Project Ladon Hackerboat Systems Architecture

# Hardware Architecture

The Hackerboat control system is built around a Beaglebone Black, located in a drybox in the forward compartment of the boat. This box contains the Beaglebone, RC receiver, AIS receiver, shore communications transceiver, GNSS receiver, IMU, and associated electronics.

In addition to the main control system box, there is a power distribution box and a motor control box. In the current iteration, the motor control box contains a set of automotive relays that control power to the different windings of the trolling motor.

The power distribution box contains switches and relays to control power distribution to different boat systems. It also contains a latching relay to control the armed status of the boat and maintain it through a power cycle.

## Main Board

The main board shall serve as a carrier for the Beaglebone Black as well as core navigation sensors and communications interfaces.

### Power Input, Conditioning, & Regulation

Battery power (12V nominal; 9-20V permitted) shall come in through a 0.375” pitch terminal strip on the top edge of the PCB. The circuit shall be protected by a 3A-5.2A SMD PTC fuse. To protect the circuit from reverse connections, a P-channel MOSFET capable of passing 12A (AO4407A) shall be connected in series with the main power rail with the gate attached to ground. The gate voltage limit of this part (-20V) shall determine the maximum input voltage of the board. This circuit must not pass any current when the power connections are reversed.

There shall be a green power status light that is illuminated when the board is powered correctly and a red one illuminated when connected backwards. In addition, there shall be a red fuse status LED that lights when the fuse is open.

#### Battery Monitor

The battery monitor circuit shall be directly connected to one of the Beaglebone’s analog inputs. Since these inputs are limited to 0-1.8V, the signal must be passed through a voltage divider and must be further protected by a 1.8V Zener diode to prevent voltage excursions from damaging the circuitry. At 12V nominal input voltage, the expected output of the voltage divider shall be 1.44V.

#### Main 5V Supply

The main 5V system supply shall be provided by a V7805-2000 5V/2A switching regulator module. This is designed as a drop-in replacement for the venerable 7805 linear regulators, except with less heat dissipation and much higher efficiency. This shall be connected to the Beaglebone and the 5V inputs of most onboard peripherals.

This power rail shall be available on pins 1-4 of the Accessory Power Connector.

#### Lighting 5V Supply

Power for the status lights shall be provided by an identical regulator to the main 5V supply. It is independent to prevent any electrical issues with the lighting from disrupting the main control system. Additionally, the power rail to the lights shall be protected by a fuse identical to the main power input.

#### Rudder Servo Supply

The rudder servo supply shall be independent and set to 7.4V to accommodate the needs of COTS RC servos. It shall be designed to be turned on only when the software and system are in a mode where outputs are active. This is to reduce the possibility of pinch injuries in the exposed reduction gear. Hitting the large red stop button shall immediately depower the servo supply.

To accomplish these aims, the enable line of the 7.4V regulator shall be driven by an AND gate with two inputs – one from a Beaglebone GPIO pin (level shifted as required) and one from the Enable line controlled by the enable/disarm relay on the power distribution board. The Beaglebone connection shall be via a level shifter connected to pin P8.17.

A green status LED shall illuminate when the 7.4V supply is active.

The output current to the servo shall be monitored by a current shunt resistor and an INA169 high-side current sense amplifier or equivalent connected to the ADC subsystem, chip U19, channel 7. Gain and shunt resistors shall be selected to provide an output of 1.65 V/A.

Backup overcurrent protection shall be provided by the saturation limits of the switching supply core.

#### 3.3V Supply

This supply is provided by the Beaglebone’s internal power regulation circuitry. It is brought out largely to supply the 3.3V rail for the level shifters that allow the Beaglebone’s GPIO pins to interface correctly with 5V input circuitry. This rail shall be provided to external circuitry on pins 7-10 of the Accessory Power Connector. It is not protected and must be used with care to prevent disruption to the Beaglebone. Indication of the status of this bus may be provided by the Beaglebone’s own power status light.

#### PoE Supply

The main board shall provide a 15V/1A power supply for Ubiquiti PoE devices. This shall be a boost supply that is always on. Overcurrent protection shall be provided by core saturation.

#### Grounding

All grounds on the board shall be connected to a common ground plane occupying both sides of the PCB. The corner mounting holes shall be plated through and connected to the common ground plane.

### Beaglebone Hardware

The Beaglebone Black shall be configured with the univ-all cape file in order to provide access to all of the pins for the internal software.

### IMU

The IMU shall be a 9 or 10 DoF unit from Adafruit. Acceptable units include <https://www.adafruit.com/products/1604>, <https://www.adafruit.com/products/1714>, and future pin-compatible replacements. The main board shall have mounting holes for a compatible header and the four corner mounting holes. The mounting holes on the PCB shall be grounded. The four corner mounting holes shall be secured to the board with appropriate conductive standoffs. The Vin connection shall be connected to the main 5V rail. The I2C lines shall be connected to the I2C-1 bus on the Beaglebone, using 4.7K pull up resistors. These are 3.3V signals already, so no level shifting is required.

The alignment of the IMU relative to the hull of the boat shall be with the X axis pointing forward, the Y axis pointing to port, and the Z axis pointing up.

### Relay Drivers

#### General

All of the relay drivers shall consist of the following components:

* A high-side driver with built-in fault, short, and overtemp detection, such as the AUIPS6041G. The fault detection line shall be of an open-drain type, and shall be connected to the 3.3V rail via a pull-up.
* A level shifter to drive the switch input from the 3.3V logic output of the Beaglebone.
* Each switch input shall be pulled down to prevent unintended activation when the Beaglebone pin is in a high-Z state.
* A current monitoring circuit arranged so that the output is 1.65 V/A.
* Status LEDs indicating both active (green) and fault (red) conditions for each channel
* Output clamp diodes.

#### Motor Controls

Controlling the motor requires driving six automotive relays. These relays shall be numbered 0-5 and connected to a standard 9-pin D-subminiature plug as the output. This connector shall be labeled as Motor Control Relays or similar legend.

These lines shall pass out of the main drybox via a seven pin SP-13 connector and enter the motor control box through the existing pair of 4-pin M12 connectors.

