Dissertation presented to the Instituto Tecnológico de Aeronáutica, in partial fulfillment of the requirements for the degree of Master of Science in the Program of Electronic Engineering and Computer Science, Area of Devices and Electronics Systems.

David Issa Mattos

AUTONOMY IMPLEMENTATIONS FOR A LOW-COST AUTONOMOUS SURFACE VEHICLE USING THE MOOS-IVP SOFTWARE

Dissertation approved in its final version by signatories below:

Prof. Dr. Cairo Lúcio Nascimento Júnior

Advisor

Prof. Dr. Douglas Soares dos Santos

Co-advisor

Prof. Dr. Luiz Carlos Sandoval Góes

Prorector of Postgraduate Studies and Research

Campo Montenegro
São José dos Campos, SP - Brazil
2016

Cataloging-in Publication Data
Documentation and Information Division

Mattos, David Issa
Autonomy Implementations for a Low-Cost Autonomous Surface Vehicle Using the MOOS-IvP Software / David Issa Mattos.
São José dos Campos, 2016.
78f.

Dissertation of Master of Science – Course of Electronic Engineering and Computer Science. Area of Devices and Electronics Systems – Instituto Tecnológico de Aeronáutica, 2016. Advisor: Prof. Dr. Cairo Lúcio Nascimento Júnior. Co-advisor: Prof. Dr. Douglas Soares dos Santos.

1. Autonomous Surface Vehicle. 2. Sensor Fusion. 3. Autonomous Navigation. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

MATTOS, David Issa. **Autonomy Implementations for a Low-Cost Autonomous Surface Vehicle Using the MOOS-IvP Software**. 2016. 78f. Dissertation of Master of Science – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

AUTHOR'S NAME: David Issa Mattos
PUBLICATION TITLE: Autonomy Implementations for a Low-Cost Autonomous Surface Vehicle Using the MOOS-IvP Software.
PUBLICATION KIND/YEAR: Dissertation / 2016

David Issa Mattos
Rua Roma 673, Apto. 145D
12.216-510 – São José dos Campos–SP
Brazil

AUTONOMY IMPLEMENTATIONS FOR A LOW-COST AUTONOMOUS SURFACE VEHICLE USING THE MOOS-IVP SOFTWARE

David Issa Mattos

Dissertation Committee:

Prof. Dr.	Wagner Chiepa Cunha	President	-	ITA
Prof. Dr.	Cairo Lúcio Nascimento Júnior	Advisor	-	ITA
Prof. Dr.	Douglas Soares dos Santos	Co-advisor	-	ITA
Prof. Dr.	Karl Heinz Kienitz	Internal Member	-	ITA
Prof. Dr.	Sidney Nascimento Givigi Jr.	External Member	-	RMCC

ITA

To my beloved Erika.

Acknowledgments

Firstly, I would like to thank my advisors Cairo Nascimento Jr. and Douglas dos Santos for all the patience, availability, guidance and the numerous contributions to this work. Their expertise and understanding added considerably to my graduate experience at ITA.

Also I would like to express my sincere gratitude to all the other members of the Laboratory of Intelligent Machines (LMI), Marcos, José Ricardo, Alessandro Paolone, Kléber Cabral, Humberto Pessoa and Elton Sbruzzi. Their numerous suggestions and reviews contributed greatly to this work.

My sincere gratitude to all professors and staff at the Electronic Engineering Division at ITA. Special thanks to professors Roberto d'Amore, Neusa Oliveira, Karl Kienitz, Wagner Chiepa, Roberto Kawakami and Irany Azevedo. Your valuable lessons have been crucial for my journey.

My thanks to the Brazilian Ministry of Education CAPES Foundation for the financial support.

My deepest gratitude is also addressed at the members of the thesis committee, for their comments and feedback on my work.

Finally and most importantly, I would like to thank my beloved girlfriend Érika, for being there and supporting me during every study hour and sleepless night. Also to my family, for all the love and assistance during so many years.

"Science is the acceptance of what works
and the rejection of what does not.
That needs more courage than we might think.'
— Jacob Bronowski

Resumo

Esse trabalho descreve uma implementação de baixo custo de um veículo autônomo de superfície. Esse veículo utiliza um software de decisão por comportamento, o MOOS-IvP. A plataforma utilizada é um barco tipo catamarã, com dois motores de corrente contínua como sistema de propulsão. Duas abordagens foram feitas e ambas são apresentadas e discutidas neste trabalho.

Na primeira abordagem foram embarcados no veículo uma placa de processamento com um microcontrolador da família Arduino, um sensor inercial de baixo custo composto por acelerômetros, giroscópios e magnetômetros, um receptor GPS e um rádio-modem serial. O barco comunica-se com a estação de controle em terra enviando dados de telemetria e recebendo comandos de navegação para os motores. A estação de controle em terra utiliza o software MOOS-IvP para implementar os procedimentos de navegação autônoma e os algoritmos de fusão sensorial utilizando o GPS e os sensores inerciais.

Na segunda abordagem algumas modificações foram feitas no hardware embarcado no veículo. Foram adicionados um microcomputador de baixo custo (Raspberry Pi 2), um adaptador WiFi e uma câmera USB para vigilância. Também foi retirado o rádio-modem serial. Nesta abordagem a principal diferença é que os procedimentos de navegação autônoma e os algoritmos de fusão sensorial estão embarcados. Dessa forma o veículo é capaz de continuar a missão mesmo com a perda de sinal com a estação de controle em terra. Apesar do barco não depender da estação de controle em terra para operação, foi utilizada uma estação para inicializar as missões e possibilitar controle manual remoto do veículo. O computador embarcado utiliza o MOOS-IvP para implementar a navegação autônoma e os algoritmos de fusão sensorial. A estação em terra utiliza o software MOOS-IvP para receber os dados de telemetria do barco e enviar comandos. A abordagem visa a criação de um sistema modular, possibilitando que o sistema seja expandido e modificado para atender aos requisitos próprios de cada tipo de missão.

Simulações demonstrando a viabilidade das missões, utilização e ajuste dos algoritmos de fusão sensorial são apresentadas e discutidas. Resultados experimentais do veículo, em condições reais, em ambas as abordagens para missões de seguimento de caminho na presença de obstáculos virtuais são apresentados e discutidos.

Abstract

This work describes the implementation of a low-cost Autonomous Surface Vehicle (ASV) using a behavior-based software, MOOS-IvP. The platform used is a catamaran boat driven by two direct current motors as the propulsion system. Two different implementations were made and both are presented and discussed in this work.

In the first approach, the ASV is embedded with a processing board with an Arduino microcontroller, a low-cost Inertial Measurement Unit (IMU) with accelerometers, gyroscopes and magnetometers, a GPS receiver and a wireless RF serial modem. The ASV communicates with a Ground Control Station (GCS) sending telemetry data and receiving navigation commands for the propulsion motors. The GCS uses the MOOS-IvP software to implement the autonomous navigation procedures and the GPS/Compass/IMU sensor fusion algorithms.

In the second approach, modifications were made in the ASV embedded hardware. A low-cost microcomputer (Raspberry Pi 2), a WiFi adapter and an USB camera for surveillance were added and the RF serial modem was removed. The main difference between the two approaches is that in the second approach the autonomous navigation procedures and the sensor fusion algorithm run embedded in the ASV. Therefore, the ASV is capable of performing a mission even if the communication link with the GCS is lost. Although the ASV does not require a GCS to operate, a GCS was used to deploy the missions and give manual remote control over the ASV. The embedded computer runs the MOOS-IvP software to implement the autonomous navigation procedures and the GPS/Compass/IMU sensor fusion algorithm. The GCS uses the MOOS-IvP software and receives telemetry data from the ASV and sends control commands. This approach aims for a modular system that allows it to be expanded and modified to meet the custom needs of specialized missions.

Simulations were used to demonstrate the viability of missions, tuning and using of the sensor fusion algorithms. In both approaches, experimental results in real conditions are presented and discussed. The experimental and simulation results consist of path following missions in the presence of virtual obstacles.

List of Figures

List of Tables

List of Abbreviations and Acronyms

ASV	Autonomous Surface Vehicle
AUV	Autonomous Underwater Vehicle
GCS	Ground Control Station
DC	Direct Current
GPS	Global Position System
IMU	Inertial Measurement Unit
MOOS	Mission Oriented Operating Suite
MOOSDB	Mission Oriented Operating Suite Database
IvP	Interval Programming
metaOS	Meta Operating System
OS	Operating System
OpenCV	Open Computer Vision
RF	Radio Frequency
ECEF	Earth Centered Earth Fixed
SVN	Apache Subversion

List of Symbols

Earth semi-major axis	a
Earth semi-minor axis	b
Latitude	λ
Longitude	ϕ
Altitude relative to sea level	h
Flattening	f
First eccentricity	e
Normal length	N
Drag forces constants	K_x, K_y and K_z
Propulsion force	F_p
Angular velocity	w_z
Velocities in the vehicle frame	V_{bx} and V_{by}
Positions in the local frame	P_x and P_y
Heading angle	ψ
Inertial moment	J
ASV mass	m
Semi-lateral distance	d

Contents

1 Introduction

1.1 Motivation

The use of autonomous surface vehicles (ASV) and autonomous underwater vehicles (AUV) are continuously drawing attention. The demands for ASV and AUV meet a wide range of applications including:

- environmental monitoring, such as water quality sampling (AUTOMARINESYS, 2016; BLUEFIN, 2016),
- coastal surveillance (CLEARPATH, 2016),
- autonomous transportation (ASVGLOBAL, 2016),
- marine targets for naval gunnery training (ASVGLOBAL, 2016),
- autonomous tow boat (ASVGLOBAL, 2016),
- automatic docking,
- search and rescue missions,
- mine counter measurements (DJAPIC; NAD, 2010).

Some of those companies provide collaborative solutions to enhance their autonomous services (ASVGLOBAL, 2016; AUTOMARINESYS, 2016; CLEARPATH, 2016). However, commercial solutions have costs in the order of tens of thousands of dollars.

Since Brazil has more than 42,000 km of rivers extension (ANTAQ, 2007), the use of ASVs can provide a cost effective way to monitor environment and water quality in rivers and water supply dams. In military missions, ASVs can be used for surveillance in difficult navigation areas for larger boats, and in human-life risk situations. Furthermore, the use of collaborative ASVs in missions can make it possible to explore a larger area in the same amount of time.

The development of low-cost ASVs can be very useful in terms of exploring and monitoring the large Brazilian river area.

1.2 Related Work

The development of an ASV provides the opportunity to create new combinations of hardware components and software interfaces with an autonomous software library. Some interesting characteristics that allow vehicles to operate as autonomous agents in dynamic environments are (SETO, 2012; KEMNA *et al.*, 2011):

- adaptability: ability to adjust itself based on the environment, such as changes in currents or higher energy consumption than expected,
- capability to perform complex missions,
- re-planning a mission because a member of the cooperative team is no longer in communication,
- self-sufficiency: ability to navigate without assistance during extended periods of time.

In the vehicle and software design, considerations such as cost and development time should be taken in account. As new vehicles are being developed, collaborative missions running vehicles from multiple developers and with different features require a platform-independent framework. The possibility of using heterogeneous vehicles should be considered, as the cost of redesigning a custom autonomous software for each vehicle might be prohibitive.

The use of a robotic meta-operating systems reduces greatly the development effort and cost. In (MADDEN, 2013), there is an extensive analysis of several robotic meta-operating systems (metaOS) for autonomous underwater vehicles as well as several vehicles simulators. A meta-operating system is program that provides services and tools expected from an operating system. Some of these services are: hardware abstraction, low-level device control, message-passing between processes and package management. It also provides tools and libraries for building and running code across multiple platforms. However, a metaOS runs under a proper operating system, such as Linux, Mac OS X, Windows.

The study by Madden was intended to select a robotics metaOS for the Australian Navy AUVs. This work evaluates the metaOS in several levels, such as the underlying operating system, abstraction of low-level device drivers, number of libraries, computer languages, open source code, among others features. Madden evaluated the following metaOS:

- MOOS (Mission Oriented Operating Suit) (MOOS, 2016),
- ROS (Robot Operating System) (ROS, 2016),
- Player (PLAYER, 2016),

- ERSP Robotics Development Platform (ERSP, 2016),
- URBI (Universal Body Interface) (URBI, 2016),
- Microsoft RDS (Robotics Developer Studio) (RDS, 2016).

The article concludes remarking MOOS as a strong candidate, as it is open source, runs on different operating systems, and has strong libraries for AUV and ASV. The work presented in this dissertation uses the metaOS MOOS. MOOS was chosen due to the several ready-to-use modules for ASVs and AUVs.

In the 2014 RobotX Competition, the challenge was to develop an ASV that was capable of doing several tasks, and each team provided a technical report. In this challenge the MIT-Olin team implemented the ASV using the software MOOS-IvP for both sensor interacting and autonomy (ANDERSON *et al.*, 2014). Communication with the shoreside computer in the GCS was done using WiFi. In the same competition, the KAIST team integrated GPS at lower rates and a IMU at higher rates to provide a better localization accuracy (KANG; et al, 2014) using a different autonomous software.

The works (DJAPIC; NAD, 2010; KEMNA *et al.*, 2011; SIDELEAU; EICKSEDT, 2010) use the MOOS-IvP software in AUV (Autonomous Underwater Vehicle) with adaptative missions. These works describe in a more detailed way how MOOS-IvP was integrated in different commercial vehicles.

Brodskiy (BRODSKIY, 2014) describes benefits of a modular approach for cooperative robots and for robots sharing computationally intensive tasks. The construction of modular AUVs are shown in (BREGE, 2011) and (SANGEKAR *et al.*, 2008). In (SANGEKAR *et al.*, 2008) it is also shown the benefits of a distributed hardware architecture.

In (SANTOS *et al.*, 2013) an ASV was implemented and its nautical structure, on-board hardware and sensors were presented. This navigation system has an architecture in which the ASV only sends telemetry data to the Ground Control Station (GCS) and receives navigation commands. The GCS uses a custom autonomous navigation software (based in Matlab) for both the decision-making and the sensor fusion algorithms. The GCS implements an Extended Kalman Filter algorithm for GPS (Global Position System), Digital Compass and IMU (Inertial Measurement Unit) fusion. The work presented in (SANTOS *et al.*, 2013) uses a costly IMU. Due to this architecture, the ASV requires a reliable communication link to run a mission.

This dissertation presents the work done to develop a low-cost ASV. This work can be divide in two parts, as it corresponds to two different designs. The first design is an intermediate step towards the final design. The first design shows the work done to extend the platform first presented in (SANTOS *et al.*, 2013) to use the software MOOS-IvP and lower cost

components compared to the ASV developed by Santos. This work started as the authors final year undergraduate project in Electronic Engineering at ITA (MATTOS, 2014). The second part shows the design of the autonomous navigation architecture used to make the ASV modular. This architecture benefits from being easy to adapt to other vehicles and to be used in collaborative missions.

1.3 Research Scope

This dissertation is concerned with the design of a low-cost ASV system to be used as a basis for autonomous missions. Figure 1.1 shows a high-level diagram representing the problem of an autonomous vehicle development. In this diagram the autonomous navigation consists of three parts

- The Vehicle: the vehicle consists of the nautical structure, the embedded hardware and the embedded software. It is responsible for interacting with the actuators and with the sensors.
- The Mission Planner: the mission planner is responsible for giving the ASV the autonomy it needs. The mission planner uses behaviors to make the navigation decisions. It also solves possible conflicts between behaviors. It receives as inputs the vehicle status and localization, the mission and outputs the desired vehicle behavior.
- The Mission: the mission consists of what is expected the vehicle to perform. The mission can have several desired vehicle behaviors such as waypoints, virtual or real obstacles among others.

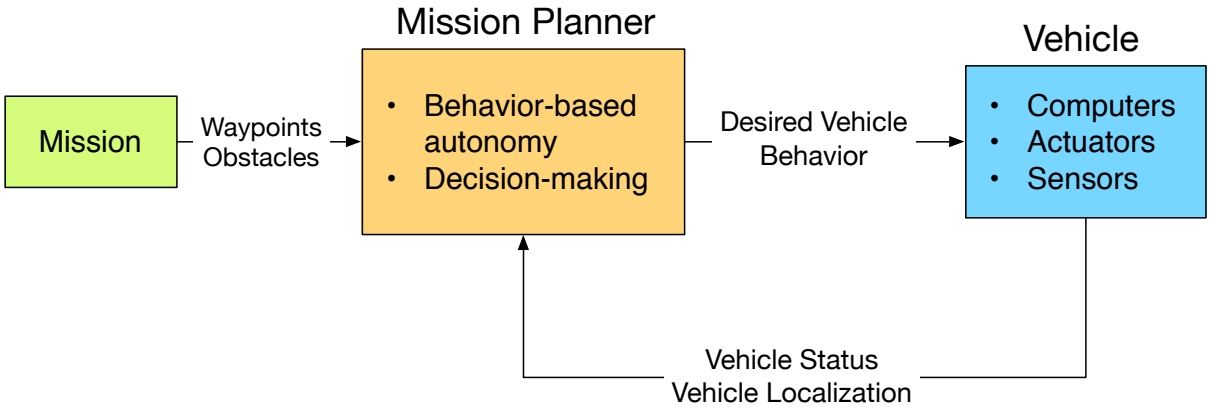

FIGURE 1.1 – This diagram represents the problem of an autonomous vehicle development in three parts, the mission, the mission planner and the vehicle.

This work consists of the hardware design for the ASV system and its software development, for both the vehicle and the mission planner. The mission planner is based on the

metaOS MOOS-IvP (MOOS-IVP, 2016).

Sensor fusion algorithms are used to improve localization of the vehicle. Simulation of missions consisting of several waypoints and virtual obstacles as well as the experimental results of these missions in real conditions are shown to demonstrate the use of the ASV system.

From this research one paper was accepted for publication. The paper, titled "Development of a Low-Cost Autonomous Surface Vehicle Using MOOS-IvP" was submitted to the IEEE System Conference 2016. This paper discusses the first architecture presented in this dissertation.

This research contributes in implementing an autonomous surface vehicle in a low-cost nautical structure using low-cost sensors. Implementing an autonomous surface vehicle on a low-cost platform has some problems, such as the development of embedded hardware and the use of algorithms to improve localization. These problems, as well as the problem of interfacing the vehicle with the software MOOS-IvP are addressed in this dissertation.

1.4 Dissertation outline

This dissertation is structure as follows:

Chapter 2 describes briefly the MOOS-IvP software. This chapter presents an overview on the software architecture, its basis and requirements to be used in a ASV. More detailed information can be found in the official documentation.

Chapter 3 describes the architecture of the ASV system developed in this dissertation. In this chapter it is also discussed the two designs for the ASV and the benefits of each one. Description on the electronic hardware and how the software MOOS-IvP is used is also given.

Chapter 4 This section explains in details the several MOOS applications developed in order to make the ASV work with the software.

Chapter 5 presents and discusses the simulation and experimental results for the developed ASV.

Chapter 6 concludes the dissertation by making some final remarks and points directions for future studies.

2 The MOOS-IvP Software

2.1 Introduction

Researchers at MIT, University of Oxford and the Naval Undersea Warfare Center (NUWC) developed the autonomy framework MOOS-IvP (MOOS-IVP, 2016). MOOS stands for Mission Oriented Operating Suite (MOOS, 2016) and it is an open-source middleware (a set of libraries and executables) that handles inter-process communications. IvP stands for Interval Programming, and it is a mathematical method for solving multi-objective optimization.

The project is based on the philosophies cited below (MOOS-IVP, 2016). The advantage of these philosophies is to develop an autonomous system capable of performing several tasks in a reduced development time.

- **Platform Independence**: The MOOS-IvP software typically runs on a dedicated computer for autonomy and sensing in the vehicle "payload" section.
- **Module Independence**: MOOS and the IvP Helm provide two architectures that enable the autonomy and sensing system to be built from distinct and independent modules.
- **Nested Capabilities**: MOOS and IvP Helm architectures both allow a system to be extended without modifying or recompiling the core software, publicly available free software.

The first topic refers to the creation an autonomy software that can be used nearly identically on vehicle platforms from different manufactures. This is reinforced by the backseat driver design paradigm. Although the software does not restrict the type of vehicle, it is mainly used in ASVs and AUVs.

The second topic refers to the ability to create modules that are not dependable on other modules.. Therefore modules can be created and updated from different developers without interfering with the existent modules.

The third topic refers to the ability to extend the software without having to modify its core. This feature allows the autonomous system to benefit from custom modules.

2.2 The backseat driver paradigm

One of the key philosophies of the MOOS-IvP software is platform independence. This independence is reinforced by using the backseat driver paradigm. The key idea of the backseat driver is the separation of vehicle control and vehicle autonomy (BENJAMIN *et al.*, 2013). The main benefit is the decoupling of the platform autonomy system from the vehicle hardware. Therefore, the vehicle manufacturer can provide navigation and control system (the frontseat), while the autonomy software computer (the backseat) provides autonomy decisions, such as desired heading, speed, depth and position. How the vehicle navigates and implements its control system is unspecified by the payload computer. Figure 2.1 represents the backseat driver paradigm.

