UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

LUCAS PIRES CAMARGO

PROJETO HYDRUS: VEÍCULO AQUÁTICO PARA MONITORAMENTO DA QUALIDADE DA ÁGUA

JOINVILLE
2017

LUCAS PIRES CAMARGO

PROJETO HYDRUS: VEÍCULO AQUÁTICO PARA MONITORAMENTO DA QUALIDADE DA ÁGUA

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Prof. Dr. Anderson Wedderhoff Spengler
Coorientador: Prof. Dr. Giovani Gracioli

JOINVILLE
2017

RESUMO

O objetivo principal do trabalho foi desenvolver um veículo aquático autônomo para monitoramento de recursos hídricos. O veículo foi projetado e construído com base em um modelo de catamarã pré-existente. Foi produzido um par de cascos de fibra de vidro, unidos por uma plataforma central. O hardware foi projetado baseando-se na utilização de componentes disponíveis no mercado, tendo o Raspberry Pi como plataforma computacional. O hardware inclui um módulo receptor GPS, magnetômetro, e conversor analógico-digital. Para a obtenção dos indicadores referentes à qualidade da água, foram agregados ao projeto sensores de pH, temperatura, e turbidez. As respostas características dos sensores em relação às grandezas de entrada foram enumeradas e o eletrodo de pH foi calibrado. O firmware do barco foi escrito na linguagem de programação C++, e sua principal característica arquitetural é a utilização do padrão Blackboard. O padrão consiste na manutenção de uma base de conhecimento comum, sobre a qual operam tarefas paralelas. Cada tarefa é especializada em algum aspecto da funcionalidade do sistema (por exemplo, tarefas de sensoreamento e navegação). As funcionalidades da plataforma de hardware são encapsuladas em classes específicas, que abstraem recursos como dispositivos, barramentos de entrada e saída, e recursos do sistema operacional. Em adição ao firmware do veículo, foi desenvolvido um software de comando, também em C++, para uso na estação-base. O software permite ao usuário monitorar o veículo, configurar rotas de navegação, e iniciar a sequência de navegação autônoma. O veículo foi testado, tanto em bancada quanto em uma situação de navegação real, e foi validada sua capacidade de navegar autonomamente, se comunicar com a estação-base, e coletar dados. Ao fim do trabalho, foram validadas as funcionalidades básicas do veículo, suas capacidades de coleta de dados, e de navegação. Concluiu-se que o trabalho atingiu seus objetivos, resultando na criação de um veículo funcional, sobre as plataformas de hardware e de software. Sugeriu-se como trabalhos futuros a aplicação da solução desenvolvida no monitoramento de um recurso hídrico real, e a melhora dos recursos de coleta e apresentação dos dados adquiridos.

Palavras-chave: Veículo de monitoramento, qualidade da água, padrão Blackboard.

ABSTRACT

The main objective of this work is to develop an autonomous aquatic vehicle for water resource monitoring. The vehicle was engineered and built using as a base a preexisting catamaran model. A pair or composite fiberglass hulls were produced, united by a central platform. The hardware was developed based on off-the-shelf parts, having as a computing platform the Raspberry Pi. The hardware includes a GPS receiver module, magnetometer, and analog-digital converter. For obtaining a set of data from the water, pH , turbidity and temperature sensors were added to the project. The firmware of the vehicle was developed in the C++ programming language, and its main architectural characteristic is the implementation of the Blackboard pattern. This software engineering pattern consists of a common knowledge base, over which independent task operate. Each tasks implements specific functionality of the system (e.g., sensing and navigation tasks). Hardware platform functionality is encapsulated in specific classes, that abstract resources such as devices, input and output buses, and operating system resources. In addition to the vehicle firmware, a command and control software was developed, also in C++, for usage in the base station. The software allows the user to monitor the vehicle, configure navigation routes, and initiate the autonomous navigation sequence. The vehicle was tested in the workbench, as well as in a real navigation scenario. Then its capacity to navigate autonomously is validated, as well as communicating with the base station, and collecting data. At the end of this paper, the basic functionalitry of the vehicle was validated, as well as its navigation and data collection capabilities. It was concluded that the work has reached its goals, resulting in the creation of a functional vehicle, built atop the hardware and software platforms. Were suggested as future works: the application of the developed solution in the monitoring of a real hydric resource, and improvements of the data collection and presentation facilities of the software.

Keywords: Monitoring vehicle, water quality, Blackboard pattern.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BLDC Brushless DC Motor

DMA Direct Memory Access

ESC Electronic Speed Controller

GPIO General-Purpose Input/Output

GPS Global Positioning System

HDMI High-Definition Multimedia Interface

I2C Inter-Integrated Circuit

NTU Nefelometric Turbidity Unit

pH Potencial Hidrogeniônico

POSIX Portable Operating System Interface

PWM Pulse Width Modulation

RAM Random Access Memory

SoC System-on-Chip

SPI Serial Peripheral Interface

USB Universal Serial Bus

SUMÁRIO

1 INTRODUÇÃO

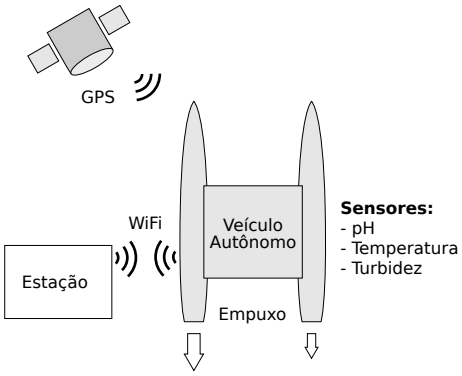
O saneamento básico é entendido como a infraestrutura necessária para garantir a qualidade de vida da população, compreendendo os serviços de tratamento de água, esgotamento sanitário, coleta e destino do lixo e da água da chuva em perímetros urbanos. A Organização das Nações Unidas (ONU, 2015) aponta o acesso ao saneamento básico como um dos direitos fundamentais para que uma população tenha uma vida digna. Segundo o Sistema Nacional de Informações sobre Saneamento (SNIS, 2014), no Brasil, apesar de 93,2% dos brasileiros terem acesso à água potável tratada, apenas 40,8% têm acesso à rede sanitária. Cerca de 3,5 milhões de brasileiros despejam esgoto irregularmente, mesmo que tenham acesso à rede de coleta. Ainda que o esgoto seja coletado, apenas 70,9% do esgoto é tratado corretamente. Isso significa que 1,2 bilhão de m^3 de esgotos não tratados por ano são despejados na natureza.

É evidente que ainda há muito a ser feito no Brasil para garantir à sua população acesso a serviços de saneamento essenciais. Neste contexto, é crucial às autoridades e às prestadoras de serviços terem à sua disposição dados que lhes permitam avaliar e monitorar a qualidade da água de bacias hidrográficas, do ponto de vista da propriedade para consumo, ou para tratamento. Atualmente, na grande maioria dos casos, a análise das propriedades da água em rios e lagos é feita manualmente, com a coleta de amostras e análise em laboratório (NASCIMENTO, 2016). Quando feita em campo, o custo do deslocamento e mão de obra especializada, somado aos equipamentos, é muitas vezes proibitivo em um regime diário ou semanal. Apesar de esses procedimentos gerarem dados precisos, eles são esparsos e custosos.

Avanços tecnológicos permitiram a redução de custo e a miniaturização de sistemas embarcados com funcionalidades complexas, que incluem coleta e processamento de dados. Como consequência, emergiram os sistemas embarcados autônomos. Temos como exemplos, carros, barcos, veículos aéreos de diversos formatos e princípios de funcionamento, entre outros.

Nos veículos aquáticos autônomos, em particular, deve ser possível a integração de sensores de qualidade da água de forma a permitir medições completamente autônomas. Busca-se com isso reduzir tempo de trabalho e custos totais para a captura dos dados dos sensores. Também objetiva-se o registo de dimensões

Figura 1 – Conceito do veículo



Fonte: do autor.

espaciais e temporais aos dados. A capacidade de se obterem gráficos temporais e espaciais pode ser de grande ajuda na detecção de tendências dos indicadores, permitindo, assim, a previsão de problemas ambientais e o planejamento de ações de correção.

1.1 Objetivo Geral

Como objetivo geral do projeto, propõe-se a construção de um veículo autônomo que possibilite medições frequentes de indicadore da qualidade da água, úteis para avaliações qualitativas e comparativas, em alternativa a medições manuais. A construção do veículo deve resultar em uma plataforma flexível, e extensível que no futuro poderá ser adaptada para prover funcionalidades ainda não previstas. A Figura 1 mostra um conceito do veículo, com suas entradas e saídas.

1.2 Objetivos Específicos

De modo a atingir o objetivo geral, foram definidos os seguintes objetivos específicos:

a. Especificação de requisitos das plataformas de hardware e software;

b. Projeto da estrutura do software e hardware, de modo a tenderem os requisitos;

c. Implementação da plataforma de software, e construção e testes do novo hardware;

d. Integração dos sistemas desenvolvidos durante o processo de desenvolvimento;

e. Testes de integração do sistema, validação de subsistemas, e testes finais de navegação e coleta de dados.

1.3 Estrutura do texto

Este trabalho está estruturado da seguinte forma: o Capítulo 2, "Fundamentação teórica", introduz a tecnologia e os conceitos básicos que fundamentam este trabalho. O Capítulo 3, "Projeto e Implementação", descreve o veículo desenvolvido, do ponto de vista físico, de hardware e de software. O Capitulo 4, "Testes e Resultados", apresenta os principais testes efetuados, e os resultados deles obtidos. Por fim, no Capítulo 5 são discutidas as contribuições realizadas e possíveis desenvolvimentos posteriores.

2 FUNDAMENTAÇÃO TEÓRICA

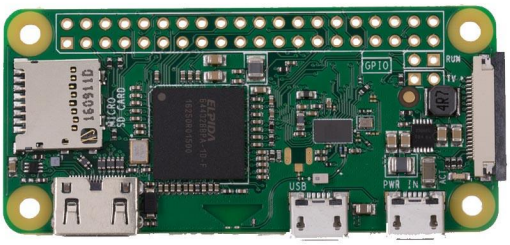
Neste capítulo, são apresentados os principais conceitos acerca de veículos autônomos e sistemas embarcados. Primeiramente, dá uma visão geral sobre veículos autônomos com enfoque em veículos aquáticos. Em seguida, descreve a plataforma de hardware Raspberry Pi, o sistema operacional Linux e a distribuição Raspbian. O capítulo aborda aspectos de hardware e subsistemas ao abordar o GPS, sensores de pH, turbidez, e o papel desses indicadores na medição de qualidade da água.

2.1 Veículos autônomos

Veículos autônomos, frequentemente denominados drones desde a segunda guerra mundial, surgiram inicialmente na forma de aviões para treinamento de pilotos militares (ZIMMER, 2016). Um dos principais usos era servir como alvos em treinamentos de combate. Desde então, veículos não-tripulados adotaram as mais diferentes formas e funcionalidades. Com o avanço da miniaturização e redução de custos dos sistemas embarcados, os drones estão cada vez menores, mais ágeis, e possuem maior poder computacional. Já existem aplicações de sensoreamento remoto, com foco em produtividade agrícola e segurança civil e militar (POBKRUT; EAMSA-ARD; KERDCHAROEN, 2016).

Há diversas pesquisas envolvendo veículos aquáticos não-tripulados, em três principais categorias: flutuantes, imersíveis, e semi-imersíveis. Ainda há uma categoria específica para hidroaviões. Dentro dessas categorias, pesquisas são realizadas com foco em telecomunicações, sensoreamento, navegação autônoma, consumo e geração de energia. Conjuntos de drones podem formar redes de sensores em malha, e agir como uma única entidade de sensoreamento, aumentando a área efetiva de monitoramento (VELEZ et al., 2015). Ou ainda, um único veículo semi-submersível pode cumprir diferentes tarefas de sensoreamento quando se encontra submerso ou não, e ser flexível o suficiente para atender a uma gama de aplicações comercias e de pesquisa (RAIMONDI et al., 2015).

Figura 2 – Uma placa Raspberry Pi Zero em tamanho real



Fonte: (RASPBERRY PI, 2017).

2.2 A plataforma Raspberry Pi

A Raspberry Pi é uma plataforma de hardware voltada a fins educacionais, de pesquisa, aplicações de baixo custo, e entusiastas. É construída em torno dos System-on-Chips (SoCs) BCM2835 e BCM2836 da Broadcom. Ambos possuem processadores baseados na arquitetura ARM11, comum em dispositivos móveis (UPTON; HALFACREE, 2012). Existem várias versões da placa, com quantidade de memória e opções de portas e conectividade variadas. (RASPBERRY PI, 2017)

O modelo Raspberry Pi Zero apresenta um SOC BCM2835, com um núcleo de processador ARM11 com clock de 1GHz, 512 MB de memória RAM, um suporte para cartão micro-SD para armazenamento de dados e do sistema operacional, saída de vídeo mini-HDMI (High-Definition Multimedia Interface), conectores Micro-USB para fornecimento de energia e conexão de periféricos, e um cabeçalho de pinos GPIO (General-Purpose Input/Output), com interfaces de comunicação nos padrões Serial Peripheral Interface (SPI) e Inter-Integrated Circuit (I2C). A figura 2 mostra a placa.

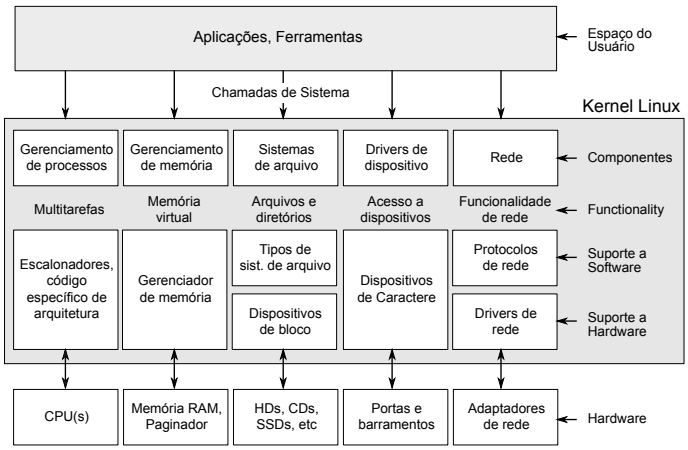
2.3 O sistema operacional Linux e a distribuição Raspbian

O Linux é um kernel de sistema operacional de código aberto, em desenvolvimento desde 1991. De design monolítico, media a interação de programas de computador com os recursos do hardware, gerenciando multiprocessamento, memória, armazenamento, e dispositivos de entrada e saída. (BOVET; CESATI, 2005)

É comumente utilizado em conjunto com as ferramentas do Sistema Operacional GNU, do projeto GNU, resultando em um sistema operacional completo denominado GNU/Linux. Na atualidade, o GNU/Linux é responsável por executar a maior parte das cargas computacionais de supercomputadores, da internet, e de dispositvos móveis, sendo a base da plataforma Android. (LINUX, 2017)

Como subsistemas importantes do Linux pode-se ressaltar o gerenciamento de memória, criação e agendamento de processos, trocas de contexto de processos, sistemas de DMA, sinais, interrupções, temporizadores, comunicação entre processos,

Figura 3 – A arquitetura do kernel Linux



Fonte: adaptado de (BOVET; CESATI, 2005).

entre outros.

O design do Linux é monolítico, onde seus vários subsistemas dividem o mesmo espaço de código e semânticas de acesso à memória. Porém, o mesmo possui um mecanismo para modularidade. Unidades de código que proveem funcionalidades específicas, denominadas módulos do kernel, podem ser carregadas e descarregadas dinamicamente. Módulos são geralmente utilizados para prover drivers de hardware, sistemas de arquivo, e funcionalidades especializadas para o espaço de usuário, onde executam os programas.

O Linux é compatível com o padrão POSIX (Portable Operating System Interface). O padrão especifica mecanismos de acesso a arquivos, sockets de rede, primitivas de multiprocessamento e sincronização, como threads, mutexes e semáforos. (WALLI, 1995) As chamadas de sistema são implementadas através de funcões da linguagem C, que ora implementam parte da funcionalidade requerida, ora servem de ponto de entrada para instruções de chamada de sistema que passam o controle do programa ao código do kernel. Por sua vez, o código do kernel executa de forma privilegiada e tem acesso direto ao hardware, efetivamente executando as operações requeridas pelo programa.

2.3.1 Raspbian

Os sistemas operacionais baseados em GNU/Linux são comumente denominados distribuições. Elas empacotam software de maneira a prover um sistema completo, com kernel, bibliotecas, interface do usuário, aplicações, e ferramentas de desenvolvimento.

A distribuição Raspbian é uma distribuição de sistema operacional livre baseada no projeto Debian, uma distribuição tradicional. Porém, foi criada especificamente para

ser a base de software padrão da plataforma Raspberry Pi. (RASPBIAN PROJECT, THE,)

O projeto consiste em um kernel configurado para o SOC BCM2835, pacotes compilados para a arquitetura ARM11 com suporte para ponto flutuante em hardware, drivers para a GPU e periféricos especiais do SOC, e uma imagem de sistema operacional com provisões de boot, compatível com o bootloader do SOC.

2.4 GPS: Global Positioning System

A crescente capacidade computacional e o avanço tecnológico permitiram a popularização do Sistema de Posicionamento Global (GPS). Essa tecnologia foi inicialmente desenvolvida pelo Departamento de Defesa americano (DoD). Na década de 80, foi liberado para uso civil, e em 1995, tornou-se completamente operacional. (KAPLAN; HEGARTY, 2006)

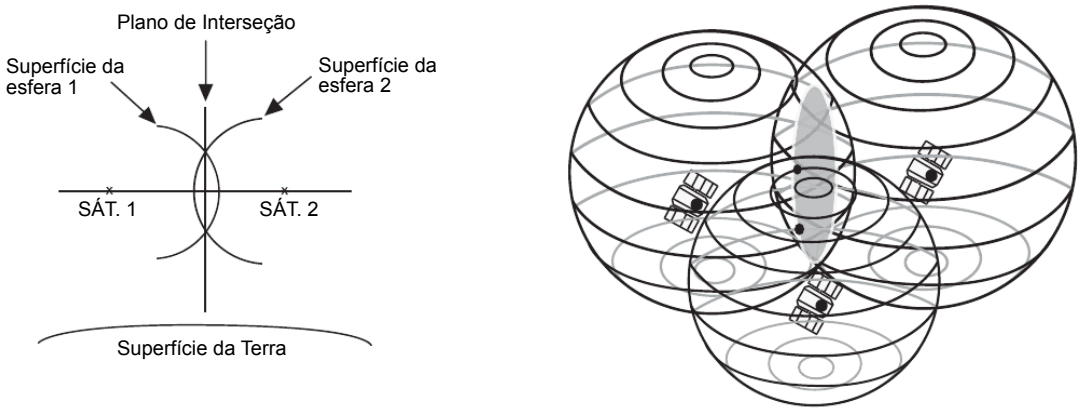
O princípio de funcionamento do sistema é baseado em satélites cuja posição é conhecida para qualquer instante de tempo determinado. Os satélites carregam relógios atômicos que são sincronizados com relógios em terra e entre si.

