IQT24 V2 Guide to Electronics (Rev 1)

Project Members:
Theo DeGuzman | Genevive Mehra | Owen Mitchell | Iris Renteria

POC for questions:
Theo DeGuzman
tdeguzm1@jhu.edu
tdeguzm1@alumni.jhu.edu

Instructors:
Rich Bauernschub
Dr. Stephen Belkoff

Sponsored by:
In-Q-Tel Labs
Logan Kunka

PROJECT OVERVIEW:

This manual is part of a Mechanical Senior Design Project for Johns Hopkins University, sponsored by In-Q-Tel Labs. IQT Labs is a branch of the IQT venture capital firm, supporting emerging technologies that may be useful to the United States.

This project is part of an effort to understand the concept of maritime domain awareness. The code and electrical schematics attached are designed to operate an autonomous vessel.

This guide is structured as a manual so that a person could reproduce our work, for minimal cost or engineering effort in terms of electronics knowledge. The full final report can be requested, but details like hull specifications, power requirements, etc. will be left as an exercise to the reader should they choose to reproduce this work. Instead, we will focus on the non-power electronics and computational aspects of the project.

Being designed for a university project, the code and processes discussed have been implemented and tested on hardware, but they are also unrefined and do not follow all coding best practices.

The scope of this manual is to provide the electrical and computational foundations of a platform that can:

1. Operate unsupervised, in open water, for an undetermined amount of time
2. Be controlled remotely, anywhere in the world, using the Iridium Satellite network
3. (Electrically) support a payload
4. Support up to 8 onboard sensors with an analog voltage output of 0-3.3V
5. Be adapted to operate on other physical platforms (relatively) easily

SYSTEMS OVERVIEW:

This code is reliant on a number of assumptions about the overall system:

Power:

Energy is provided to the system from a system with a nominal voltage of around 24V. This power is also assumed to both deplete and recharge, through a solar panel array or other means. To protect the batteries, the user sets some lower bound out output voltage, below which the system enters a safe mode to protect the batteries.

Propulsion:

The system has some form of propulsion, controlled by devices that accept a 3.3V PWM input between 1000 and 2000. These range values can be adjusted by hardware or code adjustments. Currently, the system uses differential thrust but can be adapted to use a thruster and rudder combination (as it did in earlier versions of the vessel).

Sensing:

The platform, as designed, allows for 8 sensors. The intent is to primarily measure battery values and temperatures, but they can be adapted to measure most anything. The provided hookups provide 3.3V, 5V, or 24V power. The inputs to the sensors take in voltage of 0 to 3.3V and convert it to an 8 or 10-bit digital value. These inputs are not regulated, so a sensor outputting more than 3.3 V can severely damage the attached microcontrollers.

Radio control:

The radio control module used, for close-range manual operation, was the Radiolink RC4GS V3. Any controller with PWM outputs can be used, but the values in the code will need to be adjusted to compensate. This controller outputs 1000 to 2000 microsecond pulses, with 1500 being neutral steer and no throttle (forwards or backward).

Satellite control:

The satellite control module for this project was the RockBLOCK Plus. This is a module that can communicate with the Iridium Satellite network, which is a constellation that covers everywhere on Earth, at all times. This can be purchased from Ground Control, and a Cloudloop account can be used to send messages to and receive messages from the module. A subscription fee and a data transfer fee will apply to usage of the network.

SOURCE CODE:

All of the source code is available at:

ELECTRICAL DETAILS:

A schematic of the electrical components can be found in GitHub repository. The code comments also detail all of the connections between the various components. It also contains a PCB that can be used to mount the boards with minimal effort or wiring troubles. This section outlines the actual boards used.

Main/Driver board: responsible for motor control, RC radio communication, satellite communication, SD card management, and battery monitoring

Arduino Due -> ARM 32-bit microcontroller with 3.3V logic levels

Peripheral boards (2x): responsible for communication with GPS and IMU (inertial measurement including accelerometers and gyroscopes), filtering this data, and providing it to the main board when requested

Arduino Nano BLE 33 Sense -> has integrated IMU and 3.3V logic levels

GPS Module (2x): Responsible for providing accurate world coordinates and communicating those back to peripheral boards

NEO-M9N, U.FL Sparkfun GPS breakout +
Bingfu Waterproof Active GPS Navigation Antenna

Watchdog: Responsible for monitoring the main board and resetting it after a specified time interval without seeing any heartbeats.

