INSTITUTO TECNOLÓGICO DE AERONÁUTICA

David Issa Mattos

Implementação do *software* MOOS-IvP em um barco autônomo

Trabalho de Graduação
2014

Curso de Engenharia Eletrônica

CDU 681.53

David Issa Mattos

Implementação do *software* MOOS-IvP em um barco autônomo

Orientadores

Prof. Dr. Cairo Lúcio Nascimento Júnior (ITA)

Prof. Dr. Douglas Soares dos Santos (ITA)

Curso de Engenharia Eletrônica

SÃO JOSÉ DOS CAMPOS
INSTITUTO TECNOLÓGICO DE AERONÁUTICA

2014

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Issa Mattos, David
Implementação do software MOOS-IvP em um barco autônomo / David Issa Mattos.
São José dos Campos, 2014.
90f.

Trabalho de Graduação – Curso de Engenharia Eletrônica–
Instituto Tecnológico de Aeronáutica, 2014. Orientadores: Cairo Lúcio Nascimento Jr., Prof. Dr. e Douglas Soares dos Santos., Prof. Dr.

1. Navegação Autônoma. 2. Barcos. 3. MOOS-IvP. 4. Controle

REFERÊNCIA BIBLIOGRÁFICA

MATTOS, David Issa. **Implementação do software MOOS-IvP em um barco autônomo**. 2014. 90f. Trabalho de Conclusão de Curso. (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DO AUTOR: David Issa Mattos
TÍTULO DO TRABALHO: Implementação do software MOOS-IvP em um barco autônomo
TIPO DO TRABALHO/ANO: Graduação / 2014

_____

David Issa Mattos
Rua Roma, 673, Ed. Citta di Roma, Ap 145D
São José dos Campos-SP, 12216-510

IMPLEMENTAÇÃO DO SOFTWARE MOOS-IVP EM UM BARCO AUTÔNOMO

Essa publicação foi aceita como Relatório Final de Trabalho de Graduação

David Issa Mattos

Prof. Dr. Cairo Lúcio Nascimento Júnior (ITA)
Orientador

Prof. Dr. Douglas Soares dos Santos (ITA)
Orientador

Profa. Dra. Neusa Maria Franco (ITA)
Coordenadora do Curso de Engenharia Eletrônica

São José dos Campos, 17 de novembro de 2014

Dedico este trabalho a Érika.

Agradecimentos

Agradeço aos orientadores professores Cairo L. Nascimento Jr. e Douglas Soares dos Santos, pelas diversas contribuições e sugestões ao longo deste trabalho.

Aos engenheiros Alessandro Paolone e José Ricardo pelas contribuições na análise de imagens.

Aos técnicos da Divisão de Engenharia Eletrônica, em especial ao Marcos, pela grande ajuda ao longo deste trabalho.

À Érika Ramos, por todo o suporte e paciência ao longo de toda a graduação.

Aos meus pais, irmãos e avós que sempre me incentivaram e apoiaram meus estudos.

Deve haver um começo para qualquer grande questão, mas continuar até o fim, até ser completamente terminado traz a verdadeira glória.
Francis Drake

Resumo

Este trabalho apresenta a implementação do *software* MOOS-IvP, de navegação autônoma, em barco de baixo custo. O *software* MOOS-IvP dá suporte a configuração e planejamento de missões autônomas e cooperativas para veículos, em particular para veículos subaquáticos.

O barco utilizado nesta aplicação é do tipo catamarã, que utiliza como sistema de propulsão rodas d'água impulsionadas por motores elétricos de corrente contínua. Foi projetado e instalado para esse barco um módulo embarcado com uma plataforma inercial (com acelerômetros, giroscópios e magnetômetros), um receptor GPS, um microcontrolador e um sistema de comunicação. O módulo embarcado se comunica com uma estação de controle em terra enviando a telemetria dos sensores e recebendo os comandos para o acionamento dos propulsores. A estação de terra utiliza o *software* MOOS-IvP como solução de navegação.

Também é mostrado neste trabalho como acoplar um módulo de transmissão de imagem, com capacidade de processamento para visão computacional, ao *software* MOOS-IvP.

Este trabalho abrange a simulação, configuração de missões e implementação no sistema real, mostrando que o barco segue uma trajetória pré-definida.

Abstract

This text presents the implementation of the autonomous navigation software MOOS-IvP in a low cost boat. The software MOOS-IvP gives capability to execute autonomous and collaborative missions for aquatic vehicles.

This work uses a catamaran boat, that has water wheels driven by direct current motors as propulsion system. It was developed an embedded system with inertial sensors (accelerometers, gyroscopes and compass), a GPS receiver, microcontroller and a communication system. This system communicates with the shoreside control station sending telemetry data and receiving navigation commands for the propulsion motors. The shoreside control station uses the software MOOS-IvP as the navigation decision maker.

It is also shown how to create an image transmission system with capability of computational vision integrated with the software MOOS-IvP.

This work covers the mission simulation, configuration and implementation in the real boat, showing that the boat can follow a pre determined waypoint mission.

Palavras–Chave: MOOS-IvP, Navegação autônoma

Keywords: MOOS-IvP, autonomous navigation

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

IMU	Unidade de Medida Inercial (*Inertial Measurement Unit*)
NED	Sistema de referência *North-East-Down*
MOOS	*Mission Oriented Operating Suite*
IvP	*Interval Programming*
GPS	Sistema de Posicionamento Global (*Global Positioning System*)

Lista de Símbolos

ψ	Ângulo de guinada do barco
ψ_{Guinada}	Ângulo de guinada do barco, no sistema de coordenadas local
P_x	Posição do barco no eixo x, no sistema de coordenadas local
P_y	Posição do barco no eixo y, no sistema de coordenadas local
V_x	Velocidade no eixo x, referencial do barco
V_y	Velocidade no eixo y, referencial do barco
m	Massa do barco
J	Momento de inércia do barco
d	Distância entre os motores e o centro de gravidade do barco
F_d	Força aplicada pelo motor direito do barco
F_e	Força aplicada pelo motor direito do barco
F_p	Força de propulsão dos motores
$M1$	Motor esquerdo
$M2$	Motor direito
TD_z	Torque de arrasto no eixo z do barco
K_z	Constante de arrasto em z
K_x	Constante de arrasto em x
K_y	Constante de arrasto em y
FD_x	Força de arrasto no eixo x do barco

FD_y	Força de arrasto no eixo y do barco
δd	Erro de distância entre o barco e um ponto de interesse
$\delta\psi_{\text{Guinada}}$	Erro de guinada entre o barco e um ângulo de interesse

Lista de Códigos

1 Introdução

1.1 Contextualização e Motivação

Atualmente há um crescente interesse em sistemas de navegação autônoma para realizar tarefas que apresentam riscos aos seres humanos ou tarefas que são de difícil acesso. Em particular o desenvolvimento de veículos aquáticos e sub aquáticos autônomos podem ser empregados em atividades como tarefas de resgate em áreas de risco, verificação de tubulação de óleo e gás, monitoramento e verificação de cabos submarinos e coletas de dados ambientais em áreas de difícil acesso (SIDELEAU; EICKSEDT, 2010) e (SPEARS; WEST; COLLINS, 2013).

Nos últimos anos diversas empresas criaram seus próprios veículos autônomos e desenvolveram seus próprios softwares para controle e navegação desses veículos. Devido a presença de diversos tipos de veículos, cada um utilizando seu próprio software, surgiu um problema de integração em missões cooperativas entre esses veículos. Neste contexto foi desenvolvido o *software open source* MOOS-IvP. Este *software* visa dar autonomia a veículos e possibilitar a integração de diversos veículos em uma única estrutura de navegação. Criando assim um ambiente favorável de desenvolvimento de missões cooperativas autônomas entre veículos.

1.2 Objetivo

Este trabalho busca implementar um barco autônomo de baixo custo utilizando o *software* MOOS-IvP. Utilizando uma estrutura de um barco do tipo catamarã, é adaptado o *hardware* embarcado de forma a possibilitar a integração do barco desenvolvido com o *software* MOOS-IvP.

Assim propõe-se inicialmente neste estudo a implementação da navegação utilizando apenas sensores de baixo custo, como uma unidade de medidas inerciais e um sistema de navegação global. A unidade de medida inercial permite a obtenção de dados do ângulo de guinada utilizando especificamente os magnetômetros e o sistema GPS permite a obtenção de dados de posição e de velocidade do veículo. Os resultados obtidos da navegação com o *software* MOOS-IvP visam a execução de missões simples, como seguir uma trajetória específica pré-determinada.

1.3 Estrutura do Texto

Este texto segue a seguinte estrutura. No capítulo 2 é descrito o barco utilizado, com uma descrição do seu modelo e do *hardware* embarcado.

O capítulo 3 traz uma breve descrição do *software* MOOS-IvP, com suas principais funcionalidades, essa descrição não busca ser exaustiva, uma descrição mais detalhada deve ser consultada (BENJAMIN *et al.*, 2013) para utilização do *software*.

O capítulo 4 mostra como foi realizada a integração do *software* MOOS-IvP com o *hardware* embarcado. Este capítulo ainda traz resultados da navegação e uma comparação do modelo simulado com o sistema real.

O capítulo 5 mostra como é possível implementar uma transmissão de imagem e visão computacional com o *software* MOOS-IvP. Neste capítulo também é mostrado como a comunicação entre o computador embarcado e o computador em terra pode ser realizada utilizando redes locais sem fio com protocolos TCP e UDP.

Por fim o capítulo 6 traz as considerações finais do trabalho.

2 O Barco

2.1 Introdução

Nesta seção será observado características específicas do barco que será utilizado, especificações de *hardware* e modelo do barco. O *software* MOOS-IvP será utilizado como solução de navegação ao barco, mas estará em execução em um computador remoto ao barco.

A solução de *hardware* proposta é de baixo custo e fácil implementação. A estrutura do barco utilizada foi desenvolvida em (SANTOS, 2011).

Esta seção esta dividida em mais duas outras seções. A seção 2.2 trata da estrutura física utilizada e o modelo matemático do barco. A seção 2.3 trata do *hardware* envolvido para implementar a solução de navegação utilizando o MOOS-IvP e o controle do barco.

Um sistema de transmissão e processamento de imagem foi implementado como uma solução a parte e não está integrada ao *hardware* necessário para o controle e a navegação do barco, portanto será considerado de forma separada e independente e está descrito no capitulo 5.

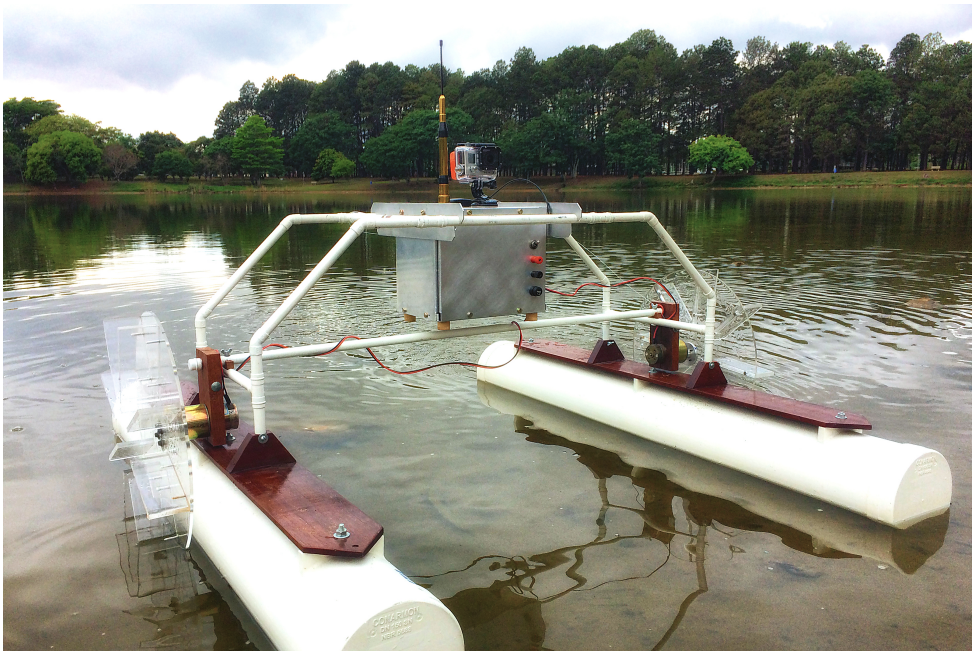

FIGURA 2.1 – Foto do barco utilizado. Fonte: (SANTOS, 2011)

2.2 A estrutura

A estrutura e o modelo matemático adotado para o barco foram desenvolvidas em (SANTOS, 2011). O barco possui uma estrutura do tipo catamarã, com dois motores de propulsão instalados cada um ao lado externo de seu flutuador. Os motores de propulsão podem assumir apenas três valores, ligado para frente, desligado e ligado para trás.

Esse tipo de estrutura permite a realização de manobras em uma área pequena e uma fácil implementação. A Figura 2.1 mostra a estrutura utilizada. Uma simplificação será feita neste estudo em relação ao proposto originalmente. Apesar do tipo de barco adotado possuir a capacidade de inversão de proa, será considerado apenas uma única proa.

Para essa estrutura em (SANTOS, 2011) foi definido o modelo matemático para o barco representado na equação 2.4. O modelo foi desenvolvido com base na representação do diagrama de forças da figura 2.2.

Neste modelo w representa a velocidade angular de rotação, V_x e V_y as velocidades

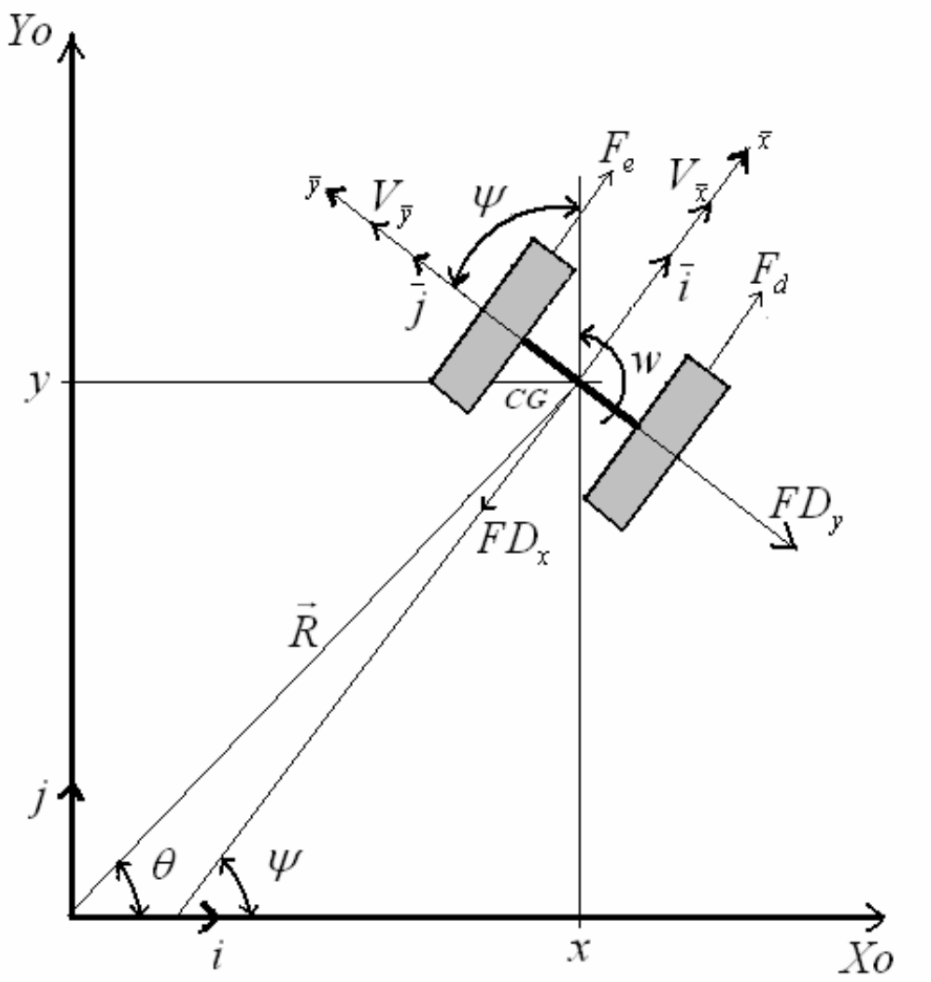

FIGURA 2.2 – Diagrama de forças atuando sobre o barco. Fonte: (SANTOS, 2011)

lineares no referencial do barco, P_x e P_y as posições em relação a origem do sistema de coordenadas local e ψ o ângulo que o eixo y do barco faz com o eixo Y_0 do sistema de coordenadas local. Observe que nesta figura o ângulo de guinada é de $\psi - 90^o$. F_d e F_e representam os motores de propulsão direitos e esquerdos do barco, m a massa do barco, J o momento de inércia do barco e d a distância entre o motor e o centro de gravidade do barco.