Assignment of each motor control relay to its end function shall be controlled by software.

|  |  |  |  |
| --- | --- | --- | --- |
| Relay Channel | Drive Pin | Fault Pin | ADC Channel |
| 0 | P8.3 | P8.4 | U18, ch 0 |
| 1 | P8.5 | P8.6 | U18, ch 1 |
| 2 | P8.7 | P8.8 | U18, ch 2 |
| 3 | P8.9 | P8.10 | U18, ch 3 |
| 4 | P8.11 | P8.12 | U18, ch 4 |
| 5 | P8.13 | P8.14 | U18, ch 5 |

#### Horn

One channel shall be dedicated to driving the horn relay, and shall be named as such. It shall use a single 2-pin C-GRID SL connection on the board, running to a 2-pin SP-13 connector in the box. The horn drive relay shall be connected to this connector, and it shall switch main battery power through the horn mechanism.

|  |  |  |  |
| --- | --- | --- | --- |
| Relay Channel | Drive Pin | Fault Pin | ADC Channel |
| Horn | P8.17 | P8.18 | U19, ch 0 |

#### Enable/Disarm

There shall be two relays devoted to switching the enable/disarm latching relay – one to enable and one to disarm. The disarm relay allows the Beaglebone software to disarm the boat in response to shore command or conditions other than someone pressing the emergency stop button. The enable relay allows the Beaglebone to arm the boat without someone pressing the external go button. This allows the boat to re-arm itself when conditions are right and there are no humans available, as for example in the deep ocean. The arm function shall be arranged so it can be controlled by making or breaking a solder jumper on the board. See the Enable/Disarm Interface, below, for more information on the behavior of these relays.

|  |  |  |  |
| --- | --- | --- | --- |
| Relay Channel | Drive Pin | Fault Pin | ADC Channel |
| Enable | P8.24 | P8.26 | U18, ch 7 |
| Disarm | P8.15 | P8.16 | U18, ch 6 |

### R/C Receiver

The S-BUS signal from the R/C receiver is serial signal at 100 kbps, 8 data bits, 2 stop bits, and even parity (8e2). It is also inverted relative to the TTL serial standard (i.e. active high rather than the standard active low). The RC receiver also requires power from the 3-pin serial signal connector.

The external connectors shall be a pair of 3-pin CGRID SL connectors. Pin 1 shall be ground, Pin 2 shall be +5V, and Pin 3 shall be signal. A power decoupling cap shall be provided. The signal input line shall be protected against voltage and current excursions with a 1K series resistor and a 5V Zener diode to ground. To invert and shift the level from 5V to 3.3V, the signal shall be run through an open drain inverting buffer with the output pulled up to the 3.3V.

Due to a shortage of serial interfaces, it shall be possible to connect the R/C receiver via solder jumper to either the GNSS or the Fona Module serial interface. This is acceptable because in the case that the Fona is installed, the GNSS is superfluous. Since the R/C interface is one way, only the receive line may be connected.

### GNSS

The GNSS shall be an Adafruit Ultimate GPS Breakout (https://www.adafruit.com/products/746) or pin-compatible equivalent. The main board shall have mounting holes for a compatible header and the two corner mounting holes. The mounting holes on the PCB shall be grounded. The corner mounting holes shall be secured to the board with appropriate conductive standoffs. The Vin connection shall be connected to the main 5V rail. The RX and TX lines shall be connected to UART4. The Fix pin shall be connected to the Beaglebone at P9.15. The Enable line shall be connected to the Beaglebone at P9.12. No protection of level shifting is required.

### Fona Module

The Fona module is an optional ship to shore interface. If installed, it shall be an Adafruit Fona 3G (<https://www.adafruit.com/products/3147>) or equivalent. The main board shall have mounting holes for a compatible header and the two corner mounting holes. The mounting holes on the PCB shall be grounded. The corner mounting holes shall be secured to the board with appropriate conductive standoffs. The serial interface shall be connected to UART5 and VIO to 3.3V. Other connections shall be as convenient. No level shifting shall be required for this module.

Note that this module has been rendered redundant by the 3G cell hotspot described in Ship to Shore Transceiver.

### ADC subsystem

One channel of the Beaglebone’s internal ADC, channel 1, shall be used to sense the Battery Monitor.

All other analog inputs shall be via a pair of ADC128D818 8-channel ADCs on a common I2C bus. They shall be powered from the 5V bus to give a 0-5V input range. Doing so requires level shifting the I2C bus; level shifting shall be provided by a single TXS0102 bi-directional level shifter. The I2C lines shall be pulled up to their respective voltages by 4.7K resistors.

The 3.3V I2C bus shall be connected to I2C-2.

Each input line shall be filtered by an RC filter consisting of a 10K series resistor and a 22 nF parallel capacitor. No protection beyond the filters described above shall be provided for these signals.

Channels 1-6 of U19 shall be brought out to the Power Mon connector. This connector shall be a 2x6 0.1” pitch pin header. It is intended to bring in current and voltage sense signals from the motor and charge current/voltage monitors as well as powering those sensors. Three pins shall be connected to +5V and three shall be connected to ground. The other six shall be designated for motor current, motor voltage, charge current, charge voltage, and auxiliary ADC channels 0 and 1.

### Auxiliary GPIO

Eight I/O pins level shifted to 5V shall be provided on the Auxiliary GPIO connector. This shall be a 2x6 0.1” pitch pin header. Two pins shall be +5V and two shall be grounded. Two of the pins shall carry the 5V I2C bus provided for the ADC subsystem. Three shall carry the SPI1 bus, and may optionally be used as GPIOs. The remaining three shall be available as plain GPIOs.

In the case of the non-I2C pins, they shall be level shifted through a TXB0108 level shifter. This part shall provide all the available I/O protection to these pins.

### RS-485

The RS-485 interface is intended for future peripheral expansions, including a more advanced motor controller architecture. The RS-485 transceiver chip shall be a half-duplex unit connected to UART1 through one of the onboard TXB0108 level shifters. The direction line shall be similarly level shifted and shall be controlled from pin P9.14.

The PCB connector shall be a 3-pin CGRID SL with the following pinout. Pin 1 shall be line B, Pin 2 shall be line A, and Pin 3 shall be ground.

### Steering

The PWM channel for the steering shall be EHRPWM1A on pin P9.16. It shall be level shifted to 5V through one of the onboard TXB0108 level shifters. The 5V output line shall be protected by a 1K series resistor and a 5V Zener diode to ground.

The PCB connector shall be a 3-pin CGRID SL with the following pinout. Pin 1 shall be power (7.4V), Pin 2 shall be the PWM output signal, and Pin 3 shall be ground.

