2016 SBESC – Intel® Embedded System Competition

Final Report

Project Name: Hydrus – Autonomous Drone for Hydrologic Monitoring

Students:	Êmili Bohrer, Guilherme Augusto Pangratz, Lucas Pires Camargo
Professor:	Giovani Gracioli
University:	Federal University of Santa Catarina
JEMS ID:	154930

Declaration of Originality

We hereby declare that this thesis and the work reported herein was composed and originated entirely by ourselves. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the references.

Professor's Signature: [illegible]
Name (in Block Letters): GIOVANI GRACIOLI
Date: 30/10/2016

Hydrus – Autonomous Drone for Hydrologic Monitoring

Abstract

Droughts, lack of basic sanitation and mismanagement of water resources are all current problems of Brazil. It is important to monitor drainage basing, water reservoirs and other sources of clean water, as well as the environment, for proper usage and conservation. Autonomous embedded technology can be used to make measurements cheaper and more efficient. The Hydrus project consists of creating an autonomous boat drone for automated water quality metrics testing, using the Intel Galileo Gen 2 platform. Details of the hardware setup and software implementation are discussed. The software is implemented using the classic blackboard architecture combined with a cyclic executive scheduler. A companion software application for programming and control is also implemented. The boat hulls and frame are also built by the team. Development challenges and solutions are discussed. Testing procedures show the drone performs as expected, and project objectives are met. As future work, the team plans to use the drone as a platform to allow for implementation of new functionality.

Keywords: Aquatic drone, Autonomous Navigation, Water quality measurement, Intel Galileo, Water quality

Contents

Acknowledgments

The preparation of this work would not have been possible without the collaboration, encouragement and dedication of many people. We would like to express all our gratitude to all those who contributed to this task come true.

The Software/Hardware Integration Lab (LISHA) and Models Laboratory throughout the structure and equipment assigned for the different project realization stages.

Professors, for their assistance whenever requested.

Mauricio Ferreira, Models Laboratory trainee, thank you for your help, support and availability throughout the construction of the boat.

1 Introduction

This report presents our project results for 2016 Intel Embedded Systems Competition. It is called **Hydrus**, named after the "male water snake" constellation of the southern sky. It consists of a boat drone that performs autonomous water quality measurements. In here, we detail the drone's subsystems and the development process.

1.1 Motivation

Basic sanitation is understood as the necessary infrastructure for guaranteeing a population's quality of life, and includes water treatment services, sewage disposal, collection and processing of garbage and waste water inside urban perimeters. The United Nations point out that access to basic sanitation as one of the fundamental rights for guaranteeing that a populations can experience a healthy and fulfilling life. According to Brazil's Institute of Geography and Statistics (IBGE), in Brazil, whilst 98% of Brazilians have access to clean, treated water, the percentage that has access to the sanitary sewage disposal or septic tanks is of only 79%, revealing more than a fifth of the population lives together with open sewage. About 3.5 million Brazilians dump sewage irregularly, even when having access to sewage plumbing. Furthermore, even if sewage is properly collected, only 44% of the raw sewage volume is treated correctly, according to data from National Sanitation Information Service (SNIS, 2014). This means that 1.2 billion m^3 of untreated sewage per year are dumped in nature.

It is clear there is still much to be done in Brazil, to guarantee its population access to essential sanitation services. In this context, it is crucial for the authorities and service providers to have at their disposal data that allows them to evaluate and monitor the water quality in drainage basins, from the point of view of suitability for consumption, or for treatment. Currently, in most cases, the analysis of water quality metrics in reservoirs and lakes is done manually, by sample collection and laboratory analysis. For analysis is done in the field, the cost of specialized work, in distant places, added to the equipment, is often prohibitive in a daily or weekly regime. Whilst these procedures generate quality data, they are costly, and sparse.

Technological advances have allowed for cost reductions and miniaturization of entire systems. As consequence, autonomous embedded systems have emerged. We have as examples cars, aerial and aquatic drones, and others. In aquatic drones in particular, it is possible to integrate water quality sensors and make those measurements completely autonomous, reducing costs and time (for capturing the sensor data for instance).

1.2 Objectives

We propose an autonomous boat drone vehicle for measurement, that would make possible frequent surveys. These surveys should be useful for qualitative and comparative analysis for water quality metrics. The possibility of obtaining spatial time series may be a big help in the detection of tendencies in the surveyed metrics, and these tendencies could indicate generalized problems in water quality. Since every measurement would be geotagged, quick action would be possible is something wrong is detected, improving quality of life for a population. Once the prototype is implemented, we hope to have in our hands a flexible platform, which we will be able to extend in the future to provide still unforeseen functionality.

1.3 Report structure

This report is structured as follows:

Section 2 presents the project's original proposal and its intended contributions. Next, we detail the developed hardware setup in section 3. Section 4 presents the software aspects of the project: how the drone firmware is structured and implemented, as well as the base station software.

A short discussion on the boat's structure and construction process is presented on section 5. Section 7 discusses some of the challenges faced by the team, and how they were overcome. Section 6 explores how the system was tested and validated. Finally, in section 8 we present our findings and final remarks about the project results.

2 The Hydrus Project

This project envisions the construction of an autonomous aquatic drone for data acquisition and water quality sensing, implemented atop Intel's Galileo Gen 2 platform. The main idea is that the vehicle will be able to follow a predetermined route, navigating autonomously. Along this route, it should be able to collect water quality metrics automatically, and later send the logged data into a supervisory system for monitoring and control when it returns and touches base. Figure 1 is a visualization of the idea.

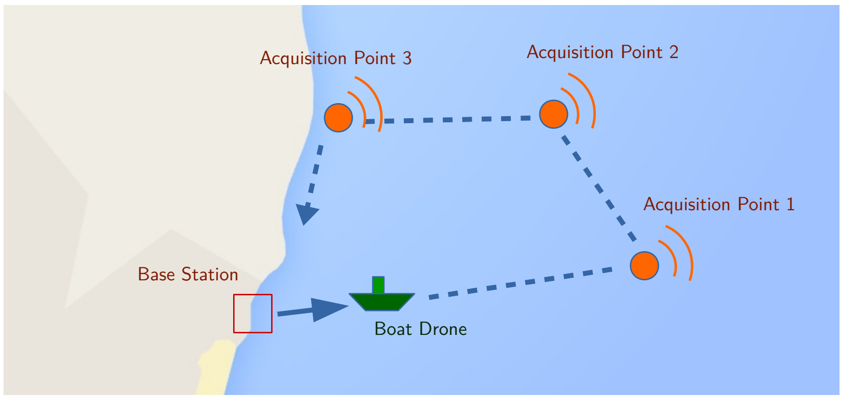


Figure 1: Aquatic drone visualization

Our main goals for the final functionality set are:

- Sensing of the following indicators: water PH, turbidity and temperature on the data collection locations.
- Autonomous navigation via GPS to data collection points and to the base station.
- Transmission of collected data and diagnostic information via WiFi.

Another visualization of the concept is presented in Figure 2. By realizing these goals, we aim to build an autonomous boat drone platform, that can later be adapted to various applications.

3 Hardware

This section is a discussion on the hardware built for the drone, and the decisions supporting the design and construction.

Our main requirements were to provide the drone with the necessary sensors, whilst still maintaining the cost low. It had to be feasible for us to build it.

Figure 3 shows everything that is connected to the Galileo Gen 2 Board.

3.1 Sensors

Here we describe the sensors used on our project, a little explanation of their working principle, and why each one of them were chosen.

