for each mission phase the set of behaviours that must be enabled as well as their parameters.

V. NAVIGATION

The localization is expected to be carried out by merging the information of several sensors with an *a priori* map of the environment. First, an imaging sonar is used to obtain a complete acoustic image where the walls of the water tank are detected and hence, it is possible to solve for the robot initial position. Then, data from the imaging sonar and the map, together with measurements from the DVL, are merged within an EKF to obtain a position estimate. In the following subsections the general structure of the localization algorithm are described.

A. Initialization

In order to perform localization using an *a priori* map it is necessary to determine the initial position of the vehicle within the mapped environment. To do this, a complete acoustic image of the surroundings is obtained and compared with the map to estimate the vehicle position. Mechanically scanning sonars perform scans in a 2D plane by rotating a sonar beam through a series of small angle steps. For each emitted beam, distance vs. echo-amplitude data is returned forming an acoustic image of the surroundings (fig 6). The scanning rate of these devices is really slow in comparison with multibeam sonars. For this reason, the vehicle movement along a complete scan usually induces important distortions in the acoustic image (fig 6 b). Extracting features from this kind of images produces inaccuracies and yield to poor results. To cope with the slow scanning rate of the low cost imaging sonars, we propose a 2 step line extraction procedure. First, the trajectory of the vehicle is estimated at the same time that the acoustic beams are grabbed. Then, when the position of each beam is known, the distortion induced by motion is compensated (fig 6 c). Since objects present (walls) in the environment appear as high echo-amplitude returns, a thresholding is applied to discard low intensity returns which contain no significant information. Then, with the remaining measurements, the Hough transform [Duda and Hart 1972] is used to extract a set of line features $\mathcal{O} = \{\mathbf{l}_{o_1}, \mathbf{l}_{o_2}, \ldots, \mathbf{l}_{o_i}\}$. As the lines that compose the *a priori* map $\mathcal{M} = \{\mathbf{l}_{m_1}, \mathbf{l}_{m_2}, \ldots, \mathbf{l}_{m_j}\}$ are perfectly known, an analytic association process can be carried out so each observed feature $\mathbf{l}_{o_i}$ can be related to its corresponding feature $\mathbf{l}_{m_j}$ in the map. Finally, an Information Filter (IF) is used to merge this information into an estimate of the initial vehicle position and its uncertainty with respect to the *a priori* map.

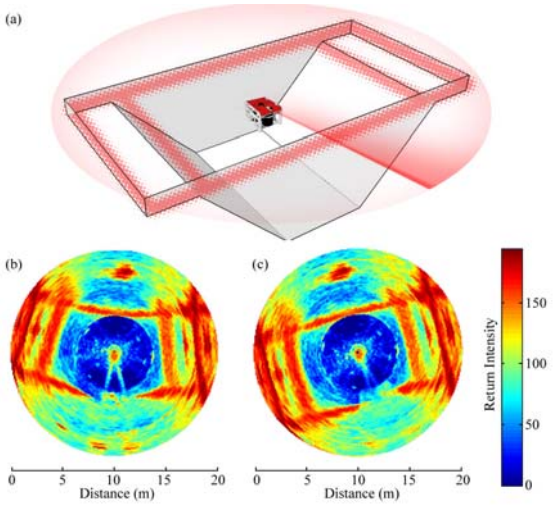


Figure 6. (a) Schematic representation of the environment where the sonar data were gathered (see fig. 13). The highlighted zones represent the expected sonar returns. Images generated from acoustic data, (b) distorted and (c) undistorted image through DVL integration.

B. Vehicle pose estimation using a DVL

The SonTek Argonaut DVL unit which includes a compass, 2 inclinometers and a depth sensor is used to estimate the robot pose (navigation problem). Imaging sonar beams are read at 30 Hz while DVL readings arrive asynchronously at a frequency within 1.5 Hz interval. An EKF is used to estimate the 6DOF robot pose whenever a sonar beam is read. DVL readings are used asynchronously to update the filter. To reduce noise inherent to the DVL measurements, a simple 6DOF constant velocity kinematics model is used instead of a more conventional dead reckoning method.

The information of the system at step *k* is stored in the state vector $\mathbf{x}_k$ with estimated mean $\hat{\mathbf{x}}_k$ and covariance $\mathbf{P}_k$:

$$\hat{\mathbf{x}}_k = \left[\hat{\eta}^B, \hat{\nu}^R\right]^T \quad \mathbf{P}_k = E\left[\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)^T\right] \quad (1)$$

with:

$$\eta^B = [x, y, z, \phi, \theta, \psi]^T; \quad \nu^R = [u, v, w, p, q, r]^T \quad (2)$$

where, as defined in [Fossen 2002], η^B is the position and attitude vector referenced to a base frame B, and ν^R is the linear and angular velocity vector referenced to the robot coordinate frame R. If the coordinate frame B is oriented to the north, the compass measurements can be straight forward integrated.