Cada satélite transmite um sinal composto por um fluxo de dados binários pseudoaleatório, porém conhecido pelo receptor, e uma mensagem com a posição do GPS. A mensagem com a posição do satélite inclui o tempo de transmissão, no referencial de tempo do sistema de posicionamento. Medindo a diferença entre o tempo de transmissão e o tempo de recepção, o receptor consegue estimar o tempo em que o sinal esteve em transmissão, e relacionando esse tempo com a velocidade da luz, consegue determinar a distância respectiva de cada satélite em sincronização.

Conhecendo a distância aproximada de quatro satélites, o receptor consegue fazer uma trilateração tridimensional de sua posição em relação aos satélites, e consequentemente, por ter como conhecidas as posições dos satélites, pode determinar sua localização na terra. Adicionalmente, tendo uma altitude conhecida, o receptor pode ainda estimar sua posição no globo terrestre calculando uma triangulação com o uso de distâncias e posições conhecidas de três satélites. A Figura 4 mostra um exemplo de trilateração com três satélites.

Devido ao GPS ser uma tecnologia militar americana, outros países do mundo desenvolveram suas próprias tecnologias e redes de satélite para fins de navegação. Por exemplo, a Rússia desenvolveu o GLONASS, a União Européia desenvolve o programa GALILEO, a China desenvolve o BeiDou, e o Japão desenvolve o QZSS. Todos os sistemas possuem o mesmo princípio de funcionamento que o GPS. Diferentes padrões de sinais de satélites são suportados por diferentes receptores. É comum aos receptores modernos o suporte ao trio GPS, GLONASS e GALILEO. (KAPLAN; HEGARTY, 2006)

Figura 4 – Princípio de trilateração com três satélites



Fonte: adaptado de (KAPLAN; HEGARTY, 2006).

2.5 Medição da Qualidade da Água

Para que seja possível o controle da poluição da água, é primeiro necessária a medição dos poluentes. Propriedades das amostras que são determinantes da qualidade da água são determinadas indicadores. A medição de poluentes é repleta de dificuldades, como poluentes desconhecidos, alterações de indicadores nas amostras, dentre outros. (WEINER; MATTHEWS, 2003, p. 81)

Medições podem ser feitas de dois modos, diretamente ou indiretamente. Medições indiretas são efetuadas a partir de amostras, parte do volume que se quer medir. Busca-se no processo de amostragem a obtenção de suficiente representatividade das grandezas originais através da amostra. Portanto, a amostragem deve ser feita com cuidado pois a maneira com que a mesma é obtida pode influenciar o resultado de um teste de indicador. (LIMA, 2006)

Amostras simples são uma parcela do material a ser estudando, eu um único ponto determinado do tempo e do espaço. Uma amostra composta é um conjunto de amostras simples que foram misturadas, afim de se obter uma amostra representativa de um perfil de profundidade, ou de múltiplos pontos no tempo ou espaço.

Devido às necessidade de obtenção de amostras consistentes e precisas, a amostragem da água de um recurso hídrico é feita seguindo-se processos de amostragem bem definidos. Para combinação de tipo de amostra e local de amostra (rio, lago, tanque, etc) há amostradores recomendados, e limitações para cada uma das situações. (ABNT, 2004)

2.5.1 Principais Indicadores

De acordo com Weiner e Matthews (2003, p. 82), os indicadores de qualidade da água mais importantes são os seguintes:

- **Oxigênio dissolvido**: oxigênio, apesar de pouco solúvel em água, é imprescindível à vida aquática. O valor de saturação á inversamente proporcional à temperatura da água. O máximo valor de saturação, com a água a 0 graus, é 14,6 mg/L. É preferível fazer medições de oxigênio dissolvido no local.

- **Demanda Bioquímica de Oxigênio (DBO)**: é a taxa em que o oxigênio é consumido por agentes decompositores, em um determinado período de tempo. Esse indicador é útil na previsão do impacto de efluentes que possuem grande quantidade de compostos orgânicos biodegradáveis. Uma demanda alta indica possível redução no oxigênio dissolvido. Uma baixa demanda indica água limpa ou contaminação por um agente tóxico ou não-biodegradável.

- **Demanda Química de Oxigênio (DQO)**: é uma versão simplificada do teste de DBO. A matéria orgânica é oxidada quimicamente, fornecendo uma aproximação da demanda bioquímica de oxigênio. Como o teste oxida mais compostos do que são biologicamente processados na natureza, tente a apresentar resultados de maior grandeza.

- **Carbono orgânico total**: é medido ao se oxidar todo o carbono orgânico em CO_2 e H_2O, em uma câmara de combustão de alta temperatura. É comum o seu uso na análise do potencial de produção de subprodutos químicos em processos de desinfecção, como cloração, por exemplo.

- **Turbidez**: é a medida com que a água absorve e espalha a luz, impedindo sua transmissão. É causada por algas, argilas, e matéria orgânica e inorgânica em suspensão. Sua importância depende das condições naturais do ambiente e dos processos em estudo.

- **pH**: uma medida da concentração de íons de hidrogênio (H^+), uma medida da acidez do meio. É importante em todas as fases de tratamento de água e efluentes, para garantir tratamento químico adequado. Organismos aquáticos são bastantes sensíveis ao valor absoluto do pH do meio e à variação do mesmo.

- **Alcalinidade**: mede a capacidade da àgua de resistir a mudanças de pH. Na natureza, é provida por um sistema de equilíbrio entre o gás carbônico (CO_2) e o ácido carbônico (H_2CO_3).

- **Sólidos**: abrangem materiais em suspensão e materiais dissolvidos na água. Influencia diretamente na turbidez e em processos biológicos. O tratamento de efluentes tem como um objetivo principal a separação dos sólidos da água.

- **Patógenos**: uma grande quantidade de doenças é transmissível pela água. As qualidades bacteriológicas da água são tão importantes como suas características

químicas, do ponto de vista da saúde pública. Como é impossível detectar grande parte dos micro-organismos nocivos, é adotado um organismo de referência, de fácil detecção. Tipicamente, é o *E. coli*, também conhecido como coliforme fecal. O processo de detecção mais comum é o de filtragem em membrana seguida de cultura in-vitro.

- **Metais pesados**: podem se acumular em cadeias alimentares e causar danos a organismos aquáticos, e ao homem. Os mais comuns são cobre, mercúrio e arsênico. Esse acúmulo nocivo pode ocorrer mesmo se a concentração for relativamente baixa, portanto os métodos de medição devem ser precisos.

2.5.2 Sensores de pH

O pH, ou Potencial Hidrogeniônico, é por definição o cologaritmo da concentração de íons de hidrogênio (H^+) em um meio. De maneira geral, o pH indica a acidez ou alcalinidade de uma solução. De acordo com Oliveira e Fernandes (2012), para soluções diluídas, o pH pode ser definido como:

$$\mathrm{pH} = -\log_{10}\left[\mathrm{H}^+\right]$$

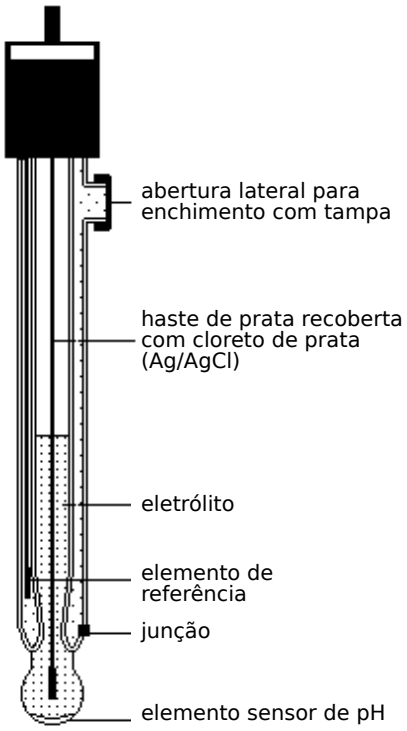
O pH de uma solução pode ser medido com a utilização de um pHmetro, geralmente composto de um eletrodo associado a um medidor de potencial. O tipo de eletrodo mais comum e largamente utilizado, é o eletrodo de vidro. Este é composto por um par de soluções de pH conhecido, separados da solução a ser medida por uma membrana de vidro dopada, sensível aos íons H^+. A diferença de potencial entre o elemento de referência e a solução interna é sensível à concentração de íons da solução a ser medida. Na figura 5 é mostrado um eletrodo combinado comum, que apresenta as duas soluções em uma única haste.

O eletrodo combinado possui dois elementos de contato, um ao elemento de referência, e outro ao eletrólito interno. Este geralmente é uma solução de cloreto de prata ou cloreto de potássio. O potencial pode ser medido diretamente, e dele, determinado o pH da solução a ser medida.

Há diversos fatores que dificultam a medição de potencial de um eletrodo de pH típico. Isso é explorado por Kuphaldt (2000):

> Even a very small circuit current traveling through the high resistances of each component in the circuit (especially the measurement electrode's glass membrane), will produce relatively substantial voltage drops across those resistances, seriously reducing the voltage seen by the meter. Making matters worse is the fact that the voltage differential generated by the measurement electrode is very small, in the millivolt range (ideally 59.16 millivolts per pH unit at room temperature). The meter used for this task must be very sensitive and have an extremely high input resistance (KUPHALDT, 2000, p. 319).

Figura 5 – Um eletrodo de ph de vidro



Fonte: adaptado de (OLIVEIRA; FERNANDES, 2012).

Por esse motivo, o autor sublinha a necessidade da utilização de um medidor amplificado, com altíssima impedância de entrada, na ordem de $10^{17}\ MOhm$.

2.5.3 Sensores de turbidez

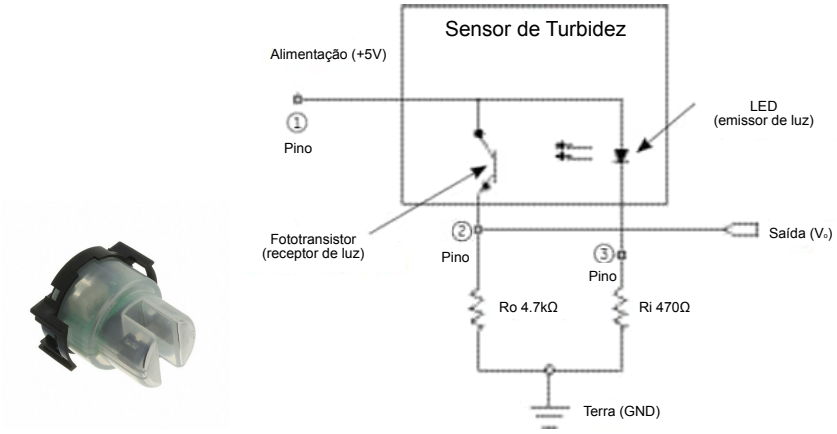
A turbidez é definida como a medida de quanto a luz é transmitida ou oclusa por um meio aquoso. Quando mais turva, ou "suja", se encontra a água, maior a turbidez. De acordo com Weiner e Matthews (2003, p. 92), vários materiais podem causar o aumento de turbidez, incluindo argilas, particulados inorgânicos em suspensão, algas e cianobactérias, e material orgânico.

No processo de tratamento de água, o monitoramento da turbidez é de grande importância, não apenas pelo aspecto estético, como também pela dificuldade em se remover ou inativar agentes biológicos nocivos quando há partículas em suspensão na solução.

O instrumento de medição para a turbidez é denominado turbidímetro, e a unidade de medida é o padrão NTU (Nefelometric Turbidity Unit). O mecanismo de funcionamento baseia-se na medição da luz transmitida pela solução em medição. A Figura 6 ilustra um turbidímetro convencional.

A luz transmitida é inversamente proporcional à turbidez do meio, pois a luz que não é transmitida é ou absorvida pelas partículas em suspensão, ou espalhada no

Figura 6 – Um turbidímetro convencional e seu circuito interno



Fonte: adaptado de (AMPHENOL ADVANCED SENSORS, 2014).

meio ao refletir de partícula em partícula. No exemplo da Figura 6, é medida apenas a luz transmitida, através de um fototransistor.

3 PROJETO E IMPLEMENTAÇÃO

Este capítulo aborda o projeto e implementação do veículo autônomo, elucidando os requisitos de projeto, características mecânicas do veículo, hardware, e software.

No primeiro momento, são enumerados os requisitos de projeto de maneira geral. Depois, são apresentadas as características mecânicas do veículo. As seções seguintes descrevem o projeto de hardware que busca atingir os requisitos, e a placa-base do veículo. É apresentada também a estrutura do firmware do veículo, o papel de cada tarefa, e como foi implementada a navegação. A seção seguinte discorre sobre como é feita a calibração e conversão dos dados dos sensores. Por fim, a última seção discute o software da estação base.

3.1 Requisitos de Projeto

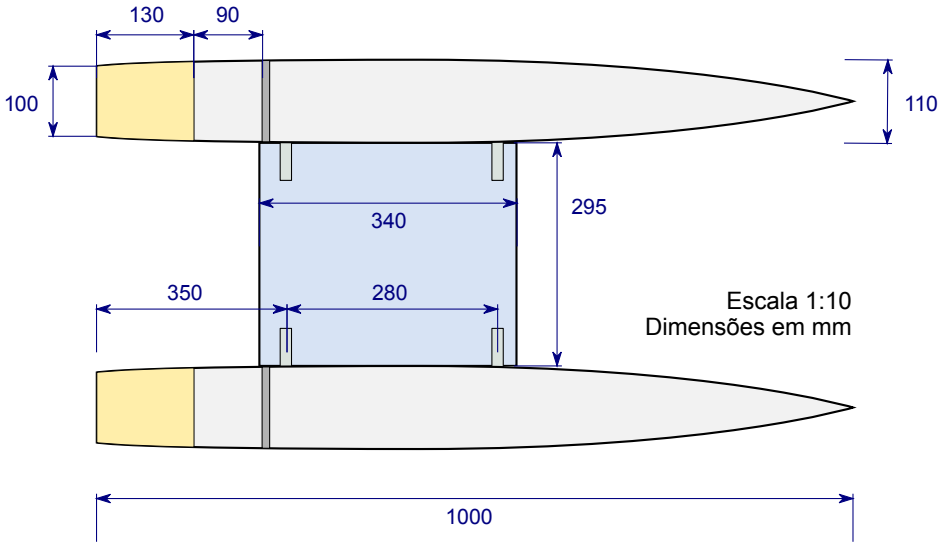
Antes da fase de projeto e construção do veículo, foram delineados os principais requisitos de projeto, de acordo com a Tabela 1. A primeira coluna denota o número identificador do requisito. A segunda define o requisito brevemente, e a terceira descreve a prioridade.

Tabela 1 – Requisitos do projeto

#	**Requisito** – O veículo deve...	**Prio.**
RF01	Navegar autonomamente por GPS	Alta
RF02	Navegar por uma rota predeterminada	Alta
RF03	Fazer medições de indicadores de qualidade da água	Alta
RF04	Armazenar os dados de medição internamente	Alta
RF04	Se comunicar com a estação-base via rede sem-fio	Alta
RF05	Prover informações de diagnóstico	Média
RNF01	Possuir custo reduzido	Alta
RNF02	Utilizar componentes comerciais amplamente disponíveis	Alta
RNF03	Ter boa performance do software	Alta
RNF04	Permitir a adição de mais sensores e atuadores	Média
RNF05	Ser controlável por um computador pessoal comum	Média
RNF06	Boa estabilidade de navegação	Alta

Fonte: do autor.

Figura 7 – Dimensões do veículo



Fonte: do autor.

Os requisitos funcionais possuem identificadores iniciando-se em RF, enquanto que os requisitos não-funcionais possuem identificadores iniciando-se em RNF.

3.2 Projeto Mecânico

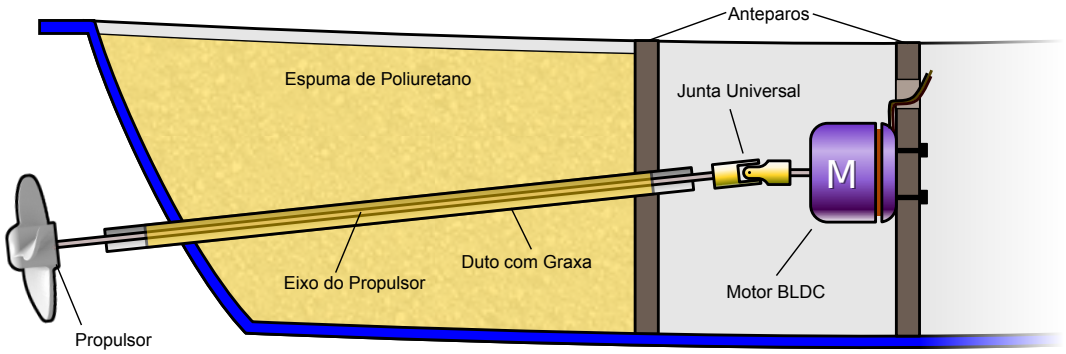
Os primeiros passos do projeto consistiram em determinar a forma construtiva do barco, de acordo com os recursos disponíveis. Esse seria um fator determinante no atendimento ao requisito RNF06, "Boa estabilidade de navegação". No Centro Tecnológico de Joinville, da Universidade Federal de Santa Catarina, onde foi realizado o trabalho, havia disponível no Laboratório de Modelagem e Construção Naval um modelo de catamarã, originalmente construído pelo discente Ricardo Bedin. O modelo estava quebrado e destinado ao descarte. Foi decidido consertar o modelo e dele retirar um molde de fibra de vidro, para uso na construção do veículo . Portanto, o veículo final teria a configuração de um catamarã.

A partir do molde confeccionado, foram obtidos dois cascos de fibra de vidro, através de um processo de laminação. Durante a laminação foram embutidos dois suportes em L de alumínio a cada casco, para que pudesse ser feita a fixação de uma plataforma central, em madeira MDF (Medium Density Fiberboard). As dimensões finais do veículo estão apresentadas na Figura 7.

3.2.1 Propulsão

Buscando o aproveitamento de materiais disponíveis na universidade, o sistema de propulsão foi construído a partir de peças de drones quadricópteros. As hélices foram

Figura 8 – Sistema de propulsão do catamarã



Fonte: do autor.

impressas em ABS a partir de um modelo adaptado de THOMAS (2015), desenvolvido expressamente para barcos de radiocontrolados.