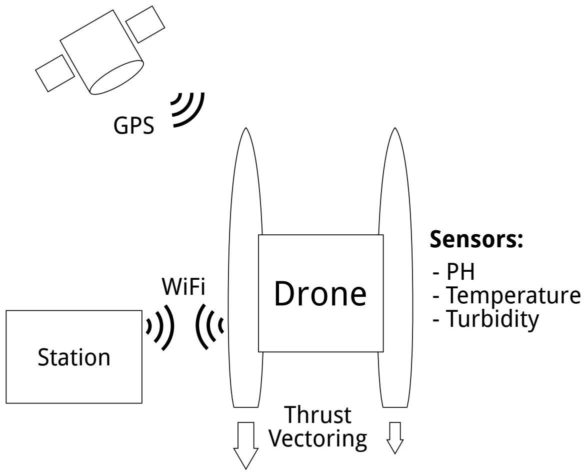


Figure 2: Drone inputs and outputs

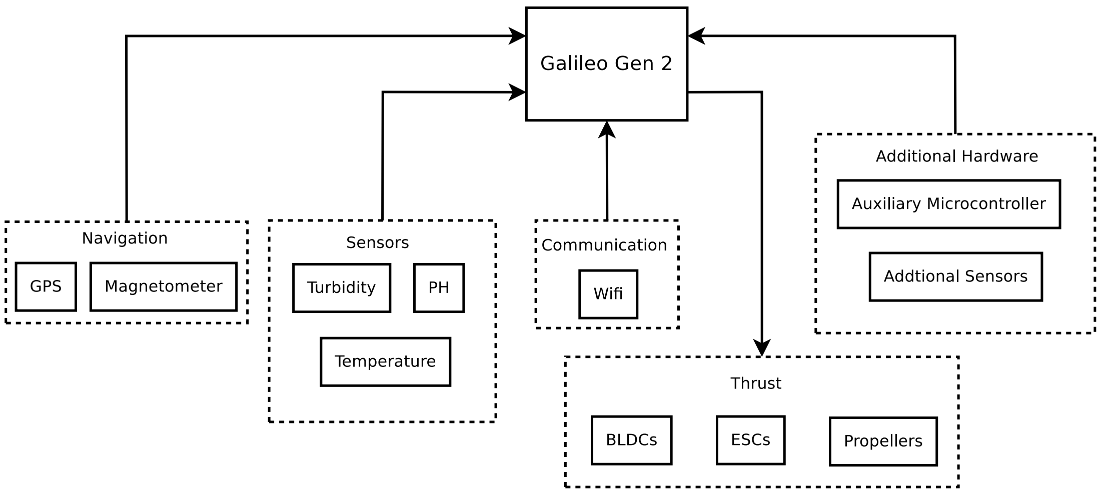


Figure 3: General hardware diagram

3.1.1 Turbidity Sensor

The turbidity sensor is used to measure the relative clarity of a fluid. The sensor operation is based on the principle that when a sample of liquid receives a beam of light, the amount of light that crosses the fluid depends on the quantity of soil particles floating in it [4]. Our sensor uses a light emitter diode as source of light, and a phototransistor to measure the amount of light that reached the other side of the gap. Figure 4 illustrates the process.

The specific model chosen for the project was the TSD-10 Turbidity Sensor, from Amphenol Thermometrics. The main factor behind the decision was the low price and wide availability.

Despite its low price, we obtained good results. The power consumption was low, generally operating with less than 10mA while being powered with 5V, and the integration is simple (it requires only a couple of additional resistors). The sensor outputs a voltage that is read directly by one of the Galileo's analog inputs.

3.1.2 pH Sensor

The pH sensor is used to measure how acid or how alkaline a liquid is, based on its hydrogen ions concentration. An acid dissolved on water form positive ions (H+), while bases when dissolved on water form negative ions (OH-).

We use a pH meter [2], constituted of a reference electrode, and another electrode that consists on a bulb made with a very special glass, generally containing potassium chloride solution, with a silver chloride wire suspended on it. Essentially, the PH meter measures the electrochemical potential between the known liquid inside the glass bulb, and the unknown liquid on the outside.

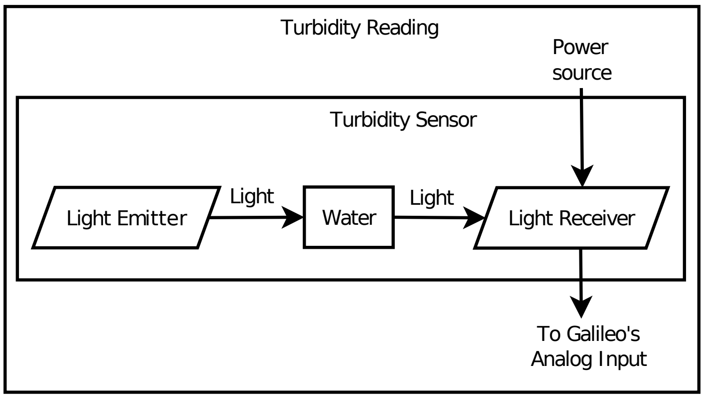


Figure 4: Turbidity reading

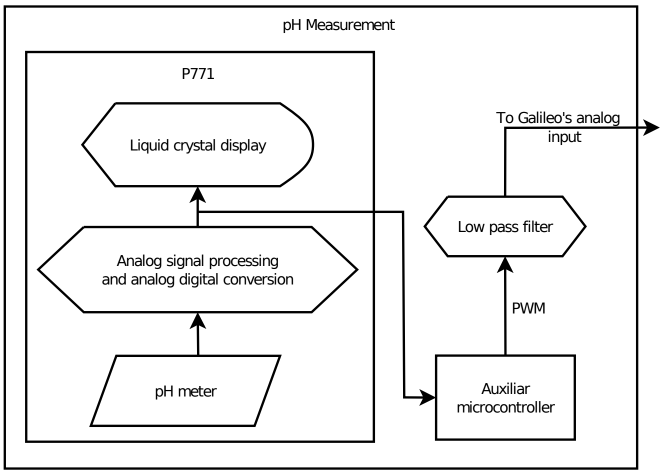


Figure 5: pH acquisition

Our solution to read pH is the P771 [12] from Anaheim Scientific, it is an complete instrument, bringing the PH meter and and integrated circuit that handles the sensor, feeding it and treating its analog output, amplifying it and converting it to a digital value, to then transforming the information in a way to be shown on a liquid crystal display.

Handling the pH meter was quiet difficult for us. The best and most practical way we've found to do it is decoding the information sent to the LCD. We removed the instrument's display, and then we use a microcontroller to read the state of its signal pins. From there we pick the pH reading value. Lastly, the auxiliary microcontroller modulates the information into a PWM signal, that is filtered by a first order low pass analog filter, and the read by the Galileo as an analog input. All the pH acquisition process is represented in the diagram on Figure 5.

3.1.3 Water Temperature Sensor

The temperature sensor used is based on semiconductor material. It uses the effect of the conductibility variation on the material, caused by the temperature variation on its body, to create a signal that is treated and amplified to produce the analog voltage output.

Our specific model is the TMP36 from Analog Devices. It just needs to be fed with a voltage between 2.7V and 5.5V to offer an linear output proportional to the temperature reading in Celsius. Another great point of this model is the very low power consumption, with the sensor using less than 50uA during its operation. Again here, the sensor measures are read by a Galileo's analog input.

We're using the TPM36 [5] to measure the water temperature, for that to work the sensor contacts were isolated with silicon, where they are soldered with the wires, keeping only the encapsulated part of the sensor in contact with the water.

3.1.4 Magnetometer

The magnetometers in general use magneto-resistive sensors to determine the influence of external magnetic fields, generally from the Earth's magnetic field, on the magnetometer's body.

The model we're using is the HMC5883l [9], from Honeywell, which offers 3-axis readings, a 12-bit ADC and an I2C interface.

The magnetometer is used during the autonomous navigation to identify the boat's heading.

3.1.5 Auxiliary Microcontroller and Additional Sensors

Since most of the galileo's pins are occupied by the boat control and the sensors responsible for taking the water quality indicatives, we've choose to use an auxiliary microcontroller to expand our possibilities. Besides the pin limitation on the Galileo, it brings some difficulties when using sensors with time requests, like sonars to measure distances, and sensors with communication protocols that require a GPIO pin to change is direction multiple times in a very short period of time, like when using the one-wire protocol.

We opted for the simplest solution, a simple microcontroller, to leave the brain job to the Galileo Gen 2, and it was and ATmega328P [3] from Atmel, embedded on an Arduino Nano board. On the ATmega 328P we've implemented the reading of two modules.

The first module on the auxiliary microcontroller is the HC-SR04 [8], an ultrasonic ranging module, which consists in an ultrasonic emitter and a receiver. The time between a wave leaving the emitter and reaching the receiver indicates the distance between the module and the obstacle that reflected the mechanical ultrasound wave. The module works with a pin called trigger, that controls when the emitter will be triggered, and a pin called echo which represents when the expected waved hits the receiver. The procedure to use the module is basically to start a timer when the trigger pin is activated, and then check the timer value when the echo pin is set high by the module. This time difference indicates the distance, based on the speed of sound in the air.