### Lights

The exact nature of the lighting interface is still in flux due to difficulties with the interface code. The connector is a 3-pin CGRID SL with Pin 1 connected to the lighting +5V, Pin 2 connected to P9.22 via one of the TXB0108 level shifters, and Pin 3 connected to ground. The signal pin shall be protected with a 1K series resistor and a 5V Zener to ground.

### Reset

The reset line into the Beaglebone is active low, and is provided with a 10K pullup to keep it inactive except as commanded. The external drive is intended to be a hall effect sensor mounted to the interior wall of the drybox, allowing the user to reset the Beaglebone with a small magnet. Therefore, the sensor is provided with 5V and ground lines. The signal line shall be high when the hall effect switch is active. The signal shall be run through an inverting buffer and then a level shifter to produce the required active low 3.3V signal at the Beaglebone.

The PCB connector shall be a 3-pin CGRID SL with Pin 1 connected to the lighting +5V, Pin 2 connected to the reset line as described above, and Pin 3 connected to ground. The signal pin shall be protected with a 1K series resistor and a 5V Zener to ground.

### Console

The console connector shall bring the receive and transmit lines of the Beaglebone serial console out to a PCB connector. It shall be a 4-pin CGRID SL with Pin 1 NC, Pin 2 connected to RX, Pin 3 connected to TX, and Pin 4 connected to ground. The RX and TX lines shall be protected with a 1K series resistor and a 5V Zener to ground, each.

### Enable/Disarm Interface

The enable/disarm interface provides a connection to the enable/disarm relay in the power distribution box. This interface shall consist of an incoming enable signal, an incoming disarm signal, and outgoing relay drivers for enable and disarm.

Each of the two inputs shall be protected by a 1K series resistor and a 5V Zener diode to ground.

The enable relay driver shall be connected if and only if solder jumper SJ2 is connected.

A high (5V) signal on the enable line shall be required to activate the servo power.

Incoming enable and disarm signals shall be level shifted from 5V to 3.3V through one of the onboard TXB0108 level shifters. The enable input to the Beaglebone shall be on pin P8.20 and the disarm input shall be on pin P8.22.

The PCB connector shall be a 5-pin CGRID SL. Pin 1 shall be connected to the input enable signal, Pin 2 to the input disarm signal, Pin 3 to ground, Pin 4 to the disarm relay output, and Pin 5 to the enable relay output.

### PoE Interface

The PoE interface shall consist of a pair of unshielded RJ45 connectors back to back. The LAN connector is intended to connect directly to the Beaglebone. The Device connector is intended to be connected to the Ubiquiti gear. Pins 1-3 and 6 shall be directly connected between the two connectors. Pins 4 and 5 on the Device connector shall be connected to the +15V PoE rail. Pins 7 and 8 of the Device connector shall be grounded. The corresponding pins on the LAN side shall be unconnected.

## Main Drybox Peripherals

The main drybox has room for several peripherals above and beyond the main board. The ones intended for current or near future installation but not previously discussed are described here.

### AIS Receiver

The AIS receiver shall be a dAISy unit. It shall be connected and powered through the Beaglebone’s USB host port (and optionally, a USB hub). The antenna connection shall be a bulkhead BNC in the outside of the drybox. The antenna shall be mounted as high as practicable.

### WiFi

There shall be two WiFi options – USB dongle and Ubiquiti. One and only one shall be installed.

#### USB Dongle

A standard USB WiFi dongle may be used to provide WiFi connectivity. If used with the AIS, it shall be plugged into an appropriate USB hub.

#### Ubiquiti

A Ubquiti Nanostation may be used to provide WiFi connectivity. If used, it shall be powered through the PoE interface and plugged from there into the Beaglebone.

### Ship to Shore Transceiver

The ship to shore transceiver shall present a WiFi hotspot that the Beaglebone and any additional equipment may connect to.

#### Current

The current ship to shore transceiver shall be a portable 3G cell hot spot. This shall be used for all in-shore work due to its high bandwidth, reliability, and low cost.

#### Future

For off-shore work, the ship to shore transceiver shall be an Iridium GO or equivalent system.

## Power Distribution Box

The power distribution box provides a common distribution and control point for all power systems on the boat. It also maintains the enable/disarm state through a two-coil latching relay.

### Board Power

At all times when the battery is connected, the power distribution board shall draw power from the power input through a 5V linear regulator. This regulator must be able to handle input voltages from 9VDC to 20VDC.

### Enable/Disarm Relay

The enable/disarm relay shall be a DPDT mechanical relay of the two-coil latching type. The drive for each coil shall be a wire-OR of the relay drive from the main board and a local relay driver activated by either the Enable or Stop button Time Delay Circuits. The enable output signal shall be connected such that when the relay’s SET coil is energized, it will be connected to +12V and to ground when the RESET coil is energized. The disarm output signal will be wired with the opposite sense.

The +12V enable and disarm signals shall be regulated down to +5V with a 1K series resistor and a 5V Zener to ground. The PCB connector for the connection to the main board shall be wired identically to the connector on the main board, as described in Enable/Disarm Interface.

### Enable and Stop Buttons

The Enable and Stop buttons shall be momentary SPDT pushbuttons sealed to IP67 or better and mounted on the outside hull. They shall be given large, obvious actuators – red for stop and green for enable. Both switches shall be wired so that their output signal is normally high.

On the PCB, each switch shall be wired to a 3-pin CGRID SL connector. Pin 1 is +5V, Pin 2 is signal, and Pin 3 is ground. The signal pin shall be protected by a 1K series resistor and clamp diodes

### Time Delay Circuits

In order to prevent inappropriate activation of the Enable and Stop buttons, each shall drive its respective relay coil through a time delay circuit. The time delay circuits shall be one-shots based around the two sides of a 556 dual timer chip. The output of each one-shot and its corresponding input signal shall be NOR’d together to produce an outgoing high pulse after the time delay. These high pulses shall trigger the relay drivers to strobe the corresponding relay coil.

The trigger signal for each channel shall be fed forward through an RC low pass network consisting of a 10K resistor and a 1 uF capacitor. This is to ensure that the fed-forward signal remains high until the timer output signal has gone high.

#### Arm Button

The delay elements of the arm button shall be a 470K resistor and a 10 uF capacitor, giving a time delay of approximately 5.2 seconds. The input shall be pulled up by a 10K resistor so that if the switch is disconnected, the Arm function will not activate.