The vehicle movement prediction is performed using a kinematic model such as:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) = \begin{bmatrix} \eta_k^B \\ \nu_k^R \end{bmatrix} = \begin{bmatrix} \eta_{k-1}^B + J(\eta_{k-1}^B)\nu_{k-1}^R T \\ \nu_{k-1}^R \end{bmatrix} \quad (3)$$

$$J(\eta) = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi & 0 & 0 & 0 \\ s\psi c\theta & s\phi s\psi s\theta + c\psi c\phi & s\psi s\theta c\phi - s\phi c\psi & 0 & 0 & 0 \\ -s\theta & c\theta s\phi & c\theta c\phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s\phi t\theta & c\phi t\theta \\ 0 & 0 & 0 & 0 & c\phi & -s\phi \\ 0 & 0 & 0 & 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \quad (4)$$

Although in this model the velocity is considered to be constant, in order to allow for slight movements, the velocity is modelled as the integral of a stationary white noise v_k with a diagonal covariance matrix $\mathbf{Q}$ in the order of magnitude of the maximum acceleration increment that the robot can experiment over a sample period.

$$\nu_k^R = \hat{\nu}_k^R + v_k T \quad (5)$$

$$E[v_k] = 0; \quad E[v_k, v_j^T] = \delta_{k_j}\mathbf{Q} \quad (6)$$

Hence, the acceleration noise is additive in the velocity (eq. 5) and propagates nonlinearly to the position. Finally, the model prediction and update is carried out as detailed below:

1) Prediction: The estimate of the state is obtained as:

$$\hat{\mathbf{x}}_k = f(\hat{\mathbf{x}}_{k-1}) \quad (7)$$

and its covariance matrix as:

$$\mathbf{P}_k = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^T + \mathbf{G}_k \mathbf{Q}_k \mathbf{G}_k^T \quad (8)$$

where $\mathbf{F}_k$ and $\mathbf{G}_k$ are the Jacobian matrices of partial derivatives of the non-linear model function *f* with respect to the state $\mathbf{x}_{R,k}^B$ and the noise v_k, respectively.

2) Update using DVL measurements: The model prediction is updated by the standard Kalman filter equations each time a new DVL measurement arrives:

$$\mathbf{z}_{S,k} = [u_b, v_b, w_b, u_w, v_w, w_w, \phi_i, \theta_i, \psi_c, z_{depth}]^T \quad (9)$$

Where subindex *b* stands for bottom tracking velocity, *w* for through water velocity, *i* for inclinometers and *c* represents the compass. The measurement model is:

$$\mathbf{z}_{S,k} = \mathbf{H}_{S,k}\mathbf{x}_{k|k-1} + w_k \quad (10)$$

$$\mathbf{H}_S = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}\ 0\ 1 & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} \end{bmatrix} \quad (11)$$

where w_k (measurement noise) is a zero-mean white noise:

$$E[w_k] = 0; \quad E[w_k, w_j^T] = \delta_{k_j}\mathbf{R}_{S,k} \quad (12)$$

Since the DVL sensor provides a status measurement for the bottom tracking and water velocity, depending on the quality of the measurements, different versions of the $\mathbf{H}$ matrix are used to fuse one (removing row 2), the other (removing row 1), or both readings (using the full matrix).

C. Correction with the a priori map and an imaging sonar.

Herein, the information obtained from the imaging sonar together with the *a priori* map $\mathcal{M}$ is used to perform a correction of the vehicle state estimate. Whenever a new single beam (not a complete acoustic image) is obtained from the imaging sonar, the highest return is chosen. This measurement is the most likely to pertain to any object present in the scene and as a consequence, to a feature in the *a priori* map. We will use this information to perform an update of the EKF presented above and correct the state estimate. The high intensity return $\mathbf{z}_{P,k}$ is a point represented in polar coordinates with respect to the vehicle frame *R*:

$$\mathbf{z}_{P,k} = [\rho_p, \theta_p]^T; \quad \hat{\mathbf{z}}_{P,k} = \mathbf{z}_{P,k} + u_k \quad (13)$$

Where $\mathbf{z}_{P,k}$ would be the value obtained if noise u_k was not present. The noise u_k is a zero-mean white Gaussian noise:

$$E[u_k] = 0; \quad E[u_k, u_j^T] = \delta_{k_j}\mathbf{R}_{P,k} \quad (14)$$

We need to determine the correspondence between the measurement and the objects in the map. First, the cartesian coordinates of the measurement $\mathbf{z}_{P,k}$ need to be obtained:

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = f(\mathbf{z}_{P,k}) = \begin{bmatrix} \rho_p cos(\theta_p) \\ \rho_p sin(\theta_p) \end{bmatrix} \quad (15)$$

$$\mathbf{S}_k = \mathbf{J}_f \mathbf{R}_{P,k} \mathbf{J}_f^T \text{ where } \mathbf{J}_f = \frac{\partial f}{\partial \mathbf{z}_{P,k}} \quad (16)$$

$\mathbf{S}_k$ represents the uncertainty of the point in cartesian coordinates and $\mathbf{J}_f$ is the Jacobian matrix of the *f* function with respect to the

measurement $\mathbf{z}_{P,k}$. The next step is to define each line $\mathbf{l}_{m_i} = [\rho_{m_i}, \theta_{(m_i)}]^T$ from the map $\mathcal{M}$ by its orientation and its distance to the origin with respect to the vehicle coordinate frame R:

$$\begin{bmatrix} \rho_{m_i}^R \\ \theta_{m_i}^R \end{bmatrix} = g(\mathbf{l}_{m_i}, \mathbf{x}) = \begin{bmatrix} \rho_{m_i} - x^B cos(\theta_{m_i}) - y^B sin(\theta_{m_i}) \\ \theta_i - \psi^B \end{bmatrix} \quad (17)$$