The second one, a DHT22 [1], which brings together a capacitive humidity sensor, to measure the relative humidity of the environment, and a thermistor to measure the temperature. The module externalizes the information through the one-wire protocol, and it's generally implemented in software on the reading unity, using a GPIO pin and a timer to know the instants where the pin's state represents a piece on incoming information, or when to change the pin level to send information to the module.

The communication between the Galileo board and the auxiliary microcontroller is made using UART. The 328P keeps reading the sensors all the time, and when the Galileo wants to get the current values it sends an character through the UART that is the signal to the 328P to start sending the readings data. This information is stored on both the Galileo and the 328P code on a structure containing the readings from all the elements on the auxiliary microcontroller, and when a data request comes from the Galileo, the 328P sends all readings encapsulated on an element of this structure via UART. Figure 6 illustrates the communication between the additional sensors modules and the auxiliary microcontroller, and then between the microcontroller and the Galileo.

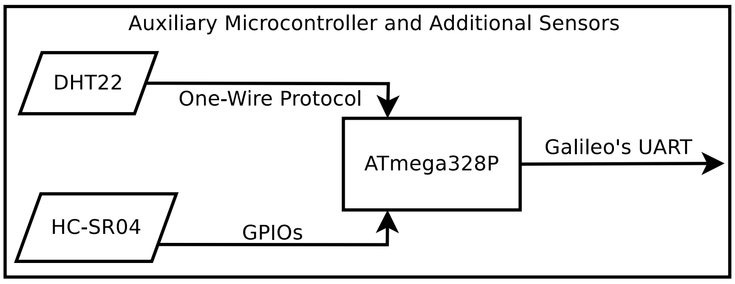


Figure 6: Auxiliary microcontroller and additional sensors

3.2 Thrust

The boat is driven by the set formed by two synchronous motors with permanent magnets (aka BLDC), two electronic speed controllers (ESCs), a lithium polymer battery, and t'wo propellers.

The energy conversion begin with the electronic speed controllers cranking the motors, which in its turn does the conversion from electrical energy to mechanical energy rotating an axis with a propeller on the end, that shifts the water on its surroundings to convert the axis rotation in movement for the boat. Figure 7 represents the energy path between the elements.

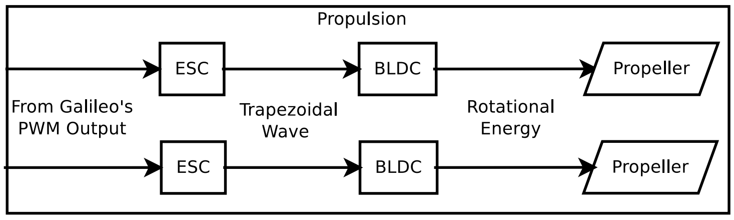


Figure 7: Propulsion

3.2.1 BLDC and ESCs

Synchronous motors with permanent magnets, often called as brushless direct current motors, brings the possibility of high rotation speeds and precision in the speed control. In order to work, they need the electronic contained on the electronic speed controller.

The ESCs [14] uses power electronics's inversion techniques to transform their direct current (DC) input in the trapezoidal waveform tension output to drive the BLDCs. Additionally, they produce a 5v DC output created by a DC converter. The speed information is sent to the ESC modulated in the form a PWM signal, generated by the main Galileo board.

3.2.2 Transmission and propellers

On boats with underwater propulsion, the rotational energy from the motors needs to be delivered to the propellers, and in order to do that some problems rise up, first, the motor's rotor axis generally isn't long enough to reach the propeller from where the motor is linked to the hull.

To solve the first problem we use a universal joint, connected to the motor on one side, and to a cylindrical bar in the other one. The universal joint is needed to handle the alignment problem between the motor's rotor axis and the bar that connects it to the propeller. The bar reaches the propeller position through a hole in the boat, and there we have a new problem, the boat can't be filled with water during the navigation.

The solution for this one is to pass the transmission axis by inside a tube, and fill this tube with grease. This way the tube's inclination and the difficulty to trespass the grease avoids the water from getting to the boat's interior.

The propellers we're using were 3D printed. They were designed to be small, being easily submerged, and shifting small amounts of water, allowing it to turn at relatively high speed.

We still have another 3D printed part, a connection to link the motor's axis into the universal joint, it's needed because there's a small diameter difference between the rotor's axis and the universal joint's input.

Even with the small propellers, the boat still not uses the full power from the ESCs and the motors, we ended up using only half of the maximum values, trying to keep the boat easy to control, and to avoid a mechanical overload.

3.3 Communication modules

We used both GPS and WiFi modules in the project. The GPS module was essential for navigation, and the WiFi module was used for communicating with the base station.

3.3.1 GPS Module

As the project's GPS solution, the A2035-H module from Maestro Wireless Solutions [13] was used. It was chosen because it was already available at our laboratory, and the integration is simple.

The module was used in UART mode, connected to the `Serial1` interface (pins 0 and 1). The reset and power control lines were connected to regular GPIOs.

3.4 System board

Given all the necessary connections to sensors and submodules, we needed to consolidate the components in a physically sound package. Thus, the team decided to create a custom printed circuit board to house all the necessary connectors for the different submodules, ESCs, and also for mounting of the GPS module.

The schematics were first assembled by hand in breadboards and tested on the fly. Then the final schematics were captured and a board layout was created in the open-source Kicad software package ??. It was then built by hand using homemade PCB construction techniques. Figure 8 shows the board layout and the finished product.

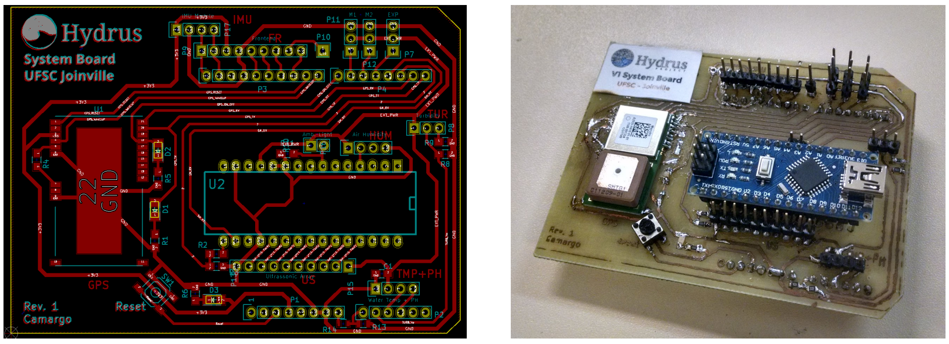


Figure 8: The system board layout and the finished product

Talk about how it came out and how it could be improved in the future.

4 Software

In this section, we describe the software developed during the project. It is divided in two parts: (i) the drone's firmware, responsible for the controlling and sensing; and (ii) the base station desktop application, responsible for configuring the drone and receiving information from it.

We start this section presenting the software requirements in Subsection 4.1, then we present an overview of the drone software architecture in Subsection 4.2, the navigation algorithm in Subsection 4.3, and the base station software in Subsection 4.4

4.1 Software Requirements

We first defined software requirements that drove the software development process. These requirements were established according to the project needs and were refined as the project evolution. The requirements for both the drone's firmware and base station are presented below.

4.1.1 Firmware requirements

- Must be implemented considering good practices of real-time software development.
- Needs to have an architecture that facilitates the integration of sensors, actuators, and controllers.
- Must have good performance.

4.1.2 Base station software requirements

- Must be easy to use.
- Must expose all the functionality on the drone.
- Should not get in the way of the user.

To accomplish the defined requirements, we choose to implement both drone and base station softwares using the C++ programming language, as it provides good performance, software reuse due to object-orientation, and support user graphic interfaces.

4.2 Software Architecture

Figure 9 shows an overview of the Drone's final software stack, based on the latest Galileo SD card image (`iot-devkit-prof-dev-image-galileo-20160606`). The Hydrus software is organized in four main components: (i) blackboard pattern; (ii) utilities; (iii) tasks; and (iv) drone platform. Below, we describe each of the components.