As forças de arrasto são dadas por:

$$TD_z = K_z w^2 \tag{2.1}$$

$$FD_x = K_x V_x^2 \tag{2.2}$$

$$FD_y = K_y V_y^2 \tag{2.3}$$

Assim o modelo pode ser escrito como:

$$\begin{pmatrix} \dot{w} \\ \dot{V}_x \\ \dot{V}_y \\ \dot{P}_x \\ \dot{P}_y \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} \dfrac{(F_d - F_e)d - \text{sign}(w)TD_z}{J} \\ wV_y + \dfrac{F_d + F_e - \text{sign}(V_x)FD_x}{m} \\ -wV_x - \dfrac{\text{sign}(V_y)FD_y}{m} \\ V_x \cos(\psi) - V_y \sin\psi \\ V_x \sin(\psi) + V_y \cos\psi \\ w \end{pmatrix} \tag{2.4}$$

Os motores de propulsão não possuem controle de velocidade e como dito anteriormente assumem um de três valores. Os motores direito e esquerdo são iguais e as forças de propulsão são assumidas como:

$$F_e = \text{M1} \cdot F_p$$

$$F_d = \text{M2} \cdot F_p$$

$$\text{M1}, \text{M2} = \{-1, 0, 1\}$$

F_p é a força de propulsão. M1 é uma constante que representa o motor esquerdo e M2 é uma constante que representa o motor direito. Os valores $(-1, 0, 1)$ que M1 e M2

TABELA 2.1 – Tabela de acionamento dos motores

M1,M2	$0 < \delta d < 3$	$\delta d \geq 3$
$5 < \delta\psi_{\text{Guinada}} \leq 90$	1,-1	1,0
$0 < \delta\psi_{\text{Guinada}} \leq 5$	0,0	1,1
$-5 < \delta\psi_{\text{Guinada}} \leq 0$	0,0	1,1
$-90 < \delta\psi_{\text{Guinada}} \leq 90$	-1,1	0,1

assumem, representam, respectivamente, os motores ligados para trás, desligados e ligados para frente.

As constantes do modelo foram medidas ou estimadas por (SANTOS, 2011) como:

$$m = 14.81 \text{ kg}$$

$$J = 3.37 \text{ kg/m}^2$$

$$F_p = 1.4 \text{ N}$$

$$d = 0.56 \text{ m}$$

$$K_z = 35.12$$

$$K_x = 7.025$$

$$K_y = 35.12$$

Para o controle dos motores é realizado pelo computador embarcado utilizando a tabela de acionamentos (Tabela 2.1), onde $\delta\psi_{\text{Guinada}}$ representa o erro de guinada e δd representa a distância entre a posição atual e a nova posição desejada.

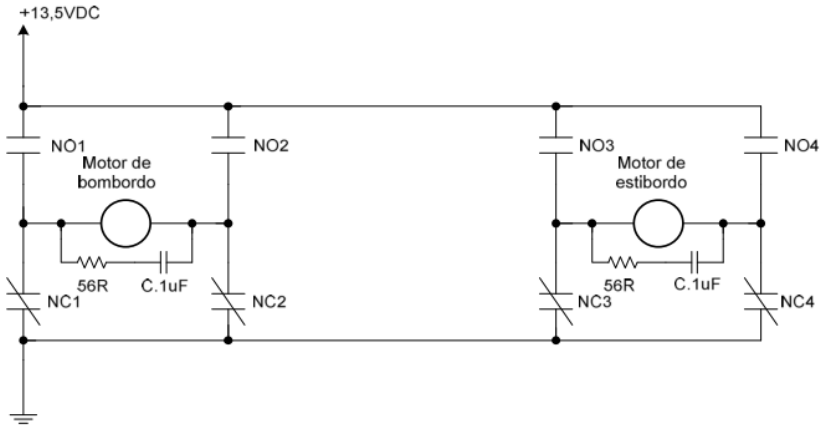

FIGURA 2.3 – Diagrama esquemático da ponte H. Fonte: (SANTOS, 2011)

2.3 O *hardware*

Esta seção descreve o *hardware* embarcado utilizado, o controlador de potência dos motores, os sensores inerciais, o GPS, o rádio-modem e o microcontrolador utilizado.

2.3.1 Controlador de potência dos motores

O controlador de potência dos motores utilizado foi o mesmo utilizado em (SANTOS, 2011). Este consiste de uma ponte H implementada com relés. As figuras 2.3 e 2.4 representam um diagrama esquemático da ponte H e a implementação desta utilizando relés, respectivamente

2.3.2 Sensores inerciais

Neste estudo foram utilizados sensores inerciais do tipo *strapdown* de custo baixo. O modelo utilizado possui dez graus de liberdade, sendo três acelerômetros, três giroscópios, três magnetômetros e um barômetro. A Figura 2.5 representa a plataforma utilizada.

Essa plataforma inercial, a IMU (*Inertial Measurement Unit*), comunica-se por meio

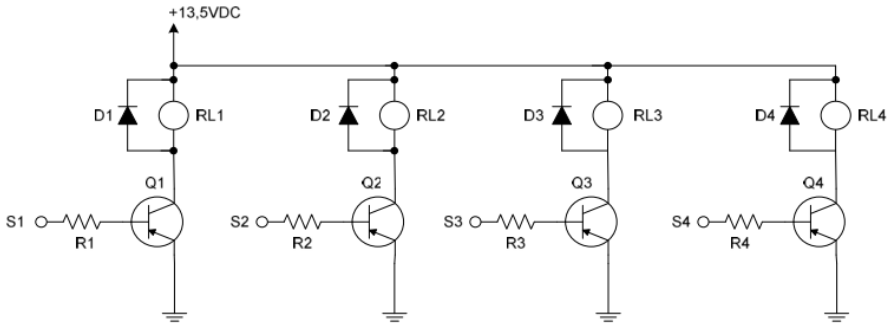

FIGURA 2.4 – Implementação da ponte H da figura 2.3 com relés. Fonte: (SANTOS, 2011)

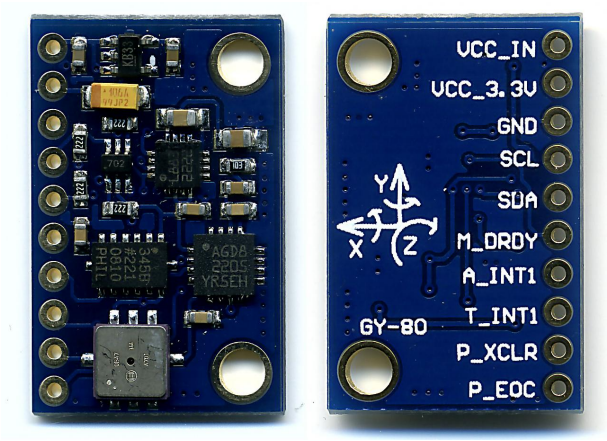

FIGURA 2.5 – Sensores inerciais utilizados, modelo GY-80. Fonte: http://technovade.pl

de uma interface I^2C e é baseada nos seguintes nos seguintes circuitos integrados:

- **Acelerômetro**: ADXL345. Endereço I^2C: $0x53$.
- **Giroscópio**: L3G4200D. Endereço I^2C: $0x69$.
- **Magnetômetro**: HMC5883L. Endereço I^2C: $0x1E$.
- **Barômetro**: BMP085. Endereço I^2C: $0x77$.

Esta plataforma inercial possui dimensões 25.8mm x 16.8mm e massa 5g.

2.3.3 GPS

O receptor de sinal GPS foi utilizado para obtenção da localização do barco. O modelo utilizado foi o ET-332, fabricado pela GlobalSat Technology Corporation (GLOBALSAT,).

Esse GPS é de baixo custo (U$35) e utiliza o *chipset* SiRF Star III, trabalha com tensão entre 3.5 a 6.5 V com consumo médio de 42 mA. As mensagens enviadas seguem o protocolo NMEA 0183 (SIRF, 2005) e se comunica por interface serial TTL.

A Figura 2.6 mostra o GPS utilizado.

O *chipset* utilizado (SiRF Star III) é desenvolvido para aplicações veiculares. Devido a esse motivo o *chipset* vem configurado como padrão a utilização da tecnologia *static navigation*. Essa tecnologia permite que o GPS mantenha uma posição estática caso a velocidade dele seja inferior a um certo limite configurável. No caso desse GPS o limite inferior é por volta de 4 km/h. Como o barco utilizado não alcança esta velocidade inferior os dados do GPS não são atualizados. Assim é necessário configurar o GPS de modo a desabilitar esta funcionalidade.

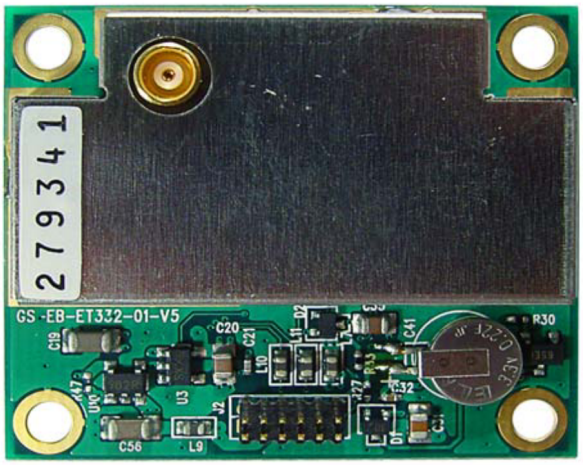

FIGURA 2.6 – GPS GlobalSat ET-332. Fonte: (GLOBALSAT,)

Os documentos (SIRF, 2007a) e (SIRF, 2007b) descrevem o conjunto de sentenças que controlam o funcionamento do GPS. O GPS é configurado inicialmente para trabalhar com sentenças NMEA, no entanto, opções de configuração mais avançadas do GPS são acessíveis apenas por meio das sentenças binárias, como controle de potência, economia de energia e o modo *static navigation*. Desse modo é necessário passar o GPS para o modo binário, configura-lo e então retorna-lo para sentenças NMEA.

Uma forma de configurar o GPS é utilizando o aplicativo para sistemas operacionais *Windows*, *SiRFdemo*. Esse aplicativo permite configurar o GPS por meio de uma interface gráfica. No entanto, caso a bateria de memória do GPS se esgote o procedimento de configura-lo deve ser feito novamente. O *software* embarcado realiza esse procedimento automaticamente na sua inicialização, para evitar reconfigurá-lo.

2.3.4 Rádio Modem

O rádio modem utilizado é o modelo SS7-900EXT da fabricante ICP Das, Figura 2.7. Este rádio modem utiliza o padrão RS-232 ou RS-485 para comunicação e é alimentado por 12V. Neste trabalho o modem foi configurado para uma conexão ponto-a-ponto a

FIGURA 2.7 – Rádio Modem utilizado.

uma taxa de transmissão de 9600 bps utilizando o protocolo RS-232. Assim utiliza-se dois rádios, um conectado ao computador embarcado e o outro conectado ao computador base.

A conversão entre sinais TTL e RS-232 no barco é feita utilizando o circuito conversor MAX232N. A conversão entre sinais RS-232 para dados serial em uma USB é feita utilizando o DB-UC001 da *Sure electronics*, Figura 2.8. Esse conversor é baseada no *chip* CP2102 da *Silabs* para comunicação USB e possui conectores DB9 macho e fêmea, e aceita sinais TTL de 5 e de 3.3V.

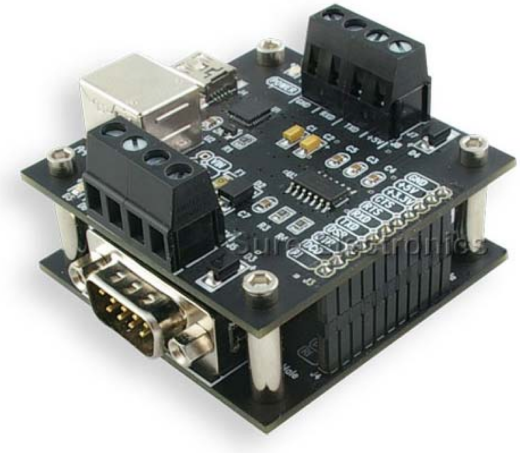

FIGURA 2.8 – Conversor USB DB-UC001. Fonte: (SURE, 2008)

2.3.5 Microcontrolador embarcado

O microcontrolador embarcado utilizado é a plataforma de desenvolvimento Arduino Uno R3. Essa plataforma utiliza o microcontrolador ATMEGA328p, da fabricante ATMEL. A Figura 2.9 mostra a plataforma de desenvolvimento utilizada.

A utilização da plataforma Arduino foi adotada pelas seguintes características:

- Baixo custo: R$ 50 (Valor levantado em 2014);
- Não necessita de gravador;
- Solução de *software* com várias bibliotecas já prontas;
- Grande base de usuários, o que facilita a obtenção de informações e de bibliotecas *open-source* para utilizar em projetos e;
- Criação de bibliotecas novas utilizando C++.

Assim devido a grande base de usuários e de bibliotecas prontas o tempo de desenvolvimento do *software* embarcado reduz significativamente.

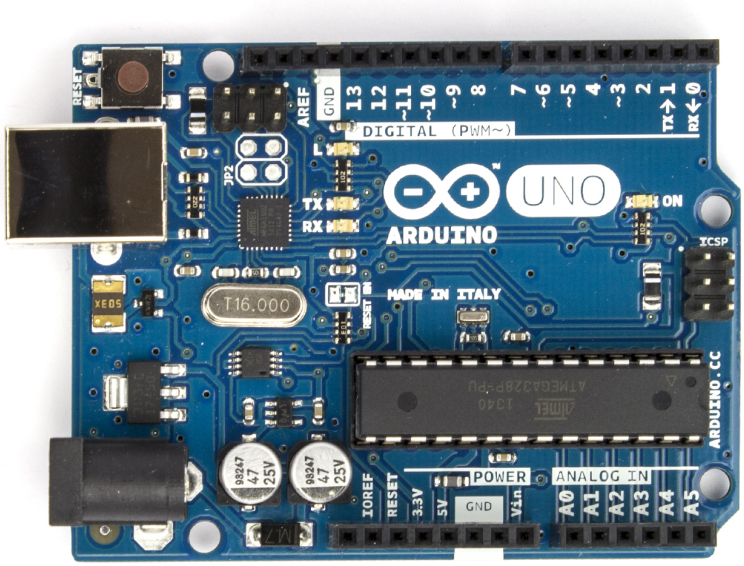

FIGURA 2.9 – Arduino Uno R3. Fonte: www.arduino.cc

O Arduino possui outros modelos com mais funcionalidades e periféricos (modelos Mega, Leonard ou Due). No entanto o Arduino Uno R3 possui periféricos e memória de programa necessárias para executar as tarefas. Uma descrição do *software* embarcado, assim como das bibliotecas utilizadas encontra-se no capítulo 4.

2.3.6 Solução integrada para o *hardware* embarcado

O *hardware* descrito nas subseções 2.3.1, 2.3.2, 2.3.3, 2.3.4 e 2.3.5 acima foi integrado em uma única plataforma, na forma de um *shield* para arduino, como mostrado nas figuras 2.10, 2.11.

A IMU conecta-se ao Arduino pelas entradas I^2C (pinos SDA e SCL). O rádio modem utiliza as entradas TX e RX da USART por *hardware*. O GPS utiliza pinos digitais simples, sendo então implementadas uma serial por *software*. A a placa controladora dos motores utiliza pinos digitais para o acionamento dos mesmos. O sistema é alimentado por uma fonte de tensão contínua de 5V.

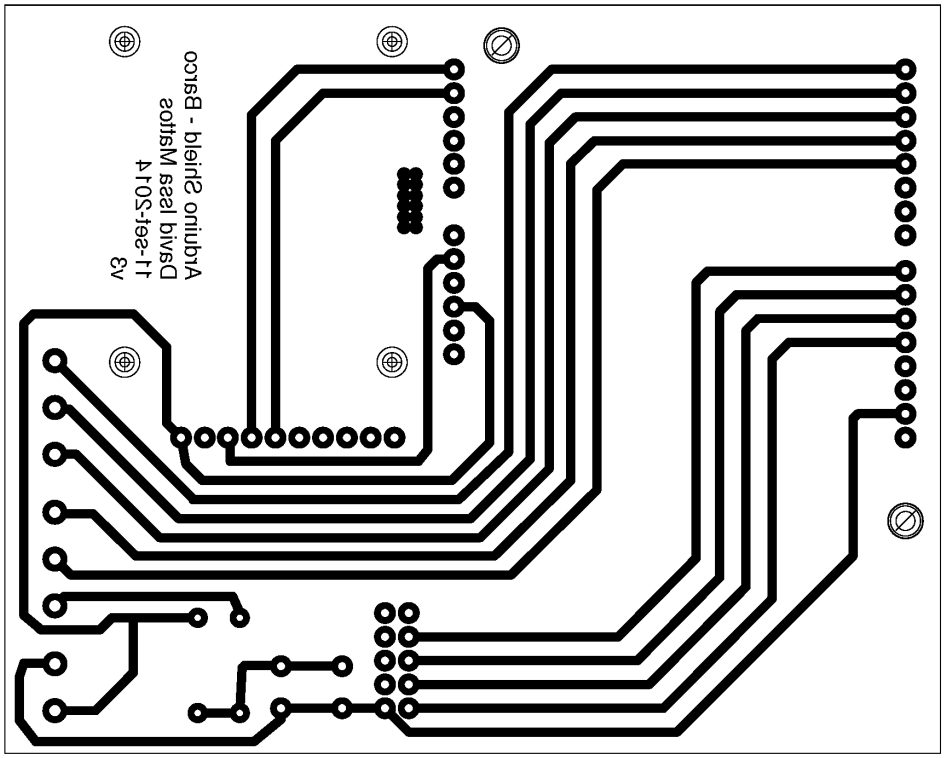

FIGURA 2.10 – *Layout* da placa de circuito impresso utilizada.