#### Stop Button

The delay elements of the arm button shall be a 470K resistor and a 2.2 uF capacitor, giving a time delay of approximately 1.1 seconds. The input shall be pulled down by a 1M resistor so that if the switch is disconnected, the Stop function will activate.

### Switch Controlled Circuits

The mechanical switches on the face of the power distribution box shall be SPDT, IP 67 or better, and rated for at least 6A and 20VDC.

#### Control Power

This switch shall control power to the main drybox. If it is off, no power shall flow to the main drybox.

#### Horn Power

This switch shall control power to the horn, allowing it to be turned off during software testing and similar activities.

#### Auxiliary Power 1

This switch shall control power to auxiliary power circuit 1.

#### Auxiliary Power 2

This switch shall control power to auxiliary power circuit 2.

### Relay Controlled Circuits

Switches for the relay controlled circuits shall switch the coil of an automotive relay rated for at least 30A.

#### Main Power

This relay shall control power to all other relays and switches. When it is in the off position, the only system that shall have power is the power distribution box’s internal circuitry.

#### Motor

The motor relay switch shall draw power from the Enable coming from the Enable/Disarm relay. Therefore, if the stop button is pushed, all power flowing to the motor must be cut.

## Motor Control Box

The motor control box shall contain seven SPDT automotive relays to switch the four motor wires in order to control motor direction and speed.

### Motor Outputs

The four motor wires shall be designated by color – red, white, yellow, and black. In forward operation, black shall be connected to ground and the other three lines shall be connected to power and each other in different combinations to produce the five forward speeds. In reverse operation, the polarity shall be flipped.

The input power lines shall be +12V (nominal) and ground. A 30A hall effect current and voltage monitor shall be placed in series with the power line and wired back to the main control box.

### Relay Functions

#### Direction

Two relays shall control the direction. Their coils shall be wired in series. Their output sides shall be wired such that when their coils are not energized, +12V (nominal) shall be supplied to the motor box’s positive rail and ground shall be connected to the negative rail. When they are energized, the polarity of the output rails shall be flipped.

#### Red

When energized, this relay shall connect the red output to the motor box’s positive rail. When not energized, it shall have no effect.

#### White

When energized, this relay shall connect the white output to the motor box’s positive rail. When not energized, it shall have no effect.

#### Yellow

When energized, this relay shall connect the yellow output to the motor box’s positive rail. When not energized, it shall have no effect.

#### Yellow-White

When energized, this relay shall connect the yellow output to the white output. When not energized, it shall have no effect.

#### Red-White

When energized, this relay shall connect the red output to the white output. When not energized, it shall have no effect.

### State Map

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Speed | Direction | Red | Yellow | White | Red-White | Yellow-White |
| F5 | Off | On | On | On | On | On |
| F4 | Off | Off | On | Off | On | Off |
| F3 | Off | Off | On | On | Off | Off |
| F2 | Off | Off | Off | On | Off | On |
| F1 | Off | Off | Off | On | Off | Off |
| Stop | Off | Off | Off | Off | Off | Off |
| R1 | On | Off | Off | On | Off | Off |
| R2 | On | Off | Off | On | Off | On |
| R3 | On | Off | On | On | Off | Off |
| R4 | On | Off | On | Off | On | Off |
| R5 | On | On | On | On | On | On |

# Software Architecture

The onboard software shall be written in C/C++ and shall run on the Beaglebone Black.

## Hardware Interfaces

1. The Beaglebone Black provides a great variety of hardware interfaces natively. Therefore, most of the interface circuitry is I/O protection, with a bit of glue logic and some high current drivers. There are also sixteen external ADC channels.

### Hardware Abstraction Layer

1. The hardware abstraction layer (HAL) shall provide a mechanism for abstracting the physical interfaces such that they can be modified at need without disturbing the rest of the code base.
2. In the case of inputs that are asynchronous with the main thread of execution, the HAL shall spawn a thread to handle the incoming data and store it as appropriate for the consumption of the main code.

#### Orientation

1. Orientation data shall come from the IMU at a rate of approximately 100 Hz. The raw data shall be in the form of three axis accelerations, three axis magnetometer readings, and three axis rotation rates. Data shall be read and processed by an OrientationInput object which encapsulates the hardware interface and reader thread. It shall present the data to the rest of the program as a pointer to its internal Orientation object. The Orientation object will, at minimum, have pitch, roll, and heading members.
2. While a full orientation solution would be nice to have, the only truly critical data coming from the IMU is heading. The pitch, roll, and heading shall be around the north, east, and down axes. Heading shall be measured in clockwise degrees from north. Pitch shall be measured in degrees from horizontal, with nose-up pitch positive. Roll shall be measured in degrees from horizontal, with starboard rail down positive. The pitch and roll shall be calculated as follows:
3. These pitch and roll values shall be calculated using the atan2() function to remove angular ambiguity.
4. The heading shall be calculated with the following formula:[[1]](#footnote-1)
5. Future implementations may use a Madgwick and Mahoney algorithm to determine orientation instead.

#### R/C receiver

1. The RCInput object shall be the interface between program logic and the the incoming serial data. It shall spawn a thread that listens to the incoming serial data, parses it, and stores it locally for later retrieval.
2. The data from the R/C receiver is packed into twenty-five (25) byte frames. The first byte of each frame is always 0x0f. The next 22 bytes (second byte through twenty-third byte) contain input data channels 0 - 15. Each channel is 11 bits wide, and they are close packed with no spacing. Channels 16 and 17 occupy the first two bits of the twenty-fourth byte. The fourth bit of the twenty-fourth byte is the failsafe bit – if it is set, then the R/C receiver is no longer receiving data from the transmitter. The last byte is the end byte, and it is always 0x00.
3. The RCInput object shall store all eighteen channels as a vector of uint16\_t ranging between 0 and 2047. It shall be possible for other code to query the value of any desired channel at any time. The values of channels 16 and 17 shall be either 0 or 2047, depending on whether the corresponding bits are cleared or set.
4. The status of the failsafe function shall be stored as a bool and shall be queriable from user code.
5. The RCInput object shall monitors its incoming datastream. If the datastream ceases or becomes corrupted, the RCInput object shall return false in response to the isValid() method. Otherwise, this method shall return true.
6. The RCInput object shall have the following methods to interpret specific channels:

* getThrottle() shall map channel 0 to an int between -5 and 5
* getRudder() shall map channel 3 to a double between -100 and 100
* getMode() shall return FAILSAFE if isFailSafe() returns true. Otherwise, it shall return RUDDER if channel 4 is far below the midpoint, COURSE is it far above, and IDLE if it is near the midpoint. Note that this corresponds to the three-position switch on the controller.
* getCourse() shall map channel 6 to a double between 0 and 360.