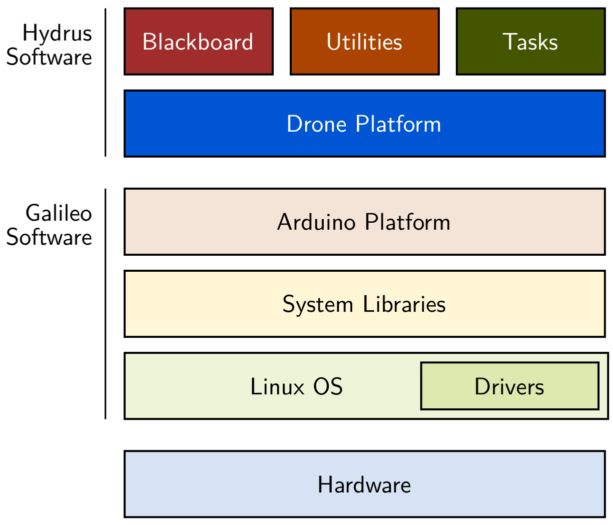


Figure 9: Drone software stack

The Blackboard pattern was adopted as the main architectural trait of the drone software. This pattern is defined as a set of specialized, loosely-coupled agents iterating over a common knowledge base, allowing for the coordination of complex control behavior.

In our implementation of the blackboard pattern, the blackboard is a central in-memory data structure and the agents are defined as independent tasks. Every task has a well-defined purpose, e.g. reading all sensors. To avoid functionality creep, tasks divide more specialized function in aggregate classes, e.g. a class representing a sensor or an actuator.

All tasks in the system, with the exception of auxiliary threads, are scheduled by a global Cyclic Executive Scheduler (CES) [7]. The CES can be understood as a table that determines which tasks have to be run at a given interval of time. It divides the time into periods, which are described by the whole table. Furthermore, it divides periods into compartments, every compartment being an equal amount of time in which the tasks are *ticked*. Figure 10 shows the classes that implement the scheduler and the base classes of the tasks.

The center of it all is the `CEScheduler` class. It is paremeterized by the total number of tasks. The CES is simple and robust real-time scheduling mechanism, used in mission-critical embedded systems, such as in the Boeing 777 airplane. Our implementation of it in the Linux user space is not hard real-time software (mainly due to non real-time behavior of Linux), but combined to the blackboard pattern, it provided a solid base for the implementation of the required functionality. Table 1 shows the full schedule. The full period is 40 milliseconds, so one of the compartments executes every 20 milliseconds. The result is thak the sytem and sensing tasks have a period of 20 milliseconds each, while the Communication and Navigation tasks have a period of 40 milliseconds.

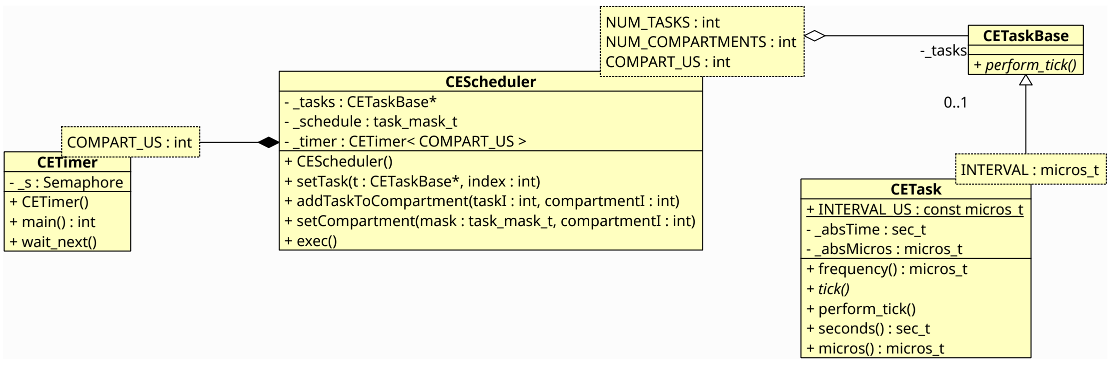


Figure 10: Cyclic Executive Scheduler infrastructure

Compartment	System	Sensing	Communication	Navigation
0	Yes	Yes	Yes	No
1	Yes	Yes	No	Yes

Table 1: The full task schedule

4.2.1 System task

This task is responsible for overseeing the overall system behaviors and interfaces. It manages OS shutdown and reboot, the interface LEDs and screen, and handles debug commands. It also initializes the blackboard, and is the first task to be run by the scheduler.

4.2.2 Navigation task

The navigation task is responsible for advancing the navigation controller, parsing and validating navigation routes, and also handling RC mode for manual motor control. Figure 11 shows an UML class diagram of the main navigation components.

A `NavController` instance is stepped manually by the task, using its `step()` method. It also aggregates the motor controller class (`Motors`).

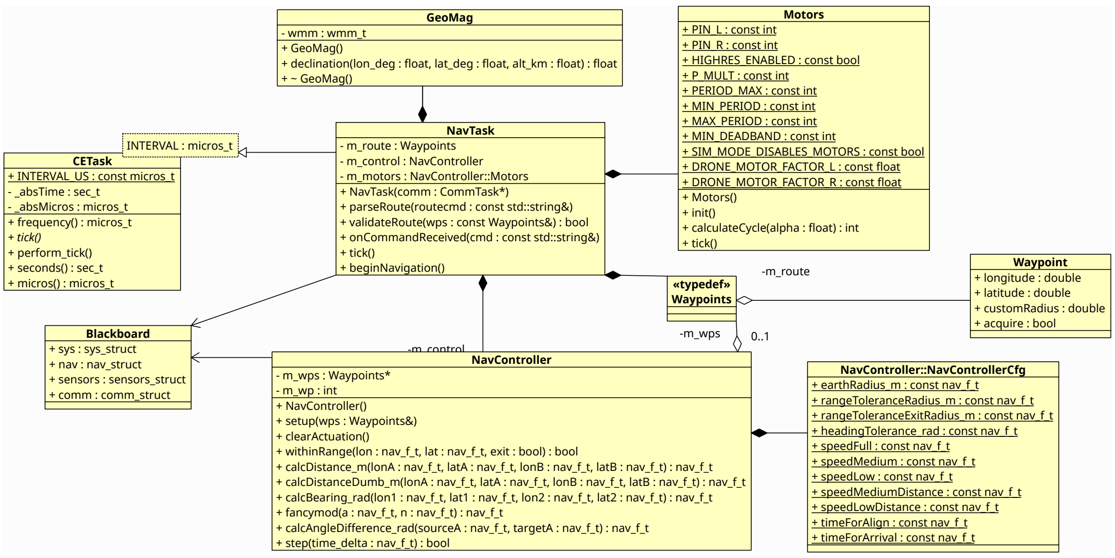

GeoMag
- wmm : wmm_t
+ GeoMag()
+ declination(lon_deg : float, lat_deg : float, alt_km : float) : float
+ ~ GeoMag()
Motors
+ PIN_L : const int
+ PIN_R : const int
+ HIGHRES_ENABLED : const bool
+ P_MULT : const int
+ PERIOD_MAX : const int
+ MIN_PERIOD : const int
+ MAX_PERIOD : const int
+ MIN_DEADBAND : const int
+ SIM_MODE_DISABLES_MOTORS : const bool
+ DRONE_MOTOR_FACTOR_L : const float
+ DRONE_MOTOR_FACTOR_R : const float
+ Motors()
+ init()
+ calculateCycle(alpha : float) : int
+ tick()
INTERVAL : micros_t
CETask
+ INTERVAL_US : const micros_t
- _absTime : sec_t
- _absMicros : micros_t
+ frequency() : micros_t
+ tick()
+ perform_tick()
+ seconds() : sec_t
+ micros() : micros_t
NavTask
- m_route : Waypoints
- m_control : NavController
- m_motors : NavController::Motors
+ NavTask(comm : CommTask*)
+ parseRoute(routecmd : const std::string&)
+ validateRoute(wps : const Waypoints&) : bool
+ onCommandReceived(cmd : const std::string&)
+ tick()
+ beginNavigation()
-m_route
«typedef»
Waypoints
Waypoint
+ longitude : double
+ latitude : double
+ customRadius : double
+ acquire : bool
-m_wps
0..1
Blackboard
+ sys : sys_struct
+ nav : nav_struct
+ sensors : sensors_struct
+ comm : comm_struct
-m_control
NavController
- m_wps : Waypoints*
- m_wp : int
+ NavController()
+ setup(wps : Waypoints&)
+ clearActuation()
+ withinRange(lon : nav_f_t, lat : nav_f_t, exit : bool) : bool
+ calcDistance_m(lonA : nav_f_t, latA : nav_f_t, lonB : nav_f_t, latB : nav_f_t) : nav_f_t
+ calcDistanceDumb_m(lonA : nav_f_t, latA : nav_f_t, lonB : nav_f_t, latB : nav_f_t) : nav_f_t
+ calcBearing_rad(lon1 : nav_f_t, lat1 : nav_f_t, lon2 : nav_f_t, lat2 : nav_f_t) : nav_f_t
+ fancymod(a : nav_f_t, n : nav_f_t) : nav_f_t
+ calcAngleDifference_rad(sourceA : nav_f_t, targetA : nav_f_t) : nav_f_t
+ step(time_delta : nav_f_t) : bool
NavController::NavControllerCfg
+ earthRadius_m : const nav_f_t
+ rangeToleranceRadius_m : const nav_f_t
+ rangeToleranceExitRadius_m : const nav_f_t
+ headingTolerance_rad : const nav_f_t
+ speedFull : const nav_f_t
+ speedMedium : const nav_f_t
+ speedLow : const nav_f_t
+ speedMediumDistance : const nav_f_t
+ speedLowDistance : const nav_f_t
+ timeForAlign : const nav_f_t
+ timeForArrival : const nav_f_t