$$\mathbf{J}_g = \frac{\partial g}{\partial \mathbf{x}_k} \quad (18)$$

$\mathbf{J}_g$ is the Jacobian of the g function. As the map is assumed perfectly known, the uncertainty only depends on the vehicle state $\mathbf{x}$. With both the sonar return and the map in the same reference frame, an implicit measurement equation stating that the point belongs to the line can be defined:

$$h(g(\mathbf{x}), f(\mathbf{z})) = \rho_{m_i}^R - x_p cos(\theta_{m_i}^R) - y_p sin(\theta_{m_i}^R) = 0 \quad (19)$$

$$\mathbf{H}_1 = \frac{\partial h}{\partial(\rho_{m_i}^R, \theta_{m_i}^R)} \cdot \mathbf{J}_g; \qquad \mathbf{H}_2 = \frac{\partial h}{\partial(x_p, y_p)} \cdot \mathbf{J}_f \quad (20)$$

Where $\mathbf{H}_1$ and $\mathbf{H}_2$ are the Jacobians of the implicit measurement function h with respect to the sonar return and a line of the map. To produce the update, an association hypothesis between the measurement and one of the lines from the map is needed. For this purpose, an individual compatibility test is performed using the presented measurement equation:

$$D^2 = h_k^T (\mathbf{H}_1 \mathbf{P}_{k|k-1} \mathbf{H}_1^T + \mathbf{H}_2 \mathbf{R}_{P,k} \mathbf{H}_2^T)^{-1} h_k < \chi_{d,\alpha}^2 \quad (21)$$

Distance D^2 is the Mahalanobis distance. The correspondence is accepted if the distance is less than $\chi_{d,\alpha}^2$, with α defined as the confidence level and $d = dim(h)$. The Nearest Neighbour (NN) selection criterion determines that among the features that satisfy eq. 21, the one with the smallest Mahalanobis distance is chosen and the association hypothesis is accepted.
Having the data association solved, an update of the vehicle state estimate can be performed using the EKF equations for an implicit measurement function:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_1^T (\mathbf{H}_1 \mathbf{P}_{k|k-1} \mathbf{H}_1^T + \mathbf{H}_2 \mathbf{R}_{P,k} \mathbf{H}_2^T)^{-1} \quad (22)$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k|k-1} - \mathbf{K}_k h_k (\hat{\mathbf{x}}_k, \hat{\mathbf{z}}_k) \quad (23)$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_1) \mathbf{P}_{k|k-1} \quad (24)$$

For experimental results of the presented localization algorithm please refer to section X B.

VI. TARGET TRACKING

Two of the tasks to be faced during the mission involve the use of image processing algorithms. Both are particular cases of the target tracking problem: (1) navigate towards a midwater buoy and impact it and (2) navigate towards a cross laying on the pool bottom and throw a marker.
Both algorithms are almost the same. In case the object to track is a colour object, the image is converted into the Hue-Saturation-Intensity (HSI) space. Then, robust regions of hue and saturation are selected as the segmentation criteria. Intensity is not used for the segmentation since it is known to be very sensible to changes in the illumination. In case the object is black, then a simple binarization is performed. After the segmentation process, we get a binarized image with two possible values for the pixel (1-object and 0-background). Then, the algorithm defines two exploration increments:

- (Δx_1, Δy_1) used to explore a small window around the object.
- (Δx_2, Δy_2) used to explore the rest of the image.

which allow to easily change the resolution of the image processing algorithm. Initially, we start the search from the left upper corner of the image, detecting clusters of candidate pixels. Of course, due to the noise, more than one cluster can be detected. Hence a constrain in the area (number of pixels of the object) is imposed hopefully filtering the noisy clusters. Once a unique cluster is available, its centre of mass and its area is returned as result. In order to foster the velocity of the algorithm, a window is defined around the object. This window is used in the next iteration to search for the object, effectively reducing the search area. In case the object is not found within the window during *n* consecutive frames, the algorithm re-starts looking for the object in the whole image.

VII. CONTROL ARCHITECTURE

The function of a control architecture is to move the robot autonomously to fulfil a set of goals in a particular order and with some constraints. The result of the whole process at every moment is the movement of the robot in each degree of freedom. Figure 7 shows the schema of the three levels that form the proposed control architecture.

A. Vehicle level: Velocity controller

Since the final task is the movement of the robot, the control architecture has, at its lower level, a classical velocity controller. This controller receives the velocity set points from the task level controller and the measured or estimated velocities from the navigator. The object in charge of doing this task is the PID velocity controller. This object reads the vehicle velocity from the Navigator object (see section 4.B) and receives the velocities set points from the Coordinator Object (see section 4.C).