#### GNSS & AIS Input

1. Both the GNSS are AIS inputs shall be processed externally by gpsd. This delivered both data streams as a series of JSON messages on a TCP port. The gpsd package provides a command line utility, gpspipe, that connects a given gpsd server instance to the command line. The GPSdInput object shall use gpspipe to receive the data and the pstreams library to manipulate the incoming stream from program code.
2. The GPSdInput object shall have a set of methods to connect, disconnect, set the host and port, and test whether the connection is good. It shall spawn a listener thread to receive GPS and AIS data and expose that data to the rest of the program.
3. As each JSON message is received, the GPSdInput object shall check if it is a GPS or AIS message. It shall then create an object of the appropriate type, and call the parse() method of that object against the incoming message. If it fails, both the new object and the message are discarded.
4. If the incoming message is a GPS message and is successfully parsed, it shall be copied to a private GPSFix object. A pointer to this internal object shall be returned by the getFix() method.
5. If the incoming message is an AIS message, it shall be compared with the internally stored map of AIS contacts. If it has the same MMSI number as an existing contact, the fields of the incoming that are present shall be copied into the existing AIS contact and the last contact time shall be set to the current time.
6. There shall be a method to prune the AIS map for stale contacts and contacts that are too far distant to have an impact on the boat.

#### Rudder (Servo)

1. R/C servos are controlled with a PWM scheme running at between 50 and 100 Hz. The positive pulse is nominally between 1000 and 2000 µsec long but can range from 600 to 2400 µsec. 1500 µsec is the nominal midpoint.
2. The PWM peripheral on the Beaglebone is controlled by writing values to the appropriate values and hooking the peripheral up with the internal pinmux. The appropriate pinmux shall be configured using the config-pin script.
3. The input shall be accepted as either a raw µsec value or as a mapped value from -100 to 100.
4. The Servo object shall handle all interface with the files required to drive the PWM peripheral.

#### Throttle

1. The Throttle object shall drive the motor control relays to translate a given throttle command into the outputs required to get that speed from the motor. See Motor Control Box for more details on the relay functions.

#### Analog Input

1. The ADCInput object shall use a listener thread to read from the battery monitor (internal ADC) and external ADCs at a rate of at least 100 Hz. The resulting data will be available as either raw codes (0-4095) or as scaled values. The channels shall be callable by descriptive names; the relation to physical channels shall be set at object construction from a configuration file.
2. The calling code shall be able to set any number of offsets or gains independently. When a new value is sampled, the scaled value shall be calculated by adding the offset and multiplying by the gain, in that order.

#### GPIO

1. The Pin object shall present an interface to any given GPIO pin. It shall handle the pinmux configuration through the config-pin script and reading/writing the appropriate configuration and value files in /sys/class/gpio/gpio\*\*. The Pin object shall include methods for translating between different representations of the same gpio pin.
2. The Pin object shall be created pointed at a specific pin. The pin shall be defined by the connecter (either 8 or 9) and pin (1-46) on that connector. The object shall contain methods to set the direction (true for output, false for input), writing, and reading the pin. The readPin() method shall return 1 for high, 0 for low, and -1 for any read error. The object shall also contain methods for setting the internal pullup and pulldowns. Changes to pin configuration (direction, pull up/down, etc) shall require a call to the init() method to take effect.

#### Relays

1. The Relay module shall consist of two classes – a Relay class for describing a single relay and a RelayMap singleton for providing access to named relays from program code. The module shall also contain a typedef for a RelayTuple.

##### RelayTuple

1. A RelayTuple shall be of type std::tuple<double, bool, bool> where the elements represent the current in amps, the drive state of the relay (true = on), and the fault state of the relay (faulted = true), respectively.

##### Relay

1. A Relay object shall encapsulate a name (type std::string), a pointer to a drive Pin, and a pointer to a fault Pin. It shall also contain a pointer to an ADCInput object that contains a channel of the same name as the relay to provide current feedback.
2. The object shall contain methods to turn the target relay on and off, check the current, check for fault states, and return the current state of the relay as a tuple.

##### RelayMap

1. The RelayMap singleton shall contain a named map of Relay objects and shall use a hard-coded initializer defined in a system level configuration file. It shall have a function that returns a pointer to the Relay object associated with a given name.

#### Test Harness

1. In order to ease unit and integration testing of the HAL code and code that depends on it, all HAL classes shall friend a test harness class.
2. Each HAL class that requires test harnessing shall be represented by a method in the test harness class. This method shall provide an interface to extract mutable pointers to each private member of the target class required to provide test functionality.

### Ship to Shore Communications

1. From the perspective of the Beaglebone, ship to shore communications shall be over a WiFi link. It must be agnostic to the exact nature of the ship to shore link, so that an Iridium Go may be substituted for a cell hotspot with no code changes.
2. The protocol used shall be MQTT. The default message broker for now shall be Adafruit.io. Connections shall be via SSL if possible.[[2]](#footnote-2)
3. Payloads may be JSON, CSV, or plain values, as noted per topic. All numeric values shall be double precision floats unless otherwise noted. The MQTT wrapper class shall provide methods to iterate through a list of publisher functions.
4. The MQTT object shall subscribe to the listed subscription topics on startup. Incoming commands shall be pushed to the command queue described in BoatState.

#### Topics Published by the Boat

|  |  |  |  |
| --- | --- | --- | --- |
| Topic Name | Format | Data | |
| SpeedLocation | CSV | | <speed, knots>,<lat, degrees>,<lon, degrees>,0 |
| Mode | CSV | | <boat mode>,<nav mode>,<auto mode>,<rc mode> |
| Bearing | Plain | | Current heading, degrees from magnetic north |
| BatteryVoltage | Plain | | battery voltage |
| RudderPostion | Plain | | rudder position, -100 to 100 |
| ThrottlePosition | Plain | | Throttle position, -5 to 5 |
| RudderK | JSON | | {“Kp”:<proportional>,”Ki”:<integral>,”Kd”:<differential>} |
| Course | Plain | | Course made good, degrees true |
| Health[[3]](#footnote-3) | JSON | | Packed JSON of HealthMonitor object |

#### Topics Subscribed to by the Boat

|  |  |  |  |
| --- | --- | --- | --- |
| Topic Name | Format | Data | |
| Command | JSON | | {“Command”:<function name>,”Arguments”:<args>} |

## Software Modules

1. This section covers the program logic, both core and supporting.
2. The main program shall run in timed frames at a rate of 50 ms per frame (20 Hz)
3. At the start of each frame, the code shall read all sensors.
4. The code shall then execute the current mode of the state machine.
5. Lastly, the code shall write to any outputs not written by the mode logic.