Cada casco do recebeu um motor de corrente contínua sem escovas BC2830-11, da RCTimer. O eixo de cada motor transmite potência para o eixo do propulsor, através de um cardã (junta universal), para ajuste de ângulo. A separação entre a câmara do motor e a água se dá através da imersão do eixo do propulsor em graxa, no duto que atravessa o casco. Dessa maneira, não há entrada de água no casco através do mecanismo. A Figura 8 ilustra a configuração final.

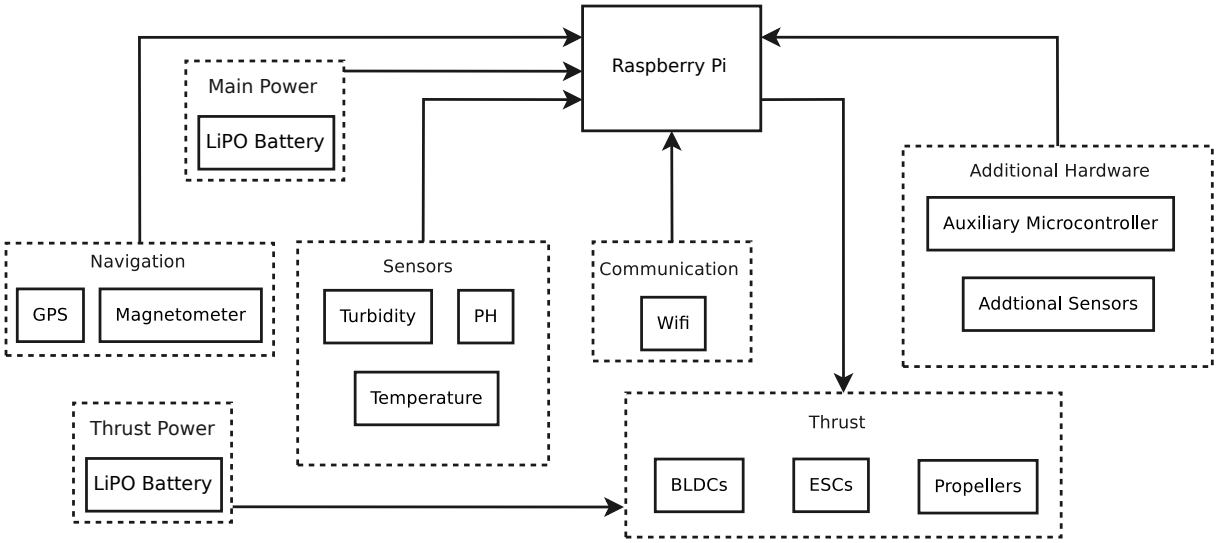
Cada motor é acionável de maneira independente. O acionamento de apenas um motor gera um momento em relação ao eixo vertical do veículo.

3.3 Análise dos Requisitos para Hardware

Para o cumprimento dos requisitos, determinou-se o conjunto de elementos de hardware que deveriam compor o sistema. De início, foi necessário definir uma plataforma de hardware para a execução do software. Em observância aos requisitos RNF01 e RNF02, foi escolhida como base do sistema a placa Raspberry Pi Zero (RASPBERRY PI, 2017). A placa possui entradas e saídas do tipo GPIO, SERIAL, e barramentos SPI, e I2C, além de USB, o que permite o interfaceamento com uma grande gama de periféricos comerciais.

Ainda em observação aos requisitos RNF01 e RNF02, determinou-se que, para a simplificação do projeto, deveria-se utilizar ao máximo módulos prontos para os subsistemas de hardware, que incluam os circuitos necessários para o correto funcionamento dos dispositivos, e que permitam a integração simplificada destes componentes ao sistema. De maneira geral, os módulos apresentam conexões de alimentação e conexões de comunicação, na forma de algum barramento padrão (serial, SPI, ou I2C).

Figura 9 – Visão-geral dos subsistemas do hardware



Fonte: do autor.

Para o atendimento do requisito RF01, "Navegar autonomamente por GPS", definiu-se que ao veículo deve-se integrar um módulo receptor de GPS e um módulo magnetômetro, que funcione como bússola, para a determinação da direção instantânea do veículo. Também determinou-se que o veículo deveria integrar algum sistema de propulsão.

Um dos requisitos mais problemáticos foi o requisito RF03, "Fazer medições de indicadores de qualidade da água", porque entrava em conflito com os requisitos RNF01 e RNF02. Grande parte dos sensores disponíveis era demasiadamente caros, e não poderiam ser adquiridos. Uma busca dos sensores disponíveis no mercado demonstrou que seria possível a integração de sensores de pH, turbidez e temperatura.

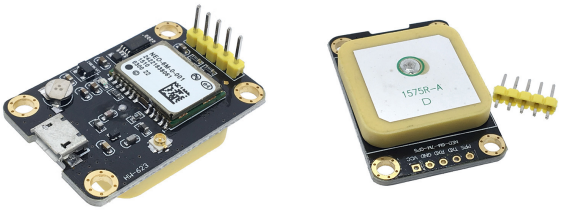
Para o atendimento dos requisitos RF04 e RF05, definiu-se que o hardware irtia comunicar-se com a estação-base via rede sem-fio padrão 802.11g (WiFi) (IEEE-CS, 2003). O requisito RNF04, "Permitir a adição de mais sensores e atuadores", é atendido com a adição de um microcontrolador adicional, conectado ao SOC principal via um barramento padrão. A Figura 9 ilustra os elementos de hardware e suas relações.

3.4 Subsistemas de Hardware

O System-on-Chip (SoC) do sistema, devido à seleção da placa Raspberry Pi Zero como unidade de processamento, é o BCM2835, fabricado pela Broadcom. A placa executa o sistema operacional Linux, que permite a criação de processos multitarefa, gerencia memória, e dispositivos de entrada e saída, além de prover drivers para os dispositivos do SOC e dispositivos USB.

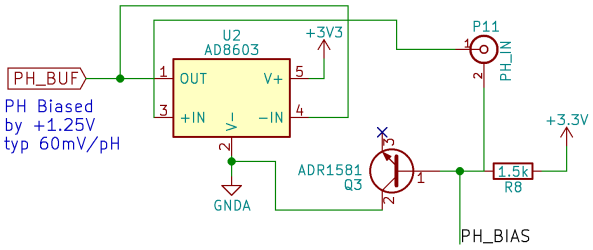
Como módulo GPS foi selecionado o u-blox NEO-6M. Este está disponível em

Figura 10 – Módulo GPS (dois lados)



Fonte: Sincere Company Store.

Figura 11 – Esquemático do circuito de bufferização e offset da sonda de pH



Fonte: do autor.

uma placa que integra uma antena cerâmica e uma bateria de backup, e expõe um interface serial e pinos de alimentação. A Figura 10 ilustra o módulo escolhido.

Para complementar o módulo GPS, foi preciso escolher um módulo magnetômetro para servir de bússola, e fornecer informações de rumo do veículo. Escolheu-se o sensor de campo magnético HMC5883L, fabricado pela Honeywell International. O sensor é triaxial, e fornece uma leitura espacial do campo magnético da terra, com sensibilidade de 2 miligauss. Considerando que o campo magnético da Terra varia entre 250 e 650 miligauss o módulo possui sensibilidade suficiente. O conversor analógico-digital interno fornece uma resolução de 12 bits, o que implica em uma precisão de ângulo de navegação entre 1 e 2 graus. (HONEYWELL, 2013)

A sonda do sensor de pH foi especificada como um eletrodo de vidro combinado padrão. Para atender as necessidades de offset e bufferização do sinal do sensor de pH, optou-se pelo projeto de um circuito dedicado a esse fim. O esquemático do circuito se apresenta na Figura 11. O circuito integrado ADR1581 é um gerador de potencial de referência, fornecendo 1.25V para servir de referencial de tensão da sonda de pH. (ANALOG, 2007) Já o circuito integrado AD8603 é um amplificador operacional que funciona como buffer do sinal da sonda de pH, impedindo a queda de tensão que seria causada no eletrodo que seria induzida pela corrente de medição do sinal. (ANALOG, 2008)

Tabela 2 – Regras de design

Regra	Rede de Sinais	Rede de Potência
Espaçamento entre cobre	0,2 mm	0,3 mm
Largura da trilha	0,5 mm	1,75 mm
Diâmetro da via	0,6 mm	0,6 mm
Furo da via	0,4 mm	0,4 mm

Fonte: do autor.

O sensor de turbidez escolhido é o TSW-10 da Amphenol (AMPHENOL ADVANCED SENSORS, 2014). Originalmente concebido para aplicação em lava-louças e máquinas de lavar, requer apenas dois resistores, e possui saída analógica. Já o sensor de temperatura escolhido é o TMP36 da Analog Devices (ANALOG, 2015). O mesmo é alimentado a partir de um barramento de 3.3V e fornece na saída uma tensão que varia linearmente com a temperatura do circuito integrado.

Para a conversão dos sinais analógicos dos sensores em sinais digitais, optou-se pela integração de um módulo conversor analógico-digital dedicado. O ADS1115 da Texas Instruments (TI, 2016) é um circuito integrado ADC de 16 bits com quatro canais de entrada, amplamente disponível no mercado em forma de módulo. A interface de comunicação com o SoC é o barramento I2C.

Escolheu-se fazer a propulsão do veículo a partir de dois ESCs (Electronic Speed Controllers) e dois motores BLDC de drones quadricópteros. Os mesmo foram escolhidos por já estarem disponíveis em laboratório, em acordância com o requisitos RNF01 e RNF02. Os controladores de velocidade são do modelo SK-30A da RCTimer (RCTIMER, 2017). Os mesmos disponibilizam uma fonte de tensão de 5V, 2A, que pode ser utilizada para fornecer potência aos componentes a partir da bateria.

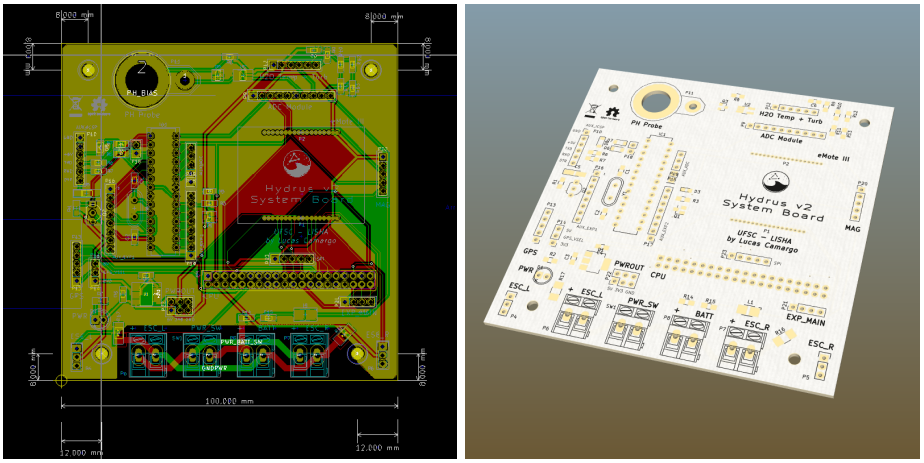
Para prover as necessidades de comunicação, optou-se pela utilização de um adaptador de rede sem-fio genérico, conectado ao veículo pela interface USB.

3.5 Placa-base

Após a definição dos componentes do sistema, foi iniciado o projeto da PCB (placa de circuito impresso) do sistema eletrônico do veículo. Através do software livre KiCad (KICAD, 2017), foi criado um esquemático da placa, disponível no Apêndice A deste trabalho.

O esquemático serviu de base para a criação do projeto de circuito impresso da placa. Buscando redução de custos de produção, em atendimento ao requisito RNF01, o tamanho físico da placa ficou limitado a 100x100 mm. As regras de design utilizadas na criação da PCB estão listadas na Tabela 2. O projeto da placa de circuito impresso, juntamente a uma previsão tridimensional, é apresentada da Figura 12.

Figura 12 – Pré-visualização 3D da placa



Fonte: do autor.

3.5.1 Circuito alternativo do eletrodo de pH

Devido à falta dos componentes necessários à implantação do circuito de pH projetado, adotou-se uma soluçao alternativa baseada no amplificador operacional TL082 (TI, 1977). Por ter uma alta impedância de entrada, na ordem de $10^12 Ohms$, o componente é adequado para ser utilizado como um buffer de sinal para o eletrodo. As tensões necessárias ao funcionamento do aplificador operacional são fornecidas por um par de baterias de 9V. Um par de potêciometros de precisão (trimpots) gera uma tensão de offset para que a tensão de saída possa ser processada pelo conversor analógico-digital. O circuito completo está ilustrado no Anexo B.

3.6 Firmware do veiculo autônomo

O firmware do veículo implementa as funcionalidades de navegação, leitura dos sensores, comunicação, armazenamento de dados, etc. De maneira geral, todas as funcionalidades específicas ao veículo.

Nesta implementação, o firmware consiste em uma aplicação desenvolvida na linguagem C++, fazendo uso de paradigmas de programação padrão da indústria, como a orientação a objeto. A aplicação executa sobre o kernel Linux em espaço de usuário, como um serviço do sistema operacional (daemon). Todo o código-fonte do projeto está disponível em um repósitório de acesso público[1].

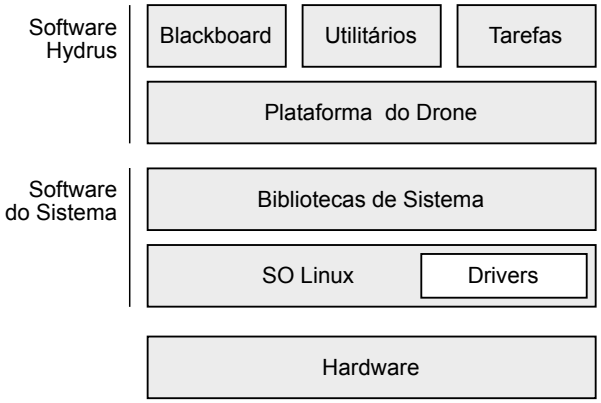
3.6.1 Estrutura do software

A Figura 13 mostra uma visão-geral da pilha de software final do veículo, baseada na distribuição de sistema operacional Raspbian. O software do Hydrus está

[1] Disponível em: https://github.com/lucaspcamargo/hydrus

organizado em quatro componentes príncipais: (i) padrão blackboard; (ii) utilitários; (iii) tarefas; e (iv) plataforma de veículo. A seguir, são descritos todos os componentes.

Figura 13 – Pilha de software do veículo

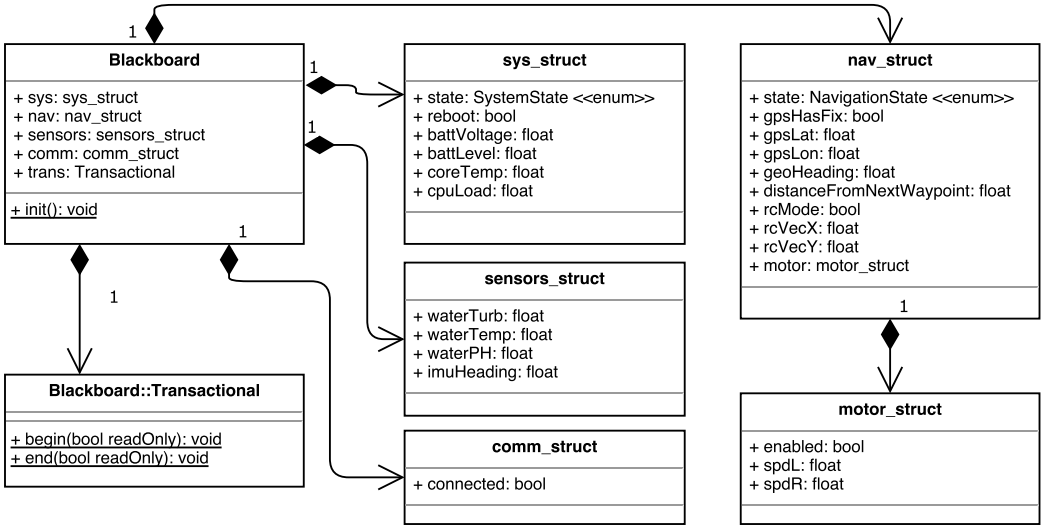


Fonte: do autor.

O padrão Blackboard for adotado para estruturar o firmware. Este padrão é definido como um conjunto de agentes especializados e independentes, que iteram sobre uma base de conhecimento comum, o que permite a coordenação de comportamentos de controle complexos (DONG; CHEN; JENG, 2005) (HO, 1990). A figura 14 ilustra a estrutura de implementação do padrão Blackboard a partir de um diagrama de classe da UML.

A classe Blackboard é composta por diversas estruturas aninhadas. A estrutura `sys_struct` contém variáveis de estado de gerenciamento do sistema: o estado do

Figura 14 – Implementação do Blackboard (diagrama de classes UML)



Fonte: do autor.

sistema, o que fazer ao fim da sequência de desligamento, tensão e nível da bateria, temperatura do SOC e carga da CPU. Já a estrutura `sensors_struct` agrega as variáveis de estado dos sensores, o que inclu temperatura, turbidez e pH da água, e o ângulo de curso magnético do veículo.

A estrutura `nav_struct` contém variáveis de estado do subsistema de navegação: o estado atual da navegação, as variáveis de estado do receptor de GPS (se há localização válida, latitude e longitude), o ângulo de curso corrigido do veículo, e a distância do próximo ponto de navegação. Também inclui as variáveis de controle do modo de radiocontrole. Ainda, a estrutura de variáveis de navegação agrega a estrutura `motor_struct`, que contém as variáveis de controle da atuação dos motores do veículo (liga/desliga, e velocidades de cada motor). Finalmente, a estrutura `comm_struct` inclui informações do subsistema de comunicação. No momento, a estrutura contém apenas uma variável, que indica se o veículo está conectado à estação-base ou não.

Como o blackboard é um recurso compartilhado, é importante o controle de acesso ao mesmo para que não haja acesso simultâneo por tarefas diferentes, o que pode causar inconsistências nos dados. A classe `Blackboard::Transactional` coordena o acesso ao Blackboard, através das funções `begin()` e `end()`. Ambas as funções recebem um argumento do tipo booleano que indica o tipo de acesso ao blackboard, que pode ser de leitura e escrita, ou de somente leitura. Todo acesso ao blackboard pelas tarefas do firmware deve ser encapsulado entre chamadas a esses métodos, o que delimita uma transação.