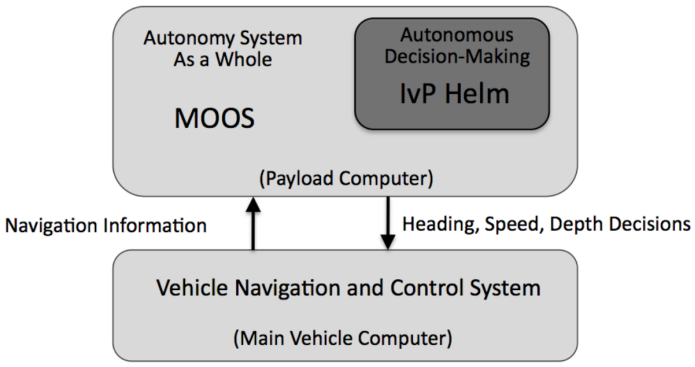

FIGURE 2.1 – Backseat driver paradigm (MOOS-IVP, 2016).

The communication between the frontseat computer and the backseat computer can be done by a module that implements the manufacturer communication protocol such as serial port, ethernet or CAN bus. This module communicates bidirectionally coordinating sensors data (such as speed, position, heading) and decision commands (desired heading, desired speed).

2.3 The Mission Oriented Operating Suite - MOOS

The MOOS software is a middleware software for robotics. It is also considered a metaOS for robotics. It runs on different operating systems, such as Linux, Mac OS X and Windows. It is fully developed in C++. However it provides ways to integrate with other development tools, such as Matlab (MATLAB, 2016) and Python (PYTHON, 2016). It provides a publish-subscribe architecture and communication protocols between processes (MOOS, 2016). The use of a publish-subscribe architecture allows the development of uncoupled modules. This pattern isolates the development of each module from the main core. This reinforces the philosophy of module independence. This way each module does not need to have access

to variables it does not use, and does not interfere in other modules.

The MOOS software is organized in a star topology, in which the central node is called the MOOSDB (MOOS Database). The MOOSDB is responsible for coordinating messages between the several other nodes (inter-process communication), the MOOS Applications. Figure 2.2 represents this star topology.

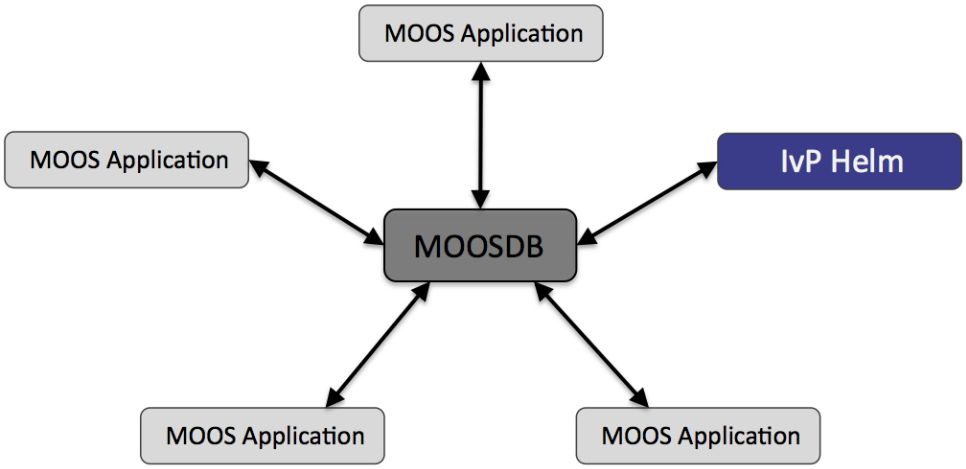

FIGURE 2.2 – Star topology for the MOOS Software (MOOS-IVP, 2016).

This publish-subscribe architecture does not allow Peer-to-Peer communication between the MOOS Applications. This restriction reinforces module independence. By not allowing direct communication between MOOS Aplications the developer is encouraged to follow the communication pattern using the MOOSDB. A MOOS Application communicates only with the MOOSDB, requesting information regarding to a certain variable. Also, the MOOS Application posts all the data it generates only to the MOOSDB. Other applications can request these data only through the MOOSDB.

A MOOS community is the group of several MOOS Applications and a single MOOSDB. Usually a single computer (and vehicle) runs only one MOOS community. Different communities can communicate with each other through special MOOS Applications.

The MOOS community is configured with a configuration file. This file (`.moos`) specifies the several MOOS Applications that will be available in the community, as well as configuration parameters of those applications. For specific details on how to use the configuration file refer to the MOOS-IvP documentation (MOOS-IVP, 2016).

2.4 The IvP helm application

The IvP helm is a MOOS Application that provides behavior-based autonomy. This application plugs into a MOOS community and subscribes to receive updates regarding to any

information necessary to make autonomy decisions. It publishes navigation variables, such as desired heading, desired speed or desired depth. The helm can be configured to generate decisions over virtually any user-defined decision space. Figure 2.3 represents this situation.

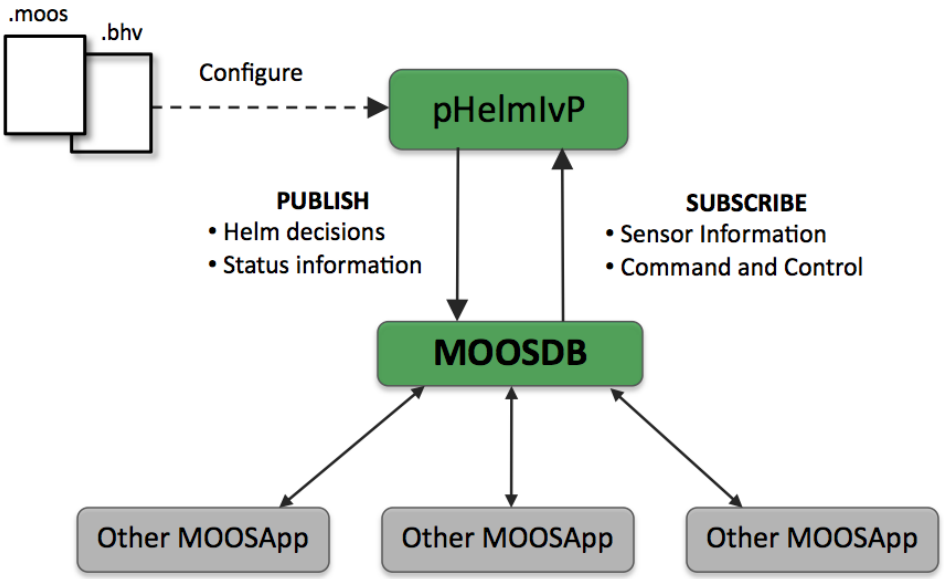

FIGURE 2.3 – The IvP helm as a MOOS application (MOOS-IVP, 2016).

The IvP helm can be configured with a mission file. This file (`.bhv`) defines the active behaviors in a hierarchical tree. The mission is defined using nodes, where each node contains one or several behaviors and the conditions that make the node active. During the mission, several conditions can occur and trigger different nodes. With each node different behaviors are active and the mission objective is changed. These nodes are organized in a hierarchical tree. The IvP searches the active behaviors in the nodes.

Using a hierarchical mission it is possible to configure a mission to be adaptative, the active behaviors can change depending on real-time sensors information. The behaviors therefore, can run simultaneously and can be grouped into sets that are active depending on conditions. Figure 2.4 represents the hierarchical structure of the MOOS-IvP missions. For specific details on how to use the mission file refer to the MOOS-IvP documentation (MOOS-IVP, 2016).

The IvP helm, available from the official SVN repository (MOOS-IVP, 2016), has several behaviors such as following waypoints, collision avoidance, or to keep the vehicle in a safety area. However, it also provides tools that make it easier to develop custom behaviors.

The IvP helm solves the conflicts between behaviors using the IvP solver, a mathematical programming technique that searches a globally optimal solution for each domain in use. This technique is fast enough to run in real-time with the vehicle (MOOS-IVP, 2016; BENJAMIN, 2002).

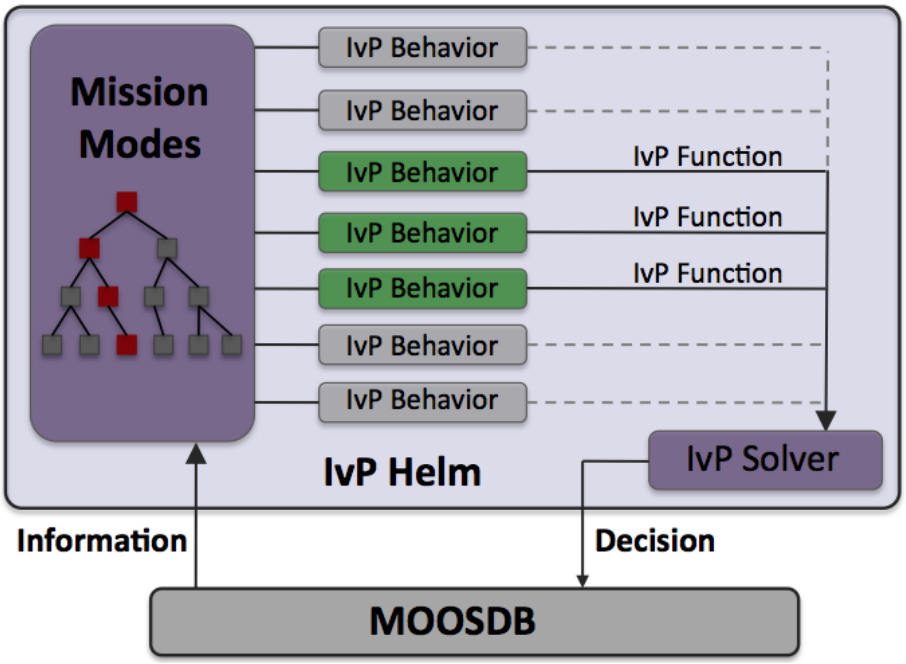

FIGURE 2.4 – The hierarchical mission structure using the IvP helm application. In the `.bhv` file it is possible to declare mission modes that uses a sets of behaviors. The IvP solver resolves possible conflicts (MOOS-IVP, 2016).

2.5 Remarks

The MOOS-IvP software was selected because it fulfills several requirements for developing a low-cost ASV and it is the de-facto metaOS used in ASVs and AUVs. Some of these requirements are:

- it is open source,
- it provides, out-of-the-box, several modules for autonomous vehicles, mainly for ASV and AUV,
- it provides, out-of-the-box, several behaviors capable of performing several autonomy missions,
- it has a large and active community of users,
- it runs on the Mac OS X and on the Linux Debian-based operating systems,
- it is lightweight and with low libraries dependencies, therefore it is easy to configure,
- it is written in C++, and it is fast enough to run in real-time in embedded vehicles computers,
- it has an extensive documentation,

- it has collaborative features for ASV and AUV.

It is important to mention that these requirements are also satisfy by other metaOS, for instance ROS.

Other design philosophies such as the backseat driver, nested capabilities and hierarchical missions structure facilitates the adoption of this software in low-cost vehicles by reducing the development cost of the autonomy software of the vehicle.

The MOOS-IvP has an extensive documentation. This chapter was intended to be a brief introduction to the software. The complete documentation including several lectures and examples on how to use the software can be found in the MOOS and MOOS-IvP website (MOOS, 2016; MOOS-IVP, 2016).

3 The Autonomous Surface Vehicle Hardware Design

3.1 The nautical structure and mathematical model

This work uses a catamaran boat with water wheels driven by DC motors as a propulsion system. It was developed and modeled in (SANTOS, 2011). The complete derivation of this model can be found in (SANTOS, 2011). Figure 3.1 shows a photograph of the boat.

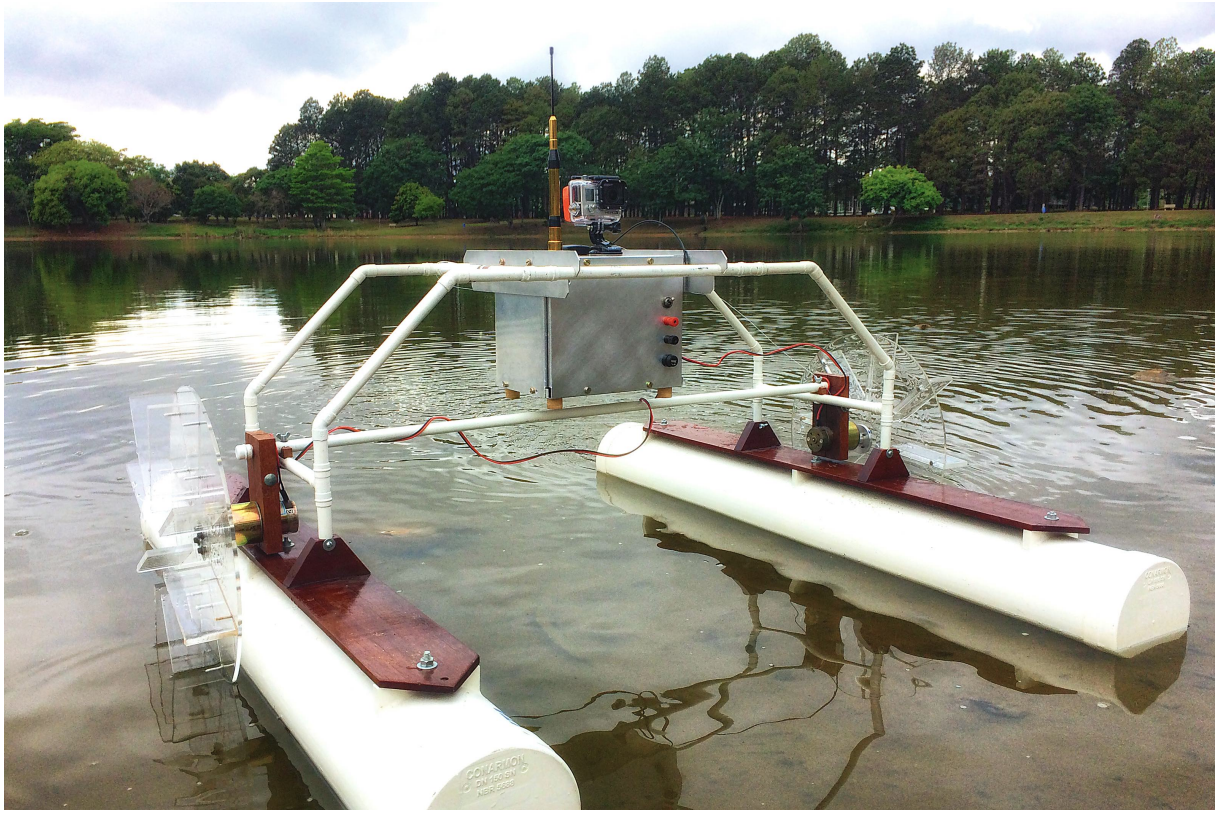

FIGURE 3.1 – A photograph of the catamaran boat used in this dissertation. The embedded hardware is contained inside the metal box.

The ASV uses direct current motors (DC) attached to the water wheels. The DC motors, ($M_{L,R}$, motor left and motor right), can only have three possible states: forward, off and back-

ward ($M_{\mathrm{L,R}} = \{1, 0, -1\}$). The motors are driven by a relay H-Bridge. The motor controller uses a lookup table which can be seen in Table 3.1. The lookup table uses as input variables: the error in position (relative to the desired position) δd , and the error in heading (relative to the desired heading) $\delta\psi$. The output variables of this table is the motor right state ($\mathrm{M_R}$) and the motor left state ($\mathrm{M_L}$).

This table was developed intuitively in (SANTOS, 2011). It is can be easily modified to fit the ASV dynamics, in both the input bounds as well as the output values. In (SANTOS, 2011) a different table is suggested based on a learning automata algorithm.

TABLE 3.1 – Lookup table for the motor controller

$\mathbf{M_L}$,$\mathbf{M_R}$	0 m$< \delta d <$ 3 m	$\delta d \geq$ 3 m
$5° < \delta\psi \leq 90°$	1,-1	1,0
$0° < \delta\psi \leq 5°$	0,0	1,1
$-5° < \delta\psi \leq 0°$	0,0	1,1
$-90° < \delta\psi \leq -5°$	-1,1	0,1

A nonlinear model of the catamaran boat was developed in (SANTOS *et al.*, 2013; SANTOS, 2011). A schematic model diagram is represented by Figure 3.2.

The nonlinear model is shown in eq. (3.1),

$$\begin{pmatrix} \dot{w_z} \\ \dot{V_{bx}} \\ \dot{V_{by}} \\ \dot{P_x} \\ \dot{P_y} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} (M_{\mathrm{R}} - M_{\mathrm{L}}) d F_p \frac{1}{J} - \mathrm{sign}(w_z) K_z w_z^2 \frac{1}{J} \\ w_z V_{by} + -\mathrm{sign}(V_{bx})(M_{\mathrm{L}} + M_{\mathrm{R}}) F_p \frac{1}{m} - \mathrm{sign}(V_{bx}) K_x V_{bx}^2 \frac{1}{m} \\ -w_z V_{bx} - \mathrm{sign}(V_{by}) K_y V_{by}^2 \frac{1}{m} \\ V_{bx}\cos(\psi) - V_{by}\sin\psi \\ V_{bx}\sin(\psi) + V_{by}\cos\psi \\ w_z \end{pmatrix} \tag{3.1}$$

where:

- F_p is the propulsion force,
- w_z is the angular velocity,
- V_{bx} and V_{by} are the velocities in the vehicle frame,
- P_x and P_y are the positions in the local frame,
- ψ is the heading angle,
- J the inertia moment,

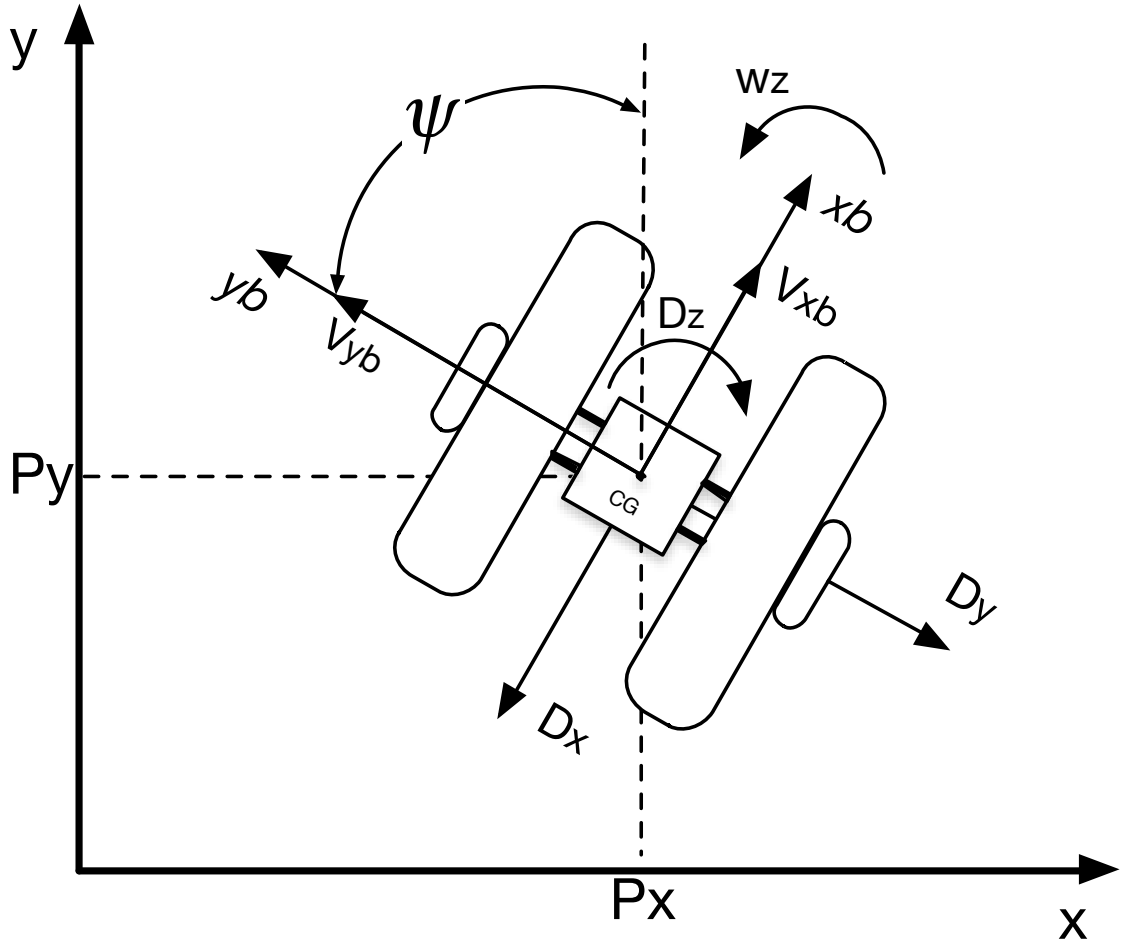

FIGURE 3.2 – A diagram representing the model of the ASV. This diagram shows the ASV frame axis, the local frame axis, the velocities and drag forces (SANTOS *et al.*, 2013).

- m the boat mass,
- d is the distance between a water wheel and the center of gravity.