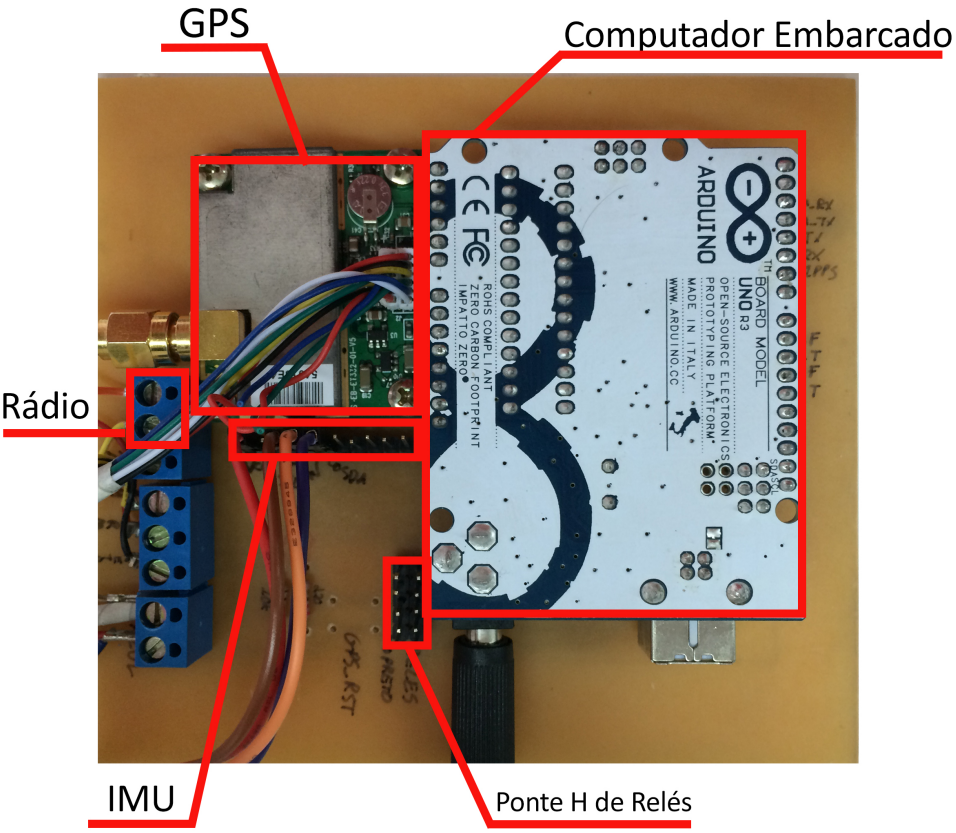

FIGURA 2.11 – Placa de circuito impresso confeccionada.

3 O MOOS-IvP

A solução de navegação utilizada neste trabalho é dada pelo *software* MOOS-IvP. Esse *software* implementa uma suíte de aplicativos para dar controle e autonomia para barcos e veículos subaquáticos. No entanto as aplicações desse *software* não se restringem a veículos aquáticos, podendo ser aplicados também a veículos terrestres e aéreos (BENJAMIN *et al.*, 2013). Uma das funções principais que o *software* MOOS-IvP implementa é o suporte a missões colaborativas entre diversos veículos.

O MOOS-IvP é composto de duas partes, o MOOS, um *software* de comunicações para implementação em robôs móveis, e o IvP, um algoritmo de tomada de decisões baseadas em comportamentos, que dá ao MOOS capacidade de navegação autônoma. O MOOS é descrito em mais detalhes na seção 3.1 e o IvP na seção 3.2.

A arquitetura do MOOS-IvP foi desenvolvida tendo em vista três princípios :

- O modelo *back-seat driver*;
- Sistema de comunicação do tipo assinante-publicador e;
- Decisões de navegação autônoma baseada em comportamentos (*behaviors*).

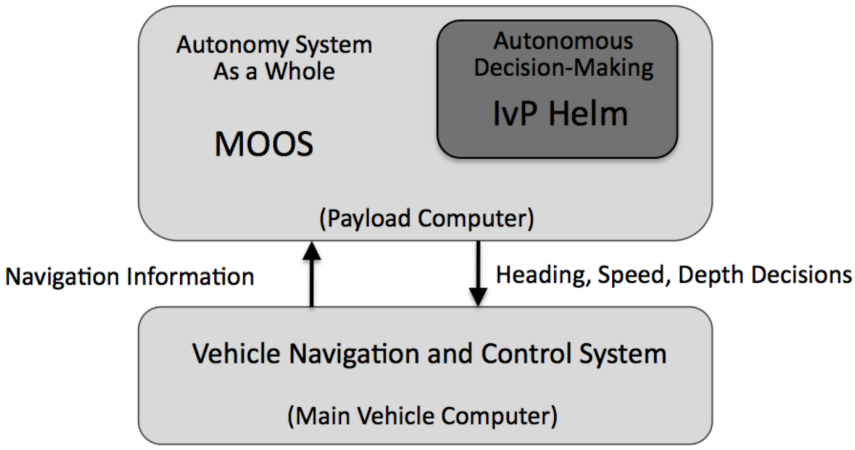

FIGURA 3.1 – Modelo *back-seat driver* para o *software* MOOS-IvP. Fonte: (BENJAMIN *et al.*, 2013)

3.0.7 O modelo back-seat driver

O modelo *back-seat driver* consiste em implementar de modo independente o controle dos atuadores do sistema e a tomada de decisão para a navegação autônoma. Assim pode-se utilizar um veículo com seus controladores próprios e não necessariamente conhecidos e o sistema de navegação, que dará aos controladores do veículos os novos *setpoints*. Dessa maneira o sistema que provê a autonomia não precisa ter conhecimento da implementação do sistema de controle. O *software* MOOS-IvP pode ser implementado tanto no mesmo computador do sistema de controle como em um computador separado.

A figura 3.1 apresenta esse modelo.

3.0.8 Sistema assinante-publicador

Essa arquitetura é definida com uma topologia estrela de comunicação onde a comunicação entre os aplicativos ocorre por meio de um *host* que coordena as mensagens trocadas. Aplicativos que desejam receber atualização de alguma variável devem assiná-la.

Aplicativos que fornecem alguma variável devem publicá-las ao *host*.

Essa é a estrutura de comunicação determinada pelo *software* MOOS e ela é descrita em mais detalhes na seção 3.1.

3.0.9 Decisões baseadas em comportamentos

Comportamentos podem ser vistos como sistemas autocontidos dedicados para uma parte específica da tomada de decisão da navegação de um veículo. Exemplos de comportamentos são, seguir uma trajetória específica, evitar colisão ou seguir outro veículo são exemplos de de comportamentos. Esses comportamentos estão ativos durante a navegação. Caso hajam comportamentos conflitantes, dependendo da prioridade do comportamento um decisor baseado na técnica computacional *Interval Programming* realiza uma otimização para conciliar os diversos objetivos e dar uma solução de navegação. A seção 3.2 descreve com mais detalhes o IvP e as decisões baseadas em comportamentos.

3.0.10 Instalação do MOOS-IvP

O *software* MOOS-IvP pode ser obtido através do *website* `http://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php`. Após o *download* do *software* os arquivos README específicos de cada sistema operacional (Linux, Mac OS X e Windows) guiam a instalação do MOOS-IvP. A instalação do MOOS-IvP no sistema operacional selecionado não acrescenta os aplicativos do *software* ao PATH. Assim antes de utilizar o MOOS-IvP pode-se utilizar o *script* abaixo (no Mac OS X ou no Linux) para acrescentar os arquivos ao PATH do sistema. Esse *script* também exclui qualquer processo MOOS que tenha ficado aberto. Este *script* está disponível também através do *website* `ftp://labattmot.ele.ita.br/`

ele/david/My_Publications/TG/Codigos/moos-ivp-extend/src/.

Código 3.1 – Inicialização para utilizar o MOOS-IvP

```
1  #Adiciona as arvores ao PATH
2    export PATH=$PATH:/Users/David/moos-ivp/bin
3    export PATH=$PATH:/Users/David/moos-ivp-extend/bin
4  #verifica se os aplicativos foram acrescentados ao PATH
5    which MOOSDB
6    which uTimerScript
7  #Mata os processos MOOS em execucao
8    killall MOOSDB
9    killall pLogger
10   killall pHelmIvP
11   killall pMarinePID
12   killall uProcessWatch
13   killall uSimMarine
14   killall pNodeReporter
15   killall pControleBarco
16   killall uBarcoDouglas
17   killall iSerial
18   killall pmyGeodesy
19 echo FIM
```

3.1 O MOOS - *Mission Oriented Operating Suit*

3.1.1 Introdução ao MOOS

O MOOS (*Mission Oriented Operating Suit*) é um programa do tipo *middleware* que coordena a comunicação entre diversos aplicativos que operam dentro do sistema de comunicação MOOS (NEWMAN, 2008). Atualmente o *MOOS* é mantido e atualizado pelo *Mobile Robotics Group* na Universidade de Oxford. As bibliotecas e o sistema MOOS estão disponíveis sob uma licença *open source GLP*.

O programa foi desenvolvido por Paul Newman em 2001 e hoje encontra-se em sua versão 10. Escrito em $C++$, o *software* é voltado para a implementação em robôs móveis. Suas principais características que o diferenciam dos outros *softwares* são:

- Independência entre plataformas, portanto pode ser executada em sistemas Windows, Linux e MAC OS X;
- Possui um sistema de comunicação robusta, trabalha com erros de comunicação, mensagens repetidas, periódicas e intermitentes;
- Voltado para a implementação em tempo real;
- Independência entre os processos, visando um desenvolvimento mais rápido e;
- Muitos processos são disponíveis com uma licença do *open source* GNU, no entanto isso não impede o desenvolvimento de módulos proprietários.

O MOOS segue uma topologia do tipo estrela, onde o elemento central (MOOSDB) coordena as trocas de mensagens. O programa funciona seguindo a filosofia de ”assinante-publicador”, onde cada processo (ou aplicativo MOOS) escolhe as mensagens que tem

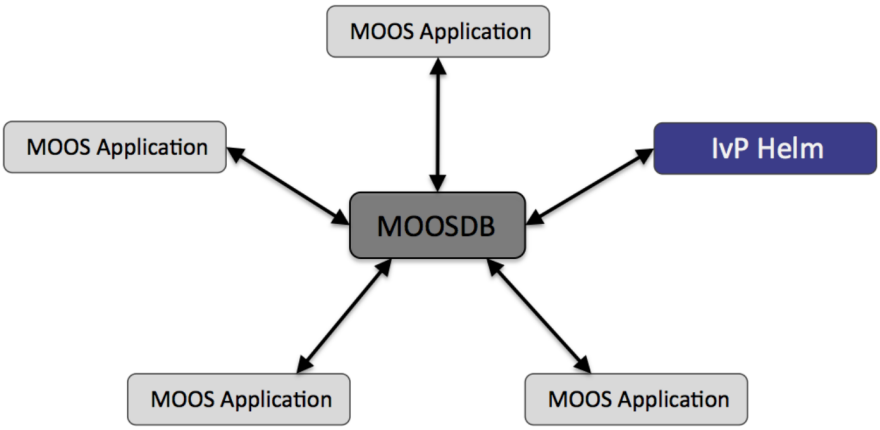

FIGURA 3.2 – Topologia em estrela do *software* MOOS. Fonte: (BENJAMIN *et al.*, 2013)

interesse em receber (o processo recebe notificações e atualizações do *status* de cada variável que foi assinada), e publica as variáveis que por ele foram processadas.

O elemento central desta topologia é o processo MOOSDB, que mantém uma lista das variáveis atualizadas, recebendo novas variáveis publicadas e enviando mensagens para os processos que assinaram variáveis específicas. As mensagens assinadas podem ter especificadas a frequência em que serão verificadas alterações em seus valores.

A figura 3.2 representa a topologia estrela do *software* MOOS. Observa-se que cada processo conecta-se e comunica-se somente com o MOOSDB, não havendo comunicação direta entre os processos. Isso permite que os desenvolvimentos de processos sejam independentes um do outro, de forma que se um processo, que troca informações com MOOSDB, for atualizado ou mesmo reescrito não há a necessidade de atualizar os outros aplicativos de modo que todos consigam se comunicar.

O desenvolvimento independente de aplicativos permite um reaproveitamento de aplicativos já bastante testados reduzindo assim o tempo de desenvolvimento do projeto. O programa em geral é executado por meio de linhas de comando, mas possui suporte para interfaces gráficas. Alguns aplicativos disponíveis para uso são:

- **pHelm**: Um aplicativo para determinação de *setpoints* de navegação. Utiliza o método *Interval Programming*. Será abordado mais na seção 3.2.

- **pLogger**: Aplicativo que permite realizar um *log* de variáveis selecionadas durante a missão. O processamento dos dados registrados é feito por meio dos aplicativos Alog-Toolbox (BENJAMIN, 2010).

- **pMarineViewer**: Aplicativo que permite visualizar graficamente a navegação do veículo em um mapa pré-definido.

- **uXMS**: Aplicativo em terminal que permite visualizar o conteúdo das variáveis do MOOSDB.

- **uMS**: Aplicativo em interface gráfica que permite visualizar o conteúdo das variáveis do MOOSDB.

- **uPokeDB**: Aplicativo em terminal que permite alterar o conteúdo das variáveis do MOOSDB.

- **uTimerScript**: Aplicativo em terminal que permite alterar o conteúdo das variáveis do MOOSDB baseado em regras e aplicadas em períodos pré-definidos.

- **pAntler**: Aplicativo utilizado para abrir diversos aplicativos em uma mesma comunidade MOOSDB.

- **iMatlab**: Aplicativo que realiza conexão com o *software* Matlab® da Mathworks.

- **pShare**: Aplicativo que implementa *sockets* para transmissão de dados em redes.

3.1.2 Comunidades MOOS

Ao utilizar o *software* MOOS é necessário iniciar o MOOSDB para realizar a comunicação entre diversos os aplicativos. Em seguida os aplicativos são abertos e conectados ao MOOSDB. Esse conjunto de aplicativos e o MOOSDB é denominado comunidade MOOS.

Cada comunidade MOOS possui apenas um MOOSDB, mas pode ter vários aplicativos conectados. Cada comunidade MOOS realiza as comunicações através de uma porta pré-definida pelo usuário através do protocolo TCP/IP em *localhost*. Assim os aplicativos MOOS conectam-se ao MOOSDB por essa porta e todas as informações trocadas por eles ocorrem nesta porta. Assim um mesmo computador pode executar várias comunidades independentes desde que as comunidades estejam realizando as comunicações em portas diferentes.

Alguns aplicativos como o **pShare** e o **pMOOSBridge** permitem que dois MOOSDB rodando em comunidades diferentes troquem informações (NEWMAN, 2009a).

3.1.3 Arquivos de configuração **.moos*

Uma forma de executar uma comunidade (MOOSDB e aplicativos) automaticamente é utilizando a ferramenta *pAntler*, (NEWMAN, 2009b) e (NEWMAN, 2013a). Essa ferramenta lê os dados de um arquivo de configuração (**.moos*) e extrai informações sobre os aplicativos e o MOOSDB e inicializa os sistemas. O arquivo de configuração traz também configurações próprias dos aplicativos.

O código 3.2 mostra um exemplo de arquivo de configuração **.moos*. Este arquivo representa o exemplo *XRelay* descrito com detalhes em (BENJAMIN *et al.*, 2013).