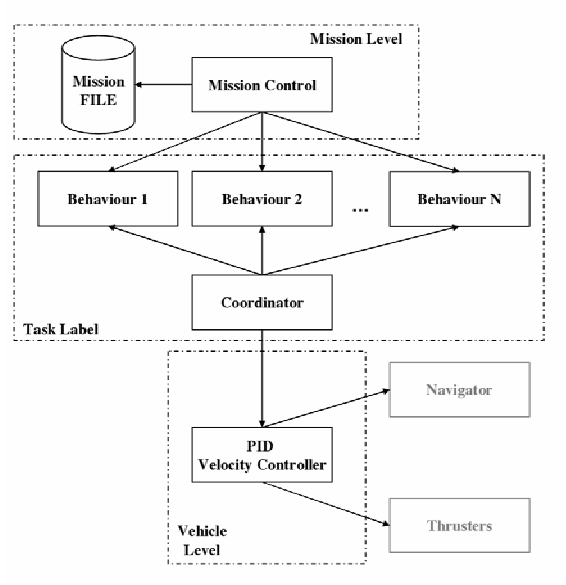


Figure.7 ICTINEU^{AUV} control architecture

B. Task level: Behaviours + Coordinator

A common methodology to implement control architectures is to build a library containing all basic functions that the robot can make. By joining these basic functions it is possible to carry out complex tasks. These basic functions have been named: vehicle primitives, tasks, command primitives … One of the most popular names to express this concept is behaviour, which was introduced in 1986 by Brooks [Brooks 1986] and has been extensively used. The task level controller presented in this paper is based on very simple behaviours. Since more than one behaviour can be activated, a coordinator is always needed in all no pre-emptive architectures. The coordinator is the object in charge of combining the outputs of all active behaviours to generate a single output. Each behaviour is an autonomous process with a particular goal. The input of a behaviour is taken from the perception module of the software architecture. The output contains:

- **The velocity for every DoF** normalized between -1 and 1.
- **The Activation level.** A value normalized between 0 and 1.
- **A priority** that together with the activation level are used by the coordinator to combine the outputs of all behaviours.
- **A blocking value** expressing if the behaviour is blocking the execution thread of the mission controller.

To initialize a behaviour, besides particular values for the parameters, it is needed to setup the following values:

- **The enable:** A boolean variable that indicates if the behaviour is activated or not and, therefore, if its output will be considered by the Coordinator.
- **The priority** that will have the output of this behaviour.
- **The Time Out** which indicates when the behaviour will block the execution thread. If *TimeOut<0*, the behaviour blocks the execution thread until its goal is fulfilled. If *TimeOut=0*, the behaviour doesn't block the execution thread. If *TimeOut>0*, the behaviour blocks the execution thread until *TimeOut* seconds or until its goal is fulfilled.

The object Coordinator is in charge of taking all the enabled behaviour outputs to combine them into a single one, which will be sent to the velocity controller. To combine all the outputs, the coordinator follows the schema showed in Figure 8. Using this coordinator, if the activation value of all active behaviours is 1 (max. value), the coordinator output corresponds to the behaviour output with more priority (pre-emptive architecture). Otherwise, if the activation values are less than 1, the final output will be the combination of all the active behaviours (collaborative architecture). Since each DoF is treated separately it is possible, by using activations levels with value 0, to program behaviours that do not affect all DoFs. The coordinator output, after combining all active behaviours, is a vector as large as the number of DoFs of the robot, where each value corresponds to a normalized velocity.

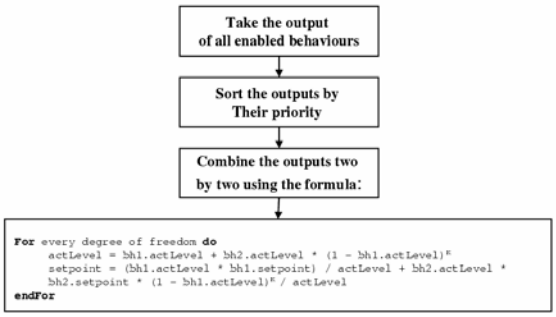


Figure. 8. Coordination algorithm.

B.Mission level: Mission Control System

The task controller decides how to guide the robot movements in each situation. However, to carry out medium/high complex missions it is very difficult to design a unique set of behaviours that can accomplish it. In these missions, it is necessary to have an autonomous system able to enable/disable and reconfigure behaviours. Our mission controller uses a Petri Net to accomplish the mission plan. A Petri Net has place nodes, transition nodes, and directed arcs that connect places with transitions. It is represented as a graph defined by a quadruple,

see equation 25:

$$PN=(P,T,I,O) \quad (25)$$

where P is a finite set of places, T is a finite set of transitions, I(p,t) is a mapping corresponding to the set of directed arcs from places to transitions, and O(t,p) is a mapping corresponding to the set of directed arcs from transitions to places. We use a kind of Petri Net called *marked graphs* which is a pure ordinary Petri net system where every place has only one input transition and one output transition, see equation 26:

$$\forall p \in P: |{}^{\bullet}p|=|p^{\bullet}|=1 \quad (26)$$

where $|{}^{\bullet}p|$ is the number of inputs of place p and $|p^{\bullet}|$ is the number of outputs of place p. In our Petri net, every place corresponds to one behaviour with a particular configuration. When a place has a token, this behaviour is enabled. When all places that go towards a transition are enabled, and their behaviours do not block the execution thread, the transition is ready to be fired. When a transition is fired, a token is removed from each of the input places of the transition and a token is generated in each output places of the same transition, see equation 27:

$$\forall p \in P: M'(p)=M(p)+O(t,p)-I(p,t) \quad (27)$$

where M is a n-dimensional integer vector which assigns a non-negative integer number of tokens to each place of the net. A Petri net can be represented as a matrix, see equation 4, called the incidence matrix (C). In addition, if active transitions and the actual state are known, it is possible to calculate the new state, see equation 28:

$$C_{ij}=O(t_j,p_i)-I(p_i,t_j) \quad (28)$$

where $1 < i < (\text{size of } P)$ and $1 < j < (\text{size of } T)$.

$$M_{i+1}=M_i+CT \quad (29)$$

Therefore, the control mission algorithm starts on the initial state M_i, checks fired transitions, applies equation 29, and repeat this process until the final state M_f is reached.