Arduino Nano IOT -> 3.3V logic levels
(and far more sophisticated than necessary for this purpose)

Buck Converter: Steps 24V battery supply down to 15V, which is usable by all boards and the satellite modem.

D-Planet 5A DC-DC Adjustable Buck Converter

Radio Controller + Transceiver: Responsible for switching the boat into manual mode at short range and allowing a user to have fine control over the boat.

Radiolink RC4GS V3

Satellite Modem: Responsible for sending and receiving byte-string messages with the Iridium satellite network.

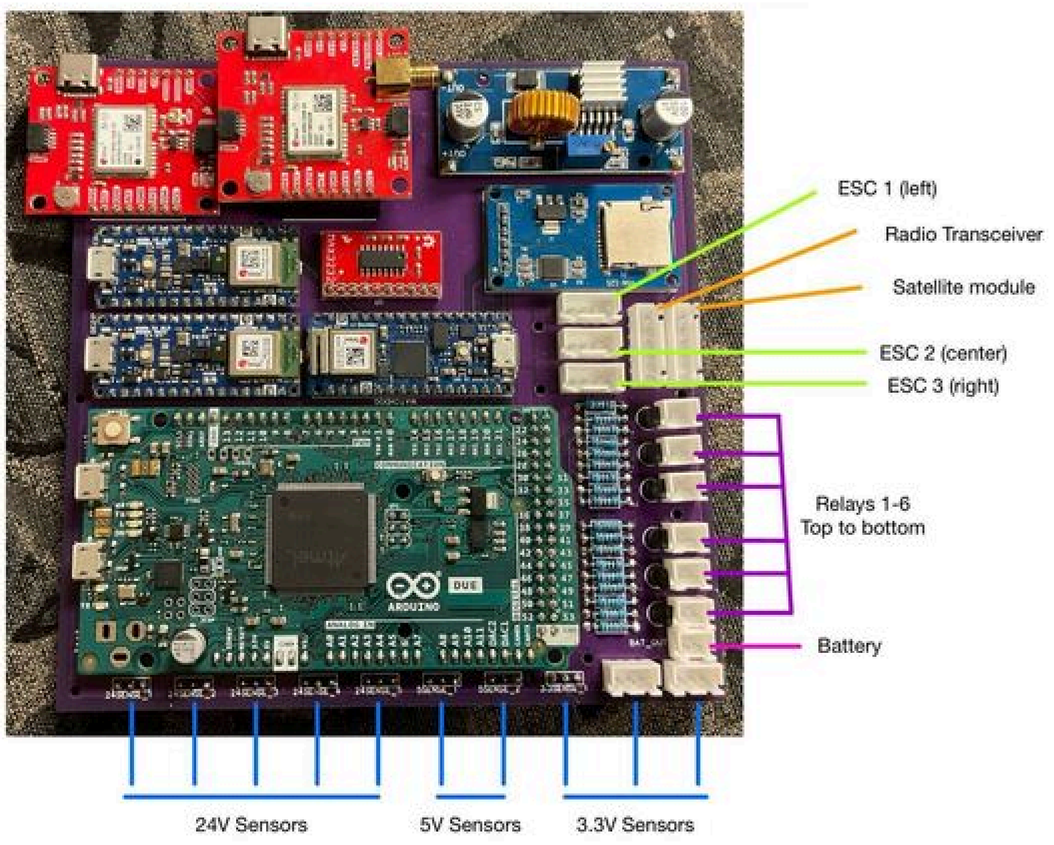
RockBLOCK Plus + Cloudloop subscription

Max_3232 Breakout: Responsible for converting between 0-3.3V UART Serial outputs from the main board to the +/- 10V RS-232 of the satellite communication protocol.

HiLetgo Micro SD Card Adapter/Reader, SPI: Responsible for sending information between the main board and the SD card

Solid State Relays: With transistor setup as shown in the schematic, main board digital IO pins can control power supply to various components, using 1864-RM1D060D50-ND relays.

In addition to the schematic, see the following photo for hooking up the system on the PCB designed.



ESCs	5V	Primary PWM	Aux PWM	GND
Sensors	3.3/5/24V	Primary PWM	Aux PWM	GND
Battery	24V	GND		
Relay	Trigger	GND		

RC	RockBLOCK
GND	GND
5V (ESC)	None
Thrust PWM	None
Steer PWM	RS-232 TX
Manual Select	RS-232 RX
5V (Arduino)	15V

SOFTWARE THEORY OF OPERATION:

The software, as designed, should work on a system, as designed with no edits. However, a high-level software overview will be provided later, to give a starting place should edits be needed for fundamental architecture changes.