Código 3.2 – Exemplo de arquivo de configuração **.moos*

```
1  // MOOS file
2
3  ServerHost = localhost
4  ServerPort = 9000
5
6  //------------------------------------------
7  // Antler configuration  block
8  ProcessConfig = ANTLER
9  {
10   MSBetweenLaunches = 200
11
12   Run = MOOSDB          @ NewConsole = false
13   Run = pXRelay         @ NewConsole = true ~ pXRelay_APPLES
14   Run = pXRelay         @ NewConsole = true ~ pXRelay_PEARS
15   Run = uXMS            @ NewConsole = true
16 }
17
18
19 //------------------------------------------
20 // First pXRelay configuration block
21
22 ProcessConfig = pXRelay_APPLES
23 {
24    AppTick    = 4
25    CommsTick = 4
26
27    OUTGOING_VAR = APPLES
28    INCOMING_VAR = PEARS
29 }
30
31 //------------------------------------------
32 // Second pXRelay configuration block
33
34 ProcessConfig = pXRelay_PEARS
35 {
36    AppTick    = 4
37    CommsTick = 4
38
39    OUTGOING_VAR = PEARS
40    INCOMING_VAR = APPLES
41 }
42
43 //------------------------------------------
44 // uXMS configuration block
45
46 ProcessConfig = uXMS
47 {
48    AppTick    = 4
49    CommsTick = 4
50
```

```
51      VAR = PEARS,   PEARS_ITER_HZ,   PEARS_POST_HZ
52      VAR = APPLES,  APPLES_ITER_HZ,  APPLES_POST_HZ
53  }
```

3.1.4 Mais informações sobre o MOOS

O Apêndice A traz uma sequência de passos de como escrever um aplicativo na API do MOOS. Esta solução é baseada na distribuição do MOOS-IvP presente no *website*: http://oceanai.mit.edu/moos-ivp/

Informações gerais sobre diversos aplicativos MOOS distribuidos com o MOOS-IvP e manuais de utilização podem ser encontrados no *website*: http://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php?n=Site.Modules

Referências gerais sobre o MOOS e sua API podem ser encontrados no *website* do MOOS: https://sites.google.com/site/moossoftware/

3.2 IvP - *Interval Programming*

3.2.1 O modelo matemático IvP - *Interval Programming*

O IvP (*Interval Programming*) é um modelo matemático para encontrar uma decisão ótima, dado um conjunto de funções objetivo que competem entre si (BENJAMIN, 2002). Nesse sentido, o IvP é uma otimização multi-objetiva baseada em comportamentos concorrentes, onde cada comportamento contribui com uma única função objetiva.

O modelo utilizado pelo IvP tenta balancear o custo computacional, a acurácia do modelo e a flexibilidade das funções utilizadas para gerar resultados dessa otimização.

Dados esses critérios o método em questão consiste em levantar funções objetivo que satisfazem um conjunto de critérios (essas funções são chamadas de funções IvP). Definido o domínio de atuação dessas funções elas são subdividas em um conjunto de funções lineares em um espaço não uniforme. As funções lineares ativas no domínio de atuação de cada uma das funções objetivas são combinadas e otimizadas através de um problema de programação linear, gerando então uma solução de decisão.

O método IvP foi utilizado como solução de navegação para o *software* MOOS-IvP. Ele é implementado como um aplicativo MOOS chamado **pHelm**.

3.2.2 O aplicativo pHelm

O aplicativo **pHelm** implementa o método IvP para tomada de decisões de uma comunidade MOOS. Junto com a implementação do **pHelm** o MOOS-IvP implementa também uma série de ferramentas (IvP *Build Toolbox*) que permite o usuário determinar seus próprios comportamentos e funções IvP . Esse trabalho, no entanto, não irá usar essas ferramentas e serão utilizados apenas os comportamentos já existentes no MOOS-IvP.

Alguns comportamentos disponíveis são:

- **Waypoint behavior**: Dados uma lista de pontos que o veículo irá navegar;
- **Loiter behavior**: Dados uma região geométrica que o veículo irá percorrer;
- **ConstantDepth behavior**: Faz o veículo subaquático manter uma profundidade predeterminada;
- **AvoidCollision behavior**: Em missões cooperativas com vários veículos esse comportamento impede a colisão entre eles e;

- **Shadow behavior**: Faz um veículo copiar a trajetória de outro.

Mais comportamentos e descrições específicas de cada um são descritos em (BENJAMIN *et al.*, 2013).

A figura 3.3 representa o funcionamento do aplicativo **pHelm** para tomada de decisão de navegação. Observa-se que o aplicativo utiliza modos de missão para levar em consideração os comportamentos ativos (subseção 3.2.3) e então o decisor IvP toma as decisões de navegação, geralmente gerando variáveis como `DESIRED_ HEADING` e `DESIRED_ SPEED` para indicar novos *setpoints* para o veículo.

Para a figura 3.3, em (1) o aplicativo **pHelm** lê (assina) as variáveis de interesse (depende dos comportamentos), mas geralmente são `NAV_ X`, `NAV_ Y`, `NAV_ HEADING` e `NAV_ SPEED` que representam as posições em x e y, direção e velocidade do veículo. Em (2) baseado no modo de missão é selecionado os comportamentos ativos para serem otimizados pelo decisor. Em (3) cada comportamento gera uma função IvP que será utilizado pelo decisor. Em (4) o decisor resolve os possíveis conflitos da otimização multi-objetiva. Em (5) os novos *setpoints* de navegação são publicados no MOOSDB, essas variáveis serão consumidas por outros aplicativos MOOS responsáveis por interfacear a solução de navegação com o veículo real, ou simulado.

3.2.3 Configuração de modos de missão

Uma das características que o aplicativo **pHelm** implementa é a hierarquização de modos de missão. Uma missão é definida hierarquicamente, onde cada nó representa um estado de operação do veículo. Em cada estado um conjunto diferente de comportamentos estará ativo. A figura 3.4 representa essa hierarquização dos modos. A missão então pode

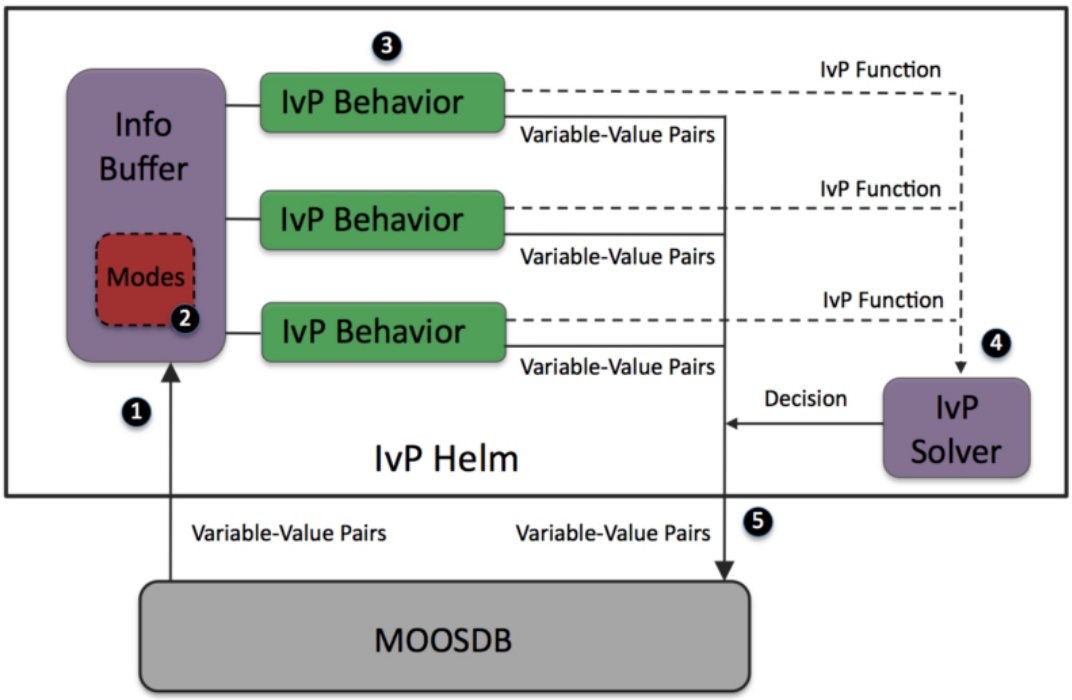

FIGURA 3.3 – O aplicativo **pHelm**. Fonte: (BENJAMIN *et al.*, 2013).

ser configurada de forma que dependendo de qual o *status* de determinada variável, ou se um objetivo for completado com sucesso a missão passa para um próximo modo, com novos comportamentos e objetivos.

3.2.4 Arquivos de configuração **.bhv*

De modo similar aos arquivos **.moos* (subseção 3.1.3) que configura os aplicativos que estarão dentro de determinada comunidade MOOS, o arquivo **.bhv*, configura os modos de missão e os comportamentos de cada modo. Esse arquivo é chamado dentro da configuração do aplicativo **pHelm** no arquivo **.moos* da comunidade.

O Código 3.3 representa um arquivo de configuração de missão (*.bhv). A missão escolhida para exemplificar é o arquivo de missão *alpha.bhv*, descrito com detalhes em (BENJAMIN *et al.*, 2013). Esta missão utiliza apenas duas variáveis representando o estado

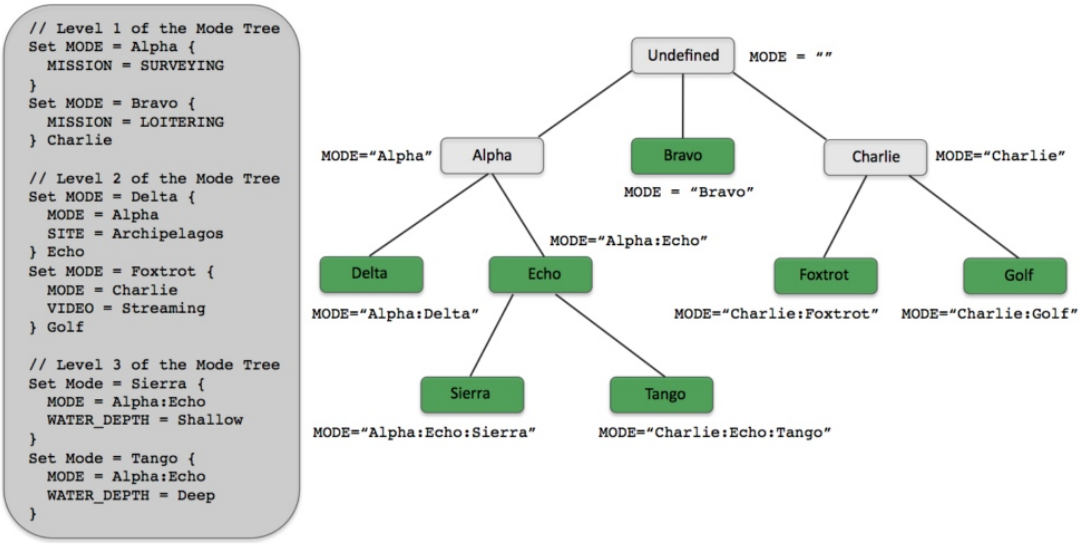

FIGURA 3.4 – Modos de missão hierarquizados. Fonte: (BENJAMIN *et al.*, 2013).

da missão (DEPLOY e RETURN) e apenas um tipo de comportamento (*waypoint behavior*), outras missões de exemplo trazem configurações de outros comportamentos como também declaração de modos de missão de forma hierarquizada.

Código 3.3 – Exemplo de arquivo de configuração **.bhv*

```
1  //--------    FILE: alpha.bhv    --------------
2
3  initialize   DEPLOY = false
4  initialize   RETURN = false
5
6  //------------------------------------------------
7  Behavior = BHV_Waypoint
8  {
9    name       = waypt_survey
10   pwt        = 100
11   condition  = RETURN = false
12   condition  = DEPLOY = true
13   endflag    = RETURN = true
14   wptflag    = WPT_HIT = $(X),$(Y)
15   UPDATES    = WPT_UPDATE
16   perpetual  = true
17
18             lead = 8
19       lead_damper = 1
20             speed = 2    // meters per second
21     capture_line = true
22   capture_radius = 5.0
23      slip_radius = 15.0
24            points = 60,-40 : 60,-160 : 150,-160 : 180,-100 : 150,-40
```

```
25            repeat = 1
26
27      visual_hints = nextpt_color=red, nextpt_lcolor=green
28      visual_hints = vertex_color=blue, edge_color=pink
29      visual_hints = vertex_size=4, edge_size=1
30  }
31
32  //------------------------------------------------
33  Behavior=BHV_Waypoint
34  {
35    name         = waypt_return
36    pwt          = 100
37    condition    = RETURN = true
38    condition    = DEPLOY = true
39    perpetual    = true
40    endflag      = RETURN = false
41    endflag      = DEPLOY = false
42    endflag      = MISSION = complete
43    UPDATES    = RETURN_UPDATE
44
45              speed = 2.0
46    capture_radius = 2.0
47       slip_radius = 8.0
48            points = 0,0
49  }
```

3.2.5 Mais informações sobre o IvP, pHelm e comportamentos

Mais informações específicas sobre IvP, **pHelm** e *IvP Build Toolbox* podem ser encontradas em (BENJAMIN, 2002), (BENJAMIN *et al.*, 2013) e (BENJAMIN, 2010). Muitas informações também podem ser consultadas em http://oceanai.mit.edu/moos-ivp/.

4 Integração do MOOS-IvP com o barco

Este capítulo traz informações de como foi implementado o *software* MOOS-IvP, (descrito no capítulo 3) com o barco (descrito no capítulo 2).

Os códigos fonte de todos os aplicativos MOOS, do *software* embarcado, configurações e missões são disponibilizados através de repositórios git, no site https://github.com/davidissamattos.

Ao fim do capítulo são discutidos os resultados obtidos por meio de simulações e de testes com o barco.

4.1 Componentes da integração

4.1.1 Introdução

Para integração entre o *software* MOOS-IvP e o barco foi utilizado a topologia presente na Figura 4.1.

Por meio desta figura é possível identificar algumas características:

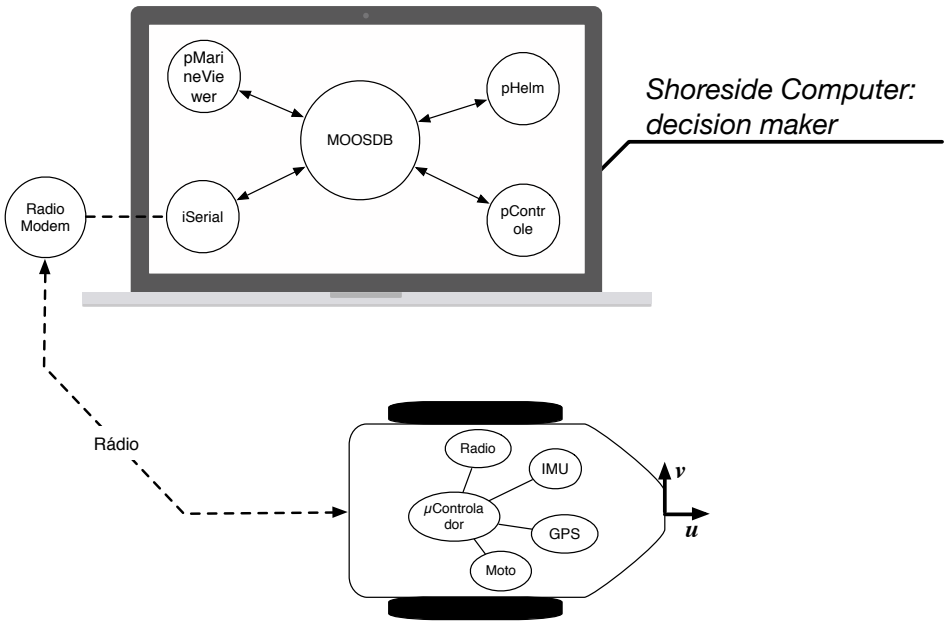

FIGURA 4.1 – Topologia adotada para integração do *software* com o barco.

- Existem dois computadores principais nesta topologia: o computador embarcado e o computador base.

- A comunidade MOOS é executada no computador base, assim todo o processo de tomada de decisão é realizada neste computador e as informações são transmitidas para o computador embarcado.

- A transmissão de informação entre o barco e o computador base é feita por meio de um rádio modem com comunicação serial.

Essa topologia permite que os custos do computador embarcado sejam reduzidos ao transferir uma boa parte do processamento para o computador base. Para poucos veículos o computador não é sobrecarregado com o processamento das tomadas de decisão de cada veículo (principalmente se a dinâmica dos mesmos forem lentas e as taxas de atualização com frequência baixa). No entanto, com o aumento do número de veículo essa abordagem torna-se computacionamelmente onerosa para o computador base. Para esse caso adota-

se um processamento distribuído, onde cada veículo possui seu computador que executa a sua própria comunidade MOOS. Assim o computador base serve como interface de visualização da missão e como ponte de transmissão e de coordenação de informação entre veículos.

Na topologia adotada, são responsabilidades do computador embarcado:

- Processar os comandos enviados pelo computador base e atuar nos motores.
- Receber e processar as informações do GPS. Eliminando dados irrelevantes e descartando dados com erros para não sobrecarregar o canal de comunicação.
- Receber e processar as informações da IMU de modo a obter o ângulo de guinada. No computador embarcado é realizado a correção da declinação magnética da Terra (HONEYWELL,).
- Verificar o enlace de comunicação. Caso haja perda de sinal os motores serão desligados.

São responsabilidades do computador base:

- Processar os dados enviados pelo computador embarcado, como dados do GPS e da IMU.
- Executar a comunidade MOOS da missão e verificar o *status* de cada aplicativo.
- Gerar os novos *setpoints* de navegação através do aplicativo **pHelm**.
- Converter os *setpoints* de navegação em valores do motor. Estes valores serão enviados ao computador embarcado.
- Gerar uma visualização da trajetória do barco.