VIII. NEPTUNE

NEPTUNE [Palomeras 2002, Hernàndez 2003, Ridao 2004] is a real-time graphical simulator with capabilities for hardware in the loop. It makes use of a virtual world based on two components: (1) a VRML file containing the topography of the scene and (2) a set of objects also defined in VRML. Internally, the topography of the scene is converted into a bathymetry grid and the objects are considered spheres of a particular radius of action. This model, together with a conic beam sonar model allows a very simple and fast geometric method for obstacle and/or collision detection.

Within NEPTUNE a simulated robot is defined through three basic files. The first is a VRML file containing the robot geometry. The second is a file which contains the robot and thruster hydrodynamics coefficients. For simulation, the hydrodynamic model described in [Fossen 2002] is used. Thrusters are simulated using the affine model [Fossen 2002]. Finally, the third file contains the file names of the previous two files plus a definition of the sensors included in the robot (sonar beams, video cameras, depth sensors, compass, DVL, DGPS, etc...). NEPTUNE is fully configurable and hence it is very easy to adapt to support the SAUC-E mission development. In order to allow a real time performance, the application is built as a distributed application including several processes: (1) the NEPTUNE main program (fig.9), (2) a robot dynamics process for the simulated robot, (3) a name server. All the components are implemented as CORBA-RT objects allowing for an easy interoperability among the involved software objects. The robot software architecture described in section 4.A can be easily interface with NEPTUNE by means of a virtual robot interface module.

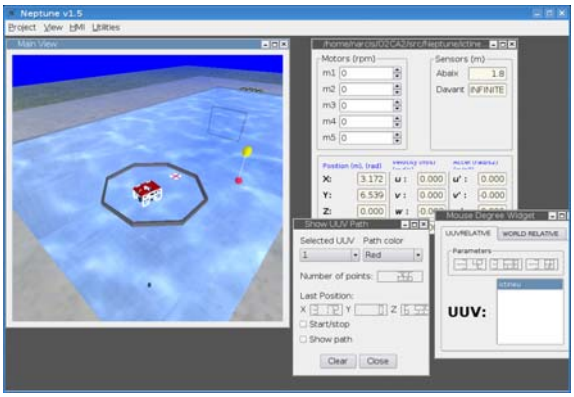


Figure 9. View of the Neptune HIL simulator.

IX. THE MISSION

As explained above, one of the big constrains of our team is the turn around time of the design-development cycle taking into account that we only have 7 months for the project. This constrain lead us to the use of concurrent engineering techniques in order to overlap the hardware and the software development. For this reason, from the very beginning of project, a simplified version of the robot software was developed and tested using hardware in the loop simulation.

The real mission as well as the simulated one occurs in a swimming pool environment with a

size of 20 meters by 10 meters, and a depth of 6 meters. Since the goal of this simulation is to verify the control architecture and the robot software, the SAUC-E mission and the required behaviours have been simplified. The simulated mission consists in (see Figure 10):

1. Moving from a launch/release point and submerging.
2. Passing through a 3x4 meters validation gate.
3. Locating a target situated on the bottom of the pool.
4. Locating a mid-water target and contact it with the AUV.
5. Surfacing at designated recovery zone.

In the following sections it is assumed that we know the validation gate and recovery zone positions and the depth of the two targets (not the position).

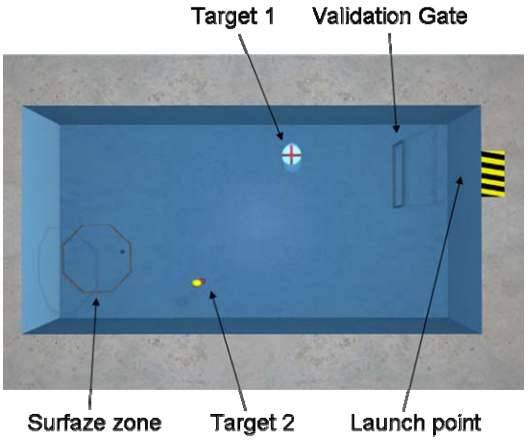


Figure 10.Sauce mission setup

A. Behaviour library

To accomplish the mission, four behaviours have been implemented:

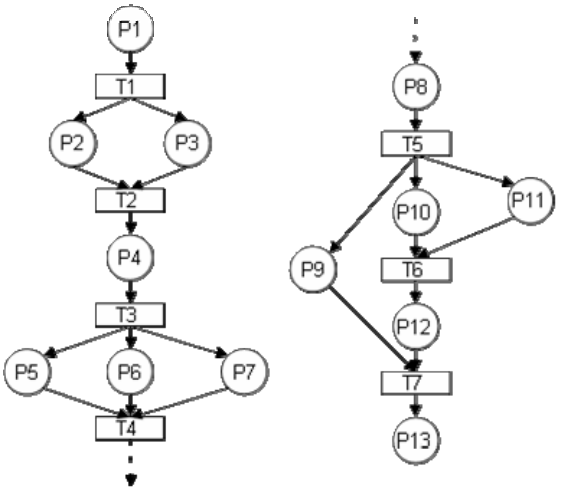
1. **KeepDepth(depth, priority, timeOut):** Keep a constant depth.
2. **MoveTo2D(X, Y, priority, timeOut):** Move the vehicle to a specific 2D point.
3. **LawnmowerMove(Xi, Yi, Xf, Yf, angle, priority, timeOut)**: Move the vehicle with a zigzagging movement.
4. **FindTarget(Xt, Yt, contactDistance, priority, timeOut):** Move the vehicle to the position of the target. It only becomes active when the vehicle is at a very short distance from the target.