A quadratic model was assumed for the drag forces:

$$D_x = K_x V_{bx}^2$$
$$D_y = K_y V_{by}^2$$
$$D_z = K_z w_z^2$$

where K_x, K_y and K_z are the drag forces constants.

Table 3.2 shows the measured values for the physical parameters of the boat. This parameters were obtained in experiments as shown in (SANTOS, 2011).

TABLE 3.2 – Value of the physical parameters

Parameters	**Value**
K_y, K_z	35.12 Ns^2/m^2
K_x	7.025 Ns^2/m^2
m	14.81 Kg
J	3.37 $\mathrm{Kg/m}^2$
F_p	1.4 N
d	0.56m

3.2 A first approach to the ASV

This section describes the first attempt to redesign the ASV, to reduce costs, and to integrate with the MOOS-IvP software. This first implementation was used to demonstrate that the former autonomous architecture could be integrated with the MOOS-IvP software. This first architecture was important to lay the foundations and learn how the MOOS-IvP software behaved in a simpler architecture. Section 3.3 shows the use of a modular autonomous architecture, with reduced costs.

Santos (SANTOS *et al.*, 2013) used a navigation system consisting of an ASV and a Ground Control Station (GCS). In this system, the ASV sends the telemetry data from the Global Position System (GPS) receiver and the Inertial Measurement Unit (IMU) to the GCS. The GCS uses a custom autonomous navigation software, developed in Matlab, to implement the decision-making, control and the sensor fusion algorithms. After processing those data, the GCS sends back to the ASV motor actuators commands. The communication link between the ASV and the GCS is a RS-232 radio link. Due to this architecture, the ASV requires a reliable communication link to run a mission

This first architecture takes this same approach as a starting point. The ASV hardware was modified in order to reduce costs. The autonomous navigation software and the sensor fusion algorithms are running in the GCS. The autonomous navigation software is the MOOS-IvP, detailed in chapter 2. Figure 3.3 shows a diagram that illustrates the ASV system.

3.2.1 Hardware Modifications

The original ASV developed by Santos (SANTOS *et al.*, 2013) used a more costly hardware, The aim of this first architecture was to lower the cost of the ASV system and integrate it with the MOOS-IvP software. Therefore, most of the embedded hardware was redesigned.

The embedded computer was replaced by a popular Arduino Uno R3 Microcontroller. The IMU and the digital compass were replaced by a lower cost 10-DoF IMU (3 axis digital

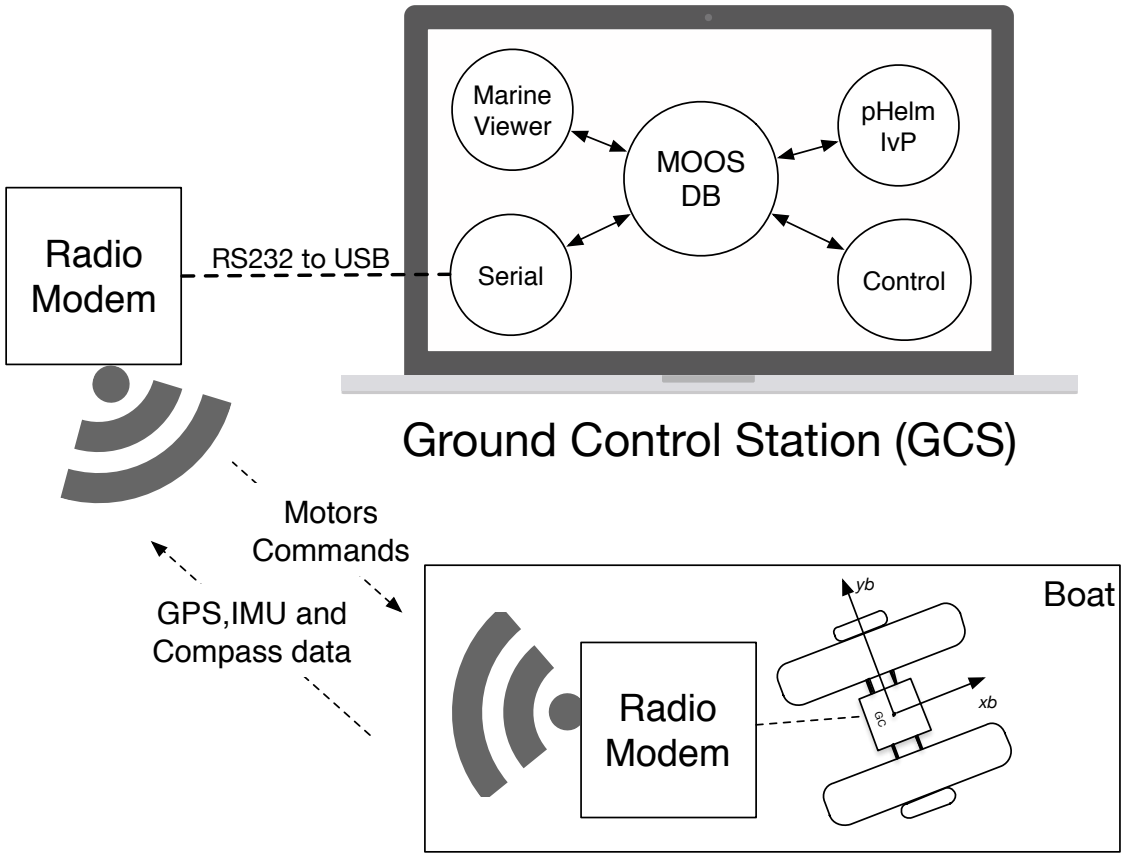

FIGURE 3.3 – The ASV system: the boat sends telemetry data to the GCS, the GCS process these data using the MOOS-IvP software and replies with navigation commands.

compass, 3 axis accelerometer, 3 axis gyroscope and a barometer), the GY-80 (FILIPEFLOP, 2016). The GY-80 is based on the chipsets ADXL345 (accelerometer), L3G4200D (gyroscope), HMC5883L (compass) and BMP085 (pressure sensor). The communication link with the GCS is done with the RF Modem SS7-900EXT ICP Das. Due to the high power consumption of the RF modem, this setup allows to have roughly 1.5 hour of autonomy.

The GCS consists of a computer connected to a RF Modem, an external battery pack for the RF Modem and a joystick for manual control.

Table 3.3 shows the list of the embedded hardware components and their cost.

TABLE 3.3 – Embedded hardware components for the first architecture.

Component	**Price in US$**
Ardunio Uno R3 Microcontroller	US$ 10
GPS Receveir Globalsat ET-332	US$ 25
IMU GY-80	US$ 20
Relay H-Bridge	US$ 5
USB to RS232 converter	US$ 5
RF Modem SS7-900EXT ICP Das	US$ 100/pair
Sealed battery Unipower 12V 4.5Ah	US$ 15

Figure 3.4 shows a diagram of the embedded electronic hardware. Figure 3.5 shows a picture of the real embedded hardware represented in the diagram of Figure 3.4.

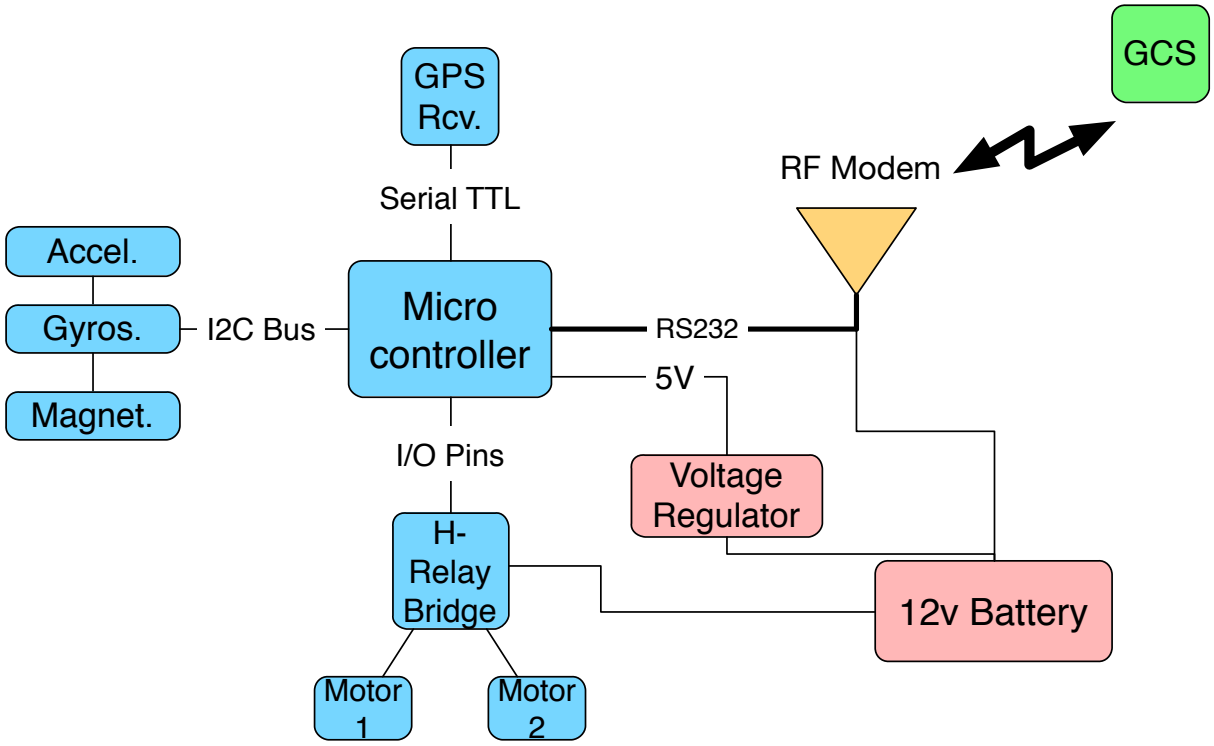

FIGURE 3.4 – Diagram of the ASV embedded hardware in the first implementation. The embedded hardware is represented in blue. The power source is represented in red. The GCS is represented in green.

3.2.2 Microcontroller Software

The microcontroller software runs on the Arduino Uno R3 Microcontroller. It was developed in a variant of C++ using the Arduino IDE. This software is responsible for commanding the motors, interacting with the GPS receiver, the IMU (digital compass, accelerometers and gyroscope) and receiving and sending data through the RF Modem.

The IMU uses the I^2C protocol and the GPS receiver uses an USART with a 9600 baud rate. The relay H-Bridge (SANTOS, 2011) is controlled with 4 digital TTL signals and the RF modem uses RS-232 at 9600 baud rate.

The microcontroller software uses several libraries. Description and examples on how to use them can be found in (MARGOLIS, 2011).

- `Wire.h`: This library is part of the Arduino common libraries. Its main job is to control the I^2C peripheral on the Atmel 328P microcontroller in the Arduino Uno R3.
- `SoftwareSerial.h`: This library is part of the Arduino common libraries. Its job is to

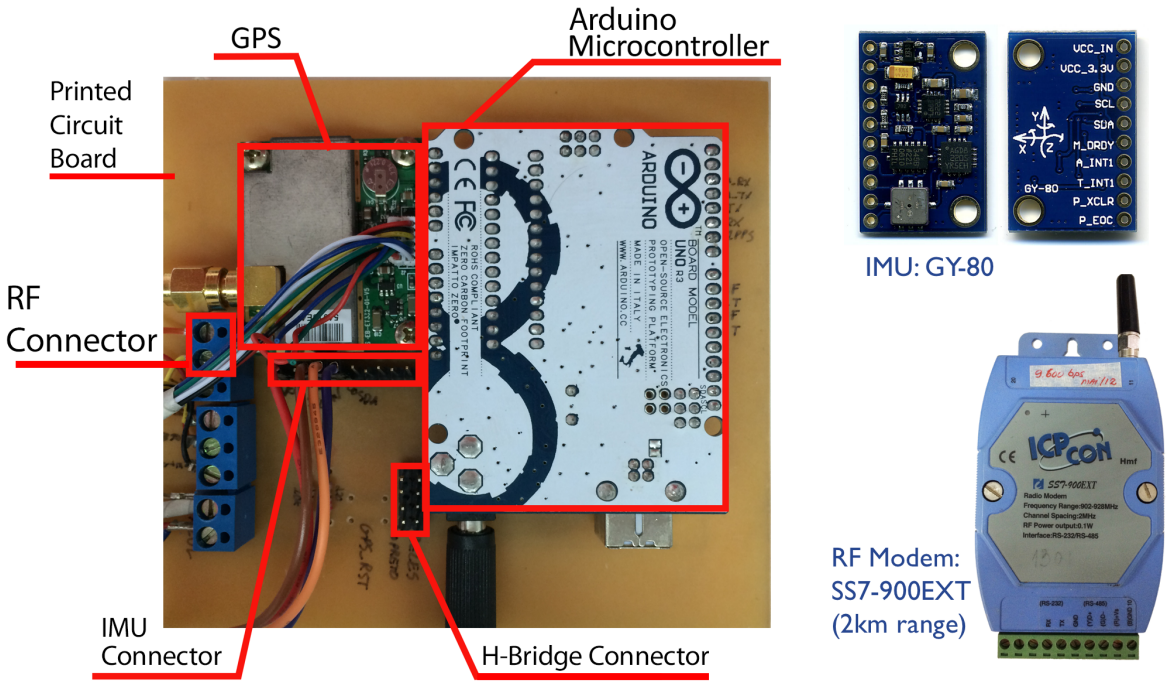

FIGURE 3.5 – Embedded hardware.

create a USART interface by software. The Atmel 328P has only one USART peripheral, and the ASV needs two independent serial interfaces.

- `TinyGPS.h`: This library is developed by Arduiniana (ARDUINIANA, 2016). This library is a GPS NMEA Parser. It tests the checksum of the NMEA sentence and extracts several data in an easy to use object.

- `stdlib.h`: This library is common to several C/C++ compilers. This version is specific for the 8-bit AVR compiler. This library is used to perform floating point operations in the 8-bit microcontroller.

- `GY80IMU.h`: This is a custom made library. This library is responsible for interfacing with the GY-80 IMU. It is based on the datasheet of the ADXL345 (accelerometer), L3G4200D (gyroscope), HMC5883L (digital compass) and BMP085 (pressure sensor). It has several options for calibrating, selecting the scale of the measurements and extracting information from all the chipsets. It was developed in C++ and makes use of the `Wire.h` library.

Figure 3.6 represents the microcontroller software diagram. The main loop runs at approximately 4 Hz, during each loop it activates the motors based on the received data, read the GPS, IMU and sends the data back. At the end of the loop it verifies the communication link, if the signal is lost the boat stops until communication is reestablished. Although the main loop runs at 4Hz, the GPS runs at 1 Hz and between this time it holds the last acquired position.

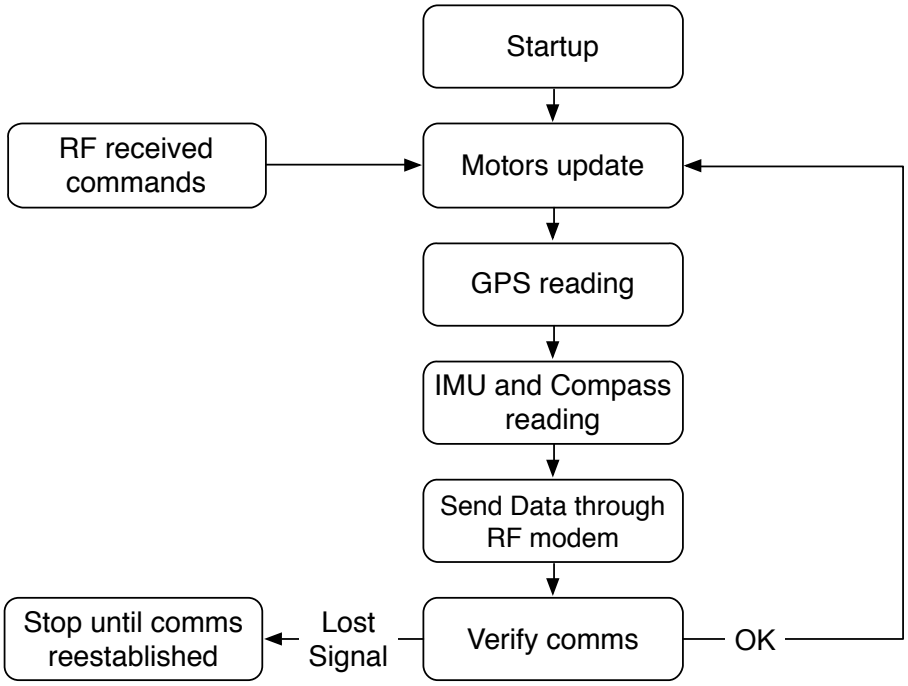

FIGURE 3.6 – Arduino embedded software. The software is composed of a single main loop that controls the motors, reads sensors and sends telemetry data.

During the start-up step in Figure 3.6, the accelerometer, the gyroscope, and the digital compass are calibrated. The GPS is set to use NMEA messages at a frequency of 1 Hz. Static navigation is also disabled. As the GPS was intended to be used in higher speed vehicles, when running in lower speeds (less than 2 m/s) it holds the position. Disabling the static navigation allows the GPS to be used in lower speed vehicles. The peripherals, I^2C and USART are also configured during the start-up routine.

A safety step verifies the communication link. If there is a loss in signal, the ASV stops until communication is reestablished. This allows the operator to try to reestablish the communication link.

3.2.3 The MOOS-IvP software

This subsection describes how the MOOS-IvP software is used in this architecture. In this first implementation it was used the MOOS-IvP software version 15.5.

The embedded hardware acts as the frontseat computer. The MOOS-IvP is running in GCS. The GCS acts as the backseat computer. In this configuration the GCS runs only one MOOS Community with several MOOS applications. Figure 3.7 represents the backseat driver paradigm with the MOOS community.

The MOOS Apps used and available in the MOOS-IvP repository are briefly described below. For more information refer to the MOOS and MOOS-IvP manual (MOOS, 2016) (MOOS-

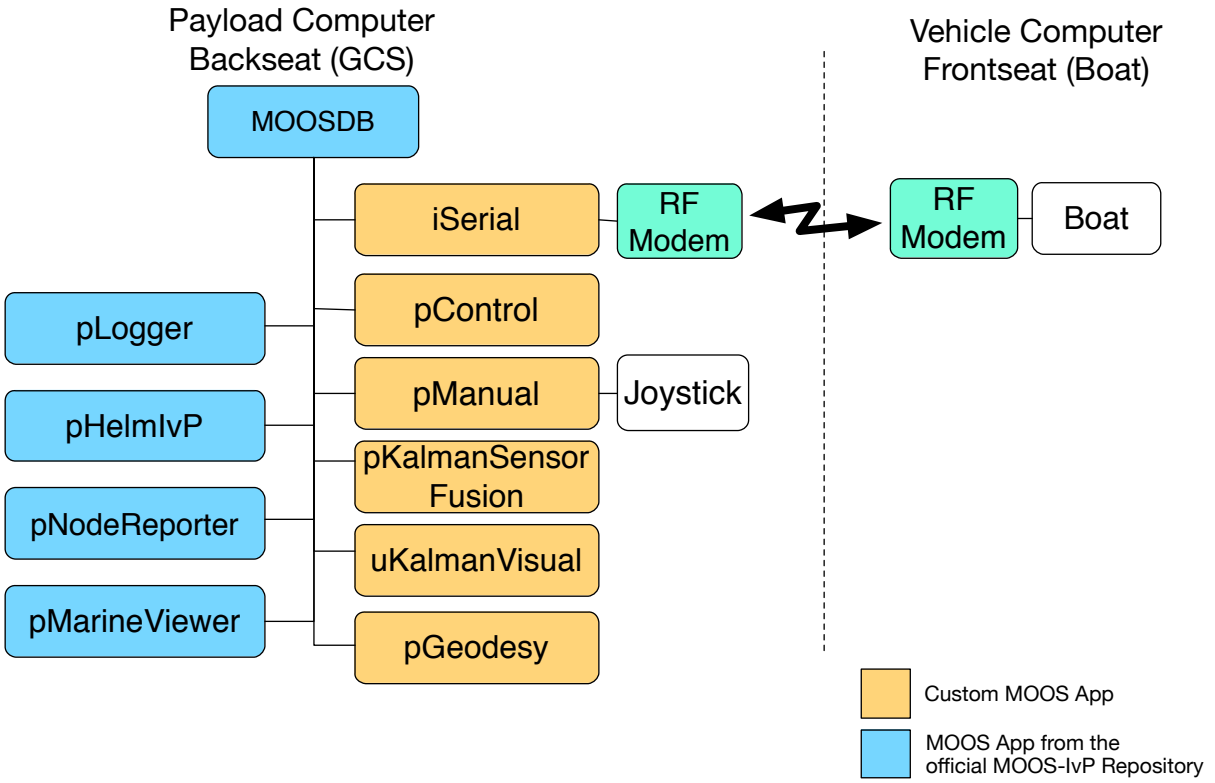

FIGURE 3.7 – MOOS-IvP Apps. In yellow, the custom MOOS Apps developed for the ASV. In blue, the MOOS Apps available in the MOOS-IvP repository.

IVP, 2016).