- Realizar um *log* dos dados da missão.
- Simular o modelo do barco e configurar as missões.
- Gerenciar a interface com o rádio-modem e enviar dados de navegação para o computador embarcado.

4.2 O *software* embarcado

Esta seção traz detalhes da implementação do *software* embarcado.

O *software* embarcado foi desenvolvido para Arduino utilizando o Arduino IDE 1.0.5, as bibliotecas padrões da IDE e bibliotecas desenvolvidas pelo autor. A linguagem utilizada pelo compilador do Arduino é uma variação de C. As bibliotecas são desenvolvidas em C++, com suporte a programação orientada a objetos.

A figura 4.2 representa o fluxograma do *software* embarcado. Uma descrição em detalhes de cada trecho do *software* é apresentado a seguir.

O código fonte completo pode ser verificado no *website*: `https://github.com/davidissamattos/Barco_SoftwareEmbarcado`.

4.2.1 Bibliotecas utilizadas

O *software* embarcado utiliza as seguintes bibliotecas listadas abaixo. Exemplos de aplicações com muitas destas bibliotecas podem ser encontradas em (MARGOLIS, 2011).

- **Wire.h**: Esta biblioteca é utilizada para comunicação I^2C com o Arduino. Nela já são implementadas rotinas para utilização do Arduino tanto como *slave* quanto

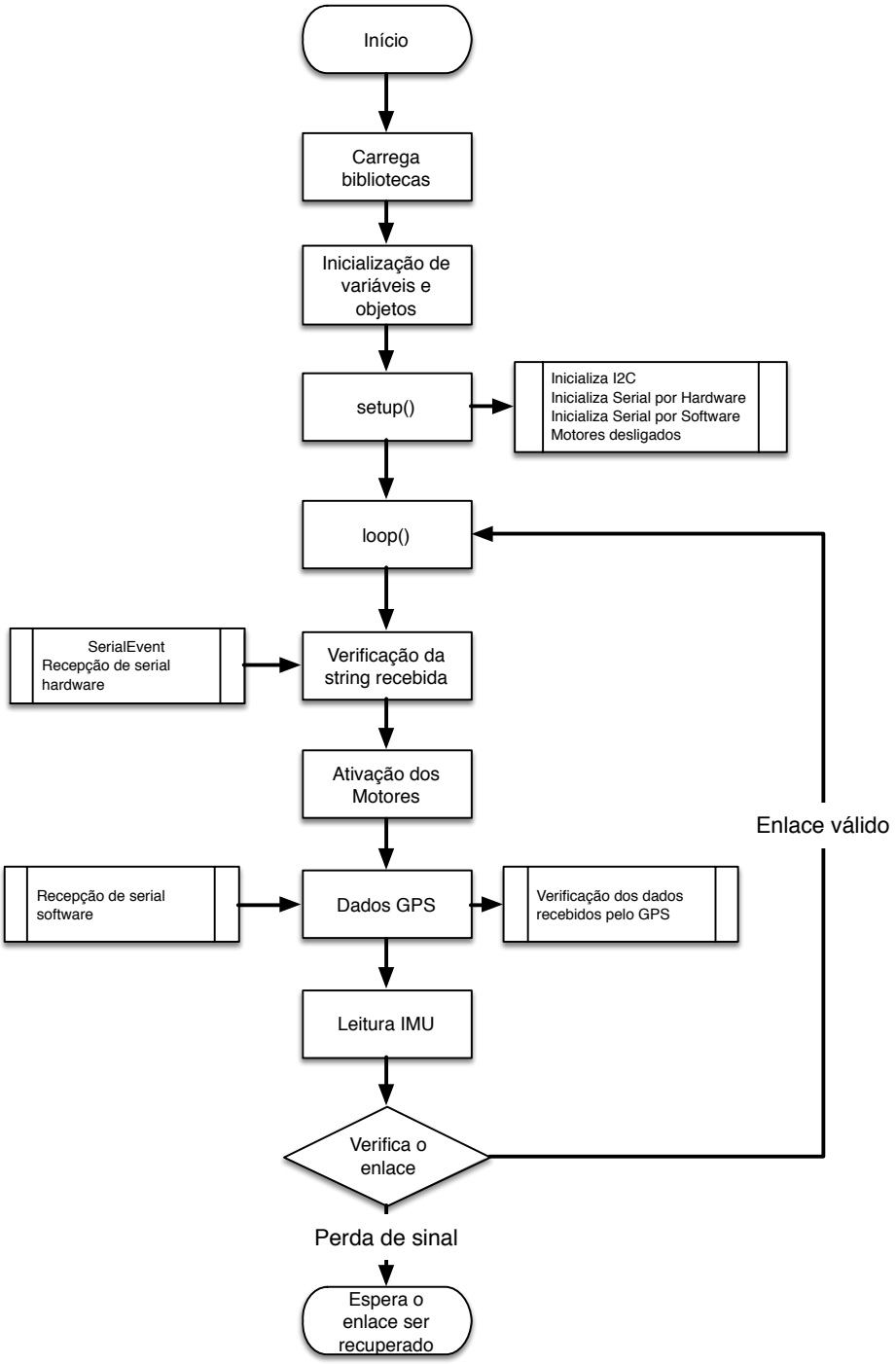

FIGURA 4.2 – Fluxograma do *software* embarcado no Arduino

master. Nela são implementadas rotinas de escrita e de leitura. Mais informações podem ser obtidas em: http://arduino.cc/en/reference/wire

- **SoftwareSerial.h**: Esta biblioteca implementa a utilização de uma serial (UART) por *software*, utilizando outros pinos IO, que não os pinos 0 e 1 (que implementam uma serial por *hardware*). Esta biblioteca é útil quando deseja-se comunicar por serial com mais de um dispositivo no Arduino UNO (este possui apenas uma interface serial por *hardware*). Esta biblioteca possui algumas limitações com taxas de transmissão e tempo de latência (efeitos não observados neste projeto), quando necessita-se de fluxos simultâneos de dados, uma implementação alternativa é proposta na biblioteca AltSoftSerial. Mais informações podem ser obtidas em: http://arduino.cc/en/Reference/softwareSerial.

- **TinyGPS.h** Esta biblioteca foi proposta por Mikal Hart. Esta biblioteca gerencia a comunicação com um dispositivo do tipo GPS que utiliza o protocolo NMEA 0183. As funções da bilbioteca verificam a validade das *strings* através de seu *checksum*, eliminando as sentenças que não são utilizadas. Por meio dessa biblioteca é possível obter dados de localização, velocidade, altitude entre outras. Possui também suporte a operações com ponto flutuante. Mais informações podem ser obtidas em http://arduiniana.org/libraries/tinygps/.

- **stdlib.h** Biblioteca padrão dos dispositivos AVR. Esta biblioteca é utilizada para conversão entre dados do tipo *float* para *strings*. Esta biblioteca é utilizada devido a um problema de conversão ainda não resolvido na função *FloatToString* da IDE. Para mais informações http://playground.arduino.cc/Main/FloatToString

- **GY80IMU.h** Biblioteca de desenvolvimento do autor para gerenciar os dados ob-

tidos pela IMU (leitura de dados e configuração dos sensores).

4.2.2 Troca de informação entre o computador embarcado e o computador base

A comunicação entre os computadores ocorre através de um enlace de rádio. As informações trocadas são enviadas em *strings* padronizadas, estas são processadas por cada computador.

O computador base envia ao computador embarcado uma *string* com o formato:

"M1=x,M2=y"

Onde x e y são caracteres que podem assumir três valores, F, T ou D. Onde F representa o estado do motor ligado para frente, T ligado para trás e D desligado. O *software* embarcado verifica a sentença recebida, se ela estiver neste padrão ele atualiza os estados do motor, caso contrário os estados permanecem o estado anterior e é enviado para o computador base a seguinte sentença: "*TransmittingError*". Assim o computador base pode utilizar esta informação para verificar a taxa de sentenças corretas enviadas.

O computador embarcado é responsável de enviar os dados obtidos nos sensores embarcados e transmiti-los ao computador base.

As sentenças enviadas pelo computador embarcado podem ser de dois tipos, uma enviando dados da IMU, outra enviando dados do GPS.

Para a IMU:

"IMU_HEADING=valor, IMU_AX=valor,IMU_AY=valor,IMU_AZ=valor,

IMU_WX=valor,IMU_WY=valor,IMU_WZ=valor,IMU_AZ=valor,

IMU_CX=valor,IMU_CY=valor,IMU_CZ=valor;"[1]

Os valores IMU_AX, IMU_AY e IMU_AZ representam os valores lidos pelo acelerômetro em cada um dos eixos. Os valores IMU_WX, IMU_WY e IMU_WZ representam os valores lidos pelo giroscópio em cada um dos eixos. Os valores IMU_CX, IMU_CY e IMU_CZ representam os valores lidos pelo magnetômetro em cada um dos eixos.

IMU_HEADING representa o ângulo de guinada em relação ao norte real, compensado a declinação magnética no local. Essa inclinação é dada pela seguinte expressão:

$$\psi_{\text{mag}} = \text{atan2}\left(\frac{\psi_y}{\psi_x}\right)$$

$$\psi_{\text{norte}} = \psi_{\text{mag}} + (-22^o 21')$$

Onde ψ_y e ψ_x são os valores lidos pelo magnetômetro nos eixos y e x, ψ_{mag} é ângulo de guinada do barco em relação ao norte magnético e ψ_{norte} é o ângulo de guinada compensada a declinação magnética do local(HONEYWELL,). A declinação de $-22^o 21'$ foir obtida através do *website*: http://magnetic-declination.com).

A *string* enviada pelo GPS é dada como:

"GPS_LAT=valor,GPS_LON=valor,GPS_TIME=valor"

Onde os valores de latitude e longitude são enviados em graus decimais (e não com minutos e segundos).

O número de casas decimais em cada um dos valores pode alterar, o *software* no computador base fica responsável por separar os valores correspondentes a cada variável.

[1]Observa-se que, inicialmente, como não há a necessidade das outras variáveis da IMU (não é implementado uma fusão sensorial) elas não estão sendo processadas nem enviadas.

4.3 Os aplicativos MOOS

Esta seção traz informações específicas dos aplicativos MOOS desenvolvidos para a simulação do barco utilizando o MOOS-IvP e para a integração/comunicação entre o computador embarcado e o computador base.

O computador base utilizado para desenvolver os aplicativos e utiliza-los foi um MacbookPro de 13 polegadas, final de 2013. O computador utiliza o sistema operacional Mac OS X 10.9.5.

O apêndice A traz informações de como desenvolver um aplicativo MOOS. No total foram desenvolvidos quatro aplicativos MOOS:

- **pControleBarco**: aplicativo que lê os *setpoints* de velocidade e de ângulo de guinada e através da tabela de acionamentos, Tabela 2.1, determina os estados futuros para os motores M1 e M2. Este aplicativo é utilizado tanto nas simulações quanto na implementação experimental.

- **iSerial**: Implementa a comunicação serial através da USB, utilizando o padrão POSIX, no MAC OS X 10.9.5. Com poucas modificações este aplicativo pode ser adaptado para outros sistemas que utilizam POSIX (como distribuições de Linux). Este aplicativo é específico para implementação experimental.

- **uBarco**: Implementa o modelo do barco descrito na Seção 2.2. Este aplicativo é específico para simulações.

- **myGeodesy**: Este aplicativo realiza a conversão entre coordenadas globais (dados do GPS) para coordenadas locais (que serão consumidas pelo **pHelm**). Este aplicativo é específico para implementação experimental.

Os códigos fonte de cada aplicativo é disponibilizado ASASDASDASDASDSASDADS.

4.3.1 pControleBarco

O aplicativo **pControleBarco** utiliza os dados de navegação publicados pelo **pHelm**, **uBarco** ou **myGeodesy** e implementa a tabela de acionamentos (Tabela 2.1).

Assim o aplicativo assina as variáveis de posição do barco "NAV_X", "NAV_Y", o ângulo de guinada "NAV_HEADING", a velocidade "NAV_SPEED"e os *setpoints* desejados "DESIRED_SPEED", "DESIRED_HEADING". O aplicativo publica as variáveis "M1"e "M2"(outras variáveis geradas não são consumidas por nenhum outro aplicativo e servem apenas para verificação e *debug*).

Para consultar os dados da Tabela 2.1, o aplicativo calcula o desvio do ângulo de guinada e a diferença de distância através do equacionamento abaixo. O tempo utilizado na integração é o tempo percorrido entre duas chamadas do aplicativo pelo *software* MOOS-IvP.

Assim, seja a velocidade desejada representada por v^d, a velocidade atual v, o ângulo de guinada desejado ψ^d e o atual ψ e as posições desejadas representadas por x^d, y^d:

$$
\begin{aligned}
v_x^d &= \sin(\psi^d) \cdot v^d \\
v_y^d &= \cos(\psi^d) \cdot v^d \\
x^d &= v_x^d \cdot \Delta t \\
y^d &= v_Y^d \cdot \Delta t \\
\Delta d &= \sqrt{(x^d)^2 + (y^d)^2} \\
\Delta\psi &= \psi^d - \psi
\end{aligned}
$$

Assim com os valores de Δd e $\Delta\psi$ o aplicativo consulta a tabela de acionamentos, Tabela 2.1, determina as novas variáveis de estado dos motores "M1"e "M2"e publica ambas no MOOSDB.

O arquivo de configuração MOOS do Código 4.1 determina apenas o nome das variáveis que serão utilizadas.

Código 4.1 – Exemplo de arquivo de configuração MOOS do aplicativo pControleBarco

```
1  ProcessConfig = pControleBarco
2  {
3      AppTick    = 4
4      CommsTick = 4
5      NAV_X = NAV_X
6      NAV_Y = NAV_Y
7      NAV_SPEED = NAV_SPEED
8      NAV_HEADING = NAV_HEADING
9      DESIRED_HEADING = DESIRED_HEADING
10     DESIRED_SPEED = DESIRED_SPEED
11
12 }
```

O código fonte completo pode ser visto em `ftp://labattmot.ele.ita.br/ele/david/My_Publications/TG/Codigos/moos-ivp-extend/src/`.

4.3.2 iSerial

O aplicativo **iSerial** utiliza duas classes. A primeira classe gerencia a interface com a porta de serial virtual criada na USB. A segunda classe implementa os métodos necessários para criação do aplicativo MOOS

O sistema operacional Mac OS X 9.5 utiliza o padrão POSIX, (IEEE,). A classe implementada tem como fundamento os códigos disponibilizados nos *websites* (SWEET,) e (APPLE,). Vale ressaltar que essas não são as únicas formas de se acessar a porta serial em um computador Mac (outras formas utilizando Objective-C e IOKit também são possíveis).

A classe que implementa os métodos necessários para criação de um aplicativo MOOS instancia um objeto do tipo porta serial (utilizando a classe desenvolvida). Este objeto é responsável por estabelecer a comunicação, transmitir e receber os dados da porta serial virtual.

O aplicativo assina para receber atualizações das variáveis ”M1”e ”M2”. Ao receber uma nova atualização ele gera a sentença ”M1=x,M2=y”, onde x e y podem ser ”T”,”F”e ”D”, como descrito na Seção 4.2, e envia os dados para o rádio modem.

Em seguida o *software* processa os dados recebidos (Seção 4.2), separando as sentenças nas variáveis ”IMU_HEADING=valor”, ”NAV_LAT=valor”, ”NAV_LON=valor”e ”NAV_TIME=valor”que serão publicadas.[2]

A Figura 4.3 representa a o fluxograma do aplicativo MOOS **iSerial**

Este aplicativo deve ser chamado pelo *software* MOOS-IvP com uma frequência alta, já que os dados que vem pela serial são enviados continuamente. Se o aplicativo for chamado a uma frequência baixa o **buffer** de recepção dos dados estoura e aparecerá lixo nas variáveis. Uma alternativa que não foi implementada é o *software* embarcado enviar dados ao computador base somente quando este requerir.

O arquivo de configuração pode ser visto no Código 4.2. Nele utilizam-se as configurações de *BaudRate*, seleção da porta serial e a opção *options* representa se a leitura das sentenças é feita com *carriage return*, como selecionado na opção ”Arduino”. As opções M1 e M2 representam o nome da variável dos motores que será assinada (no caso é o mesmo nome).

Código 4.2 – Exemplo de arquivo de configuração MOOS do aplicativo iSerial

```
1 ProcessConfig = iSerial
```

[2]Observa-se que, inicialmente, como não há a necessidade das outras variáveis da IMU (não é implementado uma fusão sensorial) elas não estão sendo processadas nem enviadas.