Figure 11: UML class diagram for the navigation task and related components

4.2.3 Communication task

This task handles communication with the base station, connecting to the network and to the TCP server. It sends blackboard and log message information, receives commands from the station and handles or routes them to the appropriate task. The communication task also handles logging to the SD card. The main classes that support communication activities in Hydrus are shown in Figure 12.

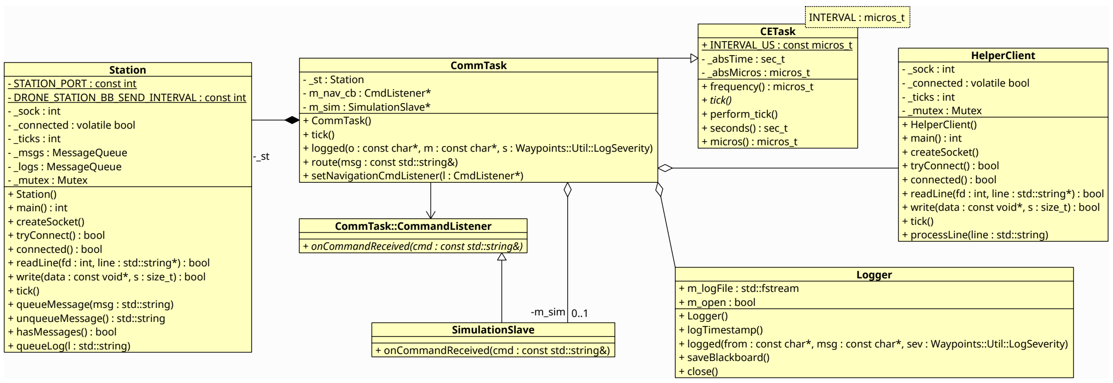

INTERVAL : micros_t
Station
- STATION_PORT : const int
- DRONE_STATION_BB_SEND_INTERVAL : const int
- _sock : int
- _connected : volatile bool
- _ticks : int
- _msgs : MessageQueue
- _logs : MessageQueue
- _mutex : Mutex
+ Station()
+ main() : int
+ createSocket()
+ tryConnect() : bool
+ connected() : bool
+ readLine(fd : int, line : std::string*) : bool
+ write(data : const void*, s : size_t) : bool
+ tick()
+ queueMessage(msg : std::string)
+ unqueueMessage() : std::string
+ hasMessages() : bool
+ queueLog(l : std::string)
-_st
CommTask
- _st : Station
- m_nav_cb : CmdListener*
- m_sim : SimulationSlave*
+ CommTask()
+ tick()
+ logged(o : const char*, m : const char*, s : Waypoints::Util::LogSeverity)
+ route(msg : const std::string&)
+ setNavigationCmdListener(l : CmdListener*)
CommTask::CommandListener
+ onCommandReceived(cmd : const std::string&)
-m_sim
0..1
SimulationSlave
+ onCommandReceived(cmd : const std::string&)
CETask
+ INTERVAL_US : const micros_t
- _absTime : sec_t
- _absMicros : micros_t
+ frequency() : micros_t
+ tick()
+ perform_tick()
+ seconds() : sec_t
+ micros() : micros_t
HelperClient
- _sock : int
- _connected : volatile bool
- _ticks : int
- _mutex : Mutex
+ HelperClient()
+ main() : int
+ createSocket()
+ tryConnect() : bool
+ connected() : bool
+ readLine(fd : int, line : std::string*) : bool
+ write(data : const void*, s : size_t) : bool
+ tick()
+ processLine(line : std::string)
Logger
+ m_logFile : std::fstream
+ m_open : bool
+ Logger()
+ logTimestamp()
+ logged(from : const char*, msg : const char*, sev : Waypoints::Util::LogSeverity)
+ saveBlackboard()
+ close()

Figure 12: UML class diagram for the communication task and related components

4.2.4 Sensing task

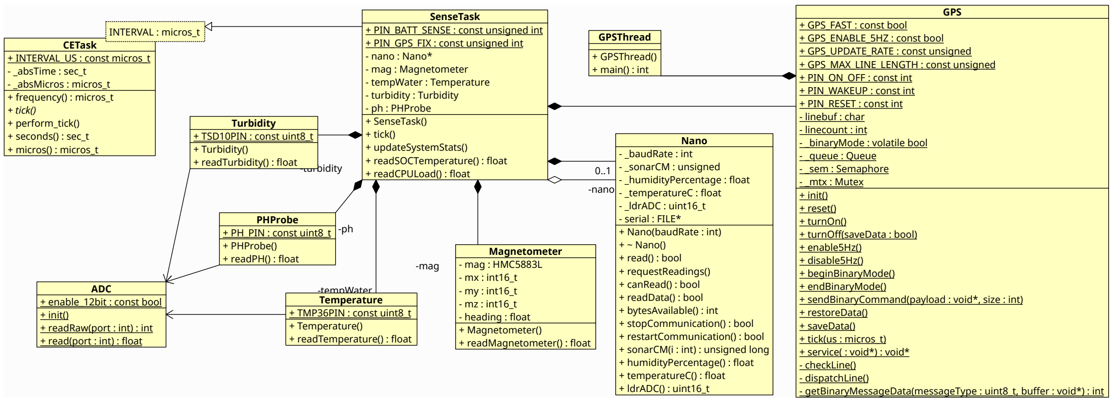


Figure 13: UML class diagram for the sensing task and related components

The sensing task is responsible for acquiring all sensor data and writing it to the blackboard. Information used for navigation, such as GPS and compass data, are acquired in this task as well. The `Turbidity`, `PHProbe`, and `Temperature` classes are wrappers for the respective hardware sensors. They acquire sensor readings via the ADC peripheral and convert them to the appropriate range. Sensor data from the auxiliary micro-controller is read using the `Nano` class, via redirection of the secondary serial port (`ttyS1`). This data is mainly distances retrieved from the ultrasonic rangefinder array. There is a compile-time toggle that enables reading humidity, ambient temperature, and ambient light level information from additional sensors.

4.3 Navigation System

The navigation controller is the heart of the drone's core functionality. It is based on a state machine. For every state, there is a corresponding behavior, or sub-controller. Therefore, top-level controller can be understood as a controller multiplexer.

The navigation states and their relationships are shown in a state diagram of Figure 14. Below we briefly describe each state and their relations.

- `NS_NOT_NAVIGATING` is the state in which the navigation controller is inactive, and does nothing. It changes from it when the navigation task tells the controller to start a navigation sequence.