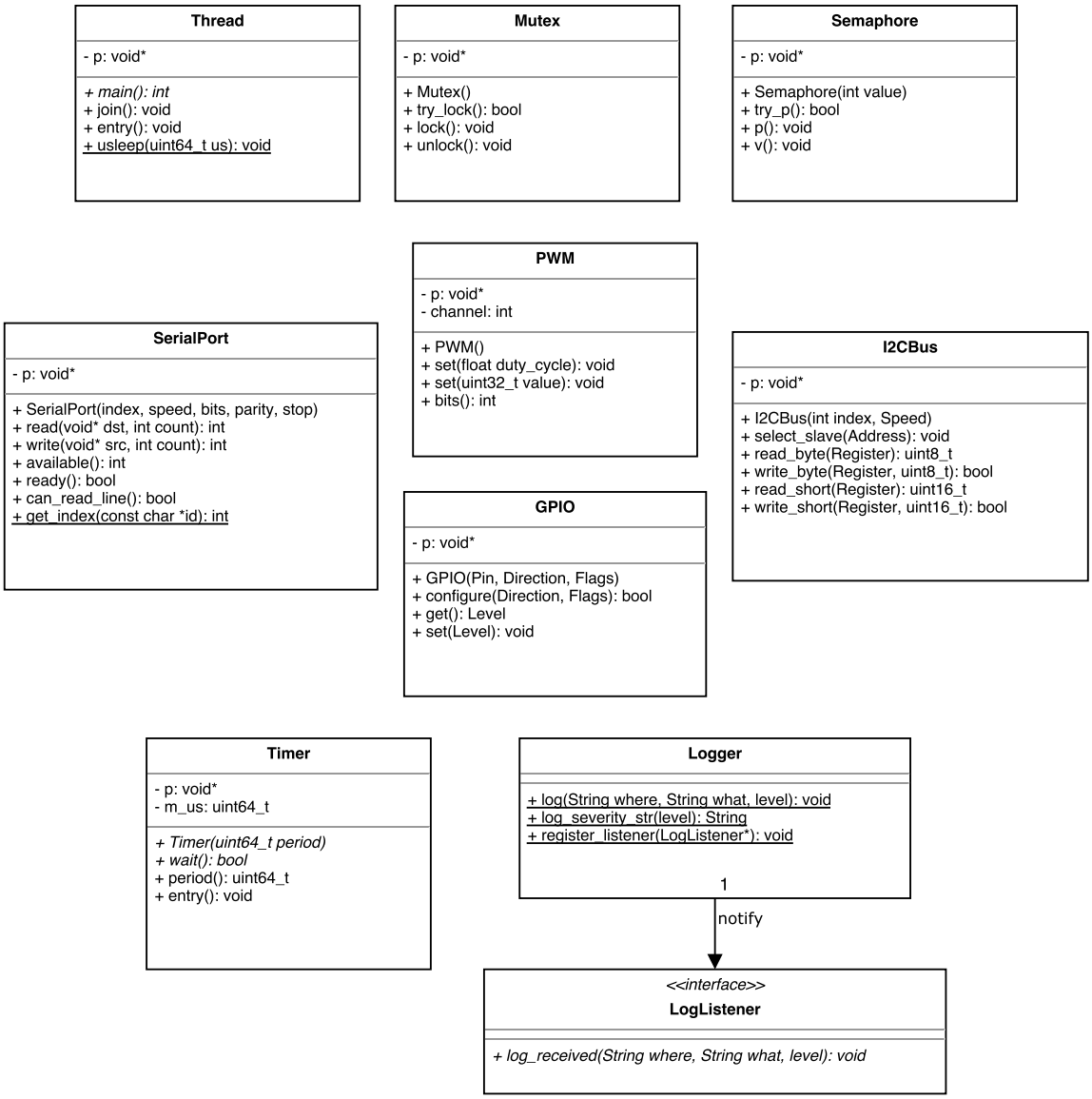
Para resguardar o recurso blackboard de acessos simultâneos, é implementado na classe Blackboard um mecanismo de bloqueio similar a uma trava Readers-writer, que é descrita por Raynal (2012, p. 74). Similarmente à trava Readers-writer, o mecanismo permite múltiplas operações de leitura simultâneas, e exclusivamente, uma operação de escrita. Para que não haja conflito entre leituras e escritas, é implementado também um mecanismo de bufferização duplo, em que uma instância de `Blackboard` é utilizada para leituras, e outra para escrita.

Na implementação do padrão Blackboard utilizada neste trabalho, o blackboard é uma estrutura central na memória, e os agentes são definidos como tarefas independentes. Os detalhes sobre a implementação das tarefas e o papel de cada tarefa é elaborado nas subseções seguintes.

3.6.2 Suporte à plataforma

Para melhorar a portabilidade do código, decidiu-se implementar uma camada de abstração dos recursos da plataforma de software do veículo. Isso inclui elementos do sistema operacional, como threads, primitivas de sincronização como mutexes e semáforos, acesso a barramentos de hardware (I2C, Serial), mecanismos de logging, entre outros. A Figura 15 mostra a interface dos componentes dessa camada de

Figura 15 – Interfaces de suporte à plataforma (diagrama de classes UML)



Fonte: do autor.

abstração através de um diagrama de classes da UML.

As classes `Thread`, `Mutex`, `Semaphore` implementam primitivas de execução paralela e sincronização. A interface busca fornecer apenas as funcionalidades necessárias para atender as necessidades da implementação do firmware. A classe `Thread` possui um método virtual `main()` que deve ser reimplementado por uma classe-filha, para prover a funcionalidade de execução desejada do thread. A classe `Mutex` implementa uma trava de exclusão mútua, com métodos para fazer um travamento, liberação, ou tentativa de travamento de um recurso compartilhado. Finalmente, a classe `Semaphore` é uma interface para um implementação de um semáforo Raynal (2012, p. 63).

Os mecanismos de entrada e saída atrelados ao hardware da plataforma são

implementados através das classes `SerialPort` (provê uma porta de comunicação no padrão serial), `PWM` (mecanismo de geração de onda modulada por pulso, utilizada no controle da motorização), `GPIO` (um pino de entrada/saída digital), e `I2CBus` (acesso ao barramento de entrada/saída no padrão I2C).

Para complementar os mecanismos de multiprocessamento e entrada e saída, são definidas as classes `Timer`, que provê um timer que expira e pode suspender um thread de execução até a expiração do período do timer, e a classe `Logger`, que encapsula o mecanismo de registo dos eventos que acontecem no veículo, para fins de gravação e diagnóstico. Associada à classe `Logger`, há a interface `LogListener`, que permite ao implementador auscultar os eventos recebidos pelo mecanismo de registro.

O padrão de software *ponteiro privado* foi utilizado para facilitar o mecanismo de implementação da camada de suporte à plataforma. Nesse padrão, cada classe tem como único membro de estrutura um ponteiro opaco, que é utilizado pela implementação da classe para referenciar uma estrutura interna, invisível ao usuário da classe. Uma vantagem desta abordagem é permitir à implementação flexibilidade quanto aos membros privados da classe e estruturas de dados auxiliares (importande devido à natureza dependente da plataforma da implementação). Porém, uma desvantagem é que isso impossibilita a alocação das instâncias das classes na memória de pilha do firmware.

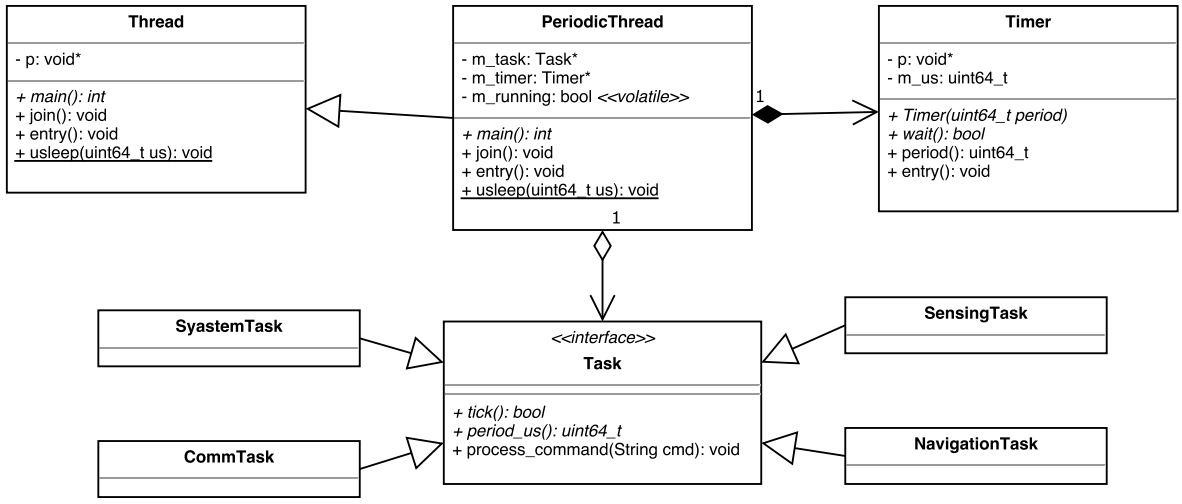
Pode-se perceber que esses componentes estão largamente desacoplados uns dos outros. Isso facilita o desenvolvimento de uma implemteção das interfaces para uma plataforma específica. Para este projeto, foi implementada apenas a plataforma Linux, voltada para a execução do software no hardware de referência (Raspberry Pi). Uma implementação consiste em um conjunto de arquivos-fonte C++ (extensão `.cpp`), que são ligados ao binário do firmware. Esses arquivos provêem as definições dos métodos das classes.

3.6.3 Suporte a tarefas

Cada tarefa do firmware é definida como uma thread de execução, e possui um papel bem-definido (por exemplo, ler todos os sensores). Para evitar sobrecarga de interfaces com funcionalidade, tarefas dividem funções mais especializadas em classes agregadas, por exemplo, classes representando um sensor ou atuador. O software do firmware define uma infraestrutura comum a todas às tarefas, ilustrada na Figura 16.

As tarefas definidas no firmware são: tarefa de sistema (`SystemTask`), de sensoreamento (`SensingTask`), de navegação (`NavigationTask`), e de comunicação (`CommTask`). Cada tarefa é uma especialização da classe `Task`, e implementa os métodos `tick()`, que executa uma iteração da tarefa, e `period_us()`, que estabelece o período de iteração da tarefa. Adicionalmente, a tarefa pode reimplementar o método `process_command()`, para receber e tratar comandos em formato de uma

Figura 16 – Infraestrutura de tarefas do firmware (diagrama de classes UML)



Fonte: do autor.

string provenientes da estação de controle.

Cada instância de tarefa é agregada à uma instancia da classe `PeriodicThread`, que por sua vez é uma especialização de `Thread`. `PeriodicThread` encapsula uma thread periódica que itera uma instância de tarefa específica, de acordo com seu período de iteração. Para esse fim, a classe faz uso do utilitário `Timer`, que bloqueia a thread em execução, e a libera periodicamente.

As subseções seguintes elaboram a implementação e as responsabilidades de cada tarefa do firmware.

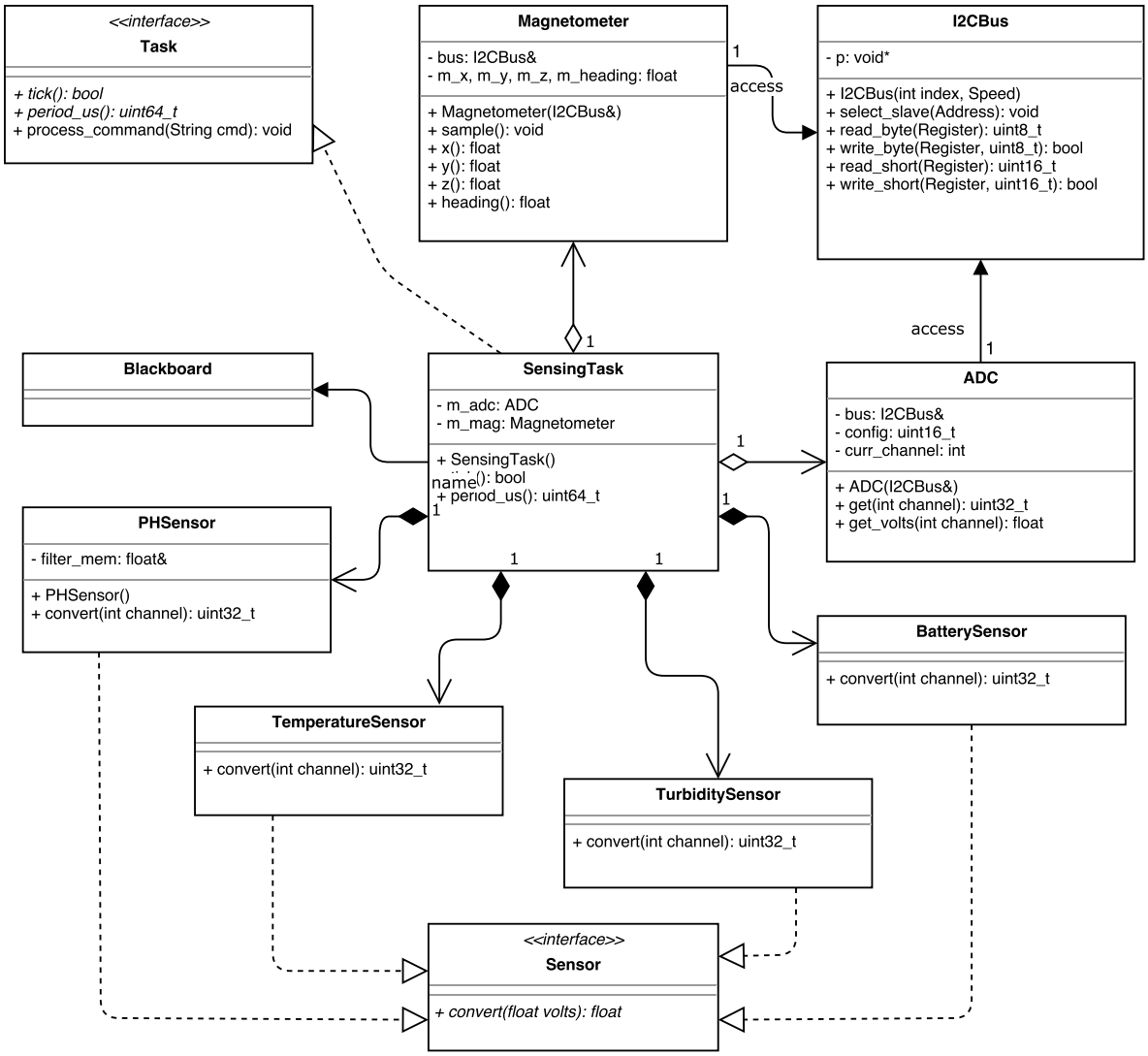
3.6.4 Tarefa de Sistema

Essa tarefa é responsável por gerenciar o comportamento e interfaces gerais do sistema. Gerencia o desligamento e reinício do sistema operacional, e manipula comandos de depuração. Também inicializa o blackboard, e é a primeira tarefa executada pelo escalonador. O principal ponto de interação desta tarefa com outras tarefas do sistema é a estrutura de dados `sys_struct` do blackboard. Das tarefas do firmware, a tarefa de sistema é a mais simples.

3.6.5 Tarefa de Sensoreamento

A tarefa de sensoreamento é responsável por obter todos os dados de sensores e escrevê-los ao blackboard. As leituras são feitas por classes que encapsulam os dispositivos: conversor analógico-digital (ADC) e magnetômetro. No caso das leituras obtidas por meio do ADC, as mesmas são convertidas de uma medição em tensão elétrica (V) para a grandeza apripriada. A Figura 17 mostra as classes relevantes.

Figura 17 – Tarefa de sensoreamento e classes associadas (diagrama de classes UML)

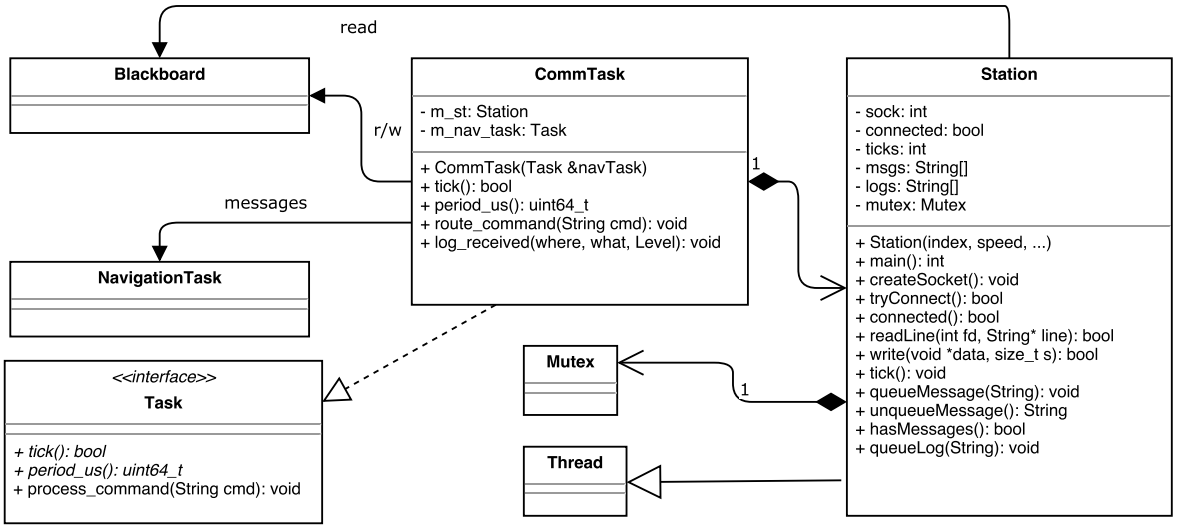


Fonte: do autor.

Os dois dispositivos de hardware encapsulados em classes são o conversor analógico digital (ADS1115, classe `ADC`) e o magnetômetro (HMC5883L, classe `Magnetometer`). Ambos os dispositivos são acessados através de um barramento I2C, encapsulado na classe `I2CBus` proveniente do suporte à plataforma. Todas as leituras são executadas de maneira sequencial, durante uma iteração da tarefa de sensoreamento.

As grandezas obtidas na leitura do conversor analógico-digital estão expressas em volts. A conversão das grandezas em unidades apropriadas (Celcius, NTU, etc) é feita por especializações da classe `Sensor`. São elas: `PHSensor`, `TemperatureSensor`, `TurbiditySensor`, e `BatterySensor`. A classe do sensor de pH agrega ainda, um filtro IIR, para amenização do ruído de leitura. As grandezas convertidas são escritas na estrutura `sensors_struct` (membro `sensors`) do blackboard.

Figura 18 – Tarefa de comunicação e classes associadas (diagrama de classes UML)



Fonte: do autor.