- **MOOSDB**: The central MOOS App that coordinates the communication between several processes.
- **pMarineViewer**: This MOOS App provides a interface to visualize the mission, the vehicles and other variables. It has a degree of customization and usually it is used to deploy the mission.
- **pLogger**: Creates a log of the whole mission which can be used to replay and analyze the mission later.
- **pNodeReporter**: Creates a custom message to be consumed by the pMarineViewer App and display the vehicle.

The custom MOOS Apps developed are briefly described below. A complete reference for the custom MOOS applications is given in Chapter 4. The source code for these applications can be found on the LMI FTP server (`ftp://labattmot.ele.ita.br/ele/david/moos-ivp-extend/`).

- **iSerial**: This MOOS App is responsible for interfacing the GCS computer with the RF

modem. It creates a USB virtual serial port and decodes the data messages sent by the ASV to the GCS.

- **pControl**: This MOOS App implements the lookup table (Table 3.1) and it is used to determine the states of each motor. Its output is transmitted to the iSerial App to be send to the boat.

- **pManual**: This MOOS App creates a manual interface to control the boat. The boat can be controlled by a keyboard or by a joystick controller.

- **pGeodesy**: This MOOS App is responsible for converting from global to local coordinates. It uses the Earth model WGS-84 and implement the equations presented at (FARRELL; BARTH, 1999).

- **pKalmanSensorFusion**: This app implements a GPS, Compass and IMU sensor fusion using an Extended Kalman Filter. The Extended Kalman Filter is a mathematical technique that uses a model of the system to estimate errors in a navigation system and combine data from different sensors in order to improve the accuracy of the position and heading. The Extended Kalman Filter used was developed in (SANTOS *et al.*, 2013).

- **uKalmanVisual**: This app implements a visual interface to observe the Extended Kalman Filter states. This app helps to easily tune the Extended Kalman Filter.

- **uBoatSimulator**: Although not present in the Figure 3.7, this app implements the boat dynamic equations. It helps in simulating the dynamics of the boat and test new MOOS Apps before deploying.

Figure 3.8 shows how the Kalman Filter is used in conjunction with the MOOS-IvP in the GCS.

The GCS computer is a Macbook Pro running the Mac OS X 10.10.5 operating system. The computer uses the processor Intel Core i5 2.4GHz, with 8 GB 1600 MHz DDR3 RAM Memory. The graphic card is an Intel Iris 1536 MB.

3.2.4 Remarks on the first implementation

As shown in the experimental results in Chapter 5, this architecture worked well. The ASV was capable of performing a full mission with the sensor fusion algorithms and different behaviors using the MOOS-IvP software.

However, the proposed architecture is not scalable for several collaborative vehicles. Collaborative vehicles would need a communication network, also using other computers in the

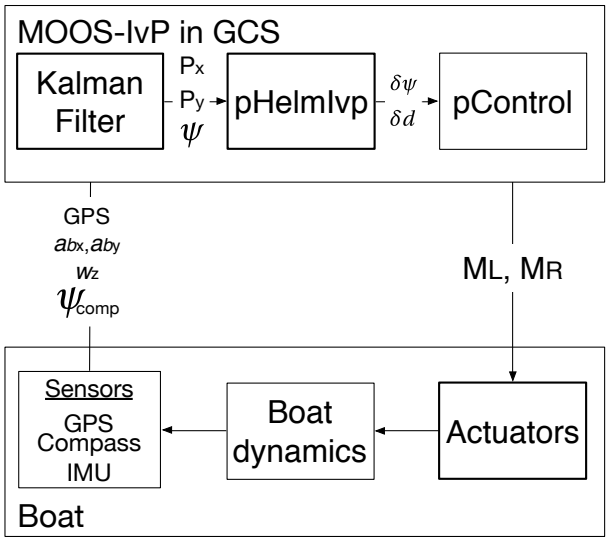

FIGURE 3.8 – This block diagram shows how the Kalman Filter is used for the sensor fusion. The GCS receives GPS, IMU and compass data (through the application iSerial), it processes those data and outputs the estimated position and heading to the pHelmIvp.

GCS is a costly solution. Other computers would be necessary, because the MOOS-IvP software runs a community for each vehicle. Running several communities, each one running a decision-making software and sensor fusion algorithms might overload the computer. A distributed approach, with each vehicle running its own autonomy software is a better alternative for a multi-vehicle mission (SETO, 2012; MOOS-IVP, 2016).

The navigation is dependable of a reliable communication link with the GCS. For autonomous systems a reliable communication link is usually a requirement. However, in the event of a communication loss the navigation is interrupted and in an open environment it may result in the loss of the vehicle.

The communication link can have delays. The presence of delays can degrade the performance of the controller. In order to address this issue in this approach it is necessary to improve the communication channel. This improvement may result in a significant cost to this low cost ASV system. However, this research has not verified the communication link or the transmission delays.

A significant part of the hardware costs is due to the expensive radio modem. Although there are cheaper alternatives, they might not be so reliable in longer distances from the GCS.

This first implementation was needed to test the MOOS-IvP with an architecture that worked well with this nautical structure (SANTOS *et al.*, 2013).

3.3 The second approach to the ASV: the modular architecture

This section describes another architecture for the ASV system. It uses several components and software modules from the first approach. However, it differs significantly in several aspects.

One of the main differences of this approach compared to the the first approach is the presence of the payload computer inside the ASV. This makes it possible to run the MOOS-IvP software embedded in the ASV. This creates a mission independence between the ASV and the GCS in terms of navigation. The GCS is not used to perform calculations for the ASV and therefore does not interfere in the decision-making and sensor fusion algorithms.

This implementation uses a modular approach for ASV. This modular approach allows the vehicles to be easily adapted for different situations, such as changes in the payloads and different sensors combinations. Because with the modular approach it is not necessary to provide new sensors interfacing, or to modify the embedded hardware. Also the use of a payload computer allows the use of different and proprietary software to interface with the payload sensors.

3.3.1 Hardware Modifications

The new implementation follows closely the backseat driver paradigm in both the software and electronic hardware architecture. The ASV electronic hardware consists of two separate parts: the embedded hardware and the payload computer.

The embedded hardware is almost the same as the previous approach, and it is responsible for motor control, power sources and navigation sensors interfacing. This part is the frontseat.

The payload computer is the backseat computer. It is responsible for navigation decisions and GPS/IMU/Compass sensor fusion algorithms. It is also responsible for interfacing with new sensors.

Figure 3.9 represents the new ASV architecture.

In this new approach, the frontseat computer communicates with the backseat computer through a USB bus, instead of the radio link. The ASV communicates with the GCS through a WiFi link. With this new communication link it is possible to add a USB camera and send real-time video feed (640x480 at 15 frame rate) from the ASV to the GCS. Due to the lower power consumption of this new setup, the ASV is capable of performing missions up to 3.5 hours of continuous navigation without the need of recharging. The GCS was also modified

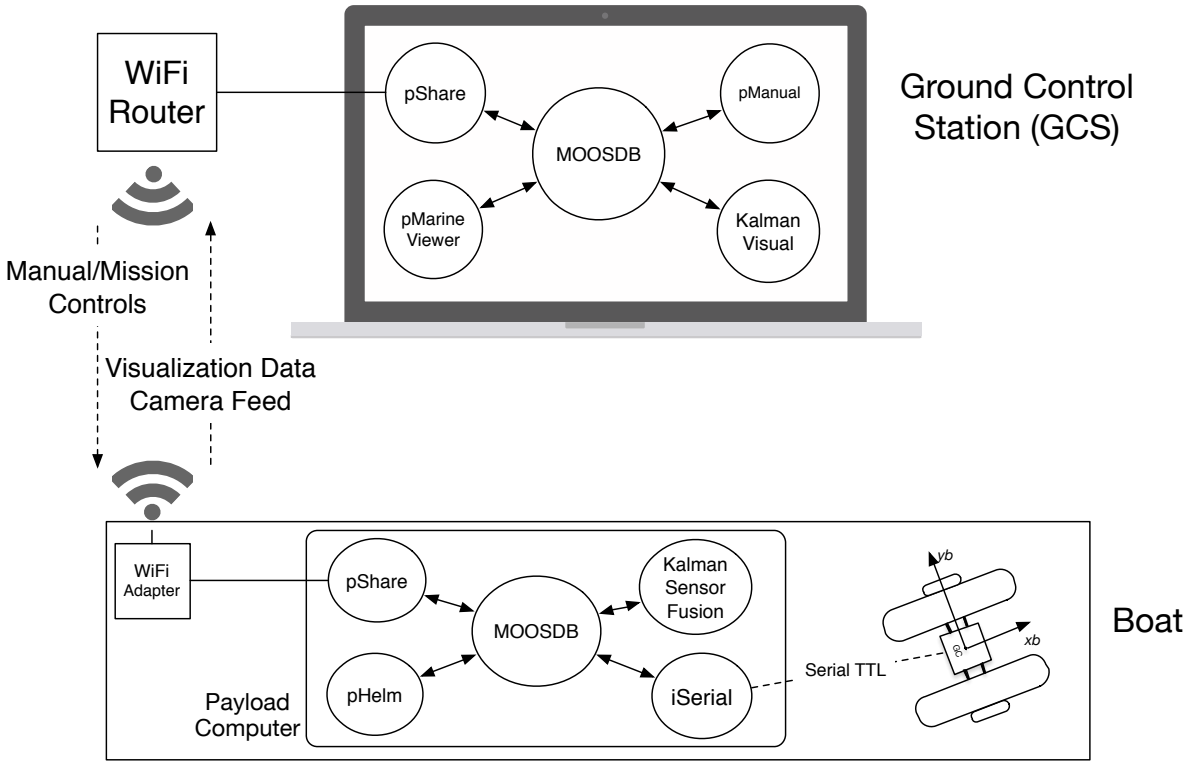

FIGURE 3.9 – Block diagram of the ASV system: the ASV and the GCS. Both the GCS and the ASV use the MOOS-IvP software. Communication between the ASV and the GCS happens through a WiFi link.

and it consists of a computer connected to a WiFi router and a joystick for manual control.

Figure 3.10 represents a block diagram of the ASV hardware.

Table 3.4 shows the list of the ASV hardware components and their cost.

TABLE 3.4 – Embedded hardware components for the second approach.

Component	**Price in US$**
Ardunio Uno R3 Microcontroller	US$ 10
GPS Receveir Globalsat ET-332	US$ 25
IMU GY-80	US$ 20
Relay H-Bridge	US$ 5
Raspberry Pi 2	US$ 30
TP-Link TL-WN8200ND WiFi Receiver	US$ 30
Logitech C525 USB Camera	US$ 40
Voltage Regulator	US$ 5
Sealed battery Unipower 12V 4.5Ah	US$ 15

The payload computer (a Raspberry Pi 2) is responsible for the autonomous decision-making, the sensor fusion algorithms and the ASV control. It runs the MOOS-IvP software. Also, it is responsible to communicate with the GCS by sending UDP messages through the USB WiFi adapter.

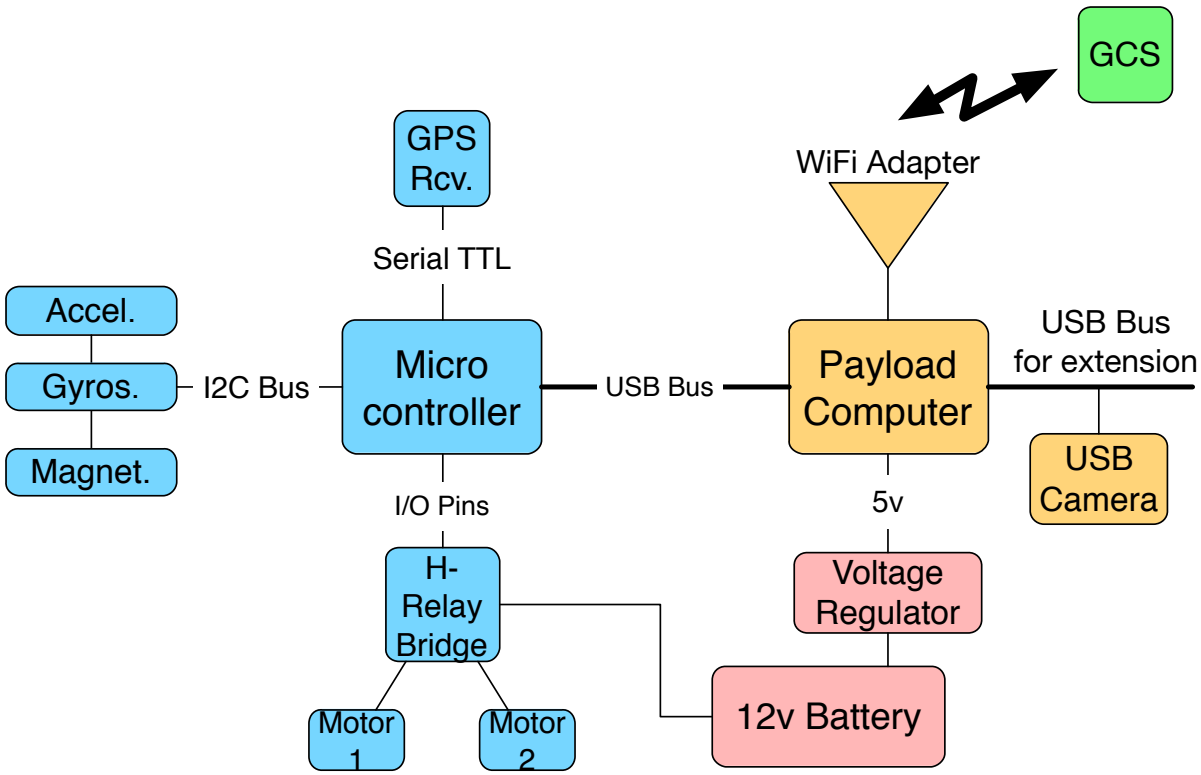

FIGURE 3.10 – Block diagram of the ASV hardware in the second approach. The embedded hardware is represented in blue. The power source is represented in red. The GCS is represented in green. In yellow it is represented the payload computer, WiFi adapter and the USB camera.

The payload computer communicates with the microcontroller (frontseat computer) using a USB bus. The use of a USB bus endorses the modular approach, as it allows an easy way to customize the payload computer, the communication channel and to add new sensors. This modular approach provides the means for vehicles to be adapted to wide range of missions. As an example, a USB camera was added to the ASV for surveillance purposes.

3.3.2 Microcontroller Software

The microcontroller software was altered slightly. It uses the same software skeleton as the previous approach and the same libraries for sensor interfacing. However, there is no need to verify if the communication signal was lost. Although exists the possibility of the MOOS-IvP software hangs, this kind of error verification was not addressed. Also it was modified to send data to the payload computer, instead of the RF modem. Figure 3.11 represents the diagram of the new microcontroller software.

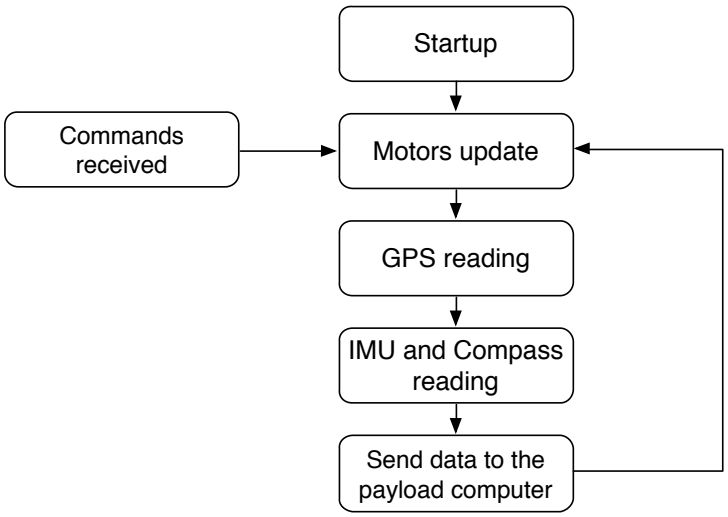

FIGURE 3.11 – Arduino embedded software. The software is composed of a single main loop that controls the motors, reads the sensors and communicates with the payload computer.

3.3.3 The MOOS-IvP software in the payload computer and in the GCS

In this new implementation, significant changes occurred in the MOOS-IvP communities. The custom applications (Chapter 4) were not modified, therefore reducing software development costs. With this new approach, two MOOS communities were used. One community is in the payload computer (on-board the ASV) and the other one is in the GCS. Figure 3.12 shows the MOOS communities and the MOOS Applications used in both the ASV and in the GCS.

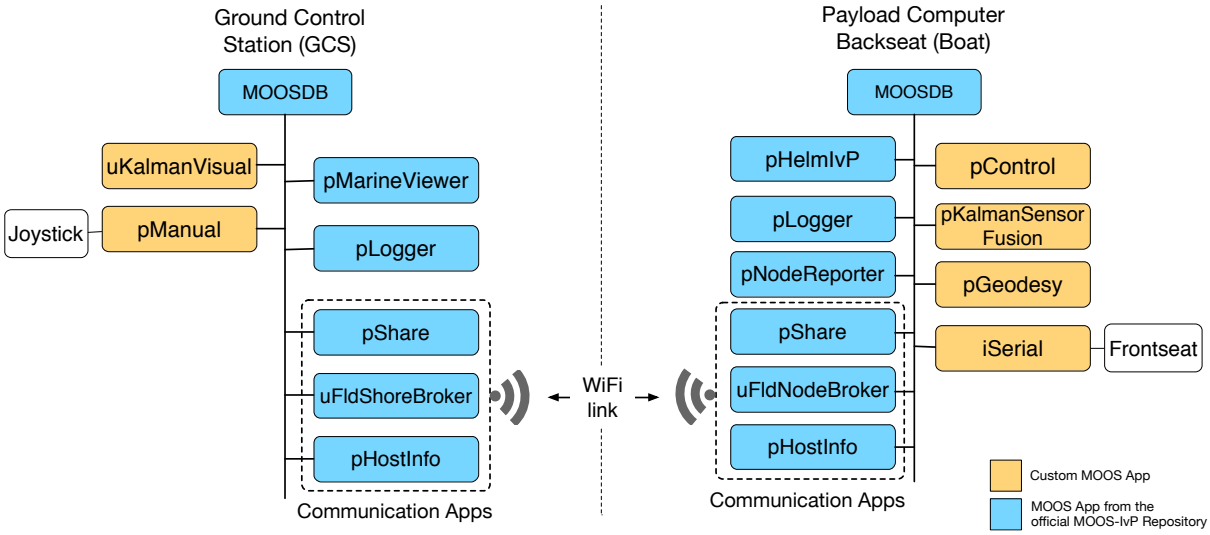

FIGURE 3.12 – MOOS-IvP Apps. In the left there is a diagram of the MOOS apps used in the GCS, in the right the diagram for the MOOS apps used in the payload computer. The yellow blocks correspond to the custom MOOS Apps developed for the ASV. The blue blocks correspond to the MOOS Apps available in the MOOS-IvP repository.

3.3.3.1 The MOOS-IvP software in the payload computer

The payload computer (a Raspberry Pi 2) uses a Linux Raspian operating system. In this second approach, it was used the MOOS-IvP software version 15.5.

The MOOS Apps used in the payload computer and available in the official MOOS-IvP repository (MOOS-IVP, 2016) are briefly described below. For more information refer to the MOOS and MOOS-IvP manual (MOOS, 2016; MOOS-IVP, 2016).

- **MOOSDB**: The central MOOS App that coordinates the communication between several processes.
- **pLogger**: Creates a log of the whole mission which can be used to replay and analyze the mission later.
- **pNodeReporter**: Creates a custom message to be consumed by the pMarineViewer App and display the vehicle.
- **pShare**: This app creates a socket to transmit MOOS messages between MOOS communities running in different machines, such as the ASV and the GCS.
- **uFldNodeBroker**: A tool for brokering connections between a node (vehicle) with the shoreside computer (GCS).
- **pHostInfo**: Discovers and shares the IP address of the machine with the MOOS community. It is necessary for the uFldShoreBroker and the uFldNodeBroker app.

The custom MOOS Apps used in the payload computer are briefly described below. A complete reference for the custom MOOS applications is given in Chapter 4. The source code for these applications can be found on the LMI FTP server (`ftp://labattmot.ele.ita.br/ele/david/moos-ivp-extend/`).

- **iSerial**: This MOOS App is responsible for interfacing with the microcontroller. It also decodes the data messages sent from the ASV.
- **pControl**: This MOOS App implements the lookup table (Table 3.1) and it is used to determine the states of each motor. Its output is transmitted to the iSerial App to be sent to the ASV.
- **pGeodesy**: This MOOS App is responsible for converting from global to local coordinates. It uses the Earth model WGS-84 and implement the equations presented at (FARRELL; BARTH, 1999).

- **pKalmanSensorFusion**: This app implements a GPS, Compass and IMU sensor fusion using an Extended Kalman Filter. The Extended Kalman Filter used was developed in (SANTOS *et al.*, 2013).

3.3.3.2 The MOOS-IvP software in the GCS

The MOOS Apps used in the GCS computer and available in the MOOS-IvP repository are briefly described below. For more information refer to the MOOS and MOOS-IvP manual (MOOS, 2016), (MOOS-IVP, 2016). The MOOS-IvP software version used in the GCS is the 15.5.