B. Petri Net

To implement the mission controller it is required to define several aspects:

- The Petri Net with the sequence of behaviours.
- The parameters of every place/behaviour.
- The initial and final states in the Petri Net.

The Petri net can be represented as a graphic or as a matrix. Figure 11 shows both representations for the simplified SAUC-E mission.



	T1	T2	T3	T4	T5	T6	T7
P1	-1	0	0	0	0	0	0
P2	1	-1	0	0	0	0	0
P3	1	-1	0	0	0	0	0
P4	0	1	-1	0	0	0	0
P5	0	0	1	-1	0	0	0
P6	0	0	1	-1	0	0	0
P7	0	0	1	-1	0	0	0
P8	0	0	0	1	-1	0	0
P9	0	0	0	0	1	0	-1
P10	0	0	0	0	1	-1	0

Figure 11. Petri Net used in the SAUC-E mission

As commented before, every place represents a behaviour. From the Petri Net graph we can observe that the number of active behaviours will go from one to three. When there is only one behaviour, this behaviour is the keepDepth and it will try to reach a specific depth to find a target, pass throw the validation gate or reach the surface. When there are two active behaviours, these behaviours are the KeepDepth and the MoveTo2D. They will try to reach a 2D point keeping a desired depth. The KeepDepth behaviour only generates one output on the heave DoF and the MoveTo2D behaviour generates outputs on surge DoF and Yaw DoF. Therefore, the coordinator will take the two outputs and combine them without modifying the velocity set points. Finally, when there are three active behaviours, these behaviours are the KeepDepth, the LawnmowerMove and the FindTarget. They cause the robot to move at a specific depth around the swimming pool looking for a target. Both the LawnmowerMove and the FindTarget generate outputs on surge and Yaw DoFs. The FindTarget priority is highest than the LawnmowerMove priority, but its activation level is greater than zero when the robot is near the

target. For this reason, the coordinator will take the FindTarget + KeepDepth outputs when the vehicle is near the desired target and the LawnmowerMove + KeepDepth outputs when the vehicle is far.
Behaviour parameters will change depending on each task of the mission. Table 2 shows the correspondence between places and behaviours, and the values of these parameters.

Place	Behaviour + Parameters
P1	keepDepth(2.2, 2, 30)
P2	keepDepth(2.2, 2, 0)
P3	moveTo2D(5, 3, 2, 30)
P4	keepDepth(4, 2, 30)
P5	keepDepth(4, 2, 0)
P6	findTarget(8, 1.5, 1.5, 1, 300)
P7	LanmowerMove(4.5, 1, 19, 9, 1.46, 2, 0)
P8	keepDepth(1.5, 2, 30)
P9	keepDepth(1.5, 2, 0)
P10	findTarget(12, 8, 1.5, 1, 300)
P11	LanmowerMove(4.5, 1, 19, 9, 1.46, 2, 0)
P12	moveTo2D(16.5, 6.5, 2, 200)
P13	keepDepth(0, 2, 30)

Table 2. Behaviours and parameters description

Finally, initial and final state vectors show the enabled and the disabled places at the beginning and at the end of the execution respectively. These vectors contain 13 Boolean values, one for each place:

Mi = [1,0,0,0,0,0,0,0,0,0,0,0,0]
Mf = [0,0,0,0,0,0,0,0,0,0,0,0,1]

X. RESULTS

In this section, preliminary results about HIL simulation of the SAUC-E mission and an offline execution of the localization algorithm are presented.

A. HIL Simulation

In order to simulate the SAUC-E mission, the four behaviours and the mission controller have been implemented and integrated in the software architecture. In this preliminary work, the mission has been simulated using Neptune instead of the real robot. Since the hydrodynamic model of the ICTINEU^{AUV} is not known, for the simulation, the hydrodynamic parameters of GARBI^{AUV} were used [Ridao 2001]. A virtual world corresponding to the SAUCE swimming pool was setup and loaded within NEPTUNE. The simulation was executed and it took 5 minutes and 45 seconds to accomplish the mission. Figure 12 shows the obtained trajectory. The simulation showed a good performance and robustness of the mission and task controllers, as well as their simplicity.

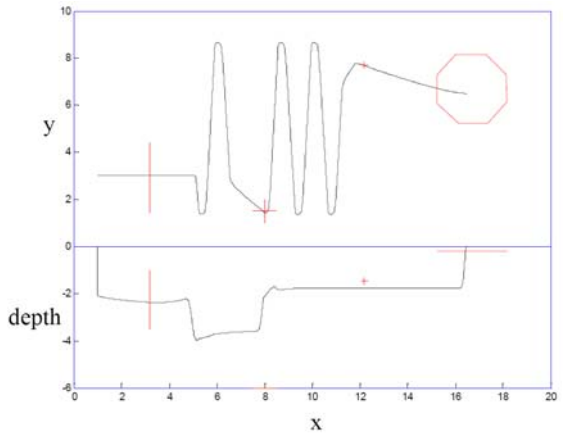


Figure 12. Trajectory obtained during the HIL simulation of the SAUC-E mission.