Os procedimentos de conversão das grandezas das leituras dos sensores, e o procedimento calibração dos mesmos, são detalhados na seção 3.7. Uma listagem de código da tarefa é apresentada no Apêndice D.

3.6.6 Tarefa de Comunicação

Esta tarefa trata da comunicação com a estação-base, conectando-se à rede e ao servidor TCP. Ela envia informações do blackboard e mensagens de registro (log), recebe comandos da estação-base, e os processa. A tarefa de comunicação também dá suporte ao relatório de informações para o sistema de arquivos. As classes principais que dão suporte a atividades de comunicação no Hydrus são mostradas na Figura 18. Uma listagem de código da tarefa é apresentada no Apêndice E.

A classe `Station`, que representa a conexao à estação-base, implementa grande parte da funcionalidade da tarefa de comunicação. Seu método `tick()` é invocado periodicamente pela classe `CommTask`. Há dois estados principais da classe: conectada e desconectada. Quando desconectada, a classe procura estabelecer uma conexão à estação-base como um cliente TCP, utilizando funcionalidades do kernel (chamadas de sistema `socket()`, `bind()`, etc). A conexão à rede WiFi é provida automaticamente pelo sistema operacional.

`Station` é uma especialização de `Thread`, e todas as operações de entrada e saída são efetuadas em um fluxo de execução separado. As mensagens passam de uma thread para a outra através de filas de mensagens, protegidas por exclusão mútua (uma instância de `Mutex`).

Quando há uma conexão válida, `Station` envia à estação-base informações

Tabela 3 – Comandos da estação-base ao veículo

Nome	Descrição	Mensagem
Desligar	Desliga o sistema operacional do veículo	$HALT
Reiniciar	Reinicia o sistema operacional do veículo	$REBOOT
Ligar radiocontrole	Ativa o controle-remoto dos motores	$RCON
Desligar radiocontrole	Desativa o controle-remoto dos motores	$RCOFF
Atualizar radiocontrole	Envia dados de radiocontrole	$RCUP,[eixo-x],[eixo-y]
Iniciar navegação	Dá início à sequência de navegação	$NAVBEGIN
Abortar navegação	Interrompe imediatamente a sequência de navegação	$NAVABORT
Rota de navegação	Envia os pontos da rota de navegação	$NAVROUTE,[lon],[lat],[lon],[lat],...
Mensagem ao GPS	Envia uma mensagem diretamente ao receptor GPS	$GPS,[mensagem]

Fonte: do autor.

do resgistro de eventos do veículo. Também envia periodicamente o blackboard do veículo, com todas as informações pertinentes ao estado do mesmo, em sua totalidade. Dessa maneira, a estação-base pode atualizar em sua interface todas as informações pertinentes ao veículo de maneira automática.

A estação-base também pode enviar mensagens ao veículo, que podem ser comandos ou rotas de navegação. Essas mensagens são disponibilizadas á tarefa de comunicação `CommTask`, que processa as mensagens, ou as envia para a tarefa apropriada. Na implementação atual, é necessário apenas rotear mensagens para a tarefa de navegação `NavigationTask`. Todos os comandos observam um padrão similar ao padrão NMEA de mensagens provenientes de equipamentos de navegação, porém sem o checksum de verificação. A Tabela 3 enumera os comandos aos quais o veículo responde.

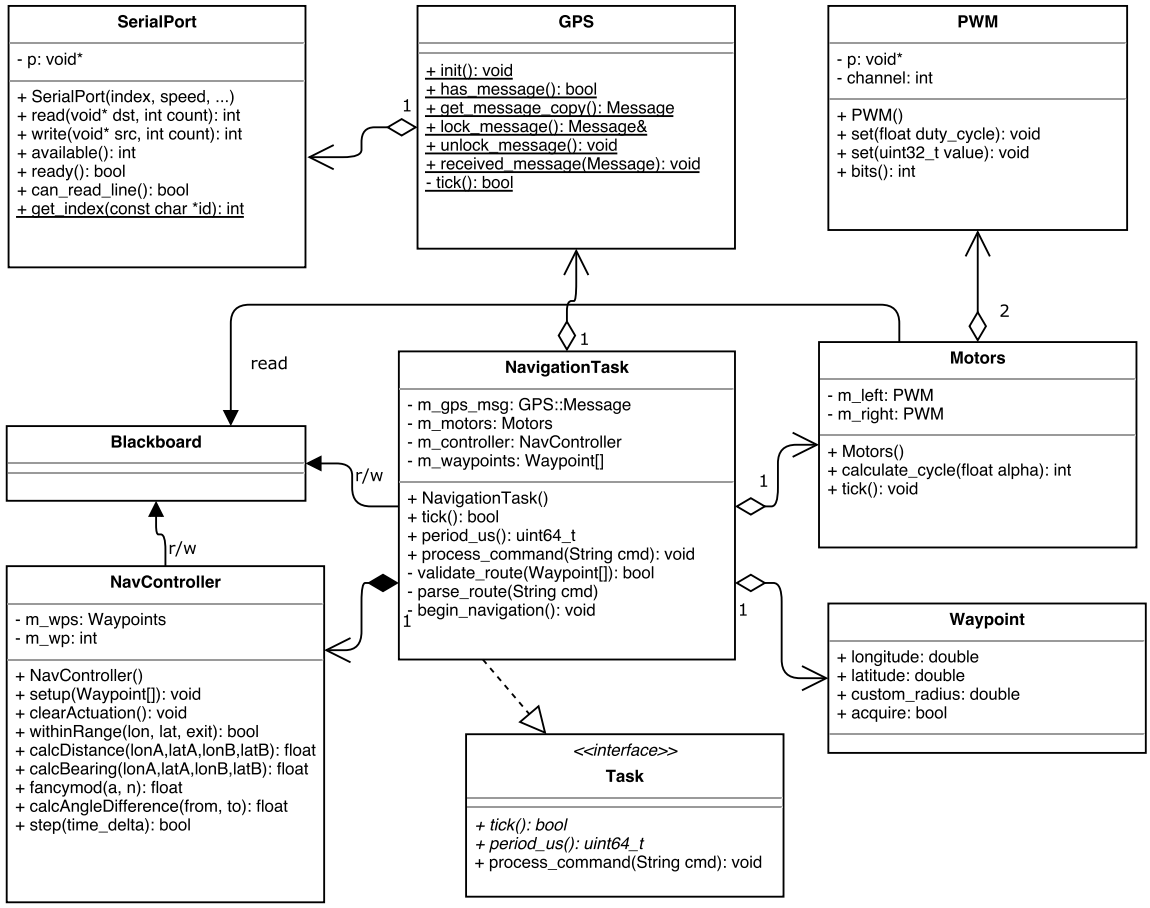
3.6.7 Tarefa de Navegaçao

A tarefa de navegação é responsável por avançar o controlador de navegação, interpretar e validar rotas de navegação, e também por processar o modo de rádio-controle para o controle manual dos motores. A Figura 19 mostra um diagrama de classe UML com os componentes principais do subsistema de navegação. Uma listagem de código da tarefa é apresentada no Apêndice F.

A classe `GPS` encapsula o receptor. A comunicação com o mesmo é feita por uma porta serial a 9600 baud, através de uma instância da classe `SerialPort`. As mensagens chegam do receptor no padrão NMEA 0183 (NMEA, 1995), e são decodificadas. Todos os métodos da classe são estáticos, por haver apenas um receptor GPS em dado sistema. Isso permite a alocação dos dados internos da classe em uma seção estática da memória do firmware, otimizando o código gerado pelo compilador. A instância da classe `NavigationTask` inicializa o módulo e processa as mensagens recebidas pelo GPS, atualizando o posicionamento do veículo no Blackboard.

Os motores do veículo estão representados pela classe `Motors`. Uma instância da mesma é agregada à instância da tarefa, que avança o controle do hardware através

Figura 19 – Tarefa de navegação e classes associadas (diagrama de classes UML)



Fonte: do autor.

da invocação periódica do método `tick()`. A classe `Motors` agrega duas instâncias da classe `PWM`, para o controle da velocidade de cada um dos motores por modualação de largura de pulso. O estado e velocidade de cada motor é determinado através da leitura direta do blackboard.

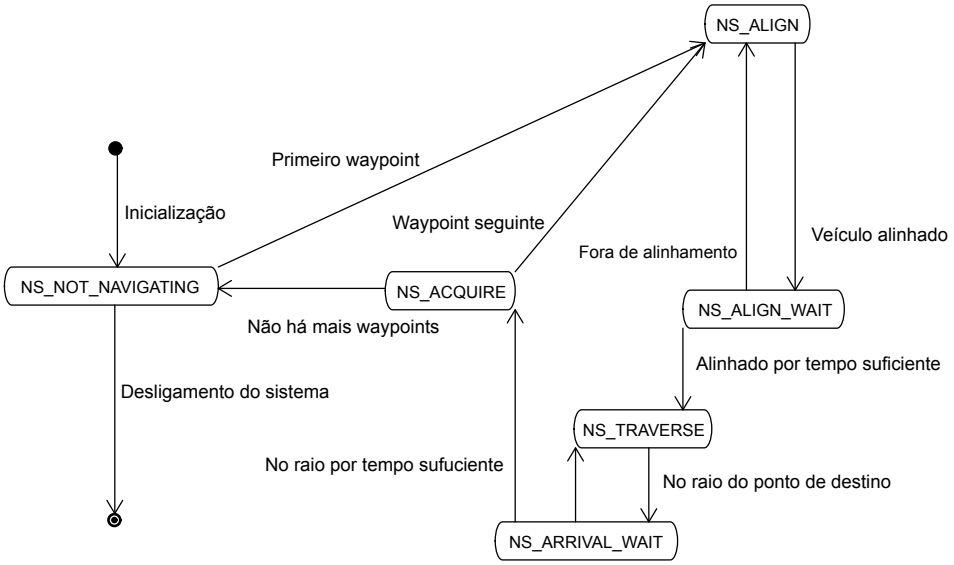
A tarefa agrega instâncias da classe `Waypoint` em uma lista, correspondendo aos pontos da rota de navageção atual programada no veículo. O controlador da navegação autônoma é implementado na classe `NavController`. A tarefa de navegação opera o controlador (métodos `setup()`, `withinRange()`, e `step()`). Porém, a lógica dos estados de navegação está encapsulada na classe controladora. A próxima seção elabora a implementação do controlador de navegação.

3.6.8 Sistema de Navegação (`NavController`)

O controlador de navegação é baseado em uma máquina de estados. Para cada estado, há um comportamento específico, ou sub-controlador. Portanto, o controlador de nível mais alto pode ser entendido como um multiplexador de controladores.

Os estados de navegação e suas relações estão mostradas em um diagrama

Figura 20 – Estados do controlador de navegação (diagrama de estados UML)



Fonte: do autor.

de estados na Figura 20. A seguir, descrevemos cada estado e seus relacionamentos.

- `NS_NOT_NAVIGATING` é o estado no qual o controlador de navegação está inativo, e não faz nada. Ele muda quando a tarefa de navegação informa ao controlador para iniciar uma sequência de navegação.
- `NS_ALIGN` é o primeiro estado de navegação de uma sequência de navegação. Ele ativa o controlador que aponta o veículo em direção ao ponto da rota (waypoint) alvo. Quando o alinhamento está dentro de uma faixa de tolerância, o estado muda para `NS_ALIGN_WAIT`. Se o alinhamento for perdido, o estado retorna para `NS_ALIGN` novamente. No entanto, se o alinhamento for mantido por tempo suficiente, o estado transiciona para `NS_TRAVERSE`.
- No estado `NS_TRAVERSE`, o controlador move o veículo em direção ao waypoint, tentando compensar quaisquer desalinhamentos vetorando a propulsão de acordo. Ele gradualmente desacelera conforme o veículo se aproxima do waypoint desejado. Enquanto o veícuulo estiover próximo do waypoint alvo, dentro do raio de tolerância, o mesmo mantém-se no estado `NS_ARRIVAL_WAIT`. Se o veículo entiver dentro do raio especificado por tempo suficiente, então é considerado dentro do waypoint alvo, e transiciona para o estado `NS_ACQUIRE`.
- No estado `NS_ACQUIRE` o controlador espera instantes para que as leituras de sensor se estabilizem, configura o waypoint seguinte como o próximo alvo, e então transiciona novamente para o estado de alinhamento `NS_ALIGN`. No entanto, se não houver mais waypoints a alcançar, termina então a sequência de navegação, tarnsicionando para o estado `NS_NOT_NAVIGATING`.

Uma listagem de código do controlador de navegação é apresentada no Apêndice D.

3.7 Leitura e calibração dos sensores

Esta seção apresenta as fórmulas de conversão de grandezas para os sensores, e o procedimento de calibração do eletrodo de pH.

3.7.1 Eletrodo de pH

Um eletrodo de pH operando em temperatura fixa possui dois parâmetros principais de calibração, uma tensão de deslocamento ΔV_{pH}, e uma sensibilidade K_{pH}, em V/pH. A sensibilidade K_{pH} é negativa, e para eletrodos combinados comerciais, como o utilizado no veículo, o valor típico é $-60\ mV/pH$. A resposta do sensor é linear. A tensão de deslocamento representa a tensão obtida do dispositivo quando o eletrodo está imerso em um meio de pH 7,0. Essa tensão inclui o offset inserido no sinal através do circuito de bufferização do sinal do eletrodo. Já a sensibilidade expressa a grandeza em pH relacionada à mudança do sinal de leitura em 1V. Dessa maneira, considerando V_{pH} a tensão obtida pelo conversor analógico/digital, a função de conversão de leitura em Volts para o respectivo valor em pH é dada por:

$$pH(V_{pH}) = \frac{7 + (V_{pH} - \Delta V_{pH})}{K_{pH}}.$$

Calibrou-se o eletrodo de pH a partir do uso de duas soluções-tampão de pH conhecido, especificamente pH 4.0 e 7.0. O procedimento consiste em medir a solução de pH 7.0 para a determinação da tensão de deslocamento ΔV_{pH}. Em seguida a sensibilidade K_{pH} é determinada ao medir-se a solução-tampão de pH 4.0. Sendo a grandeza de tensão obtida com pH 4.0 $V_{pH}^{4.0}$, a sensibilidade pode ser expressa por:

$$K_{pH} = \frac{\Delta V_{pH} - V_{pH}^{4.0}}{7.0 - 4.0}.$$

Com esses, dois pontos, é estabelecida uma curva de calibração (uma reta). Para que não haja efeitos da temperatura, a calibração foi feita em ambiente de temperatura controlada ($T_{env} = 25^oC$). A Tabela 4 mostra os valores obtidos no processo de calibração da sonda de pH. Destaca-se o valor de sensibilidade $K_{pH} = -55,6mV/pH$, consistente com o valor nominal esperado ($-60mV/pH$).

3.7.2 Sensor de turbidez

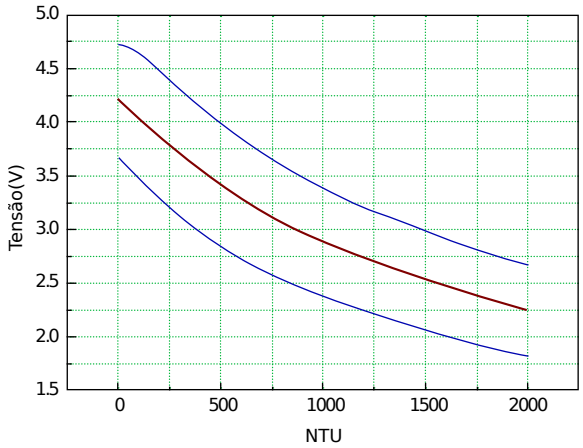
O sensor de turbidez utilizado no veículo, o TSW-10, possui uma curva de resposta especificada em sua folha de dados (Figura 21). Porém, não foi fornecida

Tabela 4 – Dados de calibração da sonda de pH

Grandeza	**Variável**	**Valor**	**Unidade**
Tensão de deslocamento	ΔV_{pH}	1,821	V
Tensão para pH 4	$V_{pH}^{4.0}$	1,988	V
Sensibilidade	K_{pH}	-55,6	mV/pH

Fonte: do autor.

Figura 21 – Curva de resposta do sensor de turbidez TSW-10



Fonte: adaptado de Amphenol Advanced Sensors (2014).

nenhuma fórmula de conversão. Verificou-se que a curva de resposta do sensor se assemelha a uma parábola. Portanto, foi utilizado o método da regressão quadrática para a obtenção de uma função que modelasse o comportamento do sensor, mapeando a tensão de saída V_{turb} em uma grandeza de turbidez T_u em NTU. A partir do gráfico da curva obteve-se a função:

$$T_u(V_{turb}) = 261,05 \cdot V_{turb}^2 - 2607,5 \cdot V_{turb} + 6367.$$

3.7.3 Sensor de temperatura

O sensor de temperatura TMP36 utilizado na medição da temperatura da água é um circuito integrado de fácil aplicação. Considerando uma tensão de deslocamento ΔV_{temp}, e uma sensibilidade K_{temp}, em $V/^oC$, a especificação ((ANALOG, 2015)) estabelece os valores $0,5\ V$ e $10\ mV/^oC$, respectivamente. Portanto, o valor da temperatura da água em função da tensão obtida é dado por:

$$T_mp(V_{temp}) = \frac{V_{temp} - \Delta V_{temp}}{K_{temp}} = 100 \cdot (V_{temp} - 0.5).$$

Figura 22 – Interface do usuário do software da estação-base

Fonte: do autor.

3.8 Software da estação-base

O software da estação-base serve de controlador para o veículo. Permite traçar as rotas de medição e iniciar o processo de navegação autônoma, e a vizualização de informações importantes do veículo, como leituras dos sensores e nível de bateria. Os principais requisitos que guiaram seu desenvolvimento são os seguintes:

- Ser fácil de usar.
- Executar em um computador pessoal com interface gráfica.
- Expor toda a funcionalidade do veículo.
- Não interromper ou distrair o usuário.

Para cumprir os requisitos acima delineados, escolheu-se implementar o software da estação-base em C++, utilizando-se do kit de ferramentas de interface do usuário Qt (QT, 2017). O uso da mesma linguagem utilizada na implementação do firmware permite ainda o reuso de código, especialmente a estrutura de dados `Blackboard`. A Figura 22 mostra uma captura de tela da janela principal do programa.

O elemento central da interface é um mapa interativo, fornecido pelo projeto de código-aberto Marble (MARBLE, 2016). Os dados cartográficos são fornecidos pela comunidade OpenStreetMap (OPENSTREETMAP, 2017), e as renderizações são fornecidas pelo Marble através da internet. Essa especialização do mapa mostra a rota planejada atualmente, a posição atual do cursor, e a posição GPS atual do veículo.