- **MOOSDB**: The central MOOS App that coordinates the communication between several process.
- **pMarineViewer**: This MOOS App provides a interface to visualize the mission, the vehicles and other variables. It has a degree of customization and usually it is used to deploy the mission.
- **pLogger**: Creates a log of the whole mission which can be used to replay and analyze the mission later.
- **pShare**: This app creates a socket to transmit MOOS messages between MOOS communities running in different machines, such as the ASV and the GCS.
- **uFldShoreBroker**: A tool for brokering connections between the shoreside computer (GCS) with several nodes (vehicles).
- **pHostInfo**: Discovers and shares the IP address of the machine with the MOOS community. It is necessary for the uFldShoreBroker and the uFldNodeBroker app.

The custom MOOS Apps used in the GCS are briefly described below. A complete reference for the custom MOOS applications is given in Chapter 4.The source code for the custom applications can be found on the LMI FTP server (`ftp://labattmot.ele.ita.br/ele/david/moos-ivp-extend/`).

- **pManual**: This MOOS App creates a manual interface to control the ASV. The ASV can be controlled by a keyboard or by a joystick controller.
- **uKalmanVisual**: This app implements a visual interface to observe the Extended Kalman Filter states. This app helps to tune the matrices of the Extended Kalman Filter.
- **uBoatSimulator**: Although not presented in Figure 3.12, this app implements the boat dynamic equations. It helps simulating the dynamics of the boat and testing new MOOS Apps before deployment.

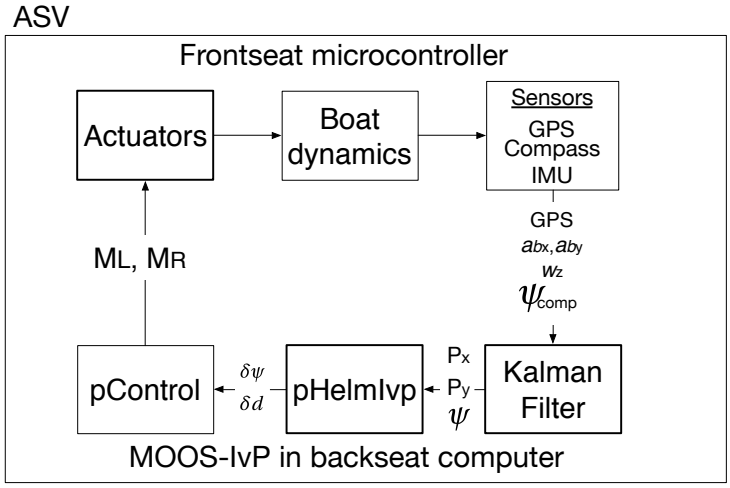

FIGURE 3.13 – This block diagram shows how the Kalman Filter was used f sensor fusion. It receives GPS, IMU and compass data (through the application iSerial), it processes those data and outputs the estimated position and heading to the pHelmIvp.

Figure 3.13 shows how the Kalman Filter is used in conjunction with the MOOS-IvP in this modular approach.

The GCS computer is a Macbook Pro running the Mac OS X 10.10.5 operating system. The computer uses the processor Intel Core i5 2.4GHz, with 8 GB 1600 MHz DDR3 RAM Memory. The graphic card is an Intel Iris 1536 MB.

3.3.4 The Surveillance Software for the USB Camera

A USB Camera, the Logitech C525 was added to the USB bus in the payload computer. It is primarily used for monitoring and visualization in real time. It is possible to add computer vision integrated to the MOOS-IvP software using the software library OpenCV (OPENCV, 2016) and (NEWMAN, 2013). However, as the camera is being used for video monitoring, for simplicity it was used the open source Motion software (MOTION, 2016). This software is easy to configure and creates a simple video webserver accessible through a specific port. The GCS can access the camera feed using the IP address of the ASV payload computer.

3.3.5 Remarks on the second approach

This is the final ASV approach used in this dissertation. This new ASV implementation provides a scalable way to deploy missions with several vehicles regardless of the communication link with the GCS. The payload computer (backseat) is responsible entirely for the navigation. The GCS acts only as a monitoring station. Therefore the vehicle can run autonomously even in the event of a lost of communication signal.

Using this modular approach new sensors can be easily added and the communication link can be changed in order to meet different requirements. The modular approach also benefits collaborative missions as several vehicles can be deployed with different payload computers and sensors without redesigning each vehicle.

Chapter 5 brings experimental results for this implementation.

4 The developed MOOS-IvP modules

This section describes in detail the several MOOS Applications developed for this dissertation. The source code for this modules is available at the `ftp://labattmot.ele.ita.br/ele/david/moos-ivp-extend/` LMI FTP server or upon direct request to the author.

4.1 The pControl Application

This MOOS Application implements the lookup table (Table 3.1) and it is used to determine the states of each motor.

It compares the current position with the desired position to determine the position error. It compares the current heading with the desired heading to determine the heading error. Using these errors this application searches the lookup table and determines what are next motors states.

To run smoothly this application should be set to run at a frequency of at least 4Hz.

4.2 The pGeodesy Application

This application is responsible for converting global to local coordinates. It uses the Earth model WGS-84 and implement the equations presented at (FARRELL; BARTH, 1999). Other methods and Earth model approximations can be used.

This application receives as input parameters from the community configuration file the height in respect with sea level. This information can found in several websites, such as, `http://www.daftlogic.com/sandbox-google-maps-find-altitude.htm`.

This application uses the ellipsoid parameters found in Table 4.1

To convert from Earth coordinates to Earth Centered Earth Fixed (ECEF) rectangular frame it is used the following equations

$$x = (N + h)\cos(\lambda)\cos(\phi) \tag{4.1}$$

TABLE 4.1 – Ellipsoid parameters for the conversion from global to local coordinates using the Earth model WGS-84

Parameter	**Symbol or Value**
Earth semi-major axis	$a = 6378137$ m
Earth semi-minor axis	$b = 6356752.314245$ m
Latitude	λ
Longitude	ϕ
Altitude relative to sea level	h
Earth flattening	$f = \frac{a-b}{a}$
Earth First eccentricity	$e = \sqrt{f(2-f)}$
Earth Normal length	$N(\lambda) = \frac{a}{\sqrt{1-e^2 \sin^2(\lambda)}}$

$$y = (N + h)\cos(\lambda)\sin(\phi) \tag{4.2}$$

$$z = (N(1 - e^2) + h)\sin(\lambda) \tag{4.3}$$

The input of this system are the latitude (λ), longitude (ϕ) and altitude (h). The output are the coordinates in the rectangular ECEF frame (x, y and z) .

Converting from the global ECEF frame to the local frame:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{Local}} = R_{e2n} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{ECEF ret}} - R_{e2n} \cdot \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}_{\text{ECEF ret}} \tag{4.4}$$

where x_0, y_0 are z_0 the origin coordinates of the local system in the ECEF rectangular frame and R_{e2n} is the matrix given below:

$$R_{e2n} = \begin{pmatrix} -sin(\lambda) & 0 & \cos(\lambda) \\ 0 & 1 & 0 \\ -\cos(\lambda) & 0 & -\sin(\lambda) \end{pmatrix} \cdot \begin{pmatrix} -\cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.5}$$

The inputs are the rectangular coordinates in ECEF (previously calculated), the origin of the system in rectangular ECEF, and the latitude and longitude (in matrix R_{e2n}). The output is the coordinates in the rectangular local frame.

It is intended to run at the same frequency of the received GPS messages, therefore, at a frequency of 4 Hz.

4.3 The iSerial Application

This MOOS Application is responsible for interfacing the GCS with the RS-232 RF Modem, as well as, interfacing with the payload computer (in the modular approach). This application creates a virtual serial port using the USB CDC profile. This application has dependencies in the popular Boost library (BOOST, 2016), specially in the `boost::asio`. Using this library it is possible to create a clean code that works on Windows, Mac OS X and Linux computers.

Beyond creating a virtual serial interface, this application is responsible for decoding messages received from the ASV. It separates the serial messages into the appropriate MOOS variables. It also subscribes to the desired motor states to encode in a message that the ASV decodes.

This application receives as configuration parameters the path to the USB serial port and the respective baudrate.

An example of a typical message received is:

`GPS_LAT=value,GPS_LONG=value,GPS_TIME=value.`

The iSerial application decodes this message, converts characters to floating numbers and posts the appropriate MOOS variables in the MOOSDB.

An example of a typical message sent to the ASV is: `M1=value,M2=value`. The embedded software in the ASV microcontroller is responsible for decoding this message and activate the motors.

Because this application does not work asynchronously (despite the fact it is using the `boost::asio` library), this application must run at high frequency.

4.4 The pManual Application

This MOOS App creates a manual interface to control the ASV. Using this application the ASV can be controlled by using a keyboard or by a joystick controller.

This application does not receive any configuration parameters on the community configuration file (`.moos`). This application has dependencies on the SDL2 library (SDL2, 2016) for communicating with a joystick. If a joystick is not connected the ASV still can be controlled with the keyboard. The SDL2 is a game library for C++ that has a module for receiving joystick inputs. This application runs with joystick in Windows, Mac and Linux computer as long as it has the correct drivers.

If the manual mode is set, the application disables the IvP Helm and pControl modules

and gives direct command to the actuators.

This application runs on two POSIX threads:

1. The first thread, the main thread is responsible for interfacing with the keyboard. It puts the terminal in non-canonical mode, therefore it will read a character when it is pressed. This way it is not necessary to hit `enter` to give the command.
2. The second thread, initiates the SDL2 joystick module and waits for commands. An axis threshold was set to prevent tiny calibration changes influences the ASV manual navigation.

The table 4.2 gives a list of all commands available

TABLE 4.2 – Manual commands for the pManual module

Command	**Keyboard**	**Joystick**
Forward	w	Axis 0
Left-Forward	q	Axis 0
Right-Forward	e	Axis 0
Backward	x	Axis 0
Right-Backward	c	Axis 0
Left-Backward	z	Axis 0
Stop	s	Release Axis 0
Start Manual Control	m	Button 8
Return Control to IvP Helm	n	Button 9

This application was tested suing different joystick controllers such as the Xbox 360 joystick, the Logitech F310 and the F510 joystick controllers. This application is intended to be running at a frequency between 1 Hz and 5 Hz.

4.5 The pKalmanSensorFusion

This application implements an Extended Kalman Filter Sensor Fusion algorithm in order to aid the localization of the ASV. This application has dependencies in the libraries `eigen3` (EIGEN, 2016) and the `boos::odeint` (BOOST, 2016). It runs on Windows, Mac and Linux operating systems.

This Extended Kalman Filter Formulation was proposed by Santos (SANTOS *et al.*, 2013). His work uses the GPS, Compass and IMU running at the same frequency of 1Hz. The implemented Kalman filter in this dissertation was modified so the IMU could run at a higher rate than the GPS.

4.5.1 The Extended Kalman Filter

The new Extended Kalman Filter equations are given below. The complete derivation of this Extended Kalman Filter can be found in (SANTOS, 2011; SANTOS *et al.*, 2013). There are different formulations for the Extended Kalman Filter. This dissertation uses only the loosely couple formulation (S. Grewal *et al.*, 2007).

The main idea of the filter is to use the inertial navigation system (INS) model with the acceleration from the accelerometers and the angular rate from the gyroscope. However, the inertial units have sources of errors such as a bias and noise. This bias, if integrated over time, diverges the estimated position from the real position. The Kalman Filter is responsible for estimating the bias errors. To estimate the bias the Kalman Filter compares the estimated position of the INS with the GPS and the digital compass inputs.

This formulation uses the state:

$$X^{\text{INS}}(t) = [V_x(t), V_y(t), P_x(t), P_y(t), \psi(t)]^T$$

In this state $V_x(t)$ and $V_y(t)$ are the velocities in the local frame. Accelerations from the vehicle frame (a_{bx} and a_{by}) to the local frame (a_x and a_y) can be converted using eqs. (4.6) and (4.7):

$$\begin{pmatrix} a_x(t) \\ a_y(t) \end{pmatrix} = \begin{pmatrix} \cos(\psi(t)) & -\sin(\psi(t)) \\ \sin(\psi(t)) & \cos(\psi(t)) \end{pmatrix} \begin{pmatrix} a_{xb}(t) \\ a_{yb}(t) \end{pmatrix} \tag{4.6}$$

$$\begin{aligned} \dot{V}_x(t) &= a_x(t) \\ \dot{V}_y(t) &= a_y(t) \end{aligned} \tag{4.7}$$

The true state and the error state vectors are defined as:

$$X^V(t) = [V_x^V(t), V_y^V(t), P_x^V(t), P_y^V(t), \psi^V(t)]^T$$

$$X^E(t) = [X^{\text{INS}}(t) - X^V(t), \text{Bias}(t)]^T$$

where:

$$\text{Bias} = [\text{bias}_{ax}, \text{bias}_{ay}, \text{bias}_{wz}]^T$$

Therefore the Kalman filter state vector has dimension 8x1.

The INS equations are represented by equation:

$$\dot{X}^{\mathrm{INS}}(t) = \begin{pmatrix} \cos(x_5^{\mathrm{INS}})\,u_1^{\mathrm{INS}} - \sin(x_5^{\mathrm{INS}})\,u_2^{\mathrm{INS}} \\ \sin(x_5^{\mathrm{INS}})\,u_1^{\mathrm{INS}} + \cos(x_5^{\mathrm{INS}})\,u_2^{\mathrm{INS}} \\ x_1^{\mathrm{INS}} \\ x_2^{\mathrm{INS}} \\ u_3^{\mathrm{INS}} \end{pmatrix}$$

The input of the IMU measurements vector, U^{INS}, is defined as:

$$U^{\mathrm{INS}}(t) = [u_1^{\mathrm{INS}}(t), u_2^{\mathrm{INS}}(t), u_3^{\mathrm{INS}}(t)]^T = [a_{xb}^{\mathrm{INS}}(t), a_{yb}^{\mathrm{INS}}(t), \psi^{\mathrm{INS}}(t)]^T$$

The measurement vector is defined as:

$$Y^{\mathrm{INS}}(t) = [P_x(t), P_y(t), \psi(t)]^T$$

This formulation assumes that the measurements of the accelerometer and the gyroscope are related to the true values using eq. (4.8).

$$\begin{aligned} a_{xb}^{\mathrm{INS}}(t) &= a_{xb}(t) + \mathrm{bias}_{ax}(t) + \mathrm{noise}_{ax}^{\mathrm{INS}}(t) \\ a_{yb}^{\mathrm{INS}}(t) &= a_{yb}(t) + \mathrm{bias}_{ay}(t) + \mathrm{noise}_{ay}^{\mathrm{INS}}(t) \\ w_{zb}^{\mathrm{INS}}(t) &= a_{zb}(t) + \mathrm{bias}_{wz}(t) + \mathrm{noise}_{wz}^{\mathrm{INS}}(t) \end{aligned} \tag{4.8}$$

The matrices R and Q are:

$$R = \begin{pmatrix} \sigma^2_{\mathrm{GPS}_x} & 0 & 0 \\ 0 & \sigma^2_{\mathrm{GPS}_y} & 0 \\ 0 & 0 & \sigma^2_{\mathrm{COMP}} \end{pmatrix}$$

$$Q = \begin{pmatrix} \sigma^2_{\text{Accel}_x} & 0 & 0 \\ 0 & \sigma^2_{\text{Accel}_y} & 0 \\ 0 & 0 & \sigma^2_{\text{Gyro}_z} \end{pmatrix}$$

The Kalman Filter algorithm was implemented using the following three steps:

Step 1: Initialization The user must provide at the mission configuration file the vectors, $X_0^{\text{INS}}, \hat{\text{B}}\text{ias}_0, P_0$ and the covariance matrix for the GPS, Compass and IMU.

$$X_0^{\text{INS}} = [V_x(0), V_y(0), P_x(0), P_y(0), \psi(0)]^T$$
$$\hat{\text{B}}\text{ias}_0 = [\hat{\text{b}}\text{ias}_{ax}(0), \hat{\text{b}}\text{ias}_{ay}(0), \hat{\text{b}}\text{ias}_{wz}(0)]^T$$
$$P_0 = \text{diag}\{\alpha_0 \cdots \alpha_7\}$$

Step 2: Kalman Filter and INS State Update As this step receives GPS and Compass data, it is running at 1 Hz.

$$Y_k^E = X_k^{\text{INS}-}[3:5] - \begin{pmatrix} P_{x,k}^{\text{GPS}} \\ P_{y,k}^{\text{GPS}} \\ \psi_k^{\text{COMP}} \end{pmatrix}$$
$$G_k = P_k^- H^T [H P_k^- H^T + R_k]^{-1}$$
$$P_k^+ = [I_{8x8} - G_k H] P_k^-]$$
$$\hat{X_k^E} = \begin{pmatrix} 0_{5x1} \\ \hat{\text{B}}\text{ias}_k^- \end{pmatrix} + G_k Y_k^E$$
$$X_k^{\text{INS}+} = X_k^{\text{INS}-} - \hat{X}_k^E[1:5]$$
$$\hat{\text{B}}\text{ias}^+ = X_k^E[6:8]$$

where R_k is the covariance matrix of the GPS and the digital compass noise.

Step 3: Kalman Filter and INS State Propagation As this step uses the accelerometer and gyroscope data with the INS model, it is running at 4 Hz. In this step the constant f_{freq} refers to the relative frequency to the GPS. In both implementations $f_{\text{freq}} = 4$ Hz.

$$U_K^{\text{INS}} = \begin{pmatrix} a_{xb,k}^{\text{INS}} \\ a_{yb,k}^{\text{INS}} \\ w_{zb,k}^{\text{INS}} \end{pmatrix} - \hat{\text{Bias}}_k$$

$$\dot{X}^{\text{INS}}(t) = f(X^{\text{INS}}(t), U^{\text{INS}}(t))$$

$$\hat{\text{Bias}}_{k+1}^{-} = \hat{\text{Bias}}_k^{+}$$

$$P_{k+1}^{-} = A_d^E P_k^{+} [A_d^E]^T + \frac{1}{f_{\text{freq}}} B_d Q_K [B_d^E]^T + \beta P_0$$

where Q_k is the covariance matrix of the IMU noise. Equation (4.9) calculates the next INS state by numerically integrating $\dot{X}^{\text{INS}}(t)$.

$$\dot{X}^{\text{INS}}(t) = f(X^{\text{INS}}(t), U^{\text{INS}}(t)) \tag{4.9}$$

The community configuration file can set the variances (GPS, compass, accelerometers and gyroscope variances), the initial P_0 matrix, the parameter β, and if the estimated heading should be used or the compass raw measurements.

Figure 4.1 shows a diagram representing these Kalman Filter steps.

This application is intended to run at a frequency of 4 Hz.

4.5.2 Obtaining the discrete Kalman Filter dynamic matrices

Computing the discrete Kalman Filter dynamic matrices can increase greatly the computational costs. In (SANTOS *et al.*, 2013), the matrices A_d^E and B_d^E were obtained using discretization function provided in the Matlab (MATLAB, 2016) software. As the MOOS-IvP is running in C++, it was not possible to use this discretization function. The continuous dynamic A^E and B^E matrices (SANTOS *et al.*, 2013) are:

$$A^E = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \alpha & -\beta & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta & \alpha & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

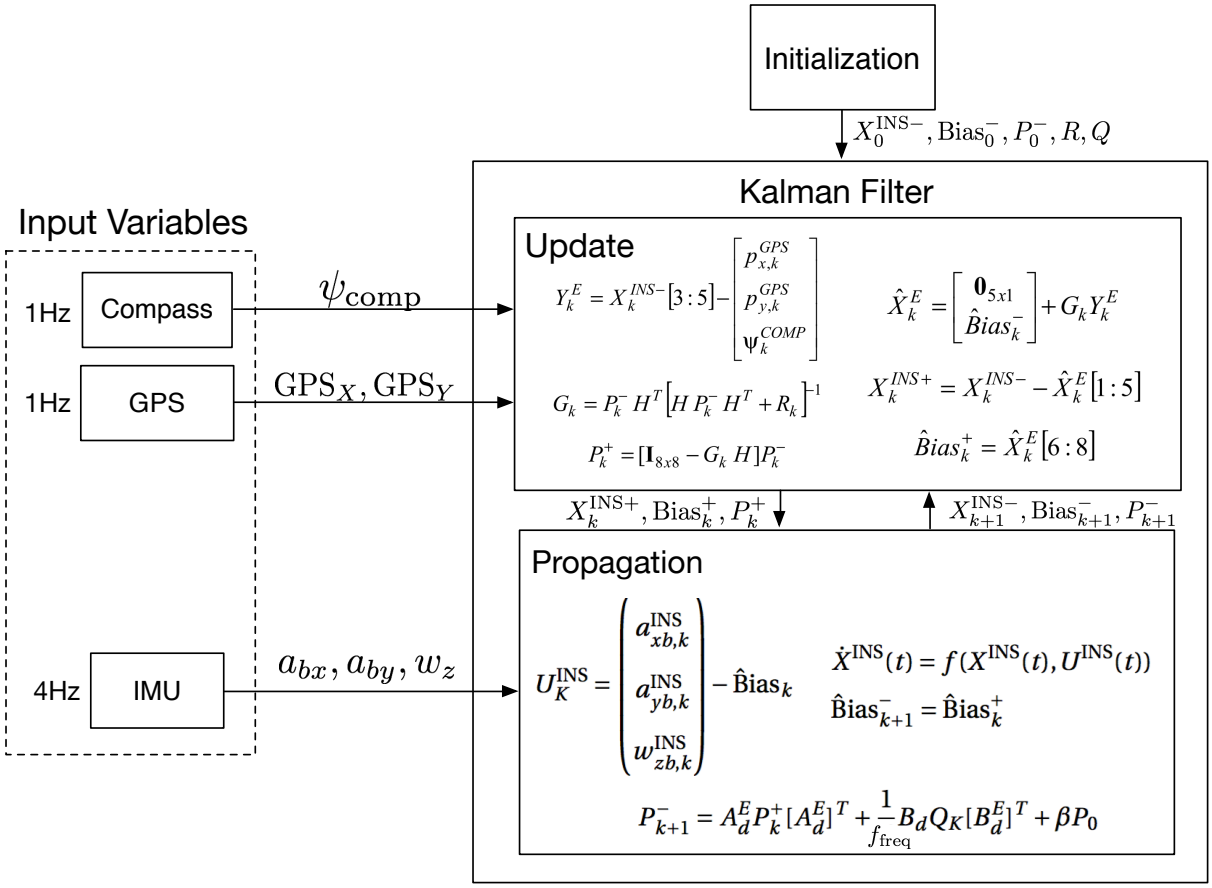

FIGURE 4.1 – Diagram showing the Kalman Filter steps. The input variables in this application come from the MOOSDB.

$$B = \begin{pmatrix} \alpha & -\beta & 0 \\ \beta & \alpha & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$D = 0$$

where:

$$\alpha = cos(x_5^{\text{INS}})$$

$$\beta = sin(x_5^{\text{INS}})$$

The exact matrices A_d^E and B_d^E are obtained through the formulas (FRANKLIN; WORKHAM, 1998):

$$A_d = e^{AT}$$
$$B_d = \int_0^T e^{A\tau} d\tau \cdot B$$

In order to reduce the computational cost, the discrete Kalman Filter dynamic matrices A_d^E and B_d^E are obtained using their exact mathematical expression, first obtained with the symbolic software Wolfram Mathematica 9 (WOLFRAM, 2016). This way it's not necessary to discretize each matrices during the loop, thus obtaining a significant performance in time of the Extended Kalman Filter. The matrices A^E and B^E (with the α and β), were inserted in the exact formulas and the function `Integrate` was used.