In this section, it will be shown how to operate the boat, in its current configuration.

Step 1. Set up a default mission in firmware.

> Should an SD card failure occur, the boat should have a fallback plan. Often this is simply a homing command, to return to some point.
>
> This can be edited by changing lines 171 to 230 in IQT_DRIVER.ino
>
> For the most part, this will simply involve changing lines 194 and 195 to change homing latitude and longitude. However, the code is set up such that an entire alternative mission can be set up if desired. It is not recommended that a long-duration mission with more than one waypoint be chosen as the default mission.

Step 2. Set up mission parameters and waypoints on the SD card.

> This is the normal mode of operation for how the vehicle should work. It also allows for the modification of parameters through satellite communication.
>
> The SD card can be opened on any computer. The mission parameters are stored as one file. Each waypoint is stored as a separate file, with the file name indicating its index.
>
> Each of the files is a .json file. Examples can be found in the repository and modified for the particular mission. Explanations for all the parameters can be found in IQT_COMMUNICATIONS.h. The major items to be changed per mission are the waypoint parameters.
>
> Important: global coordinates are stored in longs and represent the degree value E7 (i.e. times ten to the seventh). Also, the convention is that north and east are positive; south and west are negative.

<u>Step 3.</u> Plug in the boat and place it in water.

Turn the RC receiver on, and ensure that it is communicating a manual mode command. Power on all power electronics and place the boat in water.

Navigate the boat to a reasonable start point, and switch out of autonomous command.

<u>Step 4.</u> Satellite communications:

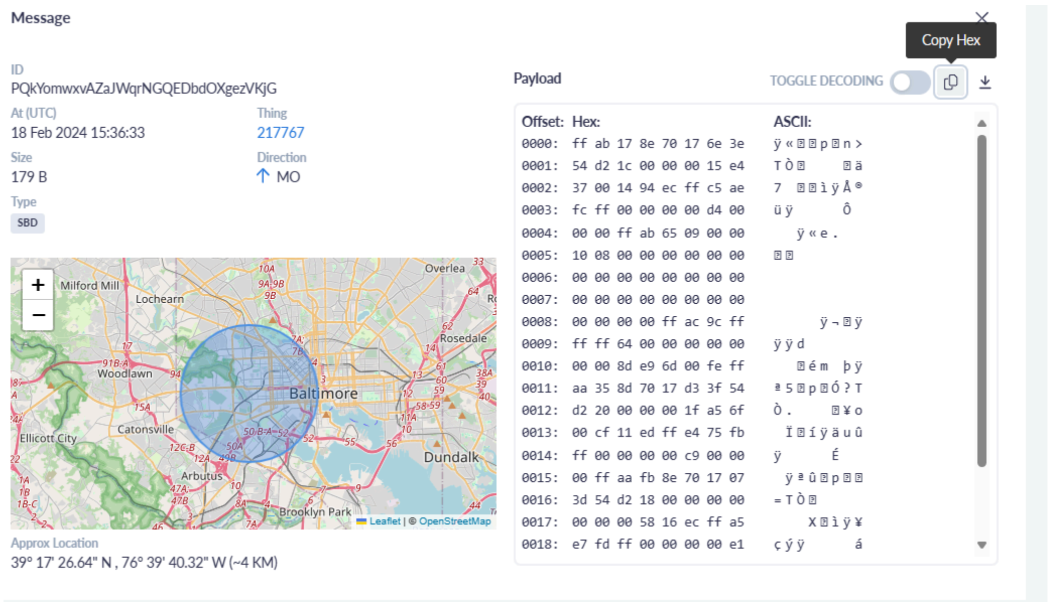
Receiving messages:

At a time interval (given in your mission parameters), the boat will send an update message to the network.

To see this message, log into Cloudloop and go to your “Data” menu.

Then go to “messages” and input your desired time interval. Each message will appear.