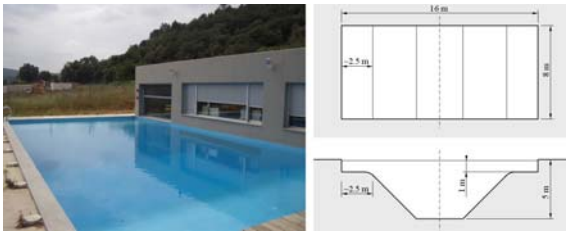


Figure 13. Water tank of the Underwater Robotics Research Center at the University of Girona

B. Results of the Map-based Navigation

We carried out an experiment in the water tank of the Underwater Robotics Research Center at the University of Girona (See fig. 13).
The vehicle was equipped with a Miniking Imaging sonar from Tritech, a sensor designed for use in underwater applications like obstacle avoidance and target tracking. It can perform scans in a 2D plane by rotating a fan-shaped sonar beam of 3º of horizontal beamwidth and 40º of vertical beamwidth. During the experiment, the sensor was set up to work within a range of 10 meters, capturing a sonar return every 0.1 meters (100 measurements per beam). Its scanning rate was set to the maximum (around 6 seconds per a 360º scan).
In order to estimate the vehicle movement an Argonaut DVL from Sontek, which measures ocean currents, vehicle speed over ground and altimetry, was used. Moreover, the unit is also equipped with a compass/tilt sensor which permits to recollect attitude data, a pressure sensor to estimate the depth and a temperature sensor for sound speed calculations.
The robot carried out a guided trajectory of around 42 meters, consisting on several loops; 161 complete sonar scans were taken.
The results are shown in fig. 14. For comparison purposes the trajectory estimated using dead reckoning of the DVL measurements (blue dash-dotted line) is

represented together with the one estimated using the imaging sonar and the localization algorithm (black line). It is also represented the map used to perform the localization. As it can be seen, the dead reckoning trajectory has an important drift, which is controlled using the proposed localization algorithm.

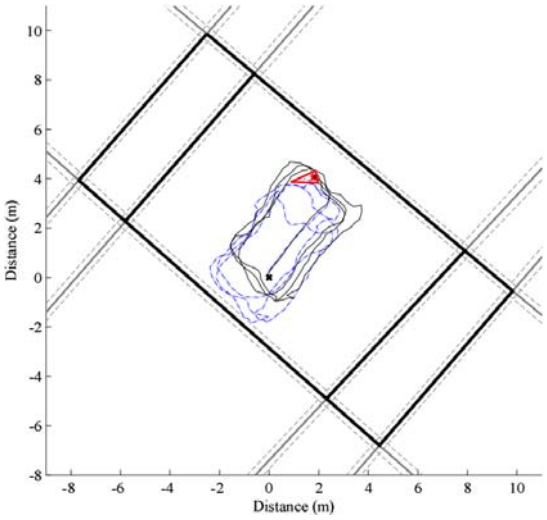


Figure 14. Navigation results using the proposed strategy.

XI. CONCLUSIONS

This paper has presented the current state of development of the ICTINEU^{AUV} robot, designed by the VICOROB-UdG^{TEAM} to face the SAUC-E challenge. The main principles of design (open frame architecture, modularity and backward compatibility) have been reported. The robot software is built as a distributed object oriented application based on CORBA-RT. The control system is organized into three levels following the principles of the hybrid control architectures. It includes a low level velocity controller (vehicle level), a behavioural layer (task level) and a Petri Net based mission controller (mission level). The paper has also described a localization method based on the use of a DVL, an imaging sonar to sense the walls of the pool and an a-priori map, which allows to accurately estimate the robot position with a bounded drift. With the help of the Neptune HIL simulator the principles of the proposed control approach have been tested with a simplified version of the SAUC-E mission.

REFERENCES

[Brooks 1986] Brooks, R. “A Robust Layered Control System for a Mobile Robot”. IEEE Journal of Robotics and Automation, vol. RA-2, No. 1, pp. 14-23.

[Duda and Hart 1972] R. Duda and P. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” Communications of the ACM, 1972.

[Fossen 2002] T. I. Fossen. Marine Control Systems, Marine Cybernetics, 2002

[Hernàndez. 2003] Henàndez E. “Enhacements for a 3D Virtual Simulator in Real-time for Underwater Robots”. Graduation Project in Computer Science. University of Girona.

[Hernàndez 2005] Hernàndez E. “Design and Implementation of a distributed objecte oriented software architecture with support for real-time execution”. Application to an Underwater Robot. Master Project in Computer Science. University of Girona.

[Palomeras 2002] N. Palomeras. “Real-Time 3D Virtual Simulator for Multiple Underwater Robots”. Graduation Project in Computer Science. University of Girona.

[Palomer e*t al* 2006] Palomeras N., Ridao P., Carreras M., Hernandez *E.*.”Design of a Mission Controller for an Autonomous Underwater Robot”. VII Workshop on physical agents 2006. Las Palmas de Gran Canaria (Spain).

[Ribas *et al* 2006] Ribas D., Neira J., Ridao P., Tardos, J. “AUV Localization in structured Underwater Environments Using an a priori Map”. Accepted for publication in the IFAC Conference on Manoeuvring and Control of Marine Crafts MCMC. Lisbon (Portugal).