The obtained A_d^E and B_d^E are:

$$A_d = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & T\alpha & -T\beta & 0 \\ 0 & 1 & 0 & 0 & 0 & T\beta & T\alpha & 0 \\ T & 0 & 1 & 0 & 0 & \frac{T^2\alpha}{2} & -\frac{1}{2}\left(T^2\beta\right) & 0 \\ 0 & T & 0 & 1 & 0 & \frac{T^2\beta}{2} & \frac{T^2\alpha}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$B_d = \begin{pmatrix} T\alpha & -T\beta & 0 \\ T\beta & T\alpha & 0 \\ \frac{T^2\alpha}{2} & -\frac{1}{2}\left(T^2\beta\right) & 0 \\ \frac{T^2\beta}{2} & \frac{T^2\alpha}{2} & 0 \\ 0 & 0 & T \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

where:

$$\alpha = cos(x_5^{\text{INS}})$$
$$\beta = sin(x_5^{\text{INS}})$$

4.6 The uKalmanVisual Application

This application implements a visual interface to observe the Extended Kalman Filter states. This app helps to tune the Extended Kalman Filter initialization matrices. The tunning process in the Extended Kalman Filter consists of filling the matrices R, Q and P_0 in the initialization step.

The MOOS software uses a custom TCP message protocol to create inter-process communication. Details on this message protocol is not well documented for the user/developer. Since developing graphical applications/plotting in C++ takes a great effort or heavily depends on graphical libraries, a different approach was used in this application.

This application consists of two parts. The first part is the graphical software, responsible for showing the Kalman Filter plots. This program is completely developed in Python 3, and uses the TKinter graphical library. The second part is the MOOS application that creates a communication bridge between the MOOS community and the Python script. The inter-process communication between those two programs is made by using the message library ZeroMQ (ZEROMQ, 2016). ZeroMQ is an asynchronous message library, built in C++ with bindings for several other languages. This message library simplifies inter-process communication and allows graphical developers to be independent from MOOS developers. Figure 4.2 shows the block diagram for this application:

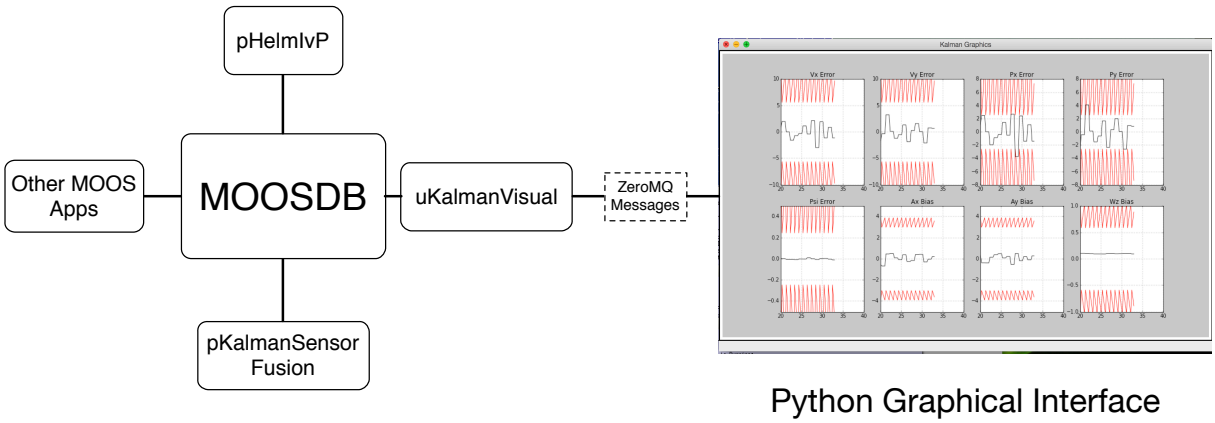

FIGURE 4.2 – The blocks diagram for the `uKalmanVisual` application. In this application the MOOS application acts as a bridge for the Python graphical interface.

The Python application runs on a different thread, called by the MOOS part. It uses the `matplotlib` for plotting and TKinter as the graphical interface. Communication between the two applications is done by a serial data stream using ZeroMQ. Figure 4.3 shows an example of these plots using the `uKalmanVisual` application.

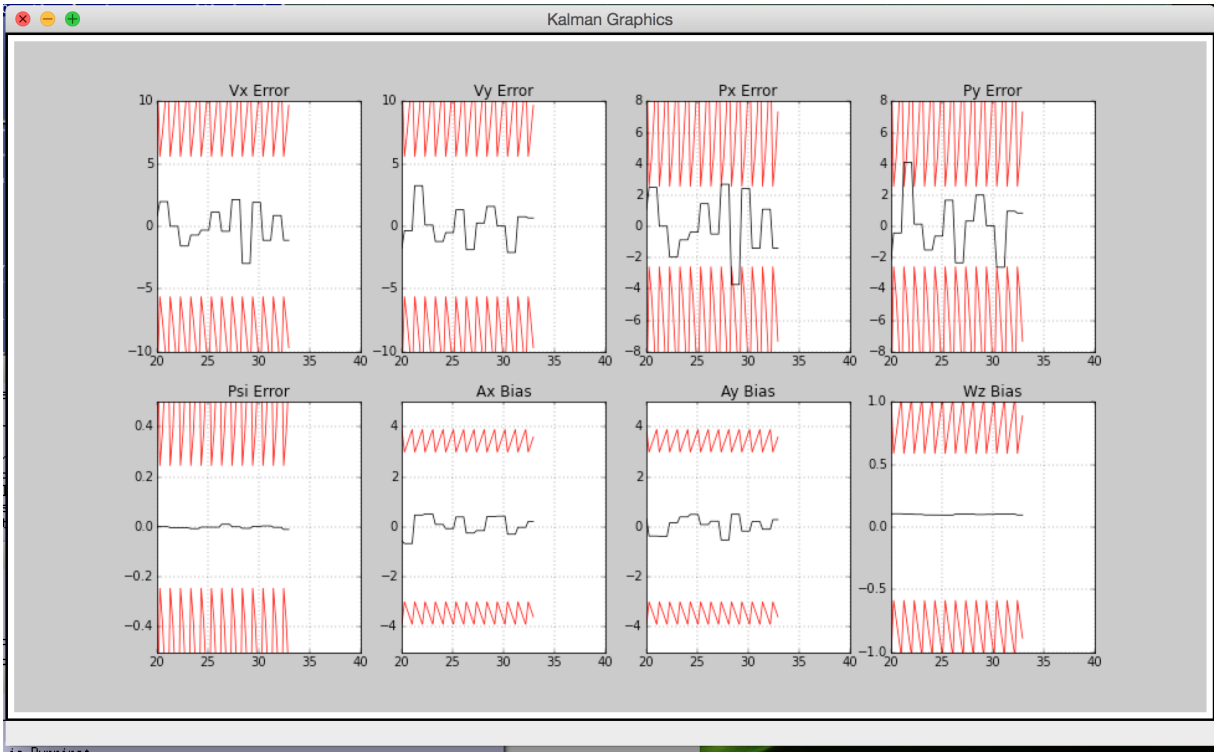

FIGURE 4.3 – A screen capture of the `uKalmanVisual` application. The red lines represent the $\pm 3\sigma$ plots and the black lines represent the error of each variable. If the error stays most of the time in the $\pm 3\sigma$ region for every value it can be said that filter converges.

4.7 The uBoatSimulator Application

This application implements the boat dynamic equations, Equation 3.1. To solve the Ordinary Differential Equations (ODE) it uses the `boost::numeric::odeint` library. For simulation purposes it was used the fourth order Runge Kutta integration method. Using this model with a simpler integration method such as the Euler method, might diverge depending of the integration step.

This application also simulates the GPS and the IMU errors. In the configuration file is possible to include the variance for the GPS, accelerometers, digital compass and gyroscope. It is also possible to introduce a constant bias for the accelerometers and for the gyroscope. In this way,imat it is possible to simulate the Kalman Filter and roughly tune it, before deploying.

This model helps to simulate the dynamics of the boat and to test new MOOS applications and missions before deployment.

5 Results

This chapter presents the simulation and experimental test results for the developed ASV. It is organized as follows.

- Section 5.1 presents the missions performed by the ASV.
- Section 5.2 presents the method used to tune the Extended Kalman Filter.
- Section 5.3 brings simulation results for the modeled ASV.
- Section 5.4 brings the experimental results for the first ASV approach. This experimental results uses the same mission file as the simulation results.
- Section 5.5 brings the experimental results for the second (modular) ASV approach. This experimental results uses a different mission file compared to the simulation results and the first implementation results.

5.1 The missions

The ASV is intended to navigate in river and dam waters in the presence of winds and light water currents. The waterways might have presence of restricted areas, islands and other obstacles. Although, the MOOS-IvP software allows the use of the ASV in the presence of other vehicles, with vehicle collision avoidance, this feature is not explored in this work. The ASV is not expected to perform high precision navigation tasks, but rather be used as a basis for a vehicle running missions under different conditions, such as surveillance or environmental sampling. In the second approach, due to the modular configuration, higher precision sensors can be added to execute specialized tasks.

From the set of behaviors supplied with the MOOS-IvP software three behaviors were used:

- **Waypoints**: Given a set of waypoints the boat has to follow these waypoints in the given order. It is possible to define a capture radius, slip radius and change the waypoints dynamically.

- **AvoidObstacle**: With this behavior it is possible to set a region to avoid, and a safety area for a virtual collision distance. This behavior was used in a way to define previously known virtual obstacles during the waypoint task.

- **StationKeep**: With this behavior the vehicle is kept in the determined area until another condition is met.

Capture radius is a waypoint parameter that defines the radius of the waypoint acceptable area. Slip radius is a waypoint parameter that is used in complex missions with conflicting objectives. For more information refer to the official documentation (MOOS-IVP, 2016).

The obstacle radius defines the size of the obstacle. The obstacle safety area is the area where the weight of the obstacle is increased in the decision space. By increasing this weight, the vehicle gives high priority to escape this area rather than accomplishing other objectives.

The mission performed in the simulation 1 and in the first implementation are the same. The mission performed in simulation 2 and in the second implementation are also the same. The ASV is set to follow the list of waypoints, in the presented order, while avoiding the static obstacles. Although there are two different missions, the mission structure in both cases remains the same. Table 5.1 presents a summary of the missions waypoints and obstacles performed by the ASV. Table 5.2 presents the other parameters of the behaviors, such as capture radius, slip radius and safety area.

TABLE 5.1 – Mission Summary

Mission	Waypoints (m)	Virtual obstacles (m)	Station keep (m)
Simulation 1	$(40,20),(10,30),(30,60),$ $(-20,40),(-5,5)$	$(24,30),(10,10),$ $(10,55),(-15,15)$	$(-5,5)$
Simulation 2	$(15,15),(-15,30),(15,60),$ $(35,40),(50,20),(10,-5)$	$(0,15),(0,30),$ $(0,45),(20,50),(30,8)$	$(10,-5)$
First approach	$(40,20),(10,30),(30,60),$ $(-20,40),(-5,5)$	$(24,30),(10,10),$ $(10,55),(-15,15)$	$(-5,5)$
Second approach	$(15,15),(-15,30),(15,60),$ $(35,40),(50,20),(10,-5)$	$(0,15),(0,30),$ $(0,45),(20,50),(30,8)$	$(10,-5)$

5.2 Tuning the Kalman Filter

One of the challenging tasks in the design of the Kalman Filter was tuning it. The tuning of the Kalman Filter refers to adjusting the matrices P_0, R and Q., This section shows how the Kalman Filter was tuned for both the simulation and experimental results.

TABLE 5.2 – Behavior parameters

Mission	Capture radius (m)	Obstacles radius (m)	Safety area radius (m)
Simulation 1	3	3	7
Simulation 2	3	2	4
First approach	3	3	7
Second approach	3	2	4

The covariances matrices of the Kalman Filter (R and Q) were filled with the covariance of the measurements from the GPS, IMU and the Compass with the vehicle in a stationary position. The matrices R and Q were filled as shown in chapter 4. Table 5.3 shows the calculated covariances.

TABLE 5.3 – Covariance values for the GPS, compass and IMU measurement noise

Variable	Covariance	Noise Standard deviation
GPS_x	0.9 m^2	0.95 m
GPS_y	0.9 m^2	0.95 m
a_{bx}	0.1 m^2/s^4	0.32 m/s^2
a_{by}	0.1 m^2/s^4	0.32 m/s^2
w_z	0.2 rad^2/s^2	25.6 °/s
ψ	0.01 rad^2	5.7 °

The P_0 matrix has eight parameters to tune. This becomes difficult to tune directly in the field. It is easier to pre-tune the Kalman Filter in simulation environment and then fine tune the Kalman Filter in field.

The convergence of the Kalman Filter can be verified using the application `uKalmanVisual`. As seen in Chapter 4 this application plots in real-time the errors for each variable and three times the standard deviation. The red lines represent the $\pm 3\sigma$ plots and the black lines represent the error of each variable. Generally speaking, if the error stays most of the time in the $\pm 3\sigma$ region for every value it can be said that the Kalman Filter converges.

Using the parameter $\beta = 0.01$ and the `uKalmanVisual` it is possible to see which variables are converging and tune them accordingly. After several iterations in simulation it was obtained pre-tune parameters for the P_0 matrix.

Using the pre-tuned parameters as a starting point the vehicles was fine tuned statically. It was obtained the following parameters for matrix P_0:

$$P_0 = \text{diag}\{10, 10, 12, 12, 0.01, 10, 10, 5\} \tag{5.1}$$

This parameters were reintroduced in the simulator to see if the Kalman Filter converges for the whole simulated mission and to test how the vehicle should behave.

5.3 Simulation Results

To simulate the behavior of the ASV the configuration file must be changed. Although most MOOS applications are used the same way, it is necessary to replace the iSerial and pGeodesy applications to the uBoatSimulator application. This last application is capable of simulating the GPS errors and the IMU errors and bias. The mission file (behavior file) remains unchanged between experimental and simulation results.

The mission used for this simulation results consists of following several waypoints while avoiding virtual obstacles. After concluding the mission successfully, the vehicle is set to station-keep at the last waypoint. This condition is set until another mission is given.

Figure 5.1 shows the mission running in the `pMarineViewer` application. In this figure it is possible to see how is the simulation environment. One can see the waypoints, some vehicle variables. With this visual interface is possible to see how the vehicle behaves in a real-time simulation and test the mission before deployment.

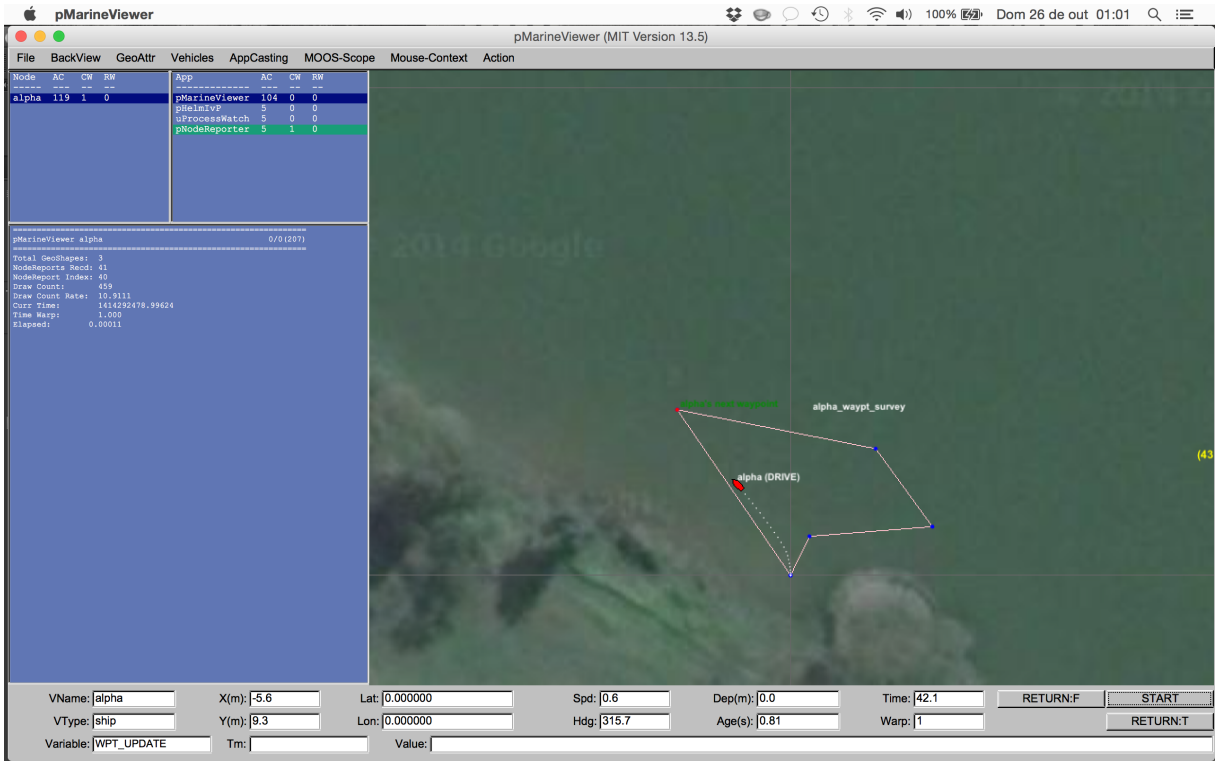

FIGURE 5.1 – A screen capture of the `pMarineViewer` application. This application allows the user to see visually how the vehicle performs.

5.3.1 Simulation 1

Figure 5.2 shows the obtained simulation results for the first simulated mission. The simulated vehicle took 540 seconds to complete the whole mission. The simulated mission does

not include possible water currents and wind conditions. In this graph, the ASV performed the whole mission, achieving all the waypoints and avoiding the safety area of the obstacles. It can be seen that the vehicle still tries to maintain a trajectory that minimizes traveled distance.