No topo, uma barra de ferramentas permite ao usuário comandar ao veículo que inicie a navegação, habilitar o modo de controle-remoto para operar o veículo através deum joystick USB, e adicionar e excluir rotas de navegação.

À esquerda do mapa, está o painel de status do veículo, que mostra informações do sistema, nível de bateria, as informações do receptor GPS, e demais informações de sensores. Na parte direita da interface, se encontra o painel de rotas, que mostra os pontos de navegação, e informações sobre a rota atual, como distância e duração estimada de navegação. O painel permite adicionar ou remover pontos de navegação a partir da posição do cursor no mapa.

A parte inferior da interface é um registro dos eventos que ocorreram no veículo, atualizado em tempo real. Um botão permite o salvamento do registro para um arquivo.

3.8.1 Modo de operação

Ao ligar o veículo, o mesmo conecta-se automaticamente à estação-base e ao software de controle. Através do painel de rotas o usuário define os pontos pelos quais o veículo deve passar, ou seleciona uma rota pré-programada. Se tudo estiver correto, o usuário pressiona o botão para iniciar o processo de navegação.

4 PRINCIPAIS RESULTADOS

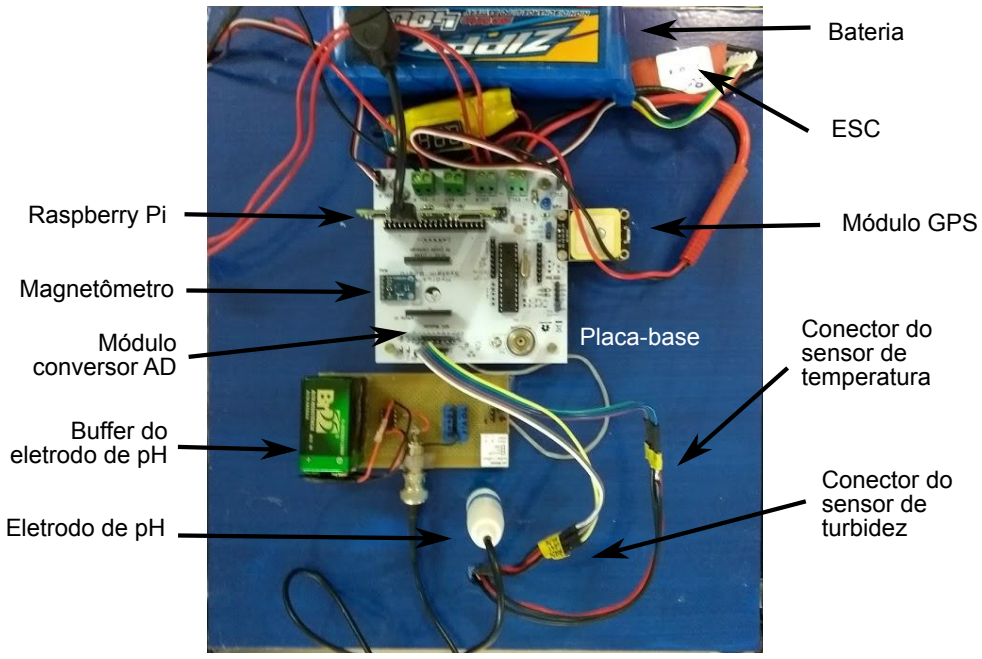
Neste capítulo é discutido o principal resultado deste trabalho, que é a validação dos sistemas implementados. A validação foi realizada primeiramente por meio de testes em bancada. Em seguida, foram feitos testes de navegação e aquisição de dados. Foi testado também o registro de grandezas de medição obtidas ao longo de uma rota.

4.1 Teste em bancada

O veículo construído foi colocado em bancada para testes. A Figura 23 ilustra a configuração final do hardware, e uma visão do veículo como um todo é apresentada no Apêndice C.

A placa base é o retângulo branco ao centro da plataforma do barco. A ela se conectam todos os componentes do sistema. Na borda de potência da placa (bornes

Figura 23 – Hardware do veículo em bancada



Fonte: do autor.

Tabela 5 – funcionalidades do veículo testadas em bancada

Funcionalidade	Procedimento	Resultado Esperado
Potência	Ligar o interruptor	Todos os sistemas devem energizar-se
Boot	Ligar o interruptor	O processo de boot do sistema deve iniciar-se
Inicialização	Ligar o interruptor	Ao fim do processo de boot o firmware deve executar
Conexão	Ligar o barco	O barco deve conectar-se automaticamente
Reconexão	Desconectar o barco da estação	O barco deve reestabelecer a conexão automaticamente
Comunicação	Conexão do barco	O barco deve enviar dados à estação constantemente
Comunicação	Conexão do barco	Todos os eventos no registro do barco são transmitidos
Sensoreamento	Imergir sonda de pH	O valor obtido do eletrodo deve ser consistente com o meio
Sensoreamento	Alterar temperatura	O valor obtido do sensor de temperatura deve ser consistente
Sensoreamento	Imergir turbidímetro	O valor obtido do sensor de turbidez deve ser consistente
Controle	Ativar o radiocontrole	O modo deve permitir o controle manual dos motores do barco
Controle	Desativar o radiocontrole	Desativar o radiocontrole deve desligar completamente os motores
Navegação	Iniciar navegação	O veículo deve interpretar a rota corretamente e mudar de estado
Passivo	Ativação dos motores	Os motores nunca devem se ativar sem motivo
Passivo	Controle dos motores	O sinal PWM aos ESCs não pode ser interrompido

Fonte: do autor.

Tabela 6 – Resultados dos testes de medição de pH

pH da solução	pH obtido	Erro	Tensão na saída (V)
3,3	3,52	0,22	2,017
4,0	4,09	0,09	1,985
7,0	6,95	-0,05	1,822

Fonte: do autor.

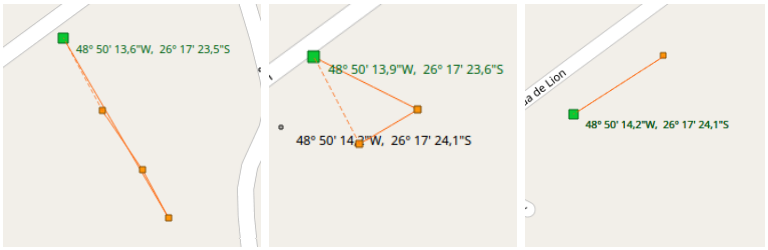
verdes) se conectam a bateria, os ESCs, e um interruptor liga/desliga. Mais ao centro placa está o Raspberry Pi, o magnetômetro. O módulo GPS se encontra em uma lateral. Na borda oposta à borda de potência, se encontra a seção analógica. Na linha que separa a seção analógica da seção digital, está o conversor analógico/digital (AD). O circuito integrado maior, acoplado diretamente à placa, é um microcontrolador auxiliar, no momento não utilizado. Esta versão do hardware utiliza o circuito alternativo para o tratamento de sinal do eletrodo de pH, apresentado no capítulo anterior e no Apêndice B.

Foram feitos diversos testes em bancada para a validação das funcionalidades do veículo. A Tabela 5 apresenta um lista completa.

4.1.1 Testes dos sensores

Para o teste de medição do pH utilizamos três soluções de pH conhecido que acompanharam um sensor de pH comercial, para servir de referência. Uma das soluções possui pH pH 3,3, outra possui pH 4,0, e a última possui pH 7,00. O teste consistiu em submergir o eletrodo da sonda de pH, e verificar as tensões geradas na saída do circuito de condicionamento de sinal. O sinal foi comparado com as tensões obtidas no conversor AD e o valor de pH calculado pelo sistema foi comparado com o valor conhecido da solução. O teste foi feito em ambiente de temperatura controlada (25 graus Celsius). A Tabela 6 apresenta os resultados.

Figura 24 – Algumas das rotas testadas



Fonte: do autor.

No teste do turbidímetro, o sensor foi testado de maneira qualitativa, uma vez que não haviam disponíveis soluções de referência. O sensor foi testado em meios aquosos com diferentes quantidades de partículas em suspensão, na ordem de menos partículas para mais partículas. A resposta do sensor e o valor obtido pelo sistema foi consistente, e respeita a curva obtida da folha de dados do componente.

4.2 Teste de integração e plotagem de dados

O veículo foi testado iterativamente durante a fase de integração, e erros de código e projeto de hardware e software foram corrigidos continuamente. O controlador de navegação e o algoritmo de vetoração de empuxo foram calibrados dessa maneira, para compensar a dinâmica do veículo.

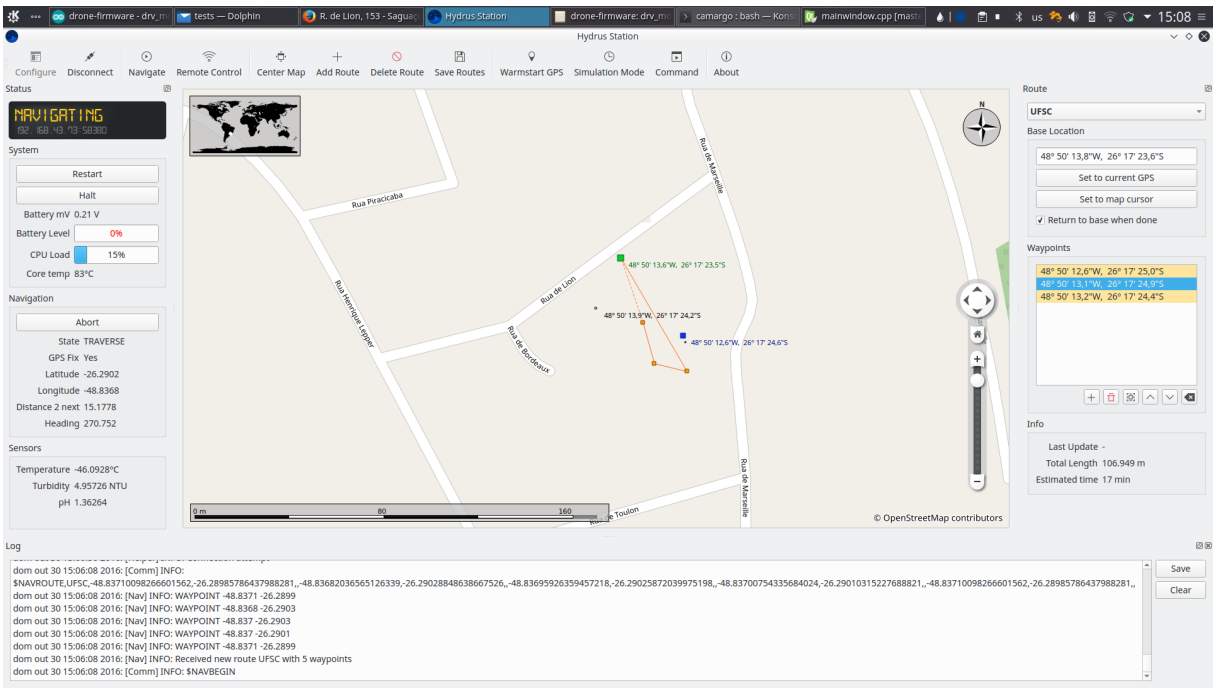
O procedimento de teste final consistiu em completar uma sequência de navegação com êxito em um lago. Pontos-alvo de navegação (waypoints) foram selecionados no mapa utilizando o software da estação-base. O computador da estação-base foi conectado ao veiculo através de rede WiFi durante o teste, o que permitiu a verificação de todas as fazes da navegação, e detectar problemas em potencial. A Figura 25 mostra uma captura de tela da estação-base durante uma sessão de navegação, para uma das rotas testadas.

Durante os teste de navegação verificou-se o correto funcionamento dos algoritmos de controle de navegação e aquisição dos dados dos sensores. Algumas das rotas testadas são mostradas na Figura 24. Estas foram selecionadas para verificar a habilidade do veículo de mudar de direção durante a navegação, fazer curvas apertadas, e manter um curso adequado durante a travessia.

Durante todo o período de testes em que o veículo estava na água, foi possível observar que os sensores estavam retornando dados, e funcionando como o esperado. Uma mudança nos valores de medição era visível em tempo real na aplicação da estação-base.

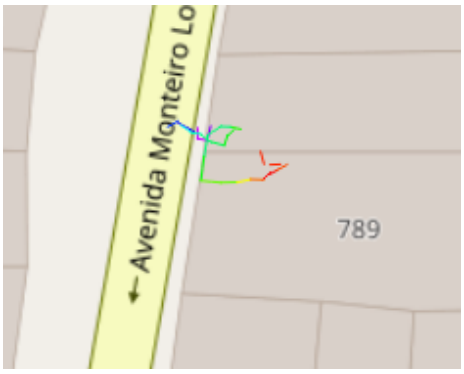
Recursos de plotagem de dados foram introduzidos à estação-base após os testes de navegação. Foi possível verificar o funcionamento correto da plotagem no

Figura 25 – O sotfware da estação-base durante uma sessão de navegação



Fonte: do autor.

Figura 26 – Dados de temperatura coletados, plotados no software da estação-base



Fonte: do autor.

mapa ao mover o barco por um caminho simulado e logar dados de sensores. Os registros foram então exportados do veículo para o computador para plotagem. A Figura 26 mostra o caminho plotado resultante. O matiz da linha plotada representa o valor em proporção aos valores mínimo e máximo, azul sendo o valor mínimo, verde um valor intermediário, e vermelho um valor máximo.

5 CONSIDERAÇÕES FINAIS

Grande parcela da população brasileira não tem acesso adequado a fontes de água potável, um cenário que muitas vezes está associado à falta de disponibilidade de saneamento básico adequado. Falta ao país garantir à totalidade de sua população esses direitos essenciais. Neste contexto, é crucial às autoridades e às prestadoras de serviços o monitoramento dos recursos hídricos.

Neste contexto, a disponibilidade de dados de qualidade sobre a saúde dos rios, lagos, e reservatórios torna-se crucial para o desenvolvimento do país. A consolidação de sistemas embarcados de mais alta complexidade e menor custo tem o potencial de contribuir com a melhora dos processos de obtenção de dados nas mais diversas áreas, inclusive no monitoramento hídrico. Observa-se a necessidade de medições constantes, distribuídas espacialmente, e automatizadas.

Este trabalho apresentou o Hydrus – um veículo aquático autônomo para fins de monitoramento hidrológico. O trabalho resultou em um veículo funcional, seu hardware associado, o firmware e software de suporte. O veículo foi projetado, construído, e testado. As funcionalidades de hardware e software foram validadas, incluindo medição de grandezas, aquisição de dados, navegação autônoma, e controle. A plataforma é capaz de fornecer dados de aquisições, associados a dimensões espaciais e temporais. Portanto, consideram-se cumpridos os objetivos deste trabalho.

Como desenvolvimento futuro, sugere-se o aprimoramento do projeto e do sistema desenvolvido, e sua aplicação em um estudo de caso de algum recurso hídrico real. Seria interessante a adição de sensores adicionais e importantes para determinação mais precisa do estado em que encontra o recurso hídrico, principalmente no que diz respeito à disponibilidade e demanda de oxigênio.

Sugere-se também o retrabalho e a melhora das ferramentas para coleta e análise de dados, que poderiam ser integradas a um sistema de gerenciamento de informações geográficas (GIS).

REFERÊNCIAS

AMPHENOL ADVANCED SENSORS. **TSW-10 Turbidity Sensor**. [S.l.], 2014. Rev. 2.

ANALOG DEVICES. **1.25 V Micropower, Precision Shunt Voltage Reference**. [S.l.], 2007. Rev. 0.

ANALOG DEVICES. **Precision Micropower, Low Noise CMOS Rail-to-Rail Input Output Operational Amplifiers**. [S.l.], 2008. Rev. C.

ANALOG DEVICES. **Low Voltage Temperature Sensors**. [S.l.], 2015. Rev. H.

ASSOCIAçãO BRASILEIRA DE NORMAS TéCNICAS. **NBR 10007**: Amostragem de resíduos sólidos. Rio de Janeiro, 2004. 22 p.

BOVET, D. P.; CESATI, M. **Understanding the Linux Kernel**. [S.l.]: O'Reilly, 2005. v. 1.

DONG, J.; CHEN, S.; JENG, J. J. Event-based blackboard architecture for multi-agent systems. In: **International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II**. [S.l.: s.n.], 2005. v. 2, p. 379–384 Vol. 2.

HO, C.-S. Development of a meta-blackboard shell. In: **[1990] Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence**. [S.l.: s.n.], 1990. p. 544–550.

HONEYWELL INTERNATIONAL. **3-Axis Digital Compass IC HMC5883L**. [S.l.], 2013. Rev. E.

IEEE COMPUTER SOCIETY. **IEEE Std 802.11g - 2003**: Wireless lan medium access control (mac) and physical layer (phy) specifications. New York, NY, 2003. 51 p.

KAPLAN, E. D.; HEGARTY, C. J. **Understanding GPS**: principles and applications. 2. ed. Norwood, Massachusetts: Artech House, 2006. v. 1.

KICAD DEVELOPER TEAM. **KiCad EDA Homepage**. 2017. Disponível em: <http://kicad-pcb.org/>. Acesso em: 1.11.2017.

KUPHALDT, T. R. **Lessons In Electric Circuits**. 5. ed. [S.l.]: Open Book Project, 2000. v. 1.

LIMA, E. P. **Planos e Técnicas de Amostragem**. 1. ed. Pelotas, RS, 2006. v. 1.

LINUX FOUNDATION, THE. **Home - The Linux Foundation**. 2017. Disponível em: <https://www.linuxfoundation.org/>.

MARBLE GLOBE PROJECT. **Marble - find your way and explore the world**. 2016. Disponível em: <https://marble.kde.org/>.

NASCIMENTO, U. R. do. **Estrevista de pesquisa concedida a Lucas Pires Camargo**. Joinville: [s.n.], 2016. Comunicação pessoal.

NATIONAL MARINE ELECTRONICS ASSOCIATION. **NMEA 0183, Standard for Interfacing Marine Electronic Devices**. NMEA National Office, 1995. Disponível em: <https://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp>.

OLIVEIRA, R. d.; FERNANDES, C. **Estudo e Determinação do "pH"**. 2012. Disponível em: <http://www.dec.ufcg.edu.br/saneamento/PH.html>. Acesso em: 1.11.2017.

OPENSTREETMAP PROJECT, THE. **OpenStreetMap Homepage**. 2017. Disponível em: <http://www.openstreetmap.org/>.

ORGANIZAçãO DAS NAçõES UNIDAS. **The Human Right to Water and Sanitation**. UN-water decade programme on advocacy and communication and water supply and sanitation collaborative council, Zaragoza, 2015. Disponível em: <http://www.un.org/waterforlifedecade/pdf/human_right_to_water_and_sanitation_media_brief.pdf>.