### Data Types

#### HackerboatState

1. All the custom datatypes shall be subclasses of the HackerboatState abstract base class. It shall define the following virtual methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Arguments | Returns | Description |
| parse | json\_t\* | bool; true on success | Populate this object from the given JSON |
| pack | void | json\_t\* | Serialize this object into JSON |
| isValid | void | bool; true if object is valid. | Determine whether the current object is in a valid state. |

1. The class shall define a public member recordTime, which shall be a timepoint as defined in the <chrono> library.
2. The class shall define two static utility methods, parseTime() and packTime(). These act as wrappers for the date library and provide a uniform conversion between timepoints and string representations of timestamps.

#### HackerboatStateStorable

1. The HackerboatStorable class shall bn abstracte a subclass of the Hackerboat class that provides a common interface for those datatypes that require database storage and retrieval.
2. The HackerboatStateStorable class shall define a new type, sequence, as an alias of int64\_t. This shall be used to store a monotonic sequence number in a protected data member. The sequence number shall increment each time this object is written to the database. There shall be a public method that returns the sequence number.
3. The class shall contain the following non-virtual but overloadable public methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Arguments | Returns | Description |
| countRecords | void | sequence | Returns number of rows in the corresponding database table. |
| writeRecord | void | bool (true if successful) | Write the contents of the current object to the corresponding database row, if it exists. |
| getRecord | sequence | bool (true if successful) | Populate the current object with the contents of the database row specified. |
| getLastRecord | void | bool (true if successful) | Populate the current object with the contents of the most recent row. |
| appendRecord | void | bool (true if successful) | Append the contents of the current object as the last row in the database. |

1. Each subclass must implement a storage() method that returns a handle to the database table corresponding to that subclass in the form of a reference to a HackerboatStateStorage object.
2. Each subclass may implement fillRow() and readFromRow() to write and read rows from the database table associated with this object. If either is implemented, both must be implemented. They shall default to writing and reading the sequence number and a text string containing the JSON representation of the object produced by the pack() method.

#### HackerboatStateStorage

1. A HackerboatStateStorage object shall encapsulate a reference to a table in a SQLite3 database file. It shall further contain prepared statements and methods for implementing all the methods described in HackerboatStateStorable .

#### Enumerations

1. The code shall use enumerations to represent values that can only take a limited number of named values.
2. All enums shall be of the type enum class.

##### BoatModeEnum

1. This enumeration shall contain the names of all the top-level modes of the boat. See Control Modes for more detail.

##### NavModeEnum

1. This enumeration shall contain the names of all the navigation modes of the boat. See Control Modes for more detail.

##### AutoModeEnum

1. This enumeration shall contain the names of all the autonomous modes of the boat. See Control Modes for more detail.

##### RCModeEnum

1. This enumeration shall contain the names of all the R/C modes of the boat. See Control Modes for more detail.

##### WaypointActionEnum

1. This enumeration shall contain the action that the Waypoint Navigation module takes when it reaches the end of the list of waypoints.

|  |  |  |
| --- | --- | --- |
| Action Name | Enum Name | Description |
| Idle | IDLE | Boat stops and does nothing – Navigation mode goes to Idle. |
| Anchor | ANCHOR | Hold position at the last waypoint. |
| Return | RETURN | Return to the origin point and hold position once there. |
| Repeat | REPEAT | Repeat the list of waypoints, starting with the first. |
| None | NONE | This is effectively the default uninitialized state. Functionally equivalent to Idle. |

#### EnumTable

1. To produce human-readable JSON, each enum shall be paired with a vector of strings corresponding to the names of the enum elements.
2. EnumTable shall have a public method which determines whether a given string names an element of the enum.
3. EnumTable shall have public methods allowing the user to get the string given the enum value or the enum value given the string.
4. The string values shall be defined in BoatState.

#### Timestamps

1. Timestamps shall conform to ISO8601 and shall have a precision of milliseconds or better. This shall be accomplished using Howard Hinnant’s date library.

#### BoatState

1. BoatState is intended to be a full log of the status of the boat, and therefore shall store the value of, or pointers to, information about every major subsystem. The intent is that a pointer to it can be passed to each mode object in the State Machine and provide the mode execute() methods with all the hooks required to control the boat.
2. BoatState shall be a subclass of HackerboatStateStorable.
3. BoatState shall maintain a colon separated list of current faults with boat systems. It shall provide methods to add faults, remove faults, clear the fault string, and count the number of entries it contains.
4. BoatState shall maintain a reference to the current state of each layer of the state machine, and shall have public accessors for this data.
5. BoatState shall have a queue for storing incoming commands from the ship to shore link. It shall be able to execute any number of queue elements from a single method call. It shall also have methods for clearing the queue and adding new members.
6. BoatState shall have a method for serializing its data as a CSV string.
7. BoatState shall have a member of type Waypoint that stores the list of waypoints as described in Waypoint Navigation. It shall also store the number of the current waypoint and the action to take when the last waypoint in the list is reached.
8. BoatState shall store the timepoints at which it last received data from the ship to shore link and from the R/C receiver.
9. BoatState shall maintain the location from which it launched.
10. BoatState shall maintain a pointer to the Health Monitoring subsystem.
11. BoatState shall maintain a copy of the last GPS fix.
12. BoatState shall maintain pointers to the Servo, Throttle, RCInput, ADCInput, GPSdInput, OrientationInput, and RelayMap subsystems.
13. BoatState shall maintain Pin objects for the Enable and Disarm inputs.
14. BoatState shall maintain a Pin object for the servo enable pin.