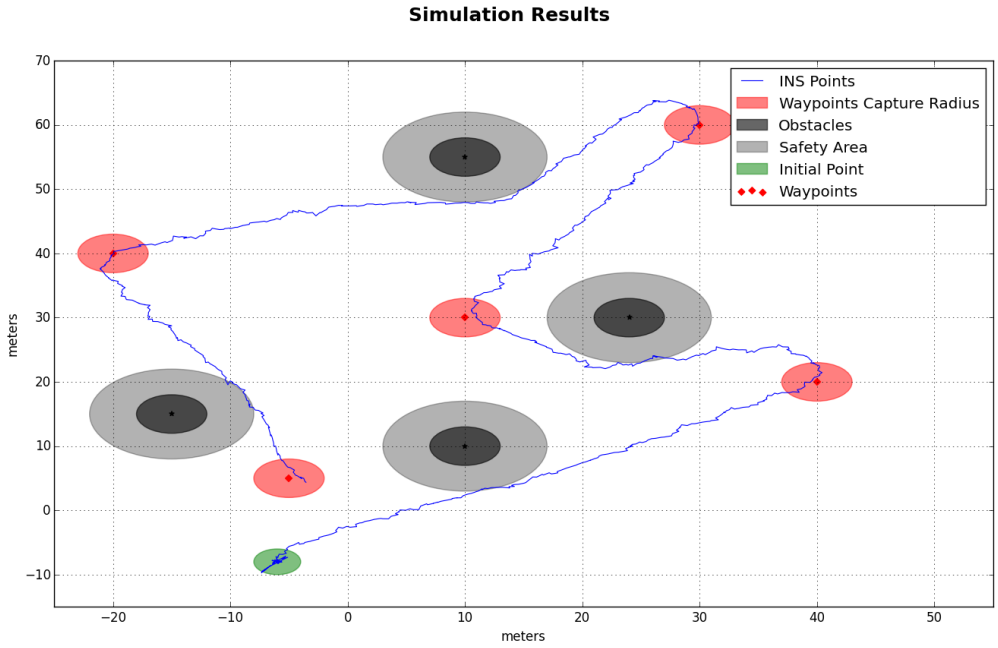

FIGURE 5.2 – Simulation results obtained with the uBoatSimulator application for the first mission.

Figure 5.3 shows the error plots of the Kalman Filter in the simulation environment for the first mission. The graph represents the values of each variable of the error state:

$$X^E(t) = [V_x(t), V_y(t), P_x(t), P_y(t), \psi(t), \hat{\text{bias}}_{ax}(t), \hat{\text{bias}}_{ay}(t), \hat{\text{bias}}_{wz}(t)]$$

In the simulation environment spikes and errors that do not fit the gaussian distribution were not simulated. The last three components of the error state vector should converge to the bias of the accelerometers and of the gyroscope.

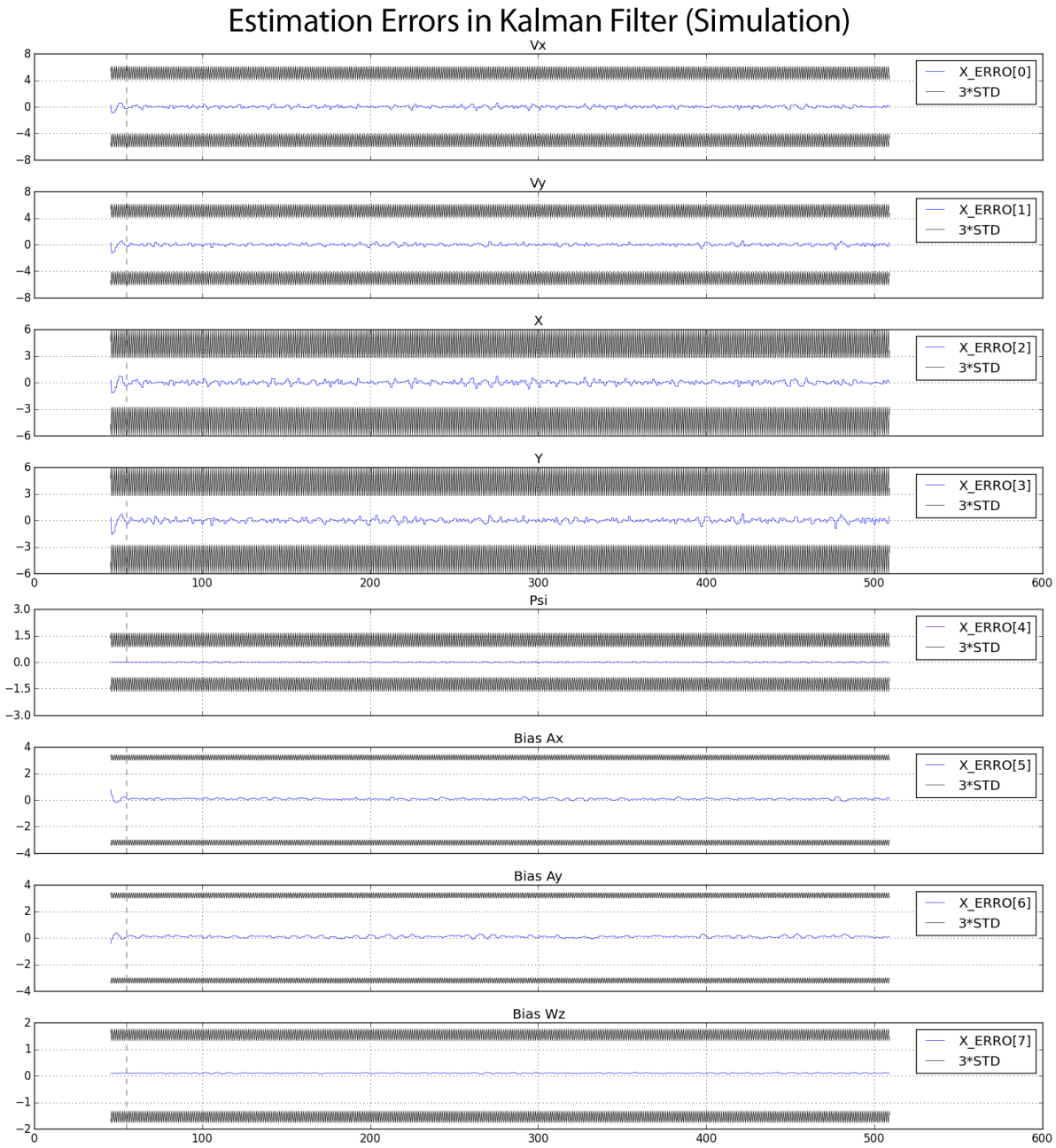

FIGURE 5.3 – The Kalman Filter errors in the simulation environment for the first mission. The blue line represents the estimated error of the Kalman Filter. The black line represents the standard deviation ($\pm 3\sigma$). This figure presents the same information that can be seen in real-time using the uKalmanVisual application.

5.3.2 Simulation 2

Figure 5.4 shows the obtained simulation results for the second simulated mission. The simulated vehicle took 473 seconds to complete the whole mission. The simulated mission does not include possible water currents and wind conditions. In this graph, the ASV performed the whole mission, achieving all the waypoints and avoiding the safety area of the obstacles.

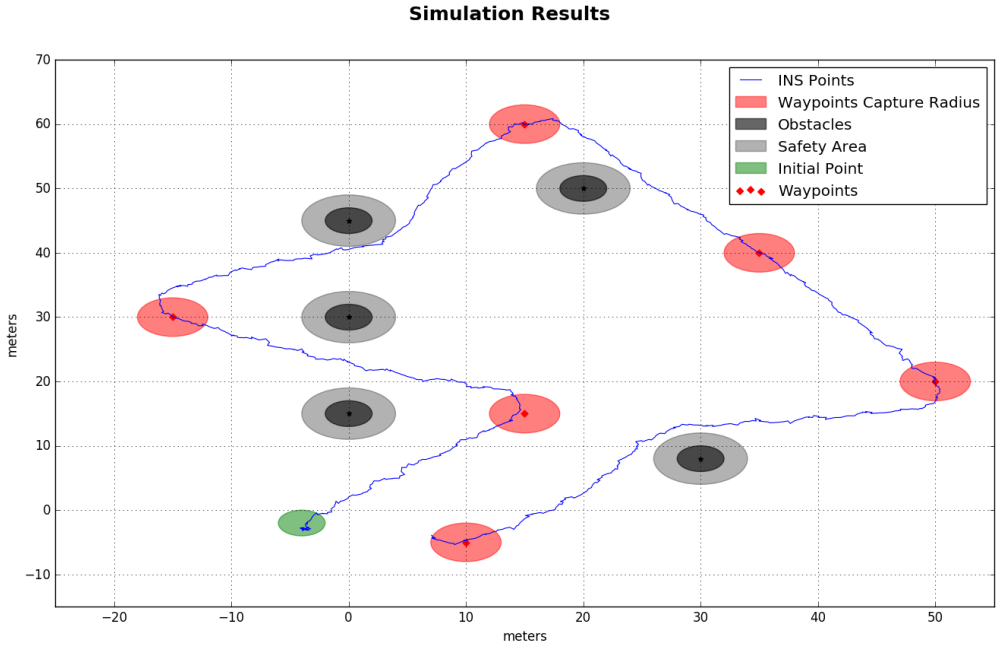

FIGURE 5.4 – Simulation results obtained with the uBoatSimulator application for the second mission.

As in the simulation, it was not simulated non gaussian errors, the Kalman Filter errors plot is quite similar to the first mission.

5.4 Experimental Results for the first implementation

Using the same mission configuration as in the simulation, the ASV was deployed in the Aeronautics Institute of Technology (ITA) lake in Brazil. Although there were no water currents or waves, the boat was submitted to wind conditions.

The vehicle took 700 seconds to complete the whole mission. This happens due to external conditions (such as winds), and the model of the ASV may not correspond exactly to the real ASV. It is worth noting that although the Extended Kalman filter provides an estimation of the velocity error, the ASV does not measure the boat velocity. Figure 5.5 shows the obtained experimental results. Figure 5.9 compares the simulation results for the first mission

with the experimental results of the first implementation.

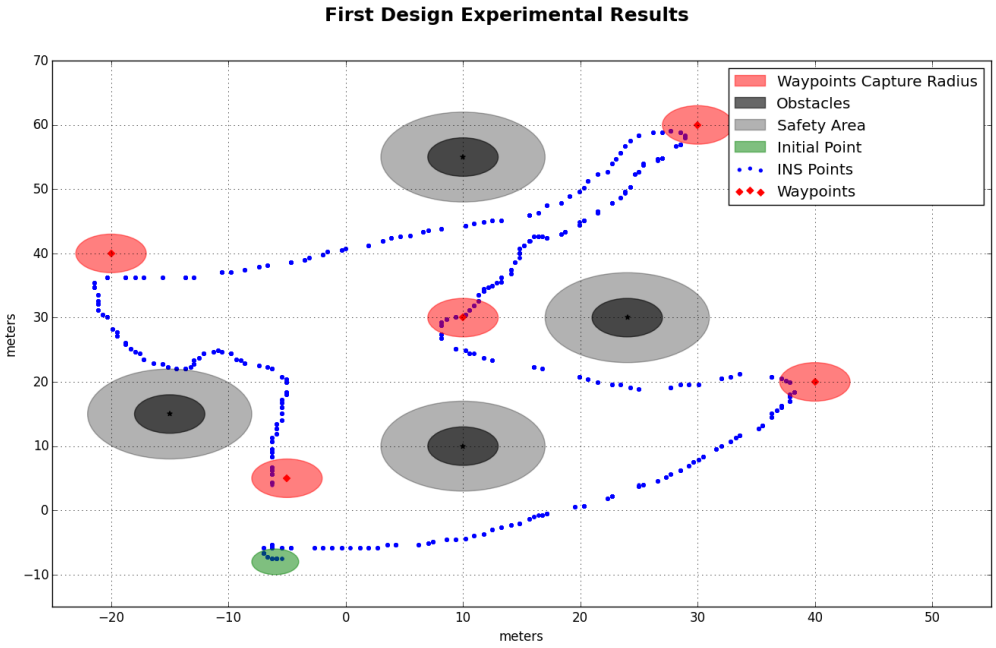

FIGURE 5.5 – Experimental results using the boat in a lake.

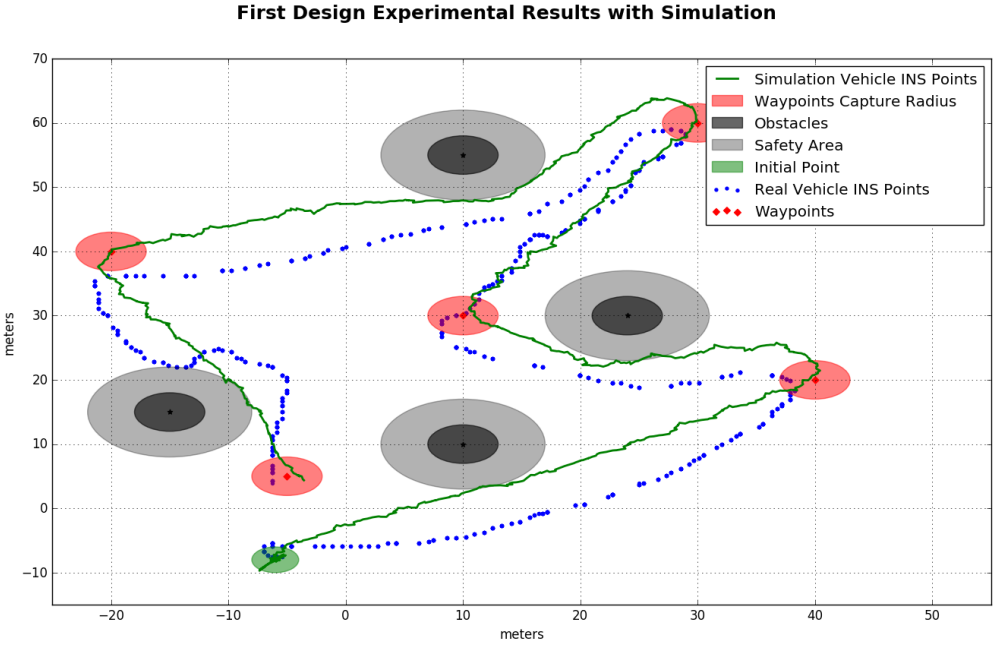

FIGURE 5.6 – Comparison between the experimental results for the first implementation with the simulation results.

As it can be seen from this experimental result, the Extended Kalman Filter worked well even using a low-cost IMU. The comparison between the simulation results and the experimental results shows can be used to verify the mathematical model of the boat. However,

each simulated result and each experimental result is a different realization, because the errors change in each step of the simulation.

One of the advantages of this inertial navigation system is to provide data between GPS measurements. Therefore the controller can run at 4 Hz also, instead of 1 Hz without the Kalman Filter.

Figure 5.7 shows the error plots of the Kalman Filter in the first implementation experimental results. The graph represents the values of each variable of the error state:

$$X^E(t) = [V_x(t), V_y(t), P_x(t), P_y(t), \psi(t), \hat{\text{b}}\text{ias}_{ax}(t), \hat{\text{b}}\text{ias}_{ay}(t), \hat{\text{b}}\text{ias}_{wz}(t)]$$

As it can be seen, the errors related to the heading (w_z and ψ) sometimes grow up. One of the ideas behind the Kalman Filter approach is based on assumptions that measurements have a Gaussian distribution, described in the measurement model. Whenever measurements are generated by a different model, outliers appears. This outliers generates large errors inside the Kalman Filter. However, if those spikes are rare the Kalman Filter can recover quickly. If the spikes start to appear regularly, an outlier-rejection scheme is necessary for a robust Kalman Filter application (AGAMENNONI *et al.*, 2011; BERMAN, 2014). Outlier-rejection was not used in this work.

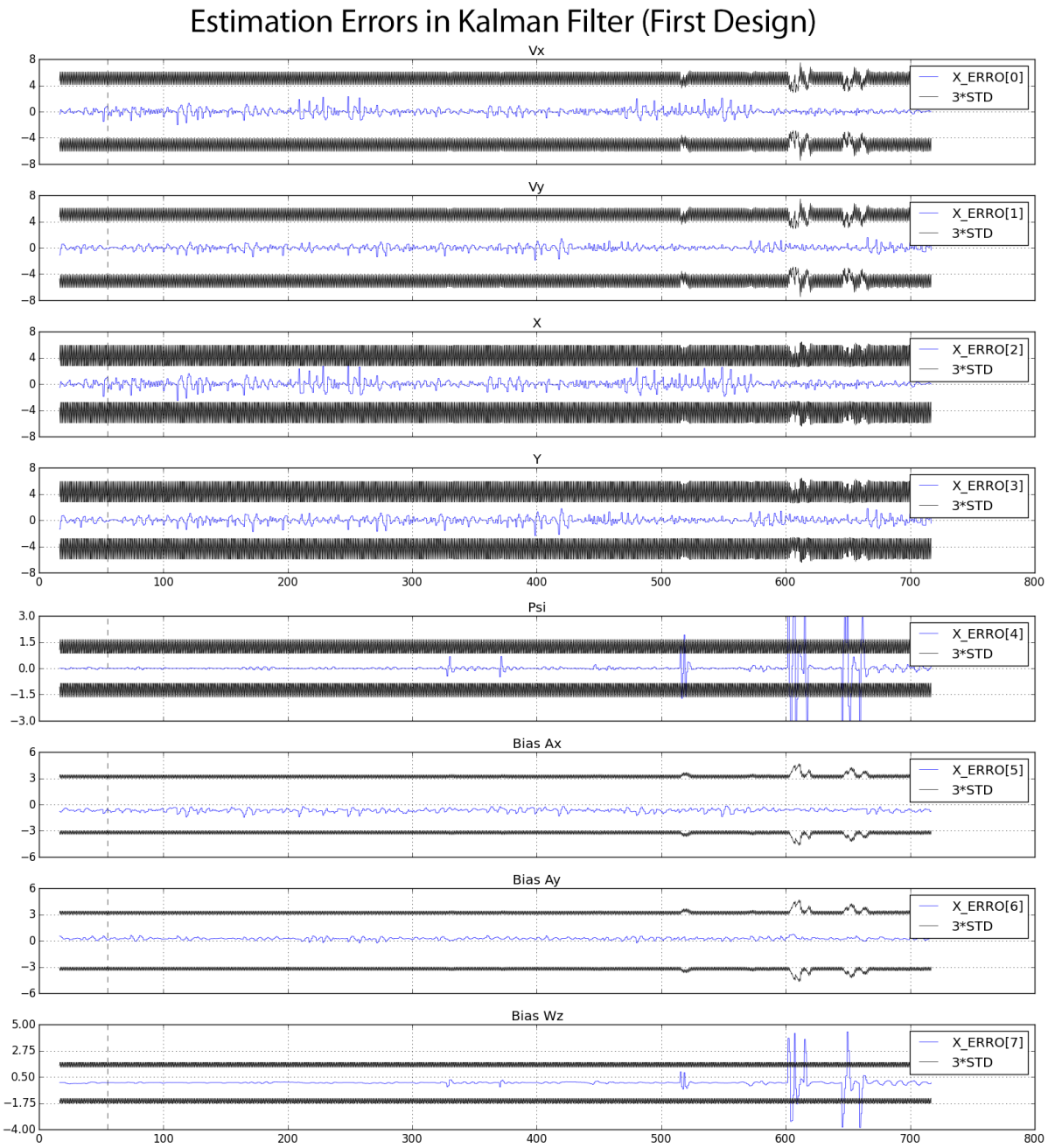

FIGURE 5.7 – The Kalman Filter errors for the first implementation. The blue line represents the estimated error of the Kalman Filter. The black line represents the standard deviation ($\pm 3\sigma$). This figure presents the same information that can be seen in real-time using the uKalmanVisual application.

5.5 Experimental Results for the second (modular) implementation

For this experimental results, the ASV was also deployed in the Aeronautics Institute of Technology (ITA) lake in Brazil. The ASV was submitted to wind conditions and rain.

The vehicle was set to perform a different mission from the previous experimental results. The trajectory consists of different waypoints and virtual obstacles. After concluding the mission successfully, the vehicle is set to station-keep at the last waypoint. This condition is kept until another mission is given.

The vehicle took 740 seconds to complete the whole mission. During the execution of the mission, communication between the vehicle and the GCS was turned off, simulating a loss of signal. The vehicle kept the mission running and later communication was reestablished. Figure 5.8 shows the obtained experimental results.

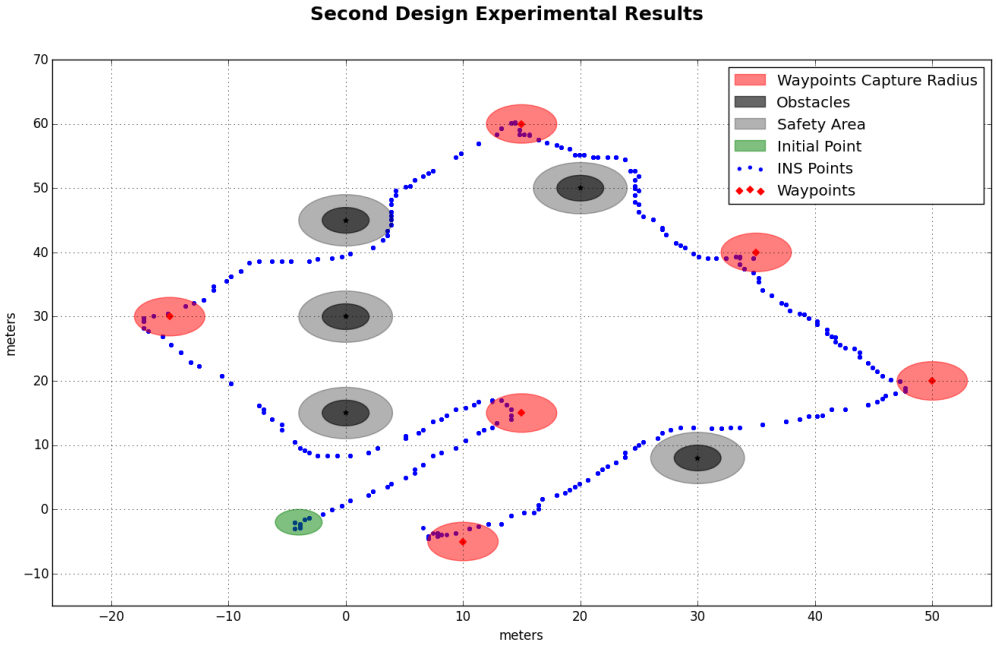

FIGURE 5.8 – Experimental results using the ASV on a lake. The vehicle is set to perform a mission consisting of several waypoints with virtual obstacles.