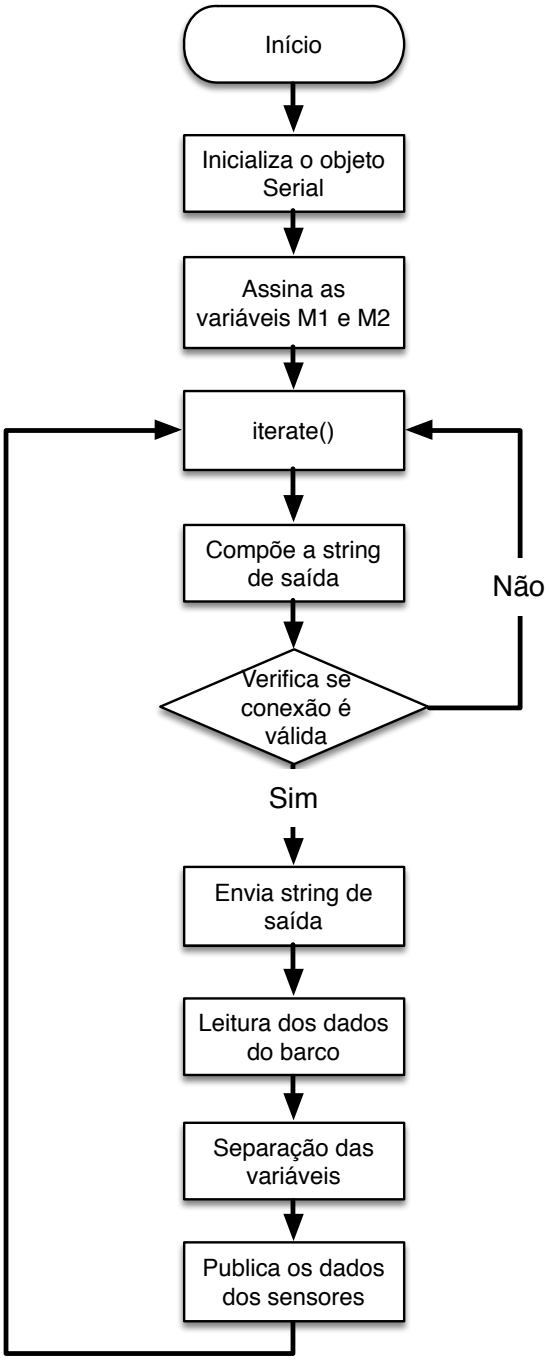

FIGURA 4.3 – Fluxograma do aplicativo MOOS iSerial

```
2  {
3    //Utiliza-se uma frequência alta para nao acumular
4    //dados no buffer da serial virtual
5     AppTick    = 0
6     CommsTick = 0
7
8     M1=M1
9     M2=M2
10    SERIAL_PORT = \dev\tty.usbmodemfd121
11    BAUDRATE = 9600
12    OPTIONS = ARDUINO
13 }
```

O nome da serial utilizada pode ser verificada utilizando um Terminal Bash com o seguinte código:

Código 4.3 – Verificar porta serial através de comandos bash

```
1 #!/bin/bash
2 ls /dev/tty.*
```

O código fonte completo de ambas as classes pode ser visto em ftp://labattmot.ele.ita.br/ele/david/My_Publications/TG/Codigos/moos-ivp-extend/src/.

4.3.3 uBarco

O aplicativo **uBarco** utiliza o modelo matemático do barco apresentado na seção 2.2 para simular o funcionamento do barco no *software* MOOS-IvP.

O método de integração utilizado foi o método de Euler. Considera-se que o barco inicialmente encontra-se em repouso, na origem.

O aplicativo assina as variáveis de posição "NAV_X", "NAV_Y"e "NAV_HEADING", as variáveis de velocidade (linear e angular) "NAV_SPEED"e "NAV_TURN"e as variáveis de *status* do motor "M1"e "M2". Assim o aplicativo implementa o modelo, integra as equações diferenciais e publica as variáveis NAV_X", "NAV_Y", "NAV_HEADING", "NAV_SPEED"e

"NAV_TURN". Lembrando que o ângulo de guinada ("NAV_HEADING") é $90 - \psi$ do modelo apresentado na seção 2.2.

O arquivo de configuração MOOS é apresentado no Código 4.4, onde as variáveis ajustam apenas os nomes das variáveis que serão utilizadas.

Código 4.4 – Exemplo de arquivo de configuração MOOS do aplicativo uBarco

```
1  ProcessConfig = uBarco
2  {
3      AppTick    = 1
4      CommsTick = 1
5      M1 = M1
6      M2 = M2
7      NAV_X = NAV_X
8      NAV_Y = NAV_Y
9      NAV_SPEED = NAV_SPEED
10     NAV_HEADING = NAV_HEADING
11 }
```

O código fonte completo pode ser visto em `ftp://labattmot.ele.ita.br/ele/david/My_Publications/TG/Codigos/moos-ivp-extend/src/`.

4.3.4 myGeodesy

Esse aplicativo é responsável por realizar a conversão entre coordenadas globais do GPS para coordenadas locais.

O *software* MOOS-IvP já possui um módulo que realiza as conversões necessárias, o MOOSGeodesy, no entanto, a falta de documentação em relação a este módulo específico dificulta sua utilização.

O modelo adotado para realizar a conversão é descrito em (FARRELL; BARTH, 1999). A conversão baseia-se no modelo de elipsóide WGS-84.

Sejam os parâmetros do elipsóide no sistema ECEF (*Earth-Centered Earth-Fixed*) geodésico:

- Semi-eixo maior: $a = 6378137$ m
- Semi-eixo menor: $b = 6356752.314245$ m
- Latitude: λ
- Longitude: ϕ
- Altitude em relação a superfície: h
- Achatamento: $f = \frac{a-b}{a}$
- Excentricidade: $e = \sqrt{f(2-f)}$
- Comprimento da normal, da superfície ao cruzamento com o semi eixo menor (z): $N(\lambda) = \frac{a}{\sqrt{1-e^2\sin^2(\lambda)}}$

A conversão para o sistema ECEF retangular é feita a partir das Equações 4.1,4.2 e 4.3:

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

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

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

A conversão para o sistema local é feita a partir de duas rotações, representadas matricialmente por R_{e2n}, onde R_{e2n} é dada pela Equação 4.4:

$$R_{e2n} = \begin{pmatrix} -sin(\lambda) & 0 & \cos(\lambda) \\ 0 & 1 & 0 \\ -\cos(\lambda) & 0 & -\sin(\lambda) \end{pmatrix} \cdot \begin{pmatrix} -\cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

Assim a conversão é dada pela Equação 4.5. Nessa equação x_0, y_0 e z_0 são as coordenadas da origem do sistema local em coordenadas ECEF retangulares.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{Local}} = R_{e2n} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{ECEF ret}} - R_{e2n} \cdot \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}_{\text{ECEF ret}} \tag{4.5}$$

O aplicativo MOOS implementa a Equação 4.5, assinando as variáveis de coordenadas globais ”NAV_LAT”e ”NAV_LON”e publica as variáveis de coordenadas locais ”NAV_X”e ”NAV_Y”. Assim este aplicativo é necessário somente na implementação experimental, já que o modelo **uBarco** fornece as variáveis de posição em coordenadas locais.

O arquivo de configuração deste aplicativo pode ser visto no Código 4.5. Os parâmetros de configuração referem-se a origem do sistema em coordenadas globais geodésicas. A altitude utilizada neste aplicativo é fixa, e pode ser obtida tanto por medições de GPS e barômetros quanto de um *website* como o http://www.daftlogic.com/sandbox-google-maps-find-altitude.htm.

Código 4.5 – Exemplo de arquivo de configuração MOOS do aplicativo myGeodesy

```
1 ProcessConfig = pmyGeodesy
2 {
3     AppTick    = 4
4     CommsTick = 4
```

```
5     LAT_ORIGIN = -23.209063 //em graus
6     LON_ORIGIN = -45.872733 //em graus
7     ALTITUDE = 611 //em m
8  }
```

O código fonte completo pode ser visto em ftp://labattmot.ele.ita.br/ele/david/My_Publications/TG/Codigos/moos-ivp-extend/src/.

4.4 Resultados simulados e experimentais

Esta seção traz os resultados obtidos através da integração entre o *software* MOOS-IvP com o barco, como mostrado nas seções anteriores deste capítulo.

As subseções 4.4.1 e 4.4.2 trazem os resultados para os dados simulados e experimentais, respectivamente. Em cada subseção é apresentado a configuração da comunidade MOOS e dos comportamentos para a obtenção dos resultados.

A missão realiza tanto em simulação quanto experimentalmente é a missão **Alpha**, esta missão consiste em realizar uma trajetória passando por *waypoints* pré definidos. Os arquivos de configuração **.moos* e **.bhv* que serão utilizados estão disponíveis através do *website* https://github.com/davidissamattos/TG_Missions/tree/master/Alpha.

4.4.1 Resultados simulados

Esta subseção traz resultados simulados obtidos com o *software* MOOS-IvP. A Figura 4.4 mostra a comunidade MOOS utilizada para simular o barco. Em seguida é apresentado uma breve discussão dos aplicativos MOOS utilizados.

O arquivo de configuração MOOS utilizado foi o *alpha.moos*. Os aplicativos utilizados são:

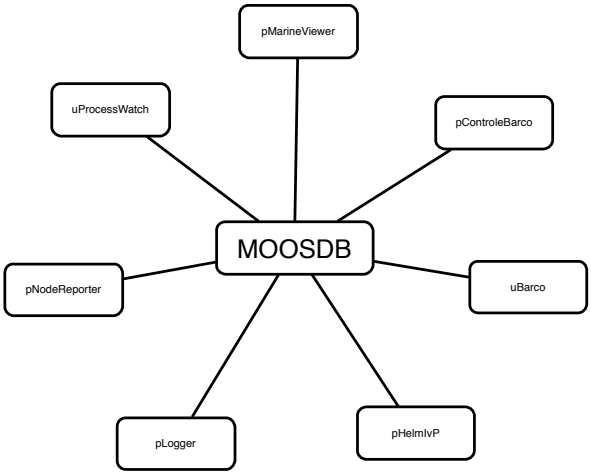

FIGURA 4.4 – Comunidade MOOS utilizada para a simulação.

- **pHelmIvP**: Aplicativo que realiza a tomada de decisões de navegação. Descrito com mais detalhes em 3.2.

- **uBarco**: Aplicativo que simula o comportamento do barco. Descrito com mais detalhes em 4.3.3.

- **pControleBarco**: Aplicativo que realiza o controle dos motores do barco. Descrito com mais detalhes em 4.3.1

- **pMarineViewer**: Aplicativo que permite a visualização da navegação do barco em uma interface gráfica, permite também interagir com mouse e alterar o estado dos nós de missão. Descrito com mais detalhes em (BENJAMIN *et al.*, 2013).

- **uProcessWatch**: Aplicativo que verifica a conexão entre os demais aplicativos com o MOOSDB avisando caso algum aplicativo seja terminado ou desconectado. Descrito com mais detalhes em (BENJAMIN, 2010).

- **pNodeReporter**: Aplicativo que encapsula as informações de navegação e posta em uma *string*. Outros aplicativos com o **pHelmIvP** e o **pMarineViewer** consomem estas informações. Descrito com mais detalhes em (BENJAMIN, 2010).

- **pLogger**: Aplicativo que realiza um *log* em arquivo da missão. Os dados são depois processados com uso dos programas presentes na *Alog-Toolbox*. Entre essas funções um dos programas presentes de uso comum é o `aloggrep`, que seleciona em um novo arquivo texto somente as informações que o usuário tem interesse. Essa *toolbox* é descrita com mais detalhes em (BENJAMIN, 2010).

A missão foi configurada para seguir pontos (*Waypoint behavior*) e está configurada no arquivo *alpha.bhv*. Neste arquivo é definido os pontos em que se deseja percorrer. Os pontos são definidos em coordenadas locais, como o aplicativo **uBarco** já fornece as coordenadas locais não é necessário a conversão. A origem das coordenas é considerada em:

$$\text{Latitude\_origem:} \quad -23.208408$$

$$\text{Longitude\_origem:} \quad -45.874717$$

A Figura 4.5 representa o simulador *pMarineViewer* executando a missão **Alpha** dada nos arquivos de configuração *alpha.moos* e *alpha.bhv*.

Um parâmetro importante na realização de uma missão de seguir *waypoints* é o raio de captura. Esse parâmetro define se o barco atingiu ou não o *waypoint*. As Figuras 4.6 e 4.7 representam a mesma missão simulada para dois raios de captura diferentes. O primeiro de 1 m e o segundo de 5 m. Observe que a trajetória obtida com um raio de captura maior (Figura 4.7) é mais suave que uma com raio de captura pequeno.

As aberturas grandes que aparecem ao realizar as curvas são devidas a duas causas, a primeira ao controlador do barco e a segunda deve-se as configurações do **pHelmIvP**.

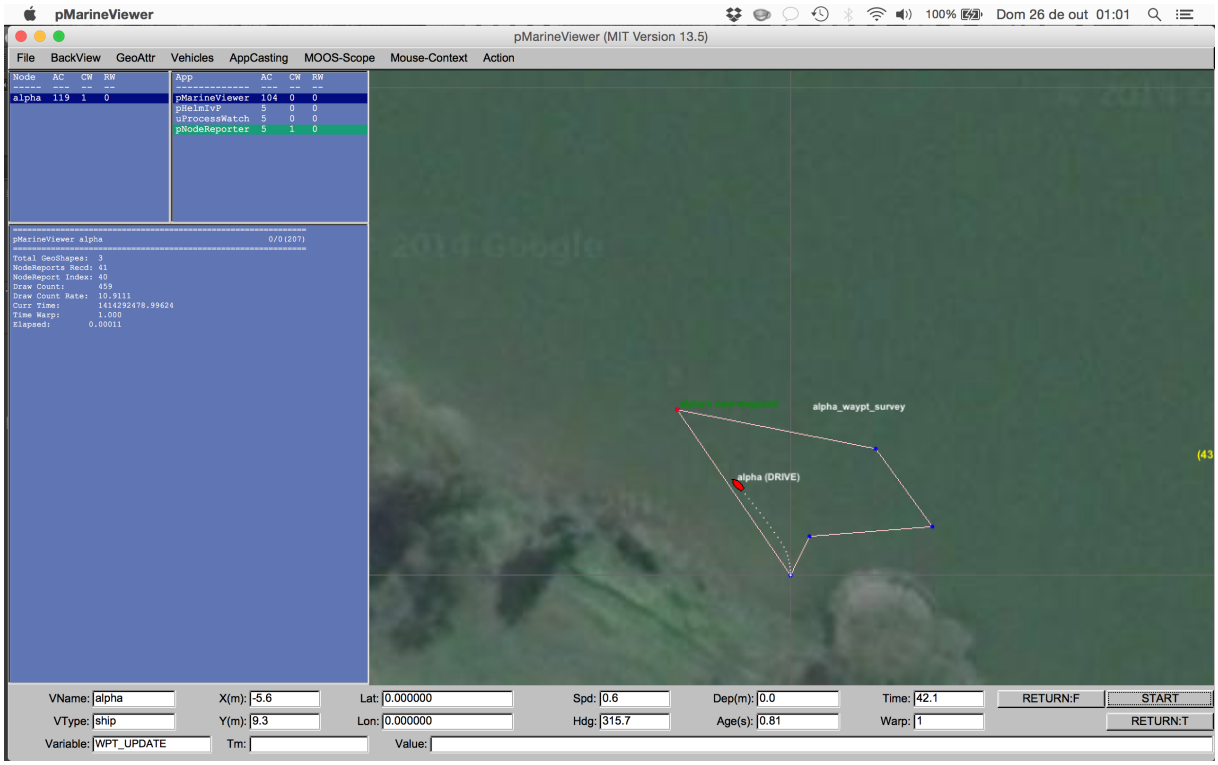

FIGURA 4.5 – Imagem do simulador **pMarineViewer** executando a missao Alpha.