POBKRUT, T.; EAMSA-ARD, T.; KERDCHAROEN, T. Sensor drone for aerial odor mapping for agriculture and security services. In: **2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)**. [S.l.: s.n.], 2016. p. 1–5.

QT COMPANY, THE. **Qt: Cross-Platform Software Development for Embedded and Desktop**. 2017. Disponível em: <http://qt.io/>. Acesso em: 3.11.2017.

RAIMONDI, F. M. et al. A innovative semi-immergible usv (si-usv) drone for marine and lakes operations with instrumental telemetry and acoustic data acquisition capability. In: **OCEANS 2015 - Genova**. [S.l.: s.n.], 2015. p. 1–10.

RASPBERRY PI FOUNDATION, THE. **Teach, Learn, and Make with Raspberry Pi**. 2017. Disponível em: <http://raspberrypi.org/>.

RASPBIAN PROJECT, THE. **FrontPage - Raspbian**. Disponível em: <https://www.raspbian.org/>.

RAYNAL, M. **Concurrent Programming: Algorithms, Principles, and Foundations**. [S.l.]: Springer, 2012. ISBN 978-3-642-32027-9.

RCTIMER POWER MODEL CO.,LTD. **RCTimer 30A SimonK Firmware Multicopter ESC SK30A**. 2017. Disponível em: <http://rctimer.com/product-779.html>. Acesso em: 3.11.2017.

SISTEMA NACIONAL DE INFORMAçõES SOBRE SANEAMENTO. **Diagnóstico dos Serviços de Agua e Esgotos – 2014**. Brasilia, 2014. Disponível em: <http://www.snis.gov.br/downloads/diagnosticos/ae/2014/Diagnostico_AE2014.zip>.

TEXAS INSTRUMENTS. **TL08xx JFET-Input Operational Amplifiers**. [S.l.], 1977. Rev. mai. 2015.

TEXAS INSTRUMENTS. **ADS111x Ultra-Small, Low-Power, I 2C-Compatible, 860-SPS, 16-Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator**. [S.l.], 2016. Rev. dec. 2016.

THOMAS, A. **Small RC boat propeller**. 2015. Disponível em: <https://www.thingiverse.com/thing:1024952>. Acesso em: 7.11.2017.

UPTON, E.; HALFACREE, G. **Raspberry Pi User Guide**. 1. ed. Chichester, West Sussex: John Wiley and Sons, 2012. v. 1.

VELEZ, F. J. et al. Wireless sensor and networking technologies for swarms of aquatic surface drones. In: **2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)**. [S.l.: s.n.], 2015. p. 1–2.

WALLI, S. R. The posix family of standards. **StandardView**, ACM, New York, NY, USA, v. 3, n. 1, p. 11–17, mar. 1995. ISSN 1067-9936. Disponível em: <http://doi.acm.org/10.1145/210308.210315>.

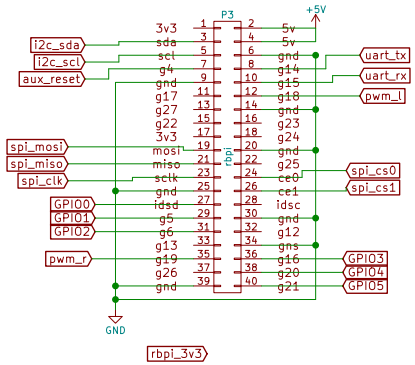
WEINER, R. F.; MATTHEWS, R. A. **Environmental Engineering**. 4. ed. [S.l.]: Elsevier, 2003. v. 1.

ZIMMER, B. **Diagnóstico dos Serviços de Agua e Esgotos – 2014**. The Wall Street Journal: [s.n.], 2016. Disponível em: <http://www.wsj.com/articles/SB10001424127887324110404578625803736954968>. Acesso em: 25 nov. 2016.

APÊNDICE A

O esquemático da placa-base é incluído a seguir.

Raspberry Pi Zero Connector

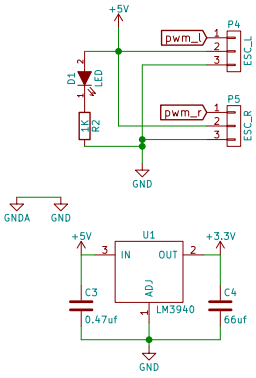


rbpi_3v3

(Better leave the Pi's 3V3 Pins alone)

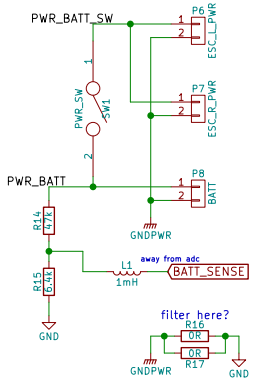
Main Processor

ESCs AND POWER



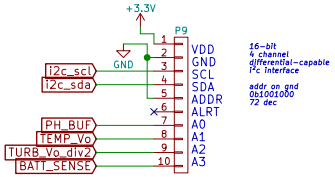
Both ESCs feed the main +5V rail

BATTERY POWER

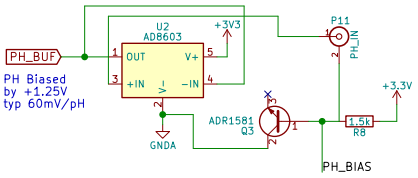


Simple battery power switch

ADC MODULE
ADS1115

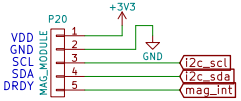


PH SENSOR + SIGNAL BUFFER

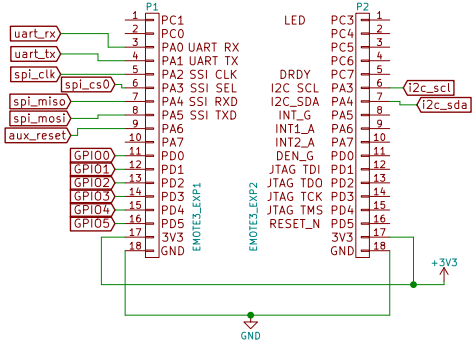


PH Biased
by +1.25V
typ 60mV/pH

MAG MODULE
HMC5883L

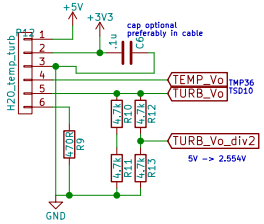


EMOTE 3 CONNECTOR



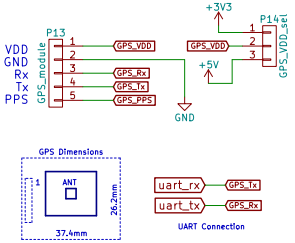
Processor for Research
Replaces Main

TURBIDITY SENSOR + WATER TEMP

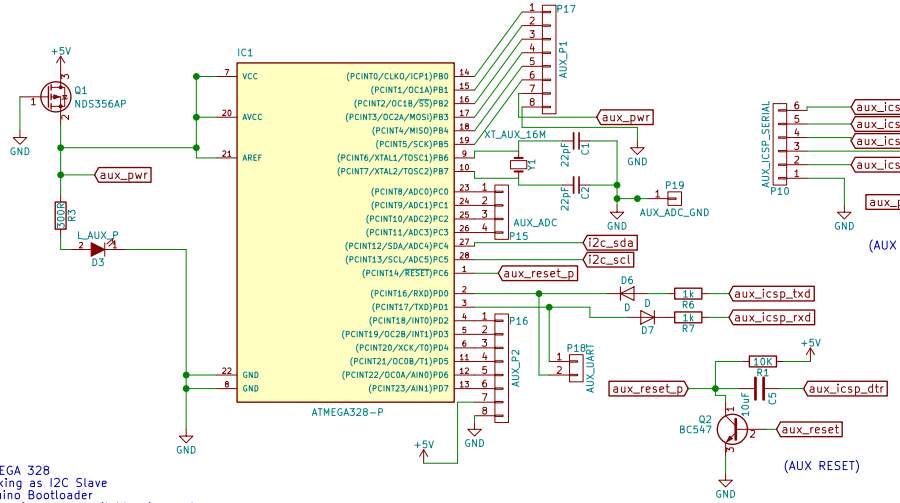


Custom cable and connector to be made, just like before

GPS MODULE



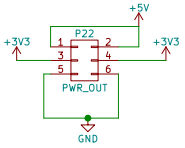
AUXILIARY MCU



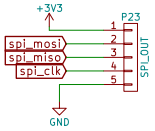
ATMEGA 328
Working as I2C Slave
Arduino Bootloader
Allows for more available pins and
connection to time-sensitive devices
(eg ultrassound) and also the device
interface (screen, buttons)

GPIO ALLOCATION

PWR_OUT



Expansion stuff



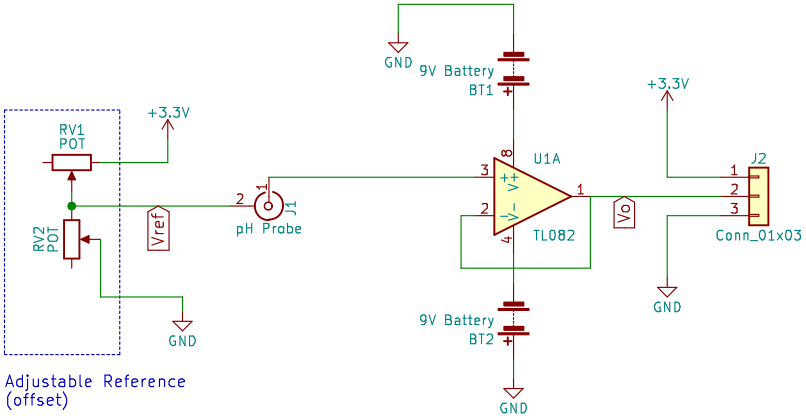
UFSC		
Sheet: /		
File: drone-v2-systemboard.sch		
Title: Hydrus V2 System Board		
Size: A3	Date: 2017	**Rev: 1**
KiCad E.D.A. kicad 4.0.7-e2-6376588ubuntu16.04.1		Id: 1/1

APÊNDICE B

O esquemático da placa de sinal alternativa para o eletrodo de pH é incluído a seguir.

SIMPLE PH METER WITH ADJUSTABLE OFFSET VOLTAGE
based on the TL082 OpAmp as a signal buffer

Adapted from http://www.66pacific.com/ph/simplest-ph.aspx



Adjustable Reference
(offset)

Vcc: 3.3V
Desired Vref: 1.65V
Trimpots are of 10 kOhm

48

Adapted from http://www.66pacific.com/ph/simplest-ph.aspx
based on the TL082 OpAmp as a signal buffer
UFSC – CTJ

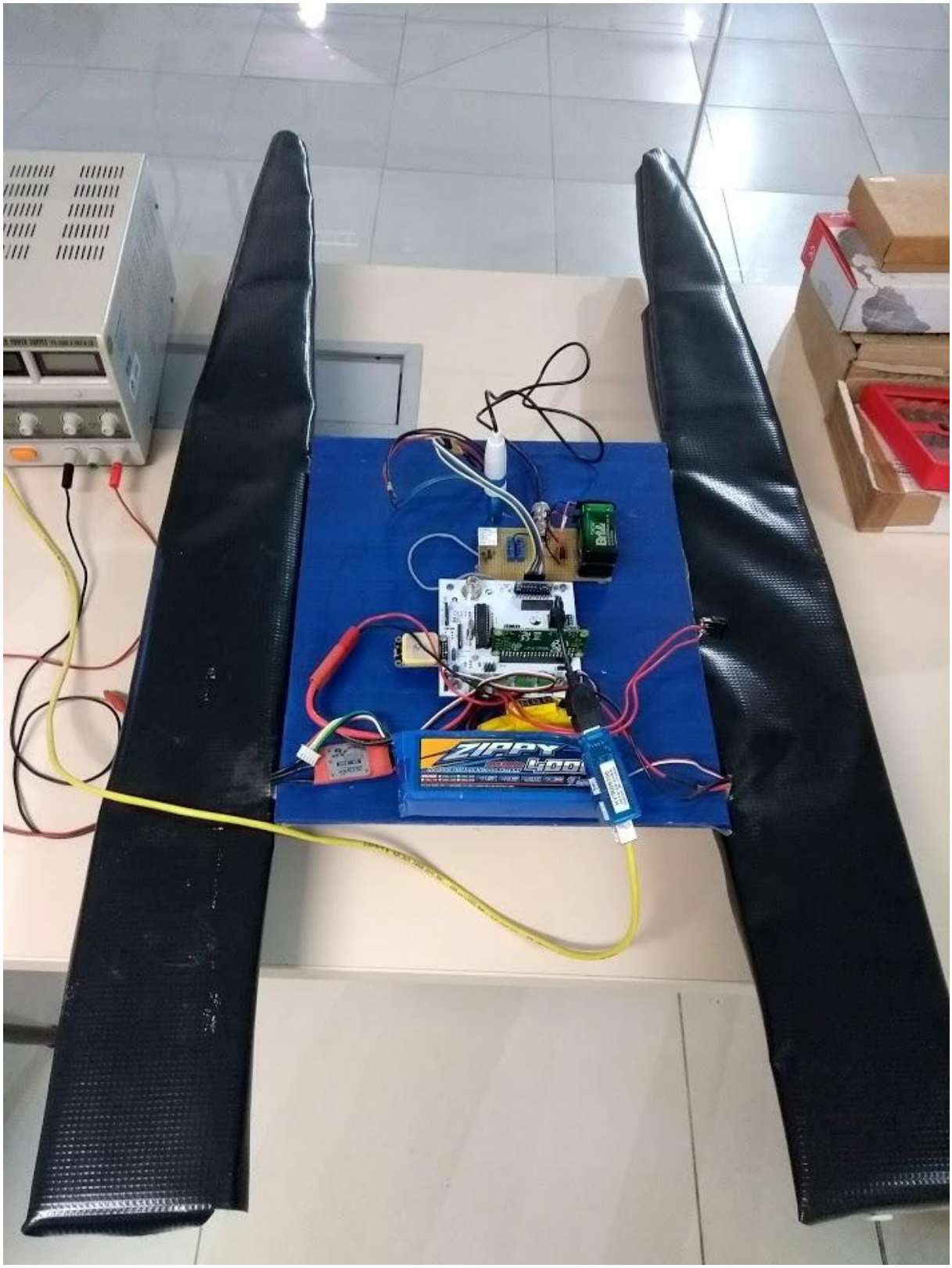
Sheet: /
File: drone-alt-phmeter.sch

Title: SIMPLE PH METER WITH ADJUSTABLE OFFSET VOLTAGE

Size: A4	Date: 2017-11-11	**Rev: 1**
KiCad E.D.A. kicad 4.0.7-e2-637658ubuntu16.04.1		Id: 1/1

APÊNDICE C

Fotografia do veículo em bancada.



APÊNDICE D

Listagem de Código: Tarefa de Sensoreamento

```
#pragma once

#include "hydrus-config.h"
#include "hydrus-traits.h"
#include "hydrus-math.h"

#include "task.h"
#include "platform/logger.h"

#include "dev-magnetometer.h"
#include "dev-adc.h"
#include "blackboard.h"
#include "sensors.h"

// #include "stdio.h"

__HYDRUS_BEGIN

class SensingTask : public Task
{
public:

    typedef Traits< SensingTask > Tr;

    SensingTask() :
    Task(),
    m_adc(P::I2CBus::getBusInstance(0)),
    m_mag(P::I2CBus::getBusInstance(0))
    {

    }

    virtual bool tick() override
    {
        static PHSensor ph_s;
        static TemperatureSensor temp_s;
        static TurbiditySensor turb_s;
```

```
38          static BatterySensor batt_s;
39
40          m_mag.sample();
41
42          float adcValues[4];
43          for(int i = 0; i < 4; i++)
44          {
45              adcValues[i] = m_adc.get_volts(i);
46          }
47
48          // make sense of adc values
49          float waterPH   = ph_s.convert(adcValues[0]);
50          float waterTemp = temp_s.convert(adcValues[1]);
51          float waterTurb = turb_s.convert(adcValues[2]);
52          float battVolts = batt_s.convert(adcValues[3]);
53
54
55          // write values to blackboard
56          BB->trans.begin();
57
58          BB->sensors.imuHeading = m_mag.heading();
59          BB->sensors.waterPH = waterPH;
60          BB->sensors.waterTemp = waterTemp;
61          BB->sensors.waterTurb = waterTurb;
62          BB->sys.battVoltage = battVolts;
63
64          BB->sys.battLevel = hClamp((battVolts - Tr::MIN_BATT_V) / (
                Tr::MAX_BATT_V - Tr::MIN_BATT_V));
65          BB->trans.end();
66
67  //          fprintf(stderr, "ADC VALUES %f %f %f %f \n", adcValues
        [0], adcValues[1],  adcValues[2],  adcValues[3] );
68  //          fprintf(stderr, "HEADING %f %f %f %f \n", BB->sensors.
        imuHeading, m_mag.x(), m_mag.y(), m_mag.z());
69
70          return true; // keep running
71      }
72
73      virtual uint64_t period_us() override
74      {
75          return Tr::THREAD_INTERVAL_us;
76      }
```

```
77
78
79 private:
80     ADC m_adc;
81     Magnetometer m_mag;
82 };
83
84 __HYDRUS_END
```

APÊNDICE E

Listagem de Código: Tarefa de Comunicação

```
#pragma once

#include "hydrus-config.h"
#include "hydrus-traits.h"

#include "task.h"
#include "platform/logger.h"

#include "support/station.h"
#include "support/util-str.h"

__HYDRUS_BEGIN

class CommTask : public Task, public P::Logger::LogListener
{
public:

    typedef Traits< CommTask > Tr;

    CommTask(Task &navTask) : Task(), m_nav_task(navTask)
    {
        P::Logger::register_listener(this);

        m_st.start();
    }

    virtual bool tick() override
    {
        m_st.tick();

        if(BB->comm.connected != m_st.connected())
        {
            BB->trans.begin();
            BB->comm.connected = m_st.connected();
            BB->trans.end();
        }