As it can be seen from this experimental result, the Extended Kalman Filter worked well even using a low-cost IMU. In this comparison between the simulated results and the experimental results are very different in the trajectory near the first obstacle. This occured because the decision-making happens in real-time depending on the vehicle localization and orientation. Therefore, depending on the current state of the vehicle a different decision can be made. In this case, the decision of the experimental realization was different from the simulation realization.

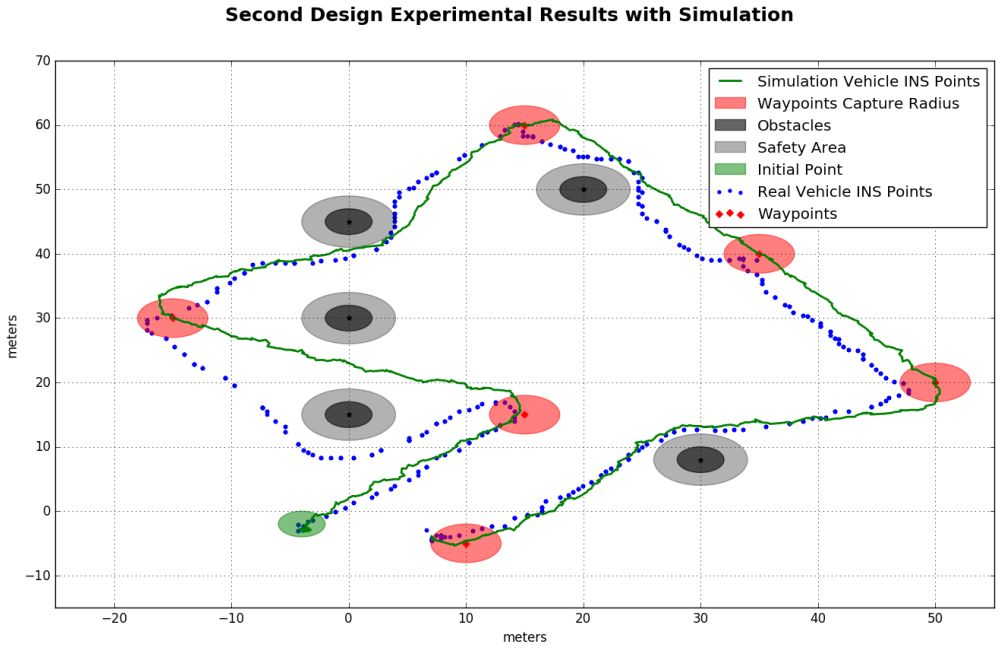

FIGURE 5.9 – Comparison between the experimental results for the second implementation with the simulation 2 results.

Figure 5.10 shows the error plots of the Kalman Filter in the modular approach experimental results.

During this experiment, a real-time video was streamed from the ASV to the GCS using the Motion software. Figure 5.11 shows a single frame of the video stream. The video lag is often less than 1 second using 640x480 video resolution with a framerate of 15 fps.

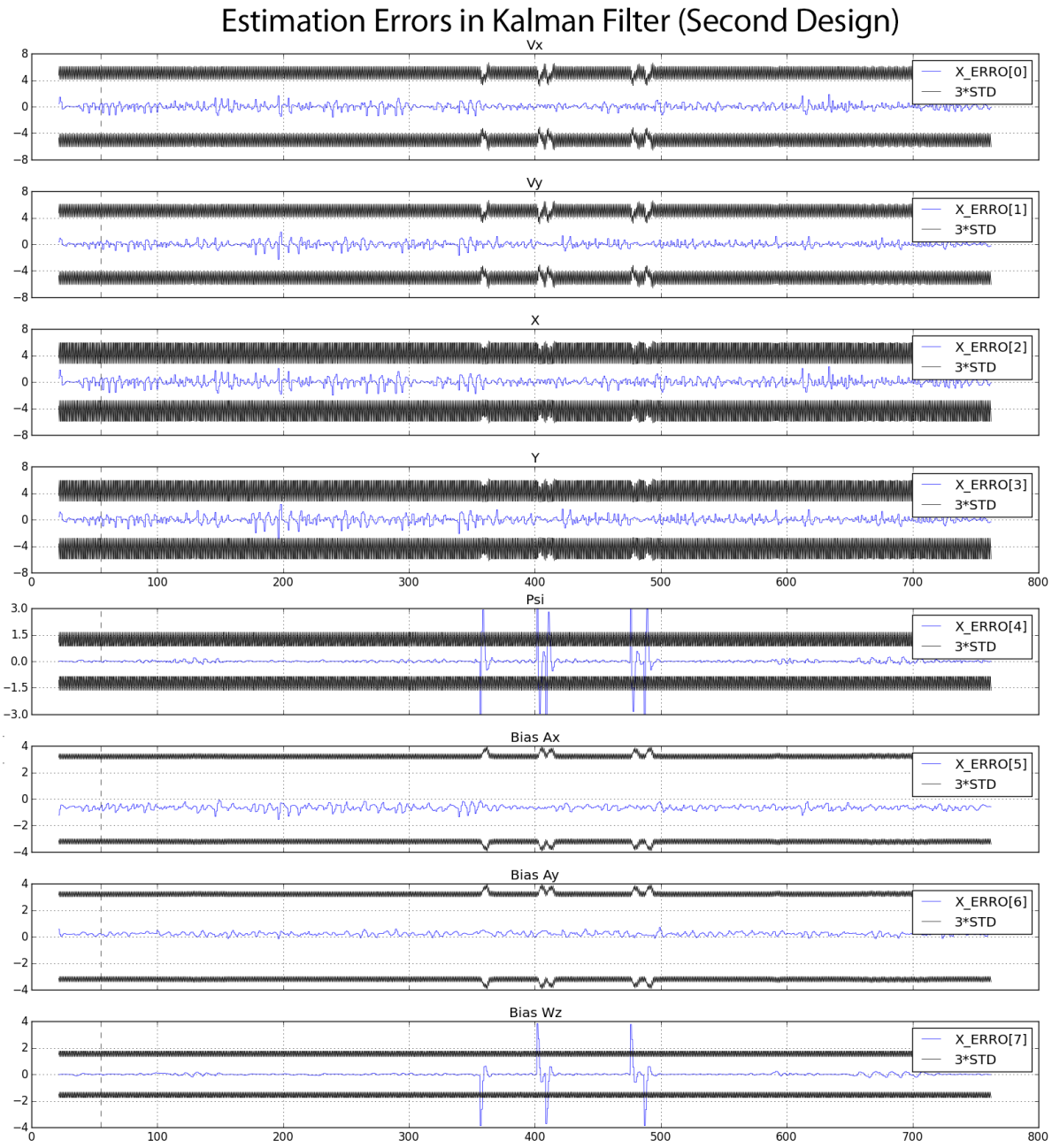

FIGURE 5.10 – The Kalman Filter errors for the second approach. The blue line represents the estimated error of the Kalman Filter. The black line represents the standard deviation ($\pm 3\sigma$). This figure presents the same information that can be seen in real-time using the uKalmanVisual application.

FIGURE 5.11 – A frame from the video stream using the Motion software. This shows an example of a surveillance ASV.

6 Final Remarks

6.1 Conclusion

This dissertation shows the simulation and also the development of an ASV using low-cost sensors. Two approaches were developed and tested in real world experiments.

The first approach took as a starting point the developed ASV in (SANTOS *et al.*, 2013). The embedded hardware was redesigned in this approach (several components were changed compared with the ASV developed by Santos). Several MOOS modules were developed to interface the ASV with the MOOS-IvP software. However, this solution is not scalable for several collaborative vehicles. The use of a single GCS computer to perform the decision-making and navigation algorithms can be computationally intensive and may not support several vehicles. Additionally, configuring more than one computer in the GCS, with each one controlling several vehicles, it may result in a setup difficult to manage. This solution is also dependent on a reliable communication link with the GCS. Without the communication link the vehicle cannot navigate and can result in a lost vehicle.

A second implementation was made in order to address the issues in the first approach, and in order to reduce the vehicle costs. This approach creates a modular ASV and follows closely the backseat driver paradigm. The redesigned ASV provides a scalable way to deploy missions with several vehicles regardless of the communication link with the GCS. The modular approach benefits collaborative missions as several vehicles can be deployed with different payload computers and sensors without redesigning each vehicle. As an example, a video camera connected to the USB bus gives the ASV surveillance capabilities.

Several MOOS-IvP modules were developed and shown in detail in Chapter 4. These modules gives the ASV manual control, improved localization with sensor fusion, communication between the backseat computer and the frontseat computer, conversion from geodesy frame to local frame, motor controller and a simulation model of the ASV.

Simulation and experimental results show the ASV working as desired. Simulation results are valuable to test and validate missions before deployment. Experimental results were presented to show that the system operated as designed in both approaches. The use of low-cost

sensors with the sensor fusion algorithms provides the necessary accuracy for navigation in surveillance missions, avoiding collisions with predetermined obstacles. In the second approach, a new sensor (a camera module) was added, making it possible to transmit a camera feed to the GCS.

The developed ASV provides a scalable way to deploy missions with several vehicles regardless of the communication link with the GCS. The developed modular approach as a basis vehicle benefits collaborative missions. Several vehicles can be deployed with different payload computers and sensors without redesigning each vehicle hardware and modifying the sensors interfaces.

6.2 Future Work

A possible line of research is implementing the MOOS-IvP software in different vehicles such as small scales of cars and AUVs. This would increase the development of custom MOOS Applications specific for different vehicles.

Other lines of research can investigate the application of other payload sensors such as water quality sensors, DVL (Doppler Velocity Logger) sensors, SONAR modules, LIDAR and the use of vision-aided navigation for areas were GPS signals are limited. The use of camera aided navigation can be useful in areas that demands high precision navigation and have visual landmarks, such as automated docking.

Another line of research is to investigate the use of a GPS receiver with a higher output data frequency, such as 10Hz, and with a real-time kinematic correction (GPS-RTK).

Real-Time Kinematic is a global position technique that relies on a precise reference station to provide real-time corrections. The GPS needs to be in the reference station area. The reference station measures the phase of the signal's carrier wave and provides over the internet real-time corrections. With these corrections it is possible to obtain centimeter-level accuracy. This kind of technique would work well in Brazil due to the number of reference stations available. Brazil has 95 (as in January 2016) reference stations in a network that is called RBMC-IP (*Rede Brasileira de Monitoramento Contínuo de Sistemas GNSS em tempo real*) (RBMC, 2016). This service provides GPS corrections for Differential GPS and for GPS-RTK using a RTMC network protocol. This service is provided by the Brazilian Government without costs to the user.

This kind of solution would provide better accuracy for missions that require a precise localization. However, improving accuracy demands more costly sensors. A low-cost solution using open source software is presented in (TAKASU; YASUDA, 2009).

This research has not investigated the problem of tolerance to failures. Autonomous sys-

tems need to consider possible failures, failure recovering and safety issues to be developed as a commercial system. Therefore, further analysis of tolerance to failures is an important topic in researches in autonomous systems.

Bibliography

AGAMENNONI, G.; NIETO, J. I.; NEBOT, E. M. An Outlier-Robust Kalman Filter. no 2006, p. 1551–1558, 2011.

ANDERSON, A.; HOWE, T.; RYPKEMA, e. a. Team MIT-Olin. **RobotX Journal Paper**, no October, 2014.

ANTAQ. 2007. Available from Internet: <http://www.antaq.gov.br/portal/>. Cited: Jan 26. 2016.

ARDUINIANA. 2016. Available from Internet: <http://arduiniana.org/libraries/tinygps/>. Cited: Jan 26. 2016.

ASVGLOBAL. 2016. Available from Internet: <http://asvglobal.com/products/commercial/>. Cited: Jan 26. 2016.

AUTOMARINESYS. 2016. Available from Internet: <http://www.automarinesys.com/datamaran/>. Cited: Jan 26. 2016.

BENJAMIN, M. **Interval programming: a multi-objective optimization model for autonomous vehicle control**. Thesis (Doctorate), 2002.

BENJAMIN, M.; SCHMIDT, H.; NEWMAN, P.; LEONARD, J. An Overview of MOOS-IvP and a Users Guide to the IvP Helm - Release 13.5. 2013.

BERMAN, Z. Outliers rejection in Kalman filtering - Some new observations. In: **Record - IEEE PLANS, Position Location and Navigation Symposium**. [S.l.: s.n.], 2014. p. 1008–1013. ISBN 9781479933204.

BLUEFIN. 2016. Available from Internet: <http://www.bluefinrobotics.com/>. Cited: Jan 26. 2016.

BOOST. 2016. Available from Internet: <https://www.boost.org//>. Cited: Jan 26. 2016.

BREGE, E. D. **Design and Construction of a Low Cost, Modular Autonomous Underwater Vehicle**. Thesis (Doctorate) — Massachussets Institute of Technology, 2011.

BRODSKIY, Y. **Robust autonomy for interactive robots**. [S.l.: s.n.], 2014. ISBN 9789036536202.

CLEARPATH. 2016. Available from Internet: <http://www.clearpathrobotics.com/kingfisher-bathymetry-unmanned-surface-vessel/>. Cited: Jan 26. 2016.

DJAPIC, V.; NAD, D. Using collaborative autonomous vehicles in mine countermeasures. **OCEANS'10 IEEE Sydney, OCEANSSYD 2010**, 2010.

EIGEN. 2016. Available from Internet: <https://www.eigen.tuxfamily.org//>. Cited: Jan 26. 2016.

ERSP. 2016. Available from Internet: <http://www.engino.com/robotics.html>. Cited: Jan 26. 2016.

FARRELL, J.; BARTH, M. **The Global Positioning System and Inertial Navigation**. [S.l.: s.n.], 1999.

FILIPEFLOP. 2016. Available from Internet: <http://www.filipeflop.com/pd-123084-sensor-gy-80-10-dof-acelerometro-giroscopio-magnetometro-barometro.html>. Cited: Fev 20. 2016.

FRANKLIN, J. D. P. G. F.; WORKHAM, M. L. **Digital Control of Dynamic Systems**. [S.l.]: Addison-Wesley, 1998.

KANG, M.; et al, S. K. Team KAIST. **RobotX Journal Paper**, p. 1–14, 2014.

KEMNA, S.; HAMILTON, M. J.; HUGHES, D. T.; LEPAGE, K. D. Adaptive autonomous underwater vehicles for littoral surveillance: The GLINT10 field trial results. **Intelligent Service Robotics**, v. 4, no 4, p. 245–258, 2011. ISSN 1861-2776.

MADDEN, C. An Evaluation of Potential Operating Systems for Autonomous Underwater Vehicles. 2013.

MARGOLIS, M. **Arduino cookbook**. [S.l.]: O'Reilly Media, 2011. ISBN 0636920022244.

MATLAB. 2016. Available from Internet: <http://www.mathworks.com/products/matlab>. Cited: Jan 26. 2016.

MATTOS, D. I. **Implementaco do software MOOS-IvP em um barco Autonomo**. 2014.

MOOS. 2016. Available from Internet: <http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/HomePage>. Cited: Jan 26. 2016.

MOOS-IVP. 2016. Available from Internet: <http://www.moos-ivp.org/>. Cited: Jan 26. 2016.

MOTION. 2016. Available from Internet: <http://www.lavrsen.dk/foswiki/bin/view/Motion/WebHome>. Cited: Jan 26. 2016.

NEWMAN, P. A MOOS-V10 Tutorial. p. 1–29, 2013.

OPENCV. 2016. Available from Internet: <http://opencv.org/>. Cited: Jan 26. 2016.

PLAYER. 2016. Available from Internet: <http://playerstage.sourceforge.net/>. Cited: Jan 26. 2016.

PYTHON. 2016. Available from Internet: <http://www.python.org>. Cited: Jan 26. 2016.

RBMC. 2016. Available from Internet:
<http://www.ibge.gov.br/home/geociencias/geodesia/rbmc/ntrip/>. Cited: Fev 20. 2016.

RDS. 2016. Available from Internet:
<https://msdn.microsoft.com/en-us/library/bb483024.aspx>. Cited: Jan 26. 2016.

ROS. 2016. Available from Internet: <http://www.ros.org//>. Cited: Jan 26. 2016.

S. Grewal, M.; R. Weill, L.; P. Andrews, A. **Global positioning systems, inertial systems, and integration**. [S.l.: s.n.], 2007.

SANGEKAR, M.; CHITRE, M.; KOAY, T. B. Hardware architecture for a modular autonomous underwater vehicle STARFISH. **Oceans 2008**, p. 1–8, 2008.

SANTOS, D. S. **Projeto e Construção de um Barco Inteligente com Integração INS/GPS e Bússola**. Thesis (Doctorate) — Instituto Tecnológico de Aeronáutica, 2011.

SANTOS, D. S.; NASCIMENTO Jr., C. L.; CUNHA, W. C. Autonomous navigation of a small boat using IMU/GPS/digital compass integration. **SysCon 2013 - 7th Annual IEEE International Systems Conference, Proceedings**, p. 468–474, 2013.

SDL2. 2016. Available from Internet: <https://www.libsdl.org//>. Cited: Jan 26. 2016.

SETO, M. **Marine Robot Autonomy**. New York, NY: Springer New York, 2012. ISBN 978-1-4614-5658-2.

SIDELEAU, S. R.; EICKSEDT, D. P. The backseat control architecture for autonomous robotic vehicles: a case study with the Iver2 AUV. **Marine technology society journal**, v. 44, no 4, p. 42–54, 2010.

TAKASU, T.; YASUDA, A. Development of the low-cost RTK-GPS receiver with an open source program package RTKLIB. **International Symposium on GPS/GNSS**, 2009.

URBI. 2016. Available from Internet: <http://www.gostai.com/products/urbi/>. Cited: Jan 26. 2016.

WOLFRAM. 2016. Available from Internet: <https://www.wolfram.com/mathematica/>. Cited: Jan 26. 2016.

ZEROMQ. 2016. Available from Internet: <http://www.zeromq.org>. Cited: Jan 26. 2016.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
DM	22 de março de 2016	DCTA/ITA/DM-010/2016	78

5. TÍTULO E SUBTÍTULO:

Autonomy Implementations for a Low-Cost Autonomous Surface Vehicle Using the MOOS-IvP Software

6. AUTOR(ES):

David Issa Mattos

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica - ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Autonomous Surface Vehicle; Sensor Fusion; Autonomous Navigation

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Veiculos autônomos; Navegação autônoma; Fusão de sensores; Motores de corrente contínua; Sistemas de propulsão; Controle; Computação.

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

ITA, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Engenharia Eletrônica e Computação. Área de Dispositivos e Sistemas Eletrônicos. Orientador: Prof. Dr. Cairo Lúcio Nascimento Júnior. Coorientador: Prof. Dr. Douglas Soares dos Santos. Defesa em 22/03/2016; 2016.

11. RESUMO:

This work describes the implementation of a low-cost Autonomous Surface Vehicle (ASV) using a behavior-based software, MOOS-IvP. The platform used is a catamaran boat driven by two direct current motors as the propulsion system. Two different implementations were made and both are presented and discussed in this work.

In the first approach, the ASV is embedded with a processing board with an Arduino microcontroller, a low-cost Inertial Measurement Unit (IMU) with accelerometers, gyroscopes and magnetometers, a GPS receiver and a wireless RF serial modem. The ASV communicates with a Ground Control Station (GCS) sending telemetry data and receiving navigation commands for the propulsion motors. The GCS uses the MOOS-IvP software to implement the autonomous navigation procedures and the GPS/Compass/IMU sensor fusion algorithms.

In the second approach, modifications were made in the ASV embedded hardware. A low-cost microcomputer (Raspberry Pi 2), a WiFi adapter and an USB camera for surveillance were added and the RF serial modem was removed. The main difference between the two approaches is that in the second approach the autonomous navigation procedures and the sensor fusion algorithm run embedded in the ASV. Therefore, the ASV is capable of performing a mission even if the communication link with the GCS is lost. Although the ASV does not require a GCS to operate, a GCS was used to deploy the missions and give manual remote control over the ASV. The embedded computer runs the MOOS-IvP software to implement the autonomous navigation procedures and the GPS/Compass/IMU sensor fusion algorithm. The GCS uses the MOOS-IvP software and receives telemetry data from the ASV and sends control commands. This approach aims for a modular system that allows it to be expanded and modified to meet the custom needs of specialized missions.

Simulations were used to demonstrate the viability of missions, tuning and using of the sensor fusion algorithms. In both approaches, experimental results in real conditions are presented and discussed. The experimental and simulation results consist of path following missions in the presence of virtual obstacles.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **CONFIDENCIAL** ( ) **SECRETO**