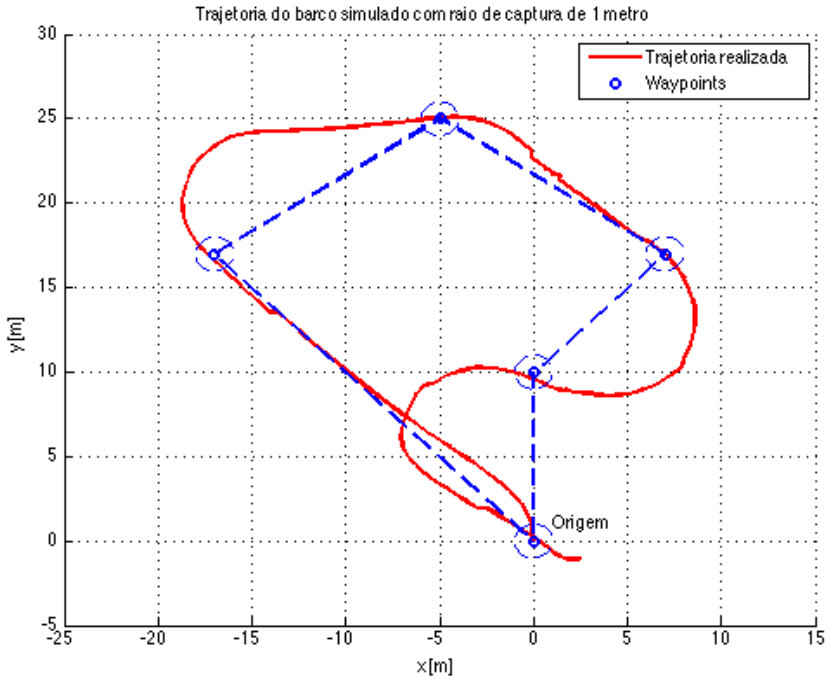

FIGURA 4.6 – Missão Alpha simulada utilizando raio de captura de 1 m

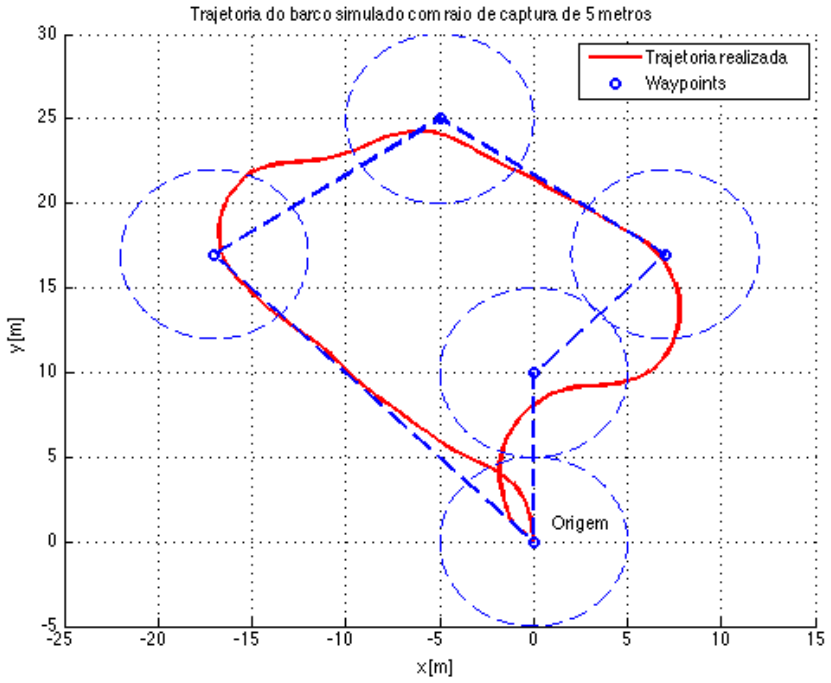

FIGURA 4.7 – Missão Alpha simulada utilizando raio de captura de 5 m

O controlador do barco trabalha sempre com o erro de distância e erro do ângulo de guinada. Quando define-se uma velocidade mínima de navegação o barco realiza o controle baseado nesta velocidade. Se o alvo encontra-se a uma distância superior a 3 m (vide Tabela 2.1) do barco o barco sempre realizará as curvas indo para frente (e não irá girar em torno do próprio erro). Uma mudança no controlador do barco, para que se o erro de guinada for superior a determinado valor faz com que ele gire em torno do próprio erro para corrigir, minimizaria esta curva.

Uma segunda forma de minimizar este erro é ao invés de utilizar apenas comportamentos de *waypoints*, onde o que interessa é atingir os pontos pré definidos e não a trajetória escolhida para isso. Se fosse utilizado outros tipos de comportamento, como seguir uma linha entre ou outra trajetória qualquer, o **pHelmIvP** geraria pontos mais próximos do barco (o que levaria ele a girar em torno do próprio eixo), para minimizar o erro de trajetória.

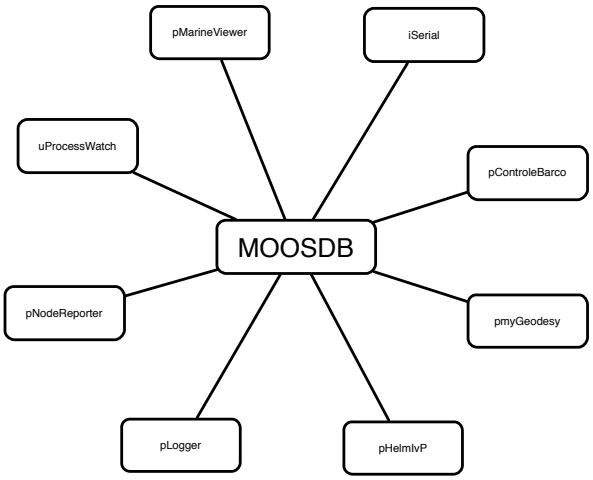

FIGURA 4.8 – Comunidade MOOS utilizada para o controle do barco experimentalmente.

4.4.2 Resultados experimentais

Esta subseção traz resultados experimentais obtidos com o *software* MOOS-IvP. A Figura 4.8 mostra a comunidade MOOS utilizada para controlar o barco.

As modificações desta comunidade em relação a comunidade da Figura 4.4 foram ao acrescentar o aplicativo **pmyGeodesy** para realizar as conversões entre coordenadas globais para coordenadas locais e ao substituir o aplicativo **uBarco** pelo aplicativo *iSerial*, que estará recebendo as informações de navegação do barco.

Para realizar o experimento o barco foi levado para a borda do lago do ITA, o rádio-modem do computador de base foi posicionado a uma distância de aproximadamente 10m da borda do lago e alimentado por uma bateria de 13.5V. No total duas missões de *waypoints* foram realizadas.

Pontos para a primeira missão em coordenadas locais(x,y):

$$10,10 : 10,20 : 0,0$$

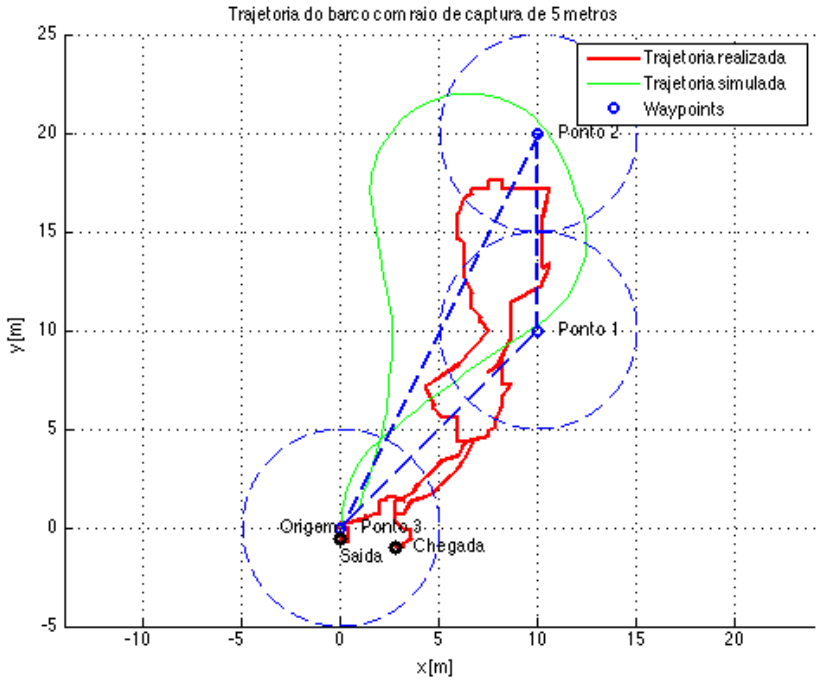

FIGURA 4.9 – Primeira missão realizada.

Pontos para a segunda missão em coordenadas locais(x,y):

$$20,20 : 20,30 : 0,0$$

As Figuras 4.9 e 4.10 mostram um gráfico comparativo entre o resultado simulado e o resultado experimental.

Observa-se que o barco em ambos os casos foi capaz de atingir os pontos desejados e retornar a margem do lago. No caso da segunda missão o barco não retorna exatamente ao raio de captura da origem, pois neste caso a margem do lago possui baixa profundidade e o barco ficou preso em pedras do fundo no ponto de chegada.

Observa-se também que os resultados experimentais estão sujeitos a erros que não estão presentes nos resultados simulados, como por exemplo a presença de vento, diferença de potência entre os motores e incerteza na medida de posição do GPS. Devido a baixa

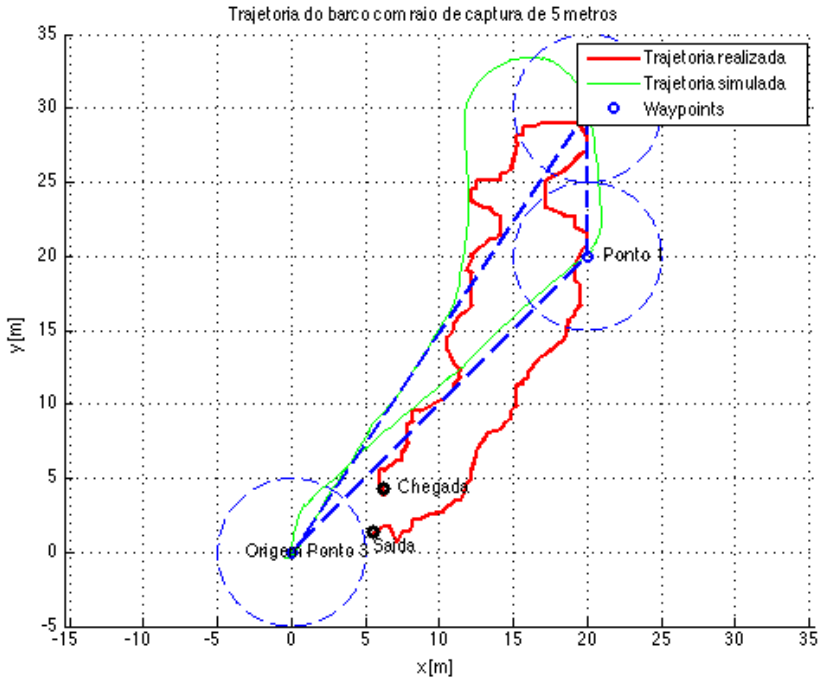

FIGURA 4.10 – Segunda missão realizada.

velocidade do barco e sua pequena massa a presença de ventos pode alterar a trajetória de forma significativa. Foi observado também em bancada uma diferença de velocidade das pás do barco em testes em bancada, assim esta diferença pode alterar a trajetória do barco em comparação com o resultado simulado.

Uma fonte de erro significativa na localização do barco é dado pela precisão do sistema GPS. O GPS utilizado não era auxiliado por nenhuma outra forma de localização como D-GPS, A-GPS ou fusão entre o GPS e a IMU, assim percebe-se uma incerteza muito grande em sua localização. A Figura 4.11 representa os dados de GPS obtidos com o sistema em completo repouso. Observa-se que a incerteza na posição pode superar 2m.

Utilizando uma IMU pode-se corrigir os erros de localização do GPS, como descrito em (SANTOS, 2011) e (GREWAL; WEILL; P., 2007). Assim as medidas de localização tornam-se mais precisas.

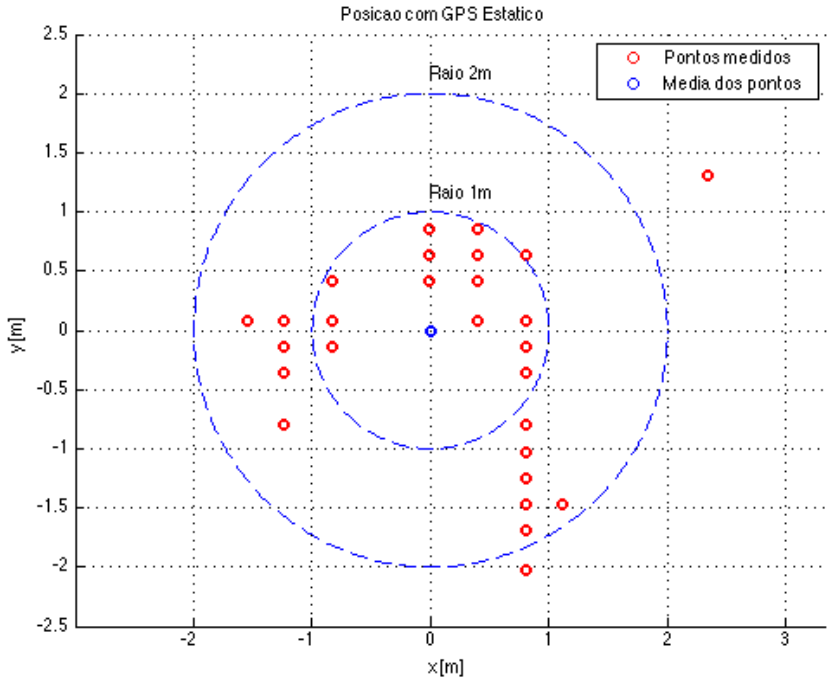

FIGURA 4.11 – Dados de posição do GPS para o sistema em repouso

Outras diferenças entre as trajetórias simuladas e a trajetórias experimentais podem ser atribuídas a uma diferença entre o modelo e o sistema real. Na implementação do modelo no aplicativo **uBarco**, foi utilizado uma integral limitada para a velocidade de angular do barco. Este limite pode estar com valor inferior ao limite do barco. O limite foi utilizado devido ao modelo divergir na sua ausência. No aplicativo **uBarco** foi utilizado o metodo de integração de Euler de passo fixo e igual a 1s. A utilização de um método de integração melhor com um passo menor pode resolver este problema, em (SANTOS, 2011) não foi utilizado nenhum batente para a velocidade angular e o modelo não apresentou esta divergência. No entanto, essas considerações não foram investigadas em mais detalhes.

Durante a execução da missão com o barco no lago, foi verificado que em poucas ocasiões o barco estava com ambos os motores ligados para frente (M1=1 e M2=1). De modo geral o barco estava com apenas um dos motores ligados (M1=1 e M2=0 ou M1=0 e M2=1), ou seja, o barco apresentava uma trajetória em S. Assim verifica-se a necessidade

de investigar uma outra tabela de acionamentos (talvez com intervalo de guinada maior ao $-5^o < \delta\psi_{\text{Guinada}} < 5^o$ utilizado). Também deve-se analizar o atraso entre o barco enviar sua posição e ângulo de guinada e ele receber os novos estados do motor, este atraso pode também estar contribuindo para a trajetória em S do barco.

5 Transmissão e processamento de imagens

5.1 Introdução

A utilização de câmeras para auxílio em navegação é bastante utilizada, seja para localização (XIA; YANG; YANG, 2006), identificação de objetos (SIDELEAU; EICKSEDT, 2010) e (SPEARS; WEST; COLLINS, 2013), para monitoramento do ambiente ou para assumir o controle manual do veículo.

A arquitetura do barco e seu computador embarcado (Seção 2.3) não prevê a utilização de uma câmera. O objetivo deste capítulo é implementar uma forma de transmissão de imagem integrada ao *software* MOOS-IvP utilizando um computador de baixo custo. A transmissão da imagem é realizada por meio de uma rede Wifi (IEEE 802.11) utilizando o protocolo UDP.

O sistema implementado também pode ser utilizado para transmitir todos os dados de comunicação entre o computador base e os computadores embarcados, caso a navegação ocorra em uma área que possua a cobertura necessária. No entanto, essa funcionalidade não foi implementada. Assim o sistema de transmissão e processamento de imagem é

FIGURA 5.1 – Computador Raspberry Pi Model B. Fonte: http://www.raspberrypi.org

independente do outro computador embarcado.

5.2 *Hardware* utilizado

O *hardware* utilizado para esta transmissão tem como objetivo manter os custos de implementação baixos, mas com características de fácil substituição, caso deseja-se melhorar o sistema e os componentes envolvidos.

Assim foi procurado de um computador de baixo custo, com um sistema operacional bastante utilizado, que forneça os *drivers* necessários para utilização de câmeras e conexão Wifi. O computador utilizado foi uma Raspberry Pi Model B, que utiliza uma distribuição de Linux Raspian (baseada no Debian). A Figura 5.1 representa o computador escolhido.

A câmera utilizada foi uma *Webcam* Logitech C525. Esta câmera possui baixo custo, *drivers* para Raspian e baixo consumo, podendo ser conectada a USB da Raspberry Pi sem utilização de um USB Hub energizado. A Figura 5.2 representa o modelo escolhido.

O Raspberry Pi possui uma interface Ethernet, no entanto não possui um sistema de transmissão Wifi. Para dar capacidade a Raspberry Pi de interfacear com roteadores

FIGURA 5.2 – *Webcam* Logitech C525. Fonte: http://www.logitech.com

FIGURA 5.3 – Wifi *dongle* TP-LINK TLWN725N. Fonte: http://www.tp-link.com.br

Wifi (IEEE 802.11) é utilizado um Wifi *dongle* USB. O modelo utilizado foi um TP-LINK TLWN725N. O adaptador possui baixo consumo, podendo ser utilizado simultaneamente com a *webcam*. A Figura 5.3 representa o modelo escolhido.

Um roteador TP-LINK foi utilizado para criar uma rede local sem-fio entre o computador base e a Raspberry Pi.