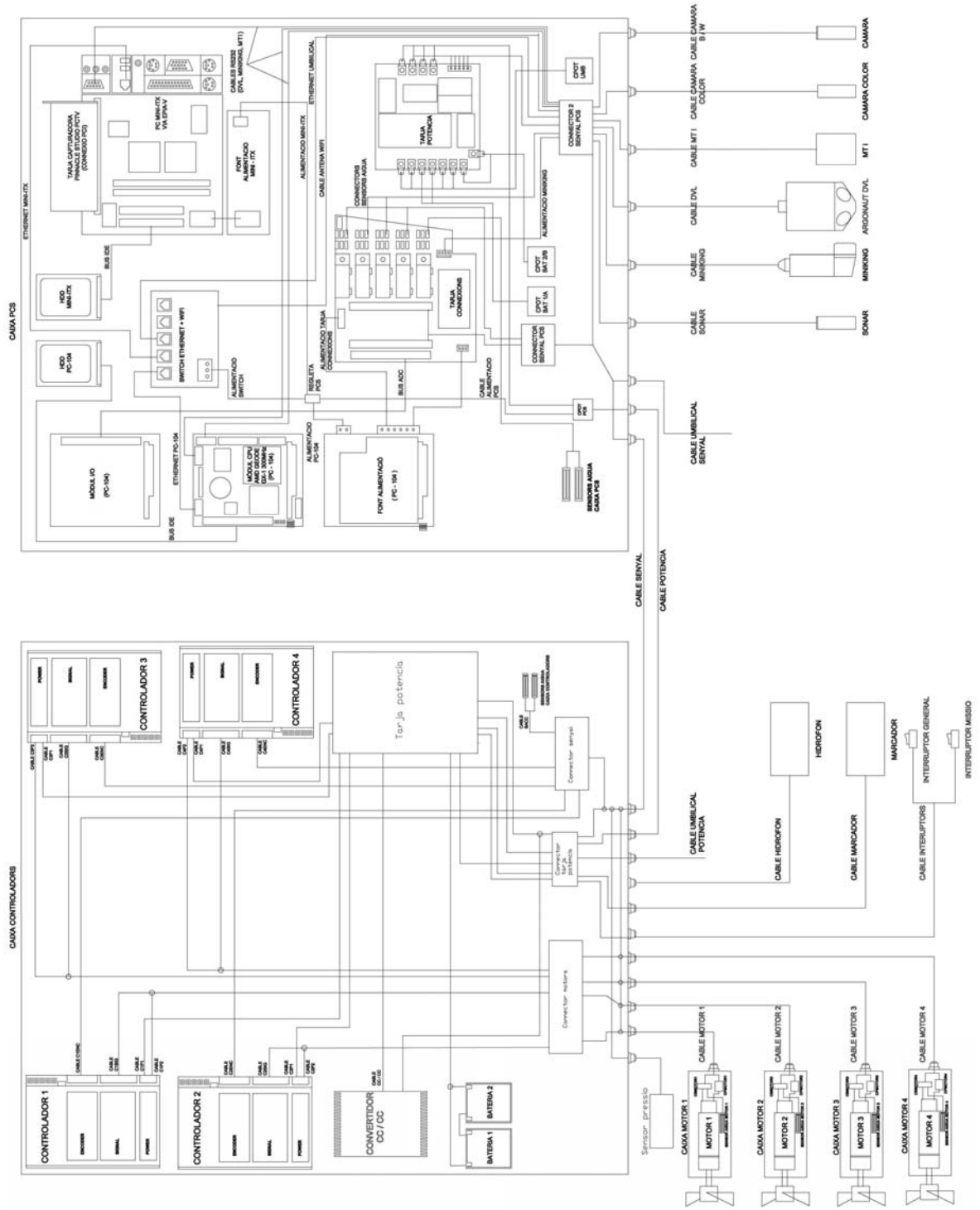
[Ridao 2001] Pere Ridao, Joan Batlle, Marc Carreras, “Model identification of a low-speed UUV with on-board sensors”. IFAC conference CAMS’2001, Control Applications in Marine Systems. Glasgow (Scotland-UK).

[Ridao 2004] Ridao, P.; Batlle, E.; Ribas, D.; Carreras, M.. “Neptune: a hil simulator for multiple UUVs”. OCEANS '04. MTS/IEEE TECHNO-OCEAN '04. Volume 1, 9-12 Nov. 2004 Page(s):524 - 531 Vol.1

ACKNOWLEDGEMENTS

Authors want to acknowledge the sponsors who made possible our participation in the SAUC-E competition. Particularly, we want to give thanks to the Catalan Artificial Intelligence Association, the Polytechnic School of the University of Girona, the GRN Telematics Services Company, the E. Ribas “industrial d’automatismes” company, the Spanish Research Network AUTOMAR, the Technical Industrial Engineers Association, the Girosacme Olot Company, the local government of the “Diputació de Girona”, the board of companies of the Polytechnic School of the University of Girona, the Social Council of the University of Girona and the students vice-presidency of the University of Girona for the monetary contribution to our project. We also want to thanks the XSens Motion Technologies company for their contribution with the MTi MRU, the EuroTech group for donating PC104 cards and Oxiter company for cutting the frame of the robot.

APPENDIX I SCHEMATIC OF THE HARDWARE DESIGN



APPENDIX I UML DESIGN