#### Command

1. The Command class is intended as a storage container for commands received on the command topic from the shore side MQTT broker.
2. A Command object shall be built with a pointer to the current BoatState (to allow it to act), the name of the command, and a json\_t\* pointer to the arguments.
3. The Command class shall have an execute() method that calls the current command. It shall take an argument indicating the number of commands from the queue to execute. It shall return the number of commands successfully executed (i.e. returning true).
4. All commands shall be implemented as private static methods of the Command class.

##### SetMode

1. This function shall request the desired boat level mode for the next iteration of the state machine. It is up to the current mode implementation whether this causes the mode to transition or not.
2. The argument JSON object shall consist of a single key named “mode”. The value of mode shall be the string name of the desired mode.

##### SetNavMode

1. This function shall request the desired navigation level mode for the next iteration of the state machine. It is up to the current mode implementation whether this causes the mode to transition or not.
2. The argument JSON object shall consist of a single member key “mode”. The value of mode shall be the string name of the desired mode.

##### SetAutoMode

1. This function shall request the desired autonomous level mode for the next iteration of the state machine. It is up to the current mode implementation whether this causes the mode to transition or not.
2. The argument JSON object shall consist of a single key named “mode”. The value of mode shall be the string name of the desired mode.

##### SetHome

1. If this function is called with no arguments, it sets the home point to the last GPS fix received.
2. This function may be called with the argument “location”. If this key contains the JSON representation of a valid Location object, the home point shall be set to the given location.

##### ReverseShell

1. This function shall open a reverse shell connection to allow fine grained control, configuration, and modification of the Beaglebone Black over the ship to shore link.
2. This function has not yet been implemented. Options include SSH over Pagekite and MOSH.

##### SetWaypoint

1. This function shall set the target waypoint number.
2. The argument shall be a single key named “number” with the value of the desired waypoint.
3. It shall check to make sure the desired waypoint exists. If it is greater than the number of waypoints, it shall set the target waypoint to the last waypoint. If it is less than zero, it shall set the current waypoint to the first waypoint.

##### SetWaypointAction

1. This function shall set the action that the boat takes on reaching the last waypoint in the list.
2. The argument JSON object shall contain a single key, “action”. Its value shall be a string representing one of the possible values of WaypointActionEnum.
3. If the argument does not contain the “action” key or the value of the key is not a valid value of WaypointActionEnum, this function does nothing and returns false.

##### DumpPathKML

1. This function shall dump a KML file containing a LineString object describing the course of the boat to date to the given web address.
2. This function is not yet implemented.

##### DumpWaypointKML

1. This function shall dump a KML file containing the waypoint list to the given web address.
2. This function is not yet implemented.

##### DumpObstacleKML

1. This function shall dump a KML file containing all obstacles known to the boat at the current time to the given web address.
2. This function is not yet implemented.

##### DumpAIS

1. This function shall dump a JSON representation of the current AIS database to the given web address.
2. This function is not yet implemented.

##### FetchWaypoints

1. This function shall fetch a new waypoint KML file from either a hard-coded default location or the given web address.
2. This function is not yet implemented. This function imposes substantial security risks and must be carefully considered before implementation.

##### PushPath

1. This is a duplicate of DumpPathKML and should be deleted.

##### SetPID

1. This function allows remote setting the rudder PID proportional, integral, and differential gains.
2. The argument JSON object may contain any combination of the keys “Kp”, “Ki”, and “Kd”, representing the proportional, integral, and differential gains, respectively.
3. The value of these keys shall be interpreted as double precision floating point numbers and written to the appropriate location in the BoatState object passed to this function.

#### HealthMonitor

1. The HealthMonitor class is intended to bring all vehicle health monitoring data into a single object, easing data acquisition, scanning, and logging.
2. HealthMonitor shall be a subclass of HackerboatStateStorable.
3. The object shall maintain a pointer to an ADCInput object used to read the incoming sensors.
4. It shall implement a single function readHealth() that populates the object with the system health data from around the boat.
5. The object shall gather and store the following vehicle health data:
   1. Servo current
   2. Battery Voltage (measured at control board)
   3. Main Battery Voltage
   4. Main Battery Current
   5. Charge Voltage
   6. Charge Current
   7. Motor Voltage
   8. Motor Current
   9. R/C RSSI value
   10. Cellular/Satellite RSSI value
   11. WiFi RSSI value

#### AIS

1. AIS data shall be received from gpsd as described in GNSS & AIS Input.
2. AIS data shall be in the format described in <http://catb.org/gpsd/gpsd_json.html> and <http://catb.org/gpsd/AIVDM.html>

##### AIS Enums

1. This module shall contain enumerations to describe the Message Type, Navigation Status, Ship Type, and EPFD Type fields described in <http://catb.org/gpsd/AIVDM.html>.

##### AISBase

1. The AIS module shall contain an abstract base class, AISBase. All AIS contact types shall be subclasses of this base class.
2. AISShip shall be a subclass of HackerboatStateStorable.
3. AISBase shall contain a virtual prune() method that, upon calling determine whether the current AIS object ought to be removed from the database.
4. AISBase shall have an MMSI member to store the contact’s identity.
5. AISBase shall contain a member storing the last time at which an AIS message was received from this contact.
6. AISBase shall contain a member storing the last reported location of the contact.

##### AISShip

1. AISShip shall contain a method for parsing the JSON packet delivered by gpsd.
2. AISShip shall be a subclass of AISBase.
3. AISShip shall contain a project() method that estimates the current location of the contact either now or at some timepoint in the future.
4. AISShip shall contain a merge() method that allows two AISShip objects with the same MMSI to be merged. The most recent data will be used when both contain valid values for a member.
5. AISShip shall contain the following public members:

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| status | AISNavStatus | The navigation status of the vessel |
| shiptype | AISShipType | The type of vessel |
| epfd | AISEPFDType | The type of EPFD mounted by the vessel |
| turn | double | Rate of turn of the vessel in degrees per min |
| speed | double | Speed of vessel in knots |
| course | double | True course of vessel in degrees from true north. |
| heading | double | Magnetic heading of vessel in degrees from magnetic north. |
| callsign | string | Callsign of vessel |
| shipname | string | Name of vessel |
| imo | int | IMO number of vessel |
| to\_bow | int | Distance from the GNSS antenna to the bow in meters |
| to\_stern | int | Distance from the GNSS antenna to the stern in meters |
| to\_port | int | Distance from the GNSS antenna to the port rail in meters |
| to\_starboard | int | Distance from the GNSS antenna to the starboard rail in meters |

1. The prune() function shall recommend removal of this contact when any of the following conditions are met:
   1. No signal from the contact has been received in AIS\_MAX\_TIME (defined in config.h)
   2. The contact is farther away than AIS\_MAX\_DISTANCE (defined in config.h)
2. When called, the prune() function shall remove the corresponding row from the database.