```

```
38          std::string msg;
39          while(m_st.hasMessages())
40          {
41              msg = m_st.unqueueMessage();
42
43              route_command(msg);
44          }
45
46          return true; // keep running
47      }
48
49      virtual uint64_t period_us() override
50      {
51          return Tr::THREAD_INTERVAL_us;
52      }
53
54      virtual void log_received(const char * o, const char * what, P
            ::Logger::Level lvl) override
55      {
56          std::string logline;
57          if(o)
58          {
59              logline.append("[");
60              logline.append(o);
61              logline.append("] ");
62          }
63          logline.append(P::Logger::log_severity_str(lvl));
64          logline.append(": ");
65          logline.append(what);
66
67          m_st.queueLog(logline);
68      }
69
70  private:
71
72      void route_command(const std::string & msg)
73      {
74          P::Logger::log("comm|recv", msg.c_str());
75
76
77          // do not route for now, just process the messages
78          if(!msg.find("$HALT"))
```

```
79          {
80              BB->trans.begin();
81              BB->sys.state = SS_SHUTDOWN;
82              BB->trans.end();
83          }
84          if(!msg.find("$REBOOT"))
85          {
86              BB->trans.begin();
87              BB->sys.reboot = true;
88              BB->sys.state = SS_SHUTDOWN;
89              BB->trans.end();
90          }
91
92          if(!msg.find("$RCON"))
93          {
94              P::Logger::log("Comm", "Turned on RC");
95              BB->trans.begin();
96              BB->nav.rcVecX = BB->nav.rcVecX = 0;
97              BB->nav.rcMode = true;
98              BB->trans.end();
99          }
100
101         if(!msg.find("$RCOFF"))
102         {
103             P::Logger::log("Comm", "Turned off RC");
104             BB->trans.begin();
105             BB->nav.motor.spdL = BB->nav.motor.spdR = 0;
106             BB->nav.rcMode = false;
107             BB->trans.end();
108         }
109
110         if(!msg.find("$RCUP,"))
111         {
112
113             stringvec_t tok;
114             Util::split(msg, ',', tok);
115
116             BB->trans.begin();
117             if(tok.size() == 3)
118             {
119                 BB->nav.rcVecX = Util::parseFloat(tok[1]);
120                 BB->nav.rcVecY = Util::parseFloat(tok[2]);
```

```
121             }
122             else
123             {
124                 BB->nav.rcVecX = 0;
125                 BB->nav.rcVecY = 0;
126             }
127             BB->trans.end();
128         }
129 
130         if(!msg.find("$NAV"))
131         {
132             m_nav_task.process_command(msg.c_str());
133         }
134 
135         // TODO restore rounting of these commands
136 
137         if(!msg.find("$SIM"))
138         {
139 //              if(!m_sim)
140 //                  m_sim = new SimulationSlave();
141 //
142 //              m_sim->onCommandReceived(msg);
143         }
144 
145         if(!msg.find("$GPS,"))
146         {
147 //              Serial1.print(msg.c_str() + 5);
148 //              Serial1.print("\r\n");
149         }
150 
151     }
152 
153     Station m_st;
154     Task &m_nav_task;
155 };
156 
157 __HYDRUS_END
```

APÊNDICE F

Listagem de Código: Tarefa de Navegação

```
#pragma once

#include "hydrus-config.h"
#include "task.h"
#include "platform/logger.h"

#include "dev-gps.h"
#include "dev-motors.h"
#include "blackboard.h"

#include "navigation/navcontroller.h"
#include "navigation/navwaypoints.h"

#include "support/util-str.h"


__HYDRUS_BEGIN

class NavigationTask : public Task
{
public:

    typedef Traits< NavigationTask > Tr;

    NavigationTask() : Task()
    {
        GPS::init();
        bzero(&m_gps_msg, sizeof(m_gps_msg));
    }


    virtual bool tick() override
    {
        bool gpsDirty = false;
        if(GPS::has_message())
        {
            m_gps_msg = GPS::get_message_copy();
```

```
38              gpsDirty = true;
39          }
40
41
42
43          if(BB->nav.rcMode)
44          {
45              BB->trans.begin();
46              BB->nav.motor.enabled = true;
47              // TODO proper trigonometry maybe
48              BB->nav.motor.spdL = fmin(1, fmax(0, +Tr::
                    RC_SPEED_FACTOR*BB->nav.rcVecX + fmax(0, Tr::
                    RC_SPEED_FACTOR*BB->nav.rcVecY)));
49              BB->nav.motor.spdR = fmin(1, fmax(0, -Tr::
                    RC_SPEED_FACTOR*BB->nav.rcVecX + fmax(0, Tr::
                    RC_SPEED_FACTOR*BB->nav.rcVecY)));
50              BB->trans.end();
51          }
52          else
53          {
54              BB->trans.begin();
55              if(BB->sys.state == SS_NAVIGATING)
56              {
57                  bool cont = m_controller.step(0.04);
58                  if(!cont)
59                  {
60                      BB->nav.state = NS_NOT_NAVIGATING;
61                      BB->sys.state = SS_READY;
62                  }
63              }
64              else
65              {
66                  BB->nav.motor.enabled = false;
67                  BB->nav.motor.spdL = 0;
68                  BB->nav.motor.spdR = 0;
69              }
70              BB->trans.end();
71          }
72
73          m_motors.tick();
74
75
```

```
76          if(gpsDirty)
77          {
78              BB->trans.begin();
79
80              BB->nav.gpsHasFix = m_gps_msg.fix;
81              BB->nav.gpsLat = m_gps_msg.lat;
82              BB->nav.gpsLon = m_gps_msg.lon;
83
84              BB->trans.end();
85
86              gpsDirty = false;
87          }
88
89          return true; // keep running
90      }
91
92      virtual uint64_t period_us() override
93      {
94          return Tr::THREAD_INTERVAL_us;
95      }
96
97      virtual void process_command(const char * cmd) override
98      {
99          if(Util::startsWith("$NAVROUTE,", cmd))
100         {
101             parse_route(cmd);
102         }
103
104         if(Util::startsWith("$NAVBEGIN", cmd))
105         {
106             begin_navigation();
107         }
108     }
109
110 private:
111
112     bool validate_route(const Waypoints & wps)
113     {
114         return wps.size() > 1; // more complex logic can be put
                here
115     }
116
```

```
117     void parse_route(const char *cmd)
118     {
119         std::string routecmd(cmd);
120         stringvec_t tok;
121         Util::split(routecmd, ',', tok);
122         int idx = 2;
123 
124         Waypoints wps;
125 
126         while(tok.size() - idx >= 3)
127         {
128             Waypoint w;
129 
130             w.longitude = Util::parseDouble(tok[idx]);
131             w.latitude = Util::parseDouble(tok[idx+1]);
132             w.acquire = tok[idx+2].size() > 0;
133             w.custom_radius = -1;
134 
135             wps.push_back(w);
136 
137             idx += 3;
138 
139 
140             std::stringstream ss;
141             ss << "WAYPOINT ";
142             ss << w.longitude << " " << w.latitude << " ";
143             ss << (w.acquire? "A" : "");
144             P::Logger::log("Nav", ss.str().c_str());
145 
146         }
147 
148         if(validate_route(wps))
149         {
150             m_waypoints = wps;
151             P::Logger::log("nav", "Received new route");
152         }
153         else
154         {
155             m_waypoints.clear();
156             P::Logger::log("nav", "Received invalid route. Current
                route cleared for safety.", P::Logger::ERROR);
157         }
```

```
158
159     }
160
161     void begin_navigation()
162     {
163         // check navigation state here
164
165         if(!m_waypoints.size())
166         {
167             P::Logger::log("nav", "No valid route specified. Can't
                    begin navigation.");
168             return;
169         }
170
171         BBro->trans.begin(true);
172         if(!BBro->nav.gpsHasFix)
173         {
174             P::Logger::log("nav", "Drone GPS has no fix. Can't
                    begin navigation.");
175             return;
176         }
177         BBro->trans.end(true);
178
179         if(!m_controller.withinRange(m_waypoints[0].longitude,
                m_waypoints[0].latitude))
180         {
181             P::Logger::log("nav", "Drone is not within range of
                    base station location");
182
183         }
184
185         m_controller.setup(m_waypoints);
186     }
187
188     GPS::Message m_gps_msg;
189     Motors m_motors;
190
191     NavController m_controller;
192     Waypoints m_waypoints;
193 };
194
195 __HYDRUS_END
```

APÊNDICE G

Listagem de Código: Controlador de Navegação

```
#include "hydrus-config.h"
#include "hydrus-traits.h"

#include "navwaypoints.h"
#include "math.h"
#include "platform/logger.h"

#include "blackboard.h"

__HYDRUS_BEGIN

#define deg2rad(x) ((x) * ((nav_f_t)0.017453292519943295))

class NavController
{
public:

    typedef Traits< NavController > Tr;
    typedef Tr::nav_f_t nav_f_t;


    NavController()
    {
        // nothing to initialize, all state in blackboard
    }

    void setup(Waypoints &wps)
    {
        m_wps = &wps;
        m_wp = 1;

        BB->trans.begin();
        BB->nav.state = NS_ALIGN;
        BB->sys.state = SS_NAVIGATING;
        BB->trans.end();
    }

```

```
38      void clearActuation()
39      {
40          BB->trans.begin();
41          BB->nav.motor.spdL = 0;
42          BB->nav.motor.spdR = 0;
43          BB->nav.motor.enabled = false;
44          BB->trans.end();
45      }
46
47      bool withinRange(nav_f_t lon, nav_f_t lat, bool exit = false)
48      {
49          BlackboardTransaction trans( true );
50          return calcDistance_m(lon, lat, BBro->nav.gpsLon, BBro->nav
                .gpsLat) < (exit? Tr::rangeToleranceRadius_m : Tr::
                rangeToleranceExitRadius_m);
51      }
52
53      /**
54       * Haversine and Forward Azimuth
55       * from http://www.movable-type.co.uk/scripts/latlong.html
56       *
57       * These arguments are taken in degrees
58       */
59      nav_f_t calcDistance_m(nav_f_t lonA, nav_f_t latA, nav_f_t lonB
           , nav_f_t latB)
60      {
61          const nav_f_t R = Tr::earthRadius_m; // metres
62          nav_f_t lat1 = deg2rad(latA);
63          nav_f_t lat2 = deg2rad(latB);
64          nav_f_t delta_lat = deg2rad(latB-latA);
65          nav_f_t delta_lon = deg2rad(lonB-lonA);
66
67          nav_f_t a = sin(delta_lat/2) * sin(delta_lat/2) +
68          cos(lat1) * cos(lat2) *
69          sin(delta_lon/2) * sin(delta_lon/2);
70          nav_f_t c = 2 * atan2(sqrt(a), sqrt(1-a));
71
72          return R * c;
73      }
74
75
76      /*
```

```
77          * These arguments are taken in radians
78          */
79         nav_f_t calcBearing_rad(nav_f_t lon1, nav_f_t lat1, nav_f_t
               lon2, nav_f_t lat2)
80         {
81
82             nav_f_t y = sin(lon2-lon1) * cos(lat2);
83             nav_f_t x = cos(lat1)*sin(lat2) - sin(lat1)*cos(lat2)*cos(
                   lon2-lon1);
84             nav_f_t ret = (-atan2(y, x)) + (M_PI*0.5);
85
86             if(ret < 0)
87                 ret += (2*M_PI);
88             return ret;
89         }
90
91         nav_f_t fancymod(nav_f_t a, nav_f_t n)
92         {
93             return fmod( (fmod(a, n) + n), n );
94         }
95
96         nav_f_t calcAngleDifference_rad(nav_f_t sourceA, nav_f_t
               targetA)
97         {
98             nav_f_t a = targetA - sourceA;
99             return fancymod((a + M_PI), 2*M_PI) - M_PI;
100        }
101
102        /**
103         * Step navigation controller
104         * @returns true if still navigating, false if not
105         */
106        bool step(nav_f_t time_delta)
107        {
108            // state timekeeping
109
110            static NavigationState prev_state = (NavigationState) -1;
111            static nav_f_t state_time  = 0;
112
113            BBro->trans.begin(true); // BEGIN reading blackboard
114
115            if(BBro->nav.state == prev_state)
```

```
116         {
117             state_time += time_delta;
118         }
119         else
120         {
121             state_time = 0;
122             prev_state = BBro->nav.state;
123         }
124 
125         // get gps state and position
126         bool gpsFix = BBro->nav.gpsHasFix;
127         nav_f_t gpsLon = BBro->nav.gpsLon,
128         gpsLat = BBro->nav.gpsLat;
129 
130         // get current heading
131         nav_f_t magneticHeading = BBro->sensors.imuHeading;
132 
133         BBro->trans.end(true); // END reading blackboard
134 
135         // get destination angle positions
136         nav_f_t destLon = ((*m_wps)[m_wp].longitude);
137         nav_f_t destLat = ((*m_wps)[m_wp].latitude);
138 
139         // calculate distance
140         nav_f_t distance = calcDistance_m(gpsLon, gpsLat, destLon,
               destLat);
141 
142         nav_f_t correctedHeading;
143         if(Tr::USE_FIXED_DECLINATION)
144             correctedHeading = magneticHeading - Tr::
                   FIXED_DECLINATION_DEG;
145         else
146             correctedHeading = magneticHeading;
147 
148         nav_f_t heading = deg2rad(correctedHeading);
149         nav_f_t angleToDestination = calcBearing_rad(deg2rad(gpsLon
               ), deg2rad(gpsLat), deg2rad(destLon), deg2rad(destLat));
150 
151         // calculate angle difference
152         nav_f_t theta = calcAngleDifference_rad(heading,
               angleToDestination);
153 
```

```
154
155         /*
156          *  Now save some navigation data to the blackboard
157          */
158
159         BlackboardTransaction t; // BEGIN writing state to the
                blackboard
160
161         BB->nav.geoHeading = correctedHeading;
162         BB->nav.distanceFromNextWaypoint = distance;
163
164         if(!gpsFix)
165         {
166             P::Logger::log("navcontrol", "Lost GPS fix. Skipping
                    control, disabling motors.", P::Logger::ERROR);
167             BB->nav.motor.enabled = false;
168             return true;
169         }
170
171         switch(BB->nav.state)
172         {
173             case NS_NOT_NAVIGATING:
174                 return false; // well that was easy
175
176
177             case NS_ALIGN:            // point vessel towards
                    waypoint
178             {
179                 // compare delta
180                 if(fabs(theta) < Tr::headingTolerance_rad)
181                 {
182                     BB->nav.state = NS_ALIGN_WAIT;
183                 }
184                 else
185                 {
186                     // angle controller
187                     bool goLeft = theta > 0;
188                     nav_f_t pulseTime = fmod( state_time, 2);
189
190                     if(false)//pulseTime > 1)
191                     {
192                         // jolt pause time
```

```
193                         BB->nav.motor.spdL = 0;
194                         BB->nav.motor.spdR = 0;
195                     }
196                     else // give it a jolt
197                     {
198                         if(goLeft)
199                         {
200                             BB->nav.motor.spdR = Tr::speedMedium;
201                             BB->nav.motor.spdL = 0;
202                         }
203                         else
204                         {
205                             BB->nav.motor.spdL = Tr::speedMedium;
206                             BB->nav.motor.spdR = 0;
207                         }
208                     }
209                     #if !DRONE_NAV_SUPRESS_MOTORS
210                     BB->nav.motor.enabled = true;
211                     #endif
212                 }
213 
214                 /*
215                  *     std::stringstream ss;
216                  *     ss << "ALIGN " << heading << " ";
217                  *     ss << angleToDestination << " ";
218                  *     ss << theta << " ";
219                  *     // TODO write location name and distance
220                  *     Util::log("NavControl", ss.str().c_str());*/
221 
222                 return true;
223             }
224 
225             case NS_ALIGN_WAIT:       // vessel is pointing towards
                    waypoint
226             {
227                 BB->nav.motor.enabled = false;
228 
229                 if(state_time > Tr::timeForAlign)
230                 {
231                     BB->nav.state = NS_TRAVERSE;
232                     clearActuation();
233                 }
```

```
234
235                 if(fabs(theta) > Tr::headingTolerance_rad)
236                 {
237                     BB->nav.state = NS_ALIGN;
238                 }
239
240                 return true;
241             }
242
243
244             case NS_TRAVERSE:         // vessel moves towards
                    waypoint
245             {
246                 nav_f_t speed = distance < Tr::speedMediumDistance?
                        (distance < Tr::speedLowDistance? Tr::speedLow
                       : Tr::speedMedium) : Tr::speedFull;
247
248                 // update actuation
249                 if(2*fabs(theta) > M_PI)
250                 {
251                     // behind me
252                     BB->nav.motor.spdL = Tr::speedFull;
253                     BB->nav.motor.spdR = 0;
254                 }
255                 else
256                 {
257                     // sane thing
258                     BB->nav.motor.spdL = 0.0 * speed;
259                     BB->nav.motor.spdR = 0.0 * speed;
260
261                     if(theta > 0) // a bit more to the left
262                         BB->nav.motor.spdR += (0.5 + 0.5*fabs(sin(
                            theta))) * speed;
263                     else          // a bit more to the right
264                         BB->nav.motor.spdL += (0.5 + 0.5*fabs(sin(
                            theta))) * speed;
265                 }
266
267                 #if !DRONE_NAV_SUPRESS_MOTORS
268                 BB->nav.motor.enabled = true;
269                 #endif
270
```

```
271                 // if within radius
272                 if(distance < Tr::rangeToleranceRadius_m)
273                 {
274                     BB->nav.state = NS_ARRIVAL_WAIT;
275                     clearActuation();
276                 }
277
278
279                 return true;
280             }
281
282             case NS_ARRIVAL_WAIT:    // vessel in in radius of
                    waypoint
283
284                 // if drifted to exit radius, realign, must be
                        close
285                 if(distance > Tr::rangeToleranceExitRadius_m)
286                 {
287                     BB->nav.state = NS_ALIGN;
288                     return true;
289                 }
290
291                 // if waited enough time, we are in the spot
292                 if(state_time > Tr::timeForArrival)
293                 {
294                     BB->nav.state = NS_ACQUIRE;
295                 }
296
297
298             case NS_ACQUIRE:         // vessel is acquiring data
299
300                 if(state_time > 1) // wait a sec
301                 {
302                     // is there another waypoint?
303                     if(m_wp == (m_wps->size() - 1))
304                     {
305                         // end of the line
306                         BB->nav.state = NS_NOT_NAVIGATING;
307                         return false;
308                     }
309                     else
310                     {
```

```
311                         // advance to next waypoint
312                         P::Logger::log("navcontrol", "reached point
                               ", P::Logger::INFO);
313                         m_wp ++;
314                         P::Logger::log("navcontrol", "going for
                               next point", P::Logger::INFO);
315
316                         // realign
317                         BB->nav.state = NS_ALIGN;
318                     }
319                 }
320
321                 return true;
322
323             case NS_HOMING_EMERGENCY: // vessel is returning home
                    because something is not right
324
325                 // TODO
326                 // set home as next point, override waypoint list
327                 // set mode to align
328
329                 return true;
330
331         }
332
333         // unhandled state, we halt navigation then
334         return false;
335     }
336
337 private:
338     Waypoints *m_wps;
339     int m_wp;
340
341 };
342
343 __HYDRUS_END
```