Click on the message, toggle decoding to “off”, and select the “copy hex” button



Message

ID
PQkYomwxvAZaJWqrNGQEDbdOXgezVKjG

At (UTC)
18 Feb 2024 15:36:33

Thing
217767

Size
179 B

Direction
↑ MO

Type
SBD

Approx Location
39° 17' 26.64" N , 76° 39' 40.32" W (~4 KM)

Payload

```
Offset: Hex:                      ASCII:
0000:  ff ab 17 8e 70 17 6e 3e    ÿ«□□p□n>
0001:  54 d2 1c 00 00 00 15 e4    TÒ□   □ä
0002:  37 00 14 94 ec ff c5 ae    7 □□ìÿÅ®
0003:  fc ff 00 00 00 00 d4 00    üÿ    Ô
0004:  00 00 ff ab 65 09 00 00      ÿ«e.
0005:  10 08 00 00 00 00 00 00    □□
0006:  00 00 00 00 00 00 00 00
0007:  00 00 00 00 00 00 00 00
0008:  00 00 00 00 ff ac 9c ff        ÿ¬□ÿ
0009:  ff ff 64 00 00 00 00 00    ÿÿd
0010:  00 00 8d e9 6d 00 fe ff      □ém þÿ
0011:  aa 35 8d 70 17 d3 3f 54    ª5□p□Ó?T
0012:  d2 20 00 00 00 1f a5 6f    Ò.    □¥o
0013:  00 cf 11 ed ff e4 75 fb     Ï□íÿäuû
0014:  ff 00 00 00 00 c9 00 00    ÿ    É
0015:  00 ff aa fb 8e 70 17 07     ÿªû□p□□
0016:  3d 54 d2 18 00 00 00 00    =TÒ□
0017:  00 00 00 58 16 ec ff a5       X□ìÿ¥
0018:  e7 fd ff 00 00 00 00 e1    çýÿ    á
```

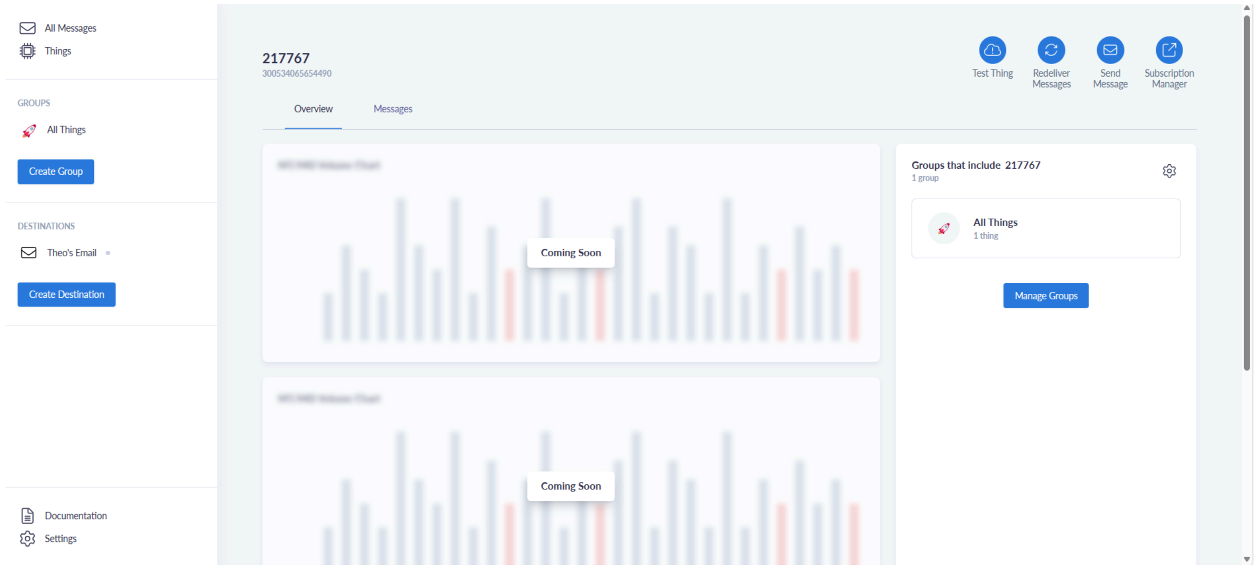
IQT_COMMUNICATIONS.H provides the communication protocol for all of these incoming messages. Similarly, a review of the IQT_COMMUNICATIONS.H and IQT_vehicle.h provides all the possible information about how to format these outgoing messages.

However, to simplify things for the user, a .pynb file is included to automatically format incoming and outgoing messages.

Simply copy the message into the appropriate code block, and the system will decode it.

Sending messages:

Similarly, parameters can be put into the correct code block to send a message to the boat, and the system will output a hex string that can be pasted into Cloudloop to send a message.