#### GPS

1. The GPSFix class shall store the details of each incoming GPS fix from gpsd.
2. GPSFix shall be a subclass of HackerboatStateStorable.
3. GPSFix shall implement a method for parsing the JSON received from gpsd as described in GNSS & AIS Input.
4. The header file containing GPSFix shall contain an enum class called NMEAModeEnum that encodes the fix mode provided with each GPS report from gpsd. Possible values are NONE, NOFIX, FIX2D, and FIX3D.
5. GPSFix shall maintain the following public data members:

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| fix | Location | The location of the current fix |
| gpsTime | sysclock | The GPS time of the current fix |
| mode | NMEAModeEnum | The mode of the current fix |
| device | string | Name of the source device (may be excluded if desired) |
| track | double | Course over ground, in degrees from true north |
| speed | double | Speed over ground, in m/s |
| alt | double | Altitude above MSL, in meters |
| climb | double | Rate of climb in m/s. |
| epx | double | Longitude error in meters, 95% confidence |
| epy | double | Latitude error in meters, 95% confidence |
| epd | double | Course error in degrees, 95% confidence |
| eps | double | Speed error in m/s, 95% confidence |
| ept | double | Timestamp error in second, 95% confidence |
| epv | double | Altitude error in meters, 95% confidence |
| epc | double | Climb error in m/s, 95% confidence |
| fixValid | bool | True if this is a valid fix |

#### Orientation

1. The Orientation class shall store the current pitch, heading, and roll of the boat.
2. The Orientation class shall be a subclass of HackerboatState.
3. The Orientation class shall be aware of the difference between true and magnetic headings, and shall have methods to convert between the two.
4. It shall use GeographicLib to calculate magnetic declination. Declination shall be recalculated only on request, because it is time consuming to do so.
5. It shall store pitch, roll, and heading as public data members of type double.
6. Pitch, roll, and heading shall default to NAN.

#### Location

1. The Location class shall provide a mechanism for storing and working with geographic locations.
2. The Location class shall be a subclass of HackerboatState.
3. The Location class shall store lat and lon as public data members of type double.
4. The header containing the Location class shall define CourseTypeEnum, which shall have the possible values of GreatCircle and RhumbLine.
5. All Location methods that calculate geographic relationships shall take a CourseTypeEnum as an argument, with the default value of GreatCircle.
6. All such methods will use GeographicLib to calculate courses and distances.
7. The Location class shall define a method bearing() which calculates the true bearing from the calling Location object to another given Location object.
8. The Location class shall define a method distance() which calculates the distance from the calling Location object to another given Location object.
9. The Location class shall define a method target() which returns a TwoVector with units of meters that transforms the calling Location to the given Location object.
10. The Location class shall define a method project() which adds a given TwoVector to the calling Location and returns a new Location object.

#### Utility Types

##### TwoVector

1. The TwoVector class shall store a two-dimensional vector.
2. All TwoVector methods shall be declared inline.
3. The TwoVector class shall implement access methods for both Cartesian and polar representations.
4. The TwoVector class shall provide rotation methods for both degree and radian input.
5. The TwoVector class shall provide scalar multiplication and division operators.
6. The TwoVector class shall provide vector addition and subtraction operators.
7. The TwoVector class shall provide the dot product as a multiplication operator.
8. The TwoVector class shall provide a unit() method which returns the unit vector.
9. The TwoVector class shall provide rad2deg() and deg2rad() static methods for angle conversions.

### Logs

The current logging code is clunky to use and doesn’t provide much in the way of protection against race conditions and the like. It is documented here for historical interest and to facilitate writing code until it is replaced.

#### Current

1. The logging class, LogError, shall be a singleton.
2. It shall implement an open() method which opens a given log file for writing.
3. The write() method shall take two arguments, a source and a message. The source argument shall name the origin of the error message and the message argument shall contain the text.
4. The write() method shall write a millisecond resolution timestamp in square brackets, followed by the source, a colon, and the message.
5. The close() method shall close the log file. Since this is a singleton and there is only one ofstream, this must be called immediately to prevent race conditions.

#### Planned

1. Logging shall be handled the EasyLogging++ library (<https://github.com/easylogging/easyloggingpp>)
2. Time format shall be [%Y-%M-%d-%H:%m:%s.%g], for example: [2016-11-03-20:12:22.136]
3. Each module may have its own custom logger in addition to the main logging facility.

### State Machine

1. The core of the software is a multi-layer state machine. Each layer has a base class that instantiates the StateMachineBase template. Each mode[[4]](#footnote-4) in that layer is a subclass of the layer’s base class. The template requires each mode to implement an execute() method. This method contains all logic for the given mode.
2. The execute() method returns a pointer to an object that is a subclass of its layer’s base class. To execute a step of the state machine, the code shall call the execute() method of the mode object that the current state machine pointer points at. It shall then replace the current state machine pointer with the new pointer returned from the call to the current mode’s execute() method.
3. The Navigation mode shall have an RC Command mode pointer and an Autonomous mode pointer as members.
4. See Control Modes for a full description of all available modes.

### Waypoint

1. The Waypoint class loads waypoints in from an external KML file, keeps track of the current waypoint, and provides that waypoint as a Location object on request.

### Obstacle Avoidance

#### Vector Summation

#### A\*

### PID Control

### Database Storage

## Control Modes

### Startup/Self-Test

### Disarmed

### Low Battery

### Fault

### Navigation

#### Idle

#### Fault

#### RC Command

##### Course

##### Rudder

##### Fail Safe

##### Idle

#### Autonomous

##### Waypoints

##### Return to Launch

##### Anchor

# External Utilities

## User Interface

## Sailing Simulator

### Phase 1

### Phase 2

1. Sourced from http://www.cypress.com/file/130456/download [↑](#footnote-ref-1)
2. Note that the SSL code has not been implemented yet, because I have not figured out how to set it up correctly for the Paho client. [↑](#footnote-ref-2)
3. This has not yet been implemented due to concerns about Adafruit.io’s maximum element length. [↑](#footnote-ref-3)
4. Note that we use modes instead of states here in order to avoid excessive overloading of the term state. [↑](#footnote-ref-4)