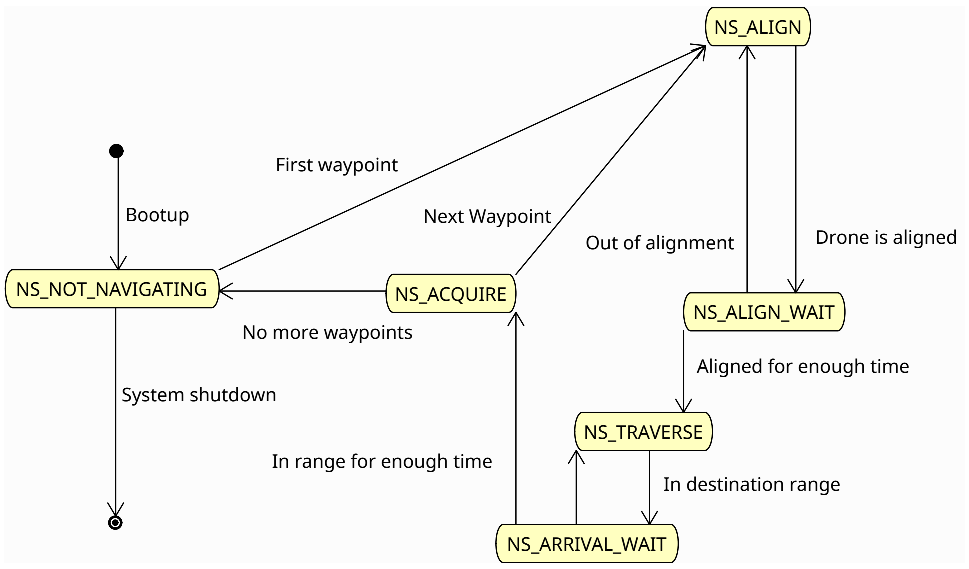


Figure 14: UML state diagram for the navigation controller

- `NS_ALIGN` is the first navigation state of a navigation sequence. It activates the controller that points the vessel towards the target waypoint. When the alignment is close enough, it changes to the `NS_ALIGN_WAIT` state. If the alignment is lost, the state falls back to `NS_ALIGN` again. However if the alignment is kept long enough, the state transitions to `NS_TRAVERSE`.
- In the `NS_TRAVERSE` state, the controller moves the vessel towards the waypoint, attempting to compensate for any misalignments by vectoring the thrust accordingly. It gradually decelerates as the drone approaches the desired waypoint. While the drone is close to the target waypoint, within the tolerance radius, it maintains the `NS_ARRIVAL_WAIT` state. If the drone is within radius for enough time, then it is deemed to be on the target point, and transitions to the `NS_ACQUIRE` state. It waits a little for sensor readings to stabilize, sets the next waypoint as the target, and then transitions back back to `NS_ALIGN`. However, if there are no further waypoints to reach, then the navigation sequence ends.

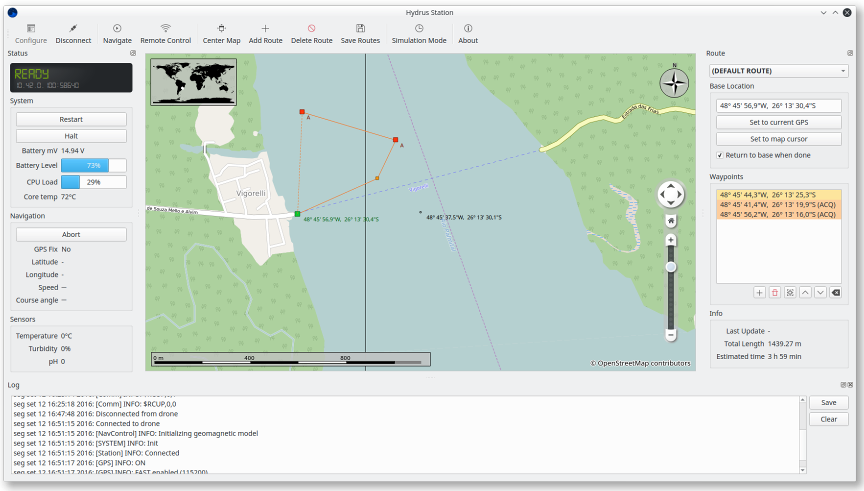
4.4 Base Station Software

The base station software is essential for the correct operation of the drone. It allows the user to program the desired route and acquire all the collected data after the drone finishes the navigation sequence. It also allows the user to inspect the data being collected by the drone in real time, check the battery level, and diagnose any problems with the sensors or the software stack. Figure 15 shows a screen capture of the base station application.

The application was written in C++, using Qt [6]. Its main view presents a map, powered by the Marble open-source project [10]. The map data is provided by the OpenStreetMap [11] community, and the renderings are provided by Marble over the internet. This instance of the map widget can display the currently planned route, the current cursor position, and the drone's current GPS position, if any. A toolbar on the top allows the user to perform the main actions the application allows: start navigation along the planned route, put the drone in RC mode and control it, and manage saved routes.

The left pane is the "Status" pane. It displays information about the current connection, and the most important state variables of the drone (battery level, CPU usage, current sensor readings, and GPS status). It also allows the user to reboot or halt the machine. The right pane is the "Route" pane. In this pane the user can edit the current route plan: add, delete, and move points, and change the position of the initial point. It also provides a rough estimation for the necessary time for the drone to complete the route.

Finally, the bottom pane displays the log of the current program execution and allows us to to diagnose problems during development.



Status

READY
10.42.0.100:58640

System

Battery mV 14.94 V
Battery Level 73%
CPU Load 29%
Core temp 72°C

Navigation

GPS Fix No
Latitude -
Longitude -
Speed –
Course angle –

Sensors

Temperature 0°C
Turbidity 0%
pH 0

Route

(DEFAULT ROUTE)

Base Location

48° 45' 56,9"W, 26° 13' 30,4"S

✓ Return to base when done

Waypoints

48° 45' 44,3"W, 26° 13' 25,3"S
48° 45' 41,4"W, 26° 13' 19,9"S (ACQ)
48° 45' 56,2"W, 26° 13' 16,0"S (ACQ)

Info

Last Update -
Total Length 1439.27 m
Estimated time 3 h 59 min

Log

```
seg set 12 16:25:18 2016: [Comm] INFO: $RCUP,0,0
seg set 12 16:47:48 2016: Disconnected from drone
seg set 12 16:51:15 2016: Connected to drone
seg set 12 16:51:15 2016: [NavControl] INFO: Initializing geomagnetic model
seg set 12 16:51:15 2016: [SYSTEM] INFO: Init
seg set 12 16:51:15 2016: [Station] INFO: Connected
seg set 12 16:51:17 2016: [GPS] INFO: ON
seg set 12 16:51:17 2016: [GPS] INFO: FAST enabled (115200)
```

Figure 15: Base Station software screen capture

5 Boat and Frame

In this section, we provide an overview of the fabrication process of the boat hulls. The boat was designed to be a model-sized catamaran, a type of dual-hull boat known to be very stable. To have a light and durable result, the team decided to create the hulls in composite fiberglass material.

The process was initiated with the fabrication of a male plug from sanded plastic material. Then, a female mold was laminated around it. Figure 16 shows the male plug and the mold. Inside the female mold, two hulls were laminated, and extracted. Their were later painted, and finalized for assembly.

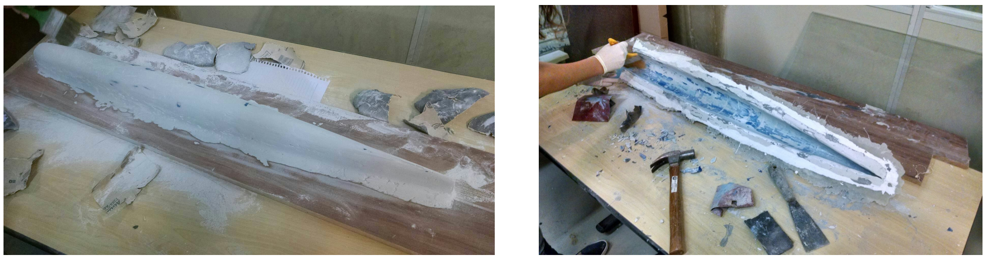


Figure 16: The male plug and the extracted fiberglass mold

Plug fabrication needed to be done carefully because we were working from plastic material, and the surface finish of the plug would transfer to the mold, and by extension, the finished hull. This process requires a lot of sanding for obtaining a smooth finish. The mold was laminated from layers of fiberglass and polyurethane resin over a gel coat layer, in a standard fabrication procedure for fiberglass hulls.

The hull lamination used the same fiberglass and polyurethane process, but inside the mold. Between lamination layers, aluminum brackets were installed for attaching the center platform. The hulls were extracted and painted with synthetic enamel. Figure 17 shows these steps.

Next, the hulls were assembled on an aluminum sheet platform, holes for the propeller axles were drilled, and top covers were made for the platform and hulls. Figure 18 shows the finished product. Overall, it was an arduous process, but the team has acquired insight on composite material fabrication techniques.

6 Testing and Validation

This section explores how the resulting drone was tested.