5.3 Transmissão de imagem com o MOOS-IvP

A transmissão de imagens utilizando o MOOS-IvP pode ser separada em duas partes, a primeira é a configuração da transmissão de informação utilizando aplicativos já desen-

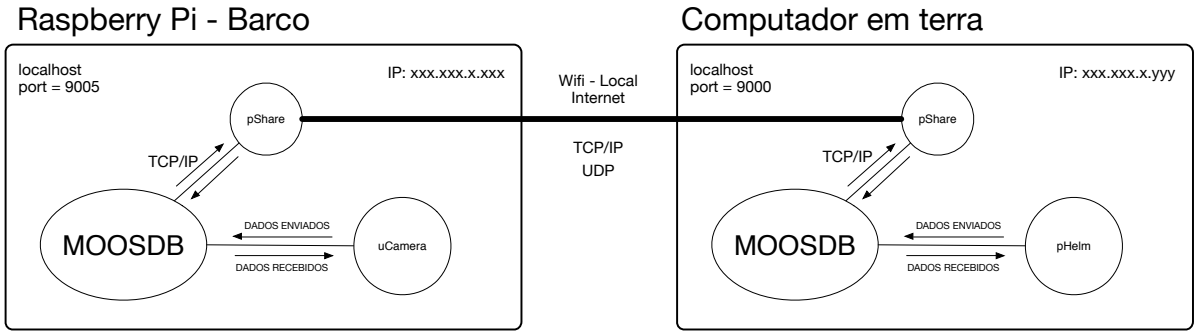

FIGURA 5.4 – Topologia para transmissao de imagem

volvidos, em particular o **pMOOSBridge** e o **pShare**. A segunda parte é a integração do conjunto de bibliotecas para *openCV* para C++ com *software* MOOS-IvP. Esta biblioteca permite interfacear o *software* com a câmera, para obtenção de imagens e vídeos e o processamento destas para obtenção de informações de navegação.

5.3.1 Transmitindo informações pelo pMOOSBridge

A transmissão entre os dois computadores segue a topologia apresentada na Figura 5.4. Essa topologia utiliza duas comunidades MOOS, cada uma sendo executada em computadores diferentes.

A comunidade presente no computador base continua executando os mesmos processos para controle e autonomia do barco. A comunidade presente no computador Raspberry Pi é responsável por interfacear com a *webcam* e realizar a captura das imagens. Ambas as comunidades são conectadas pelos aplicativos **pMOOSBridge** ou **pShare**, que implementa *sockets* para transmissão de informação via protocolos UDP e TCP.

Os aplicativos **pMOOSBridge** e **pShare** suportam transmissão de informação binária para transmissão de imagens ou vídeos. Informações sobre ambos aplicativos podem ser encontradas em (NEWMAN, 2009a) e (NEWMAN, 2013b). O Código 5.1 mostra como o

aplicativo é configurado para troca de informações entre duas comunidades.

Código 5.1 – Configuração MOOS do aplicativo pMOOSBridge na Raspberry Pi

```
1  ProcessConfig = pMOOSBridge_RbPi //Configuracao na comunidade da
       raspberry pi
2  {
3    UDPListen          = 9101 //informacoes sao recebidas nessa porta
4    BridgeFrequency    = 0
5
6    UDPSHARE = [IMAGEM] -> compbase @ 192.168.0.5:9201 [IMAG]
7
8    //Tipo de transmissao = [Variavel a ser transmitida] ->
9    //nome_da_comunidade_receptora @
10   //ip_do_computador:porta_recepcao [Nome da variavel]
11 }
```

Observa-se que neste arquivo configura-se as variáveis que serão transmitidas, o IP do receptor, a porta de recepção das informações, em que variável a informação será postada e em que porta será recebida informações.

5.3.2 Integrando openCV ao MOOS-IvP

A integração do openCV com o MOOS-IvP é feita com aplicativos MOOS que utilizam as bibliotecas openCV. O manual do MOOS V10 (NEWMAN, 2013a) traz um exemplo de um aplicativo que publica para o MOOSDB imagens em preto e branco. Nesta subseção é mostrado uma variação deste exemplo utilizando o *pMOOSBridge* para transmissão de uma imagem colorida entre duas comunidades MOOS.

Para que o código gerado seja compilado junto com os outros aplicativos MOOS, as bibliotecas openCV devem ser chamadas utilizando o cMake, mais informações podem ser obtidas em `http://opencv.org`. A versão de openCV utilizada nos exemplos é a 2.4.9.

Supõe-se instalado as bibliotecas openCV tanto no computador base quanto na Raspberry Pi. Para detalhes de como instalá-los vide (BRAHMBHATT, 2013), (BRADSKI; KAEHLER, 2008), (RICHARDSON; WALLACE, 2013) e `http://opencv.org`.

Os *links* a seguir traz a implementação de dois aplicativos MOOS, o primeiro para a captura e transmissão da imagem, o segundo para a vizualização da imagem. Ambos são testados em um exemplo semelhante ao aplicativo **pXRelay** (BENJAMIN *et al.*, 2013), daí a nomenclatura **pXRelayImage** e **pXRelayImageDisplay**. O primeiro possui também funções específicas para iniciar a captura e salvar a imagem obtida em um formato jpg com qualidade definida pelo usuário.

Aplicativo MOOS para captura da imagem: https://github.com/davidissamattos/pImageXRelay

Aplicativo MOOS para visualização da imagem transmitida: https://github.com/davidissamattos/pImageXRelayDisplay

Para a transmissão da imagem um arquivo de configuração como o mostrado no código 5.1 (com as devidas adaptações para os nomes das variáveis e as portas corretas) pode ser utilizado.

5.4 Processamento de imagem e localização utilizando *openCV*

Um dos objetivos de se escolher a biblioteca *openCV* para a captura e integração da câmera com o *software* MOOS-IvP é a utilização de outras funções específicas da biblioteca para realizar um processamento das imagens obtidas.

As imagens obtidas e transmitidas pelos processos descritos nas seções 5.1, 5.2, 5.3 podem ser processadas de modo a obter informações sobre o ambiente, como localização baseada em câmera, ou identificação de objetos.

Neste trabalho não foi explorado o processamento das imagens obtidas.

6 Conclusões

6.1 Resumo

Por meio deste trabalho foi possível mostrar a implementação de um sistema de navegação autônomo em um barco utilizando o *software* MOOS-IvP.

Foram abordados diferentes aspectos da implementação. Discutiu-se a implementação do *hardware* embarcado visando uma plataforma de baixo custo.

Descreveu-se as principais características e funcionalidades do *software* MOOS-IvP. Essa descrição acompanhada dos manuais de utilização do *software*, referenciados na bibliografia, permitem que um usuário tenha um primeiro contato e desenvolvimento de aplicativos para o *software* MOOS-IvP.

O capítulo 4 traz em detalhes a implementação do *software* embarcado e como integrar o *hardware* do barco com o *software* do computador base em terra. Assim são discutidos as configurações necessárias e a estrutura do *software* para se realizar a simulação do barco e os testes experimentais. Os testes experimentais comprovam as funcionalidades básicas de navegação do MOOS-IvP, os testes também ressaltam a necessidade de implementar uma forma mais precisa de lidar com o problema de localização, devido aos erros inerentes a sistemas de posicionamento global.

Por fim é discutido uma forma de implementar um sistema de comunicação por meio de rede Wifi local. Assim a transmissão de informação pode ser feita utilizando um protocolo UDP de forma a integrar missões cooperativas entre diversos veículos. Quando vários veículos realizam missões colaborativas, uma estrutura de comunicação entre diversas comunidades MOOS se faz necessário, e esta abordagem é a utilizada para este tipo de missão. Por fim este capítulo mostra uma forma de integrar as bibliotecas *openCV* a aplicativos MOOS.

Todos os códigos utilizados são disponibilizados através de um repositório *git*: `https://github.com/davidissamattos`, ou através de contato com o autor ou com o Laboratório de Máquinas Inteligentes no Instituto Tecnológico de Aeronáutica.

6.2 Trabalhos Futuros

São sugeridos para trabalhos futuros explorar melhor uma forma de melhorar a localização do barco, principalmente utilizando fusão sensorial entre a IMU e o GPS através de um filtro de Kalman. Sugere-se também a proposta de um controlador que se adeque melhor ao barco. A utilização de visão computacional traz diversas oportunidades de pesquisa, tanto na utilização da câmera para posicionamento e localização do barco (no caso de um atracamento), quanto na geração de objetivos da missão, como identificação padrões e vigilância. Por fim a investigação de missões cooperativas integra diversos dos conceitos apresentados neste trabalho e sua utilização mostra o grande potencial do *software* MOOS-IvP.

Referências Bibliográficas

APPLE, I. **Mac Developer Library - Performing Serial I/O**. Disponível em: <https://developer.apple.com/library/mac/samplecode/SerialPortSample/.html>.

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

BENJAMIN, M. MOOS-IvP Autonomy Tools Users Manual. 2010.

BENJAMIN, M. *et al.* An Overview of MOOS-IvP and a Users Guide to the IvP Helm - Release 13.5. 2013.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV.** [S.l.: s.n.], 2008. ISBN 9780596516130.

BRAHMBHATT, S. **Practical Opencv**. 1. ed. [S.l.]: Technology in Action, 2013. ISBN 9781430260790.

FARRELL, J.; BARTH, M. **The global positioning system and inertial navigation**. [S.l.: s.n.], 1999.

GLOBALSAT, T. C. GPS Engine Board - ET-332. p. 1–15.

GREWAL, S.; WEILL, R.; P., A. **Global positioning systems, inertial systems, and integration**. [S.l.: s.n.], 2007.

HONEYWELL, T. Compass heading using magnetometers.

IEEE, S. A. **Padrão POSIX**. Disponível em: <http://standards.ieee.org/develop/wg/POSIX.html>.

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

NEWMAN, P. MOOS - Mission Orientated Operating Suite. **Massachusetts Institute of Technology, Tech. Rep**, 2008.

NEWMAN, P. Bridging Communities with pMOOSBridge. p. 1–6, 2009.

NEWMAN, P. Launching Processes and Running Mission Scripts with pAntler. p. 1–17, 2009.

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

NEWMAN, P. Bridging MOOS Communities with pShare. p. 1–12, 2013.

RICHARDSON, M.; WALLACE, S. **Getting Started with Raspberry Pi**. [S.l.: s.n.], 2013.

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

SIDELEAU, S.; EICKSEDT, D. The backseat control architecture for autonomous robotic vehicles: a case study with the Iver2 AUV. **Marine technology society journal**, v. 44, n. 4, p. 42–54, 2010.

SIRF, T. I. NMEA Reference Manual. v. 1, n. January, 2005.

SIRF, T. I. SiRF Binary Protocol Reference Manual. v. 1, p. 119, 2007.

SIRF, T. I. SiRF NMEA Reference Manual. v. 1, 2007.

SPEARS, A.; WEST, M.; COLLINS, T. Autonomous Control and Simulation of the VideoRay Pro III Vehicle Using MOOS and IvP Helm. **mtsjournal.org**, 2013.

SURE, E. **USB to RS232 Module User Guide**. 2008.

SWEET, M. R. **Serial Programming Guide for POSIX Systems**.

XIA, T.; YANG, M.; YANG, R. Vision Based Global Localization for Intelligent Vehicles. **2006 IEEE Intelligent Vehicles Symposium**, Ieee, p. 571–576, 2006.

Apêndice A - Desenvolvimento de um aplicativo MOOS

Este apêndice traz um passo a passo de como desenvolver aplicativos MOOS, no entanto ele não pretende ser auto suficiente e portanto estar em contato com a documentação é fundamental para o desenvolvimento de um aplicativo.

A.0.1 Aumentando a árvore do MOOS-IvP

A instalação do MOOS-IvP é feita em um diretório `moos-ivp`. Este diretório contém todos os aplicativos MOOS e comportamentos IvP padrões. Alterar os diretórios ou modificá-los pode trazer consequências difíceis de rastrear, assim é recomendado criar uma nova árvore para desenvolvimento dos aplicativos. Uma estrutura pronta está disponível para *download* através do repositório do MIT. Essa estrutura é a `moos-ivp-extend`, ela é uma estrutura simplificada da árvore do MOOS e possui já diversos *shell scripts* e *CMakeLists* para facilitar a compilação.

A árvore pode ser acessada por:

Código A.1 – *Download* da árvore `moos-ivp-extend`

```
1 #!/bin/bash
2 svn co https://oceanai.mit.edu/svn/moos-ivp-extend/trunk moos-ivp-
    extend
3 cd moos-ivp-extend
4 ./build.sh
```

Se a compilação ocorreu sem problemas o *setup* inicial está feito. Lembrando que ambas as árvores (`moos-ivp` e `moos-ivp-extend`) devem estar adicionadas ao PATH.

Para criação do aplicativo um *script* está disponível para gerar as pastas e os arquivos base em *C++* e *CMakeLists*. Este *script* requer o nome do aplicativo, prefixo (NEWMAN, 2013a) e nome do autor:

Código A.2 – *Download* da árvore `moos-ivp-extend`

```
1 #!/bin/bash
2 cd moos-ivp-extend/src
3 GenMOOSApp Nome_do_aplicativo p "Autor_do_aplicativo"
```

Observe que no diretório `moos-ivp-extend/src` foi criado uma pasta com o nome do aplicativo e vários arquivos (*.cpp, *.h, *.moos e CMakeLists).

Antes de iniciar o desenvolvimento do aplicativo, o diretório do aplicativo deve ser acrescentado ao *CMakeLists* do diretório `moos-ivp-extend/src`. A adição pode ser feita acrescentando a linha de código ao arquivo CMakeLists.txt:

Código A.3 – Acrescentando o diretório ao CMakeLists

```
1 ADD_SUBDIRECTORY(MyFolderName)
```

A.0.2 Desenvolvendo o aplicativo

O desenvolvimento de um aplicativo MOOS consiste em criar uma classe e reescrever alguns de seus métodos para que o aplicativo realize as funções esperadas.

O *script* `GenMOOSApp` realiza uma boa parte deste trabalho, gerando a base da classe, com seus arquivos de cabeçalho, os métodos que devem ser reescritos entre outras.

Como indicado na documentação (BENJAMIN *et al.*, 2013) e (NEWMAN, 2013a), é necessário reescrever os métodos:

- **Constructors** e **Destructors**.
- **OnNewMail**: Nesta função escolhe-se em que variável da classe será salvo as novas notificações.
- **OnConnectToServer**: EM geral nesta função não é realizado nenhuma modificação, ela chama apenas a função **RegisterVariables**.
- **Iterate**: Nesta função que escreve-se a rotina principal do aplicativo.
- **OnStartUp**: Nesta função define-se as configurações iniciais do aplicativo, por exemplo a leitura das configurações do arquivo **.moos*.
- **RegisterVariables**: Essa função não é obrigatória, ela pode ser executada dentro de **OnConnectToServer**, função que a chama. No entanto, mantendo o padrão dos outros aplicativos, nesta função é que se registra pelas variáveis que deseja-se receber notificações.

A melhor forma de se aprender a criar um aplicativo MOOS é comparando e tentando entender a estrutura de outros aplicativos MOOS. O *website* `https://github.`

com/davidissamattos traz vários aplicativos desenvolvidos e que podem servir como base para outros. A árvore `moos-ivp` também possui o código fonte de todos os aplicativos MOOS distribuídos com o *software*, e que também servem como uma boa fonte de referência.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. REGISTRO N°	4. N° DE PÁGINAS
TC	18 de novembro de 2014	DCTA/ITA/TC-044/2014	90

5. TÍTULO E SUBTÍTULO:

Implementação do software MOOS-IvP em um barco autônomo.

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:

Navegação Autônoma, MOOS-IvP

9.PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Barcos; Navegação autônoma; Motores de corrente continua; Controle; Veiculos aquáticos; Computação; Controle.

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

ITA, São José dos Campos. Curso de Graduação em Engenharia Eletrônica. Orientadores: Cairo Lúcio Nascimento Junior; Douglas Soares dos Santos. Publicado em 2014.

11. RESUMO:

Este trabalho apresenta a implementação do software MOOS-IvP, de navegação autônoma, em barco de baixo custo. O software MOOS-IvP dá suporte a configuração e planejamento de missões autônomas e cooperativas para veículos, em particular para veículos subaquáticos. O barco utilizado nesta aplicação é do tipo catamarã, que utiliza como sistema de propulsão rodas d'água impulsionadas por motores elétricos de corrente contínua. Foi projetado e instalado para este barco um módulo embarcado com uma plataforma inercial (com acelerômetros, giroscópios e magnetômetros), um receptor GPS, um microcontrolador e um sistema de comunicação. O módulo embarcado se comunica com uma estação de controle em terra enviando a telemetria dos sensores e recebendo os comandos para o acionamento dos propulsores. A estação em terra utiliza o software MOOS-IvP como solução de navegação. Também é mostrado neste trabalho como acoplar um módulo de transmissão de imagem, com capacidade de processamento para visão computacional, ao software MOOS-IvP. Este trabalho abrange a simulação, configuração de missões e implementação no sistema real, mostrando que o barco segue uma trajetória pré-definida.

12. GRAU DE SIGILO:

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