To send a message, click on “things”-> ”##### (of your modem)” -> “Send message” -> select “SBD” -> paste your payload into the box -> press “Send”

With this, the boat should be operable to complete whatever mission you choose.

SOFTWARE DETAILS:

Diving into software details, we will focus almost entirely on the main board and ways that may be beneficial to modify the related procedures. Additionally, there will be a brief coverage of guidance, navigation, and control topics, as they are implemented on the vehicle, should the user wish to refine them beyond their current state.

The GPS peripheral boards and watchdog board are very simple and well-explained in the code documentation

The main board goes through the following process on every loop:

1. Check if in manual mode on the last time step, if so, skip steps 2-4
2. Check GPS/INS peripherals for vehicle position, orientation, speed, etc.
3. Estimate vehicle state and heading based on that information
4. Read battery values and other sensor data
5. Determine mode
6. Determine if a new waypoint has been reached
7. Execute appropriate motor and relay commands
8. Record vehicle and controller state back to SD card (if not in manual mode)
9. Check for incoming satellite messages and send messages if appropriate
10. Send out heart beat signal

Detailed documentation on every one of these processes is available in the code comments.

Possible changes:

If your RC transceiver outputs different PWM ranges than given:

Go to IQT_vehicle.cpp and edit the map command on line 640

Change map(ch, 1000, 2000, minLimit, maxLimit) to

map(ch, <your min pwm>, <your max pwm> , minLimit, maxLimit)

If your ESC output is of a different range than 1000 to 2000:

Go to the begin() method in IQT_controller.cpp

Edit the this->t#.attach(t#_pin, 1000, 2000) to instead be

this->t#.attach(t#*pin,* <esc min pwm>, <esc max pwm>)

If your architecture is not a differential thrust system or uses different configurations:

Add the relevant servo information in the begin() method in IQT_controller.cpp

Edit the method command_motors to change the actuator outputs to your desired values

this->thrust provides a thrust command between 0% and 100%

this->steering provides a steering command between -100% and 100% where positive is a right-hand turn

this->trim provides a steering correction based on accumulated errors

To add different sensors to your system:

Go to read_battery() in the IQT_vehicle.cpp

Edit the meaning of the analog read values (and map them to longs)

Remember that the expected inputs are in the range of 0 to 3.3V and overvoltage can cause electrical damage to the microcontrollers

Navigation:

The GPS modules use built-in functions to determine global position based on GPS satellites. The NEO-N9M modules use several constellations and have empirically shown very precise values with high accuracy. There is also some internal filtering that assists with this precision. The main controller aggregates the readings from two separate peripheral modules by using a weighted average of the GPS readings, with weights being the complement of the dilution of precision. The current heading is determined as the finite difference between two unique fixes.

Guidance:

The guidance portion of the algorithm operates, in general as follows:

1. Define the start position as the previous waypoint and your end position as the target waypoint
2. Calculate the distance (d) from the start to the current boat position
3. Find the point that is d away from the start, along the line (great circle) that connects the start and finish
4. Progress some additional distance (lead distance) along that line, moving toward the finish
5. Make this point as the target/aim point
6. Determine the heading between the current point and the aim point

Control

The control loop develops values for thrust and steering/trim using two separate control loops. These are returned as percentages, so they can be used regardless of propulsion/steering architecture.

The thrust control loop operates as follows:

1. The difference between the current speed and the target speed is calculated as the error in the speed
2. If the error is big (greater than the threshold provided in mission parameters)
    a. Set the thrust value to 50%, and skip steps 3 and 4
3. If the speed needs to be increased, the thrust value is incremented by some small, fixed value
4. If the speed needs to be decreased, the thrust value is decremented by some small, fixed value

The steering/trim control loop operates as follows:

1. The difference between the current heading and the target heading is calculated as the error in the heading (constrained to the smaller absolute value between -180 and 180 degrees)

2. If the error is big (greater than the threshold provided in mission parameters)
    a. Run a proportional control law for steering, using the gain *steer_big_p_gain*

3. If the error is small (less than the threshold provided in mission parameters)
    a. Run a control loop with the following properties:
        i. Steering value runs as a proportional control law *steer_small_p_gain*
        ii. Trim is set to zero on initialization, and then increments by a small, fixed value *steer_i_gain* based on the sign of the error.
            1. If the error changes sign and then also exceeds a magnitude in the other direction, reset the trim to zero.