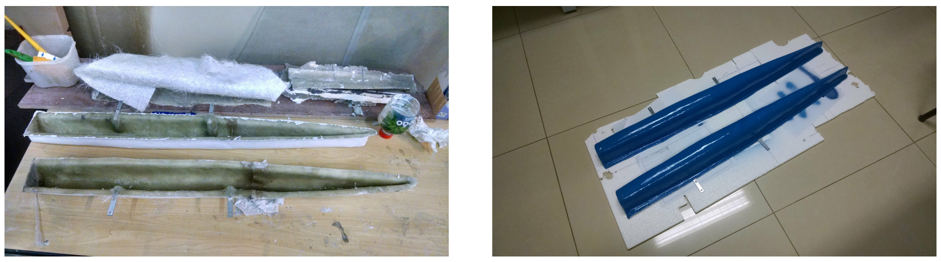


Figure 17: The hulls as extracted from the mold, and the finished product

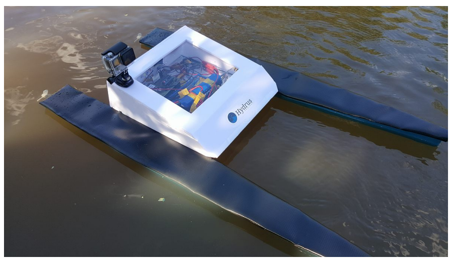


Figure 18: The finished boat

6.1 Subsystems tests

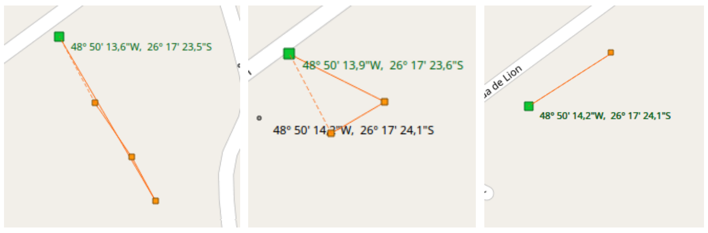
For the pH measuring testing we used three solutions that comes in the package of the pH sensor. Theses solutions brings us different known values of pH, one with pH 3.3, another one with pH 4.0 and the last one with pH 7.00. The test consists in submerging the cathodes of the pH meter, and with an oscilloscope check the voltage value on the low-pass filter. After the pH instrument calibration, the output electric tension value from the low-pass filter is proportional to the pH reading, approximately 1.19V for pH 3.30, 1.43V for pH 4.00, and 2.50V for pH 7.

In the turbidimeter testing, the sensor was placed together with the resistor needed for its operation, and then it was submerged in solutions with different NTU values, first clean water, to check the highest voltage corresponding to the lowest NTU output value, and then in extremely dirty water, to check the lowest voltage and highest NTU output. The component's datasheet offers a calibration curve for the sensor, since the output values from the tests with the two solutions were coherent with the expected, we just needed to use the manufacturer's information.

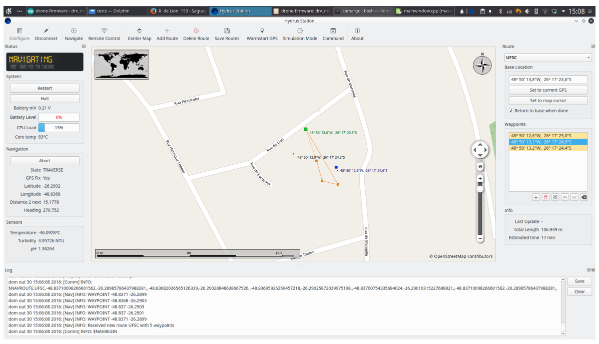
6.2 Integration testing and data plotting

The drone was iteratively tested during the integration phase, and bugs in the hardware and software were continually fixed. The navigation controller and the thrust vectoring algorithm were fine-tuned in this way, to compensate for changing boat dynamics.

The final testing procedure consisted on successfully performing a navigation sequence in a nearby lake. Navigational waypoints were selected in the map using the base station software. The base station PC was connected to the drone via WiFi during the while test, which allowed the team investigate all phases of the navigation. And spot potential problems. Figure 20 shows a screenshot of the base station software during a navigation session, for one of the tested routes.


48° 50' 13,6"W, 26° 17' 23,5"S
48° 50' 13,9"W, 26° 17' 23,6"S
48° 50' 14,2"W, 26° 17' 24,1"S
a de Lion
48° 50' 14,2"W, 26° 17' 24,1"S
Figure 19: Some of the routes that were tested

During the route testing we were able to verify the correct functioning of the navigation control algorithms and the sensor data acquisition. Some of the tested routes are shown in Figure 19. They were selected to verify the boats' ability to change direction mid-navigation, perform sharp turns, and maintain optimal orientation during traversal.



Status
NAVIGATING
192.168.43.73:58380

System
Battery mV 0.21 V
Battery Level 0%
CPU Load 15%
Core temp 83°C

Navigation
State TRAVERSE
GPS Fix Yes
Latitude -26.2902
Longitude -48.8368
Distance 2 next 15.1778
Heading 270.752

Sensors
Temperature -46.0928°C
Turbidity 4.95726 NTU
pH 1.36264

Route
UFSC
Base Location
48° 50' 13,8"W, 26° 17' 23,6"S
✓ Return to base when done
Waypoints
48° 50' 12,6"W, 26° 17' 25,0"S
48° 50' 13,1"W, 26° 17' 24,9"S
48° 50' 13,2"W, 26° 17' 24,4"S

Info
Last Update -
Total Length 106.949 m
Estimated time 17 min

Log
```
dom out 30 15:06:08 2016: [Comm] INFO:
$NAVROUTE,UFSC,-48.83710098266601562,-26.28985786437988281,,-48.83682036565126339,-26.29028848638667526,,-48.83695926359457218,-26.29025872039975198,,-48.83700754335684024,-26.29010315227688821,,-48.83710098266601562,-26.28985786437988281,,
dom out 30 15:06:08 2016: [Nav] INFO: WAYPOINT -48.8371 -26.2899
dom out 30 15:06:08 2016: [Nav] INFO: WAYPOINT -48.8368 -26.2903
dom out 30 15:06:08 2016: [Nav] INFO: WAYPOINT -48.837 -26.2903
dom out 30 15:06:08 2016: [Nav] INFO: WAYPOINT -48.837 -26.2901
dom out 30 15:06:08 2016: [Nav] INFO: WAYPOINT -48.8371 -26.2899
dom out 30 15:06:08 2016: [Nav] INFO: Received new route UFSC with 5 waypoints
dom out 30 15:06:08 2016: [Comm] INFO: $NAVBEGIN
```

Figure 20: The station software during a navigation session

During the entire testing period while the boat was in the water, we could observe that the sensors were returning data, and functioned as expected. The change in measurement values could be seen in real time in the base station application. Figure 21 shows the boat during an autonomous navigation session.

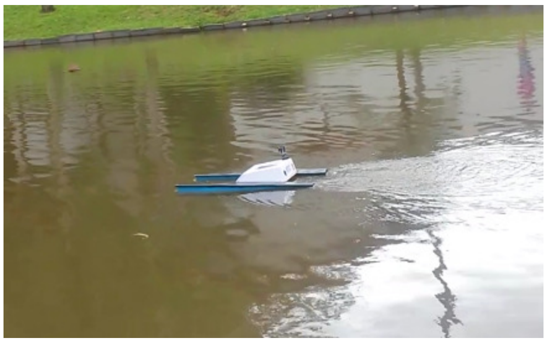


Figure 21: The boat during a navigation session

Data plotting facilities were introduced to the base station application after the navigation tests. We were able to verify the correct functioning of the map plotting by moving the drone hardware around a mock path and logging sensor data. The logs were then exported from the drone to the computer for plotting. Figure 22 shows the resulting plotted path. The line color's hue represents the value in proportion to the minimum and maximum values, blue being the minimum value, green an intermediary value, and red the maximum temperature value.

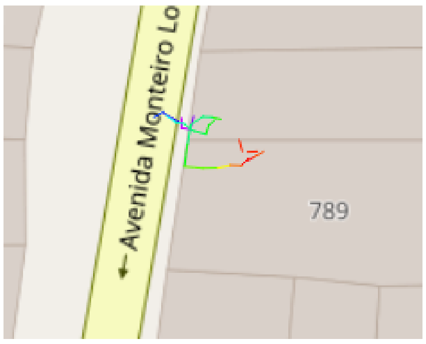


Figure 22: Collected temperature data plotted in the drone station app.

7 Challenges and Solutions

The Hydrus project faced many challenges in its development process. In retrospect, the team considers it to have been excessively horizontal, embracing a wide range of development and construction tasks. That resulted in a heavy workload for team members. Coordinating that much work between ourselves was a big challenge, and a more focused approach would greatly benefit our workflow in the future.

Working with the Galileo Gen 2 board was a mixed experienced. Our main problem was their fragility. They are easily bricked and the power conversion stages are easy to damage. In one episode, plugging it to a battery pack at 14.5 volts caused the first-stage regulator (a TPS63130, `U6` in the official schematic) to burst into flames. The team believes this was due to excessive inrush current, that a common power supply would normally prevent. However, the Li-PO battery can easily provide the excess current. Other boards were damaged as well, in a similar manner, and this threatened the completion of the project. A couple of damaged boards could still be utilized by feeding 5V into the `5V_ALWAYS_ON` circuit net directly, using a separate switching regulator.

An interesting effect is that an external interference on the I2C bus can prevent the sketch from executing. If any of the bus lines can't be pulled low, and the devices in the bus do not respond, the kernel driver that controls the external peripherals constantly times out, dragging the execution of any sketch to a halt. This was an occurrence that puzzled the team, and took some time to debug and fix, by removing a damaged device from the bus.

There were several occasions in which sketches would fail to upload repeatedly, and an attempt to reset the running sketch would result in the serial connection being dropped, requiring a complete system reboot. The team considered this to be a minor annoyance.

8 Final Considerations

In this project, the team has accomplished the following results:

- designed the hardware and constructed a custom Galileo shield for housing modules and connectors;
- constructed all of the hardware interconnects;
- designed and wrote the drone's custom firmware, which includes GPS interfacing, sensor data acquisition, navigation control, and extra functionalities;
- implemented companion desktop software for interfacing with and programming the drone; and
- Validated and tested sensor data acquisition and GPS navigation.

The team has built an autonomous boat drone for water quality measurements, fulfilling all of the initial project goals. The Hydrus project was a challenging task, but we are very satisfied with our results.

For future work, we now have an extensible platform made by ourselves, that can be adapted for other functionality and goals. Investigation is ongoing on how to adapt the platform to perform ultrasound bathymetry for usage in river beds and port dredging applications. We also plan to conduct a viability study of commercial usage of the drone.

Figure 23: From left to right, Lucas, Eloi (a friend of the team), Êmili, Guilherme, and Giovani, together in the final testing day.

References

[1] Ltd Aosong(Guangzhou) Electronics Co. Digital-output relative humidity & temperature sensor/module am2303.

[2] Edwin P Arthur and John E Leonard. Ion measurement apparatus and method, 1965.

[3] Atmel. Atmega48a/pa/88a/pa/168a/pa/328/p, 2015.

[4] GE Measurement & Control. Tsd-10 turbidity sensor, 2013.

[5] Analog Devices. Low voltage temperature sensors tmp35/tmp36/tmp37, 2015.

[6] Digia and the Qt Project. Qt documentation, 2016.

[7] B.P. Douglass. *Real-time Design Patterns: Robust Scalable Architecture for Real-time Systems*. Number v. 1 in Addison-Wesley object technology series. Addison-Wesley, 2003.

[8] ElecFreaks. Ultrasonic ranging module hc-sr04.

[9] Honeywell. 3-axis digital compass ic hmc5883l, 2013.

[10] Marble Globe Project. Marble - find your way and explore the world, 2016.

[11] The OpenStreetMap Project. Openstreetmap, 2016.

[12] Anaheim Scientific. User manual model p771 ph meter, 2012.

[13] Maestro Wireless Solutions. A2035-h gps module.

[14] Lynxmotion UAV. Simonk esc user guide.

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Pró-Reitoria de Graduação

Departamento de Administração Escolar

ATESTADO DE MATRÍCULA

Atesto, a requerimento da parte interessada e segundo consta em nossos arquivos, que EMILI BOHRER, identidade: 7110361784/RS - SSP, C.P.F.: 03231149037, foi admitido(a) nesta Universidade através de vestibular optante no curso de ENGENHARIA MECATRÔNICA [Campus Joinville], sob o número 15159546, estando regularmente matriculado(a) no segundo semestre de 2016.

Disciplina	Turma	Nome da Disciplina	Aulas	Horários/Locais
EMB5109	05604	Gestão Industrial	4	2.1330-2 / JOI-A209 5.0820-2 / JOI-A210
EMB5111	05605	Introdução ao Controle	4	4.1510-3 / JOI-A209 6.1010-2 / JOI-A215
EMB5603	05605	Introdução às Estruturas de Dados	6	2.0910-3 / JOI-A215 5.1330-3 / JOI-A215
EMB5108	05603A	Circuitos Elétricos	4	3.1330-2 / JOI-A111 5.1010-2 / JOI-B104
EMB5320	09603A	Empreendedorismo e Inovação	2	3.1010-2 / JOI-E205

Carga horária semanal do curso: Mínimo=16, Médio=26, Máximo=28, Cursando=20

Florianópolis, 30 de Outubro de 2016

UNIVERSIDADE FEDERAL DE SANTA CATARINA
Pró-Reitoria de Graduação
Departamento de Administração Escolar

ATESTADO DE MATRÍCULA

Atesto, a requerimento da parte interessada e segundo consta em nossos arquivos, que GUILHERME AUGUSTO PANGRATZ, identidade: 5924381/SC - SSP, C.P.F.: 07625239993, foi admitido(a) nesta Universidade através de vestibular optante no curso de ENGENHARIA MECATRÔNICA [Campus Joinville], sob o número 15159554, estando regularmente matriculado(a) no segundo semestre de 2016.

Disciplina	Turma	Nome da Disciplina	Aulas	Horários/Locais
EMB5602	07605	Controle Digital	4	5.1510-3 / JOI-E114 5.1830-1 / JOI-E114
EMB5606	07605	Hardware para Sistemas Embarcados	4	2.1010-2 / JOI-E215/C 6.1010-2 / JOI-E215/C
EMB5607	07605	Processamento Digital de Sinais	4	3.0820-2 / JOI-E114 5.1010-2 / JOI-E114
EMB5608	08605	Sistemas Operacionais e de Tempo Real	6	2.1330-3 / JOI-E114 4.1330-3 / JOI-E114
EMB5628	07605	Sistemas Motrizes II	3	4.0730-3 / JOI-E114

Carga horária semanal do curso: Mínimo=16, Médio=26, Máximo=28, Cursando=21

Florianópolis, 30 de Outubro de 2016

A autenticidade desse documento pode ser verificada no endereço "http://www.cagr.ufsc.br/autenticidade" informando o seguinte código: 356768-45000002050578

**UNIVERSIDADE FEDERAL
DE SANTA CATARINA**

**Pró-Reitoria de Graduação
Departamento de Administração Escolar**

ATESTADO DE MATRÍCULA

Atesto, a requerimento da parte interessada e segundo consta em nossos arquivos, que LUCAS PIRES CAMARGO, identidade: 5874993/SC - SSP, C.P.F.: 07276599957, foi admitido(a) nesta Universidade através de concurso vestibular no curso de ENGENHARIA MECATRÔNICA [Campus Joinville], sob o número 11103082, estando regularmente matriculado(a) no segundo semestre de 2016.

Disciplina	Turma	Nome da Disciplina	Aulas	Horários/Locais
EMB5602	07605	Controle Digital	4	5.1510-3 / JOI-E114 5.1830-1 / JOI-E114
EMB5605	07605	Eletrônica de Potência	4	3.1010-2 / JOI-E114 6.0820-2 / JOI-E114
EMB5606	07605	Hardware para Sistemas Embarcados	4	2.1010-2 / JOI-E215/C 6.1010-2 / JOI-E215/C
EMB5609	06605	Sistemas de Comunicação	4	3.1330-2 / AUX-ALOCAR 5.0820-2 / AUX-ALOCAR
EMB5618	08605	Planejamento do Trabalho de Conclusão de Curso	2	2.0820-2 / AUX-ALOCAR

Carga horária semanal do curso: Mínimo=16, Médio=26, Máximo=28, Cursando=18

Florianópolis, 30 de Outubro de 2016