Systems Architecture

# Hardware Architecture

The Hackerboat control system consists of three major components -- a shore station, a Beaglebone, and an Arduino Mega. The Beaglebone and the Arduino are located in a control box on the boat. They communicate with the shore station via a two-legged network. The Beaglebone is attached to a common carrier board with the Arduino. It communicates with the Arduino with two serial channels. It is connected to the network, housed in the radio box, via a WiFi link. The control box also contains an IMU (read by the Arduino) and a GPS (read by the Beaglebone).

The radio box contains two radios, the WiFi router and the 900 MHz router. Both are running OpenWRT. The 900 MHz router connects to the shore station, which is another, similar 900 MHz router.

See <https://github.com/JeremyRuhland/hackerboat/blob/restful_refactor/doc/systems/Hackerboat-Systems-Diagram.graphml> for a complete systems diagram.

# Software Architecture

## Responsibilities

The [Beaglebone](#h.6bveacn8xwaj) shall be the dominant partner between the two computers. The Arduino shall obey commands coming from it implicitly. It shall do all the high level course planning etc when the boat is autonomous mode. When the boat is in manual mode, it shall surrender high-level control to the shore station and shall take over if and only if the shore station stops transmitting commands. The Beaglebone shall receive position data from the GPS and shall poll ship status data from the Arduino.

The [Arduino](#h.tm15x7868x7p) shall do all of the low level processing. It shall take compass data from the IMU and shall receive a target heading from either the [Beaglebone](#h.6bveacn8xwaj) or the shore station. It also shall record and report data from the various ship status sensors (voltages, currents, and so on) and the two buttons (stop and go). The Arduino shall control the throttle, the steering servo, the status lights, and the horn.

The shore station shall provide a method for the operator to arm the boat, provide steering and throttle commands, and to plot a course for the boat to follow in autonomous mode. It also shall provide the interface for the user to acquire data from the boat.

## Communications Channels

All primary communications between the three elements of the control system (shore, Beaglebone, and Arduino) shall be via REST requests from higher levels to lower ones. I.e. the shore station may send requests to the Beaglebone and the Arduino. The Beaglebone may send requests to the Arduino. Requests may not be sent in the other direction; only responses.

The communication links shall be as follows:

* TCP/IP connection between the shore station and the Beaglebone, as described in the Hardware Architecture section (above).
* The Control Serial link between the Beaglebone and the Arduino. This link shall be Serial1 on the Arduino side and UART2 on the Beaglebone side.
* The Log Serial link between the Arduino and the Beaglebone. This link shall be Serial on the Arduino side and UART1 on the Beaglebone side.

Requests from the shore station to the Beaglebone shall be sent over the TCP/IP link with http.[[1]](#footnote-1) The Beaglebone shall run a webserver to receive and dispatch these requests either to its own operating software or to the Arduino. Requests bound for the Arduino shall be dispatched to it via the Control Serial link, and responses are expected on the same serial link. Once a response is received by the Beaglebone from the Arduino, it shall pass that response along to the shore station.

The Beaglebone may send its own REST requests over the Control Serial link, and shall handle them as otherwise described in this document.

The Beaglebone shall also record all data transmitted by the Arduino on the Log Serial link to non-volatile storage for later retrieval and analysis. This may be done by the simple expedient of opening the serial port and piping all output to a disk file.

## Arduino Software

The Arduino software shall take the inputs described in this section, process them according to the rules laid out here, and produce the outputs described in the appropriate section. It shall be written in the standard Arduino language, and shall use only those libraries included as part of the Arduino core (i.e. by the statement #include <Arduino.h>) or listed below.

### Hardware

The Arduino hardware shall be an Arduino Mega 2560 or equivalent.

### Inputs

The Arduino shall read the follow inputs:

* Control Serial, on the Serial1 device
* IMU, on the I2C bus
* Start button, on pin 37
* Stop button, on pin 36
* Power input voltage, on pin A0
* Motor current, on pin A13
* Motor voltage, on pin A15

### Outputs

The Arduino shall write to the following outputs:

* Log Serial, on the Serial device
* Control Serial, on the Serial1 device
* Arduino status lights, on pin 39
* Beaglebone status lights, on pin 38
* Throttle relay drivers, as follows:
  + Direction relay, on pin 52
  + White speed relay, on pin 51
  + Yellow speed relay, on pin 48
  + Red speed relay, on pin 47
  + Red/White crossover relay, on pin 44
  + Red/Yellow crossover relay, on pin 43
* Horn relay driver, on pin 40
* Steering servo, on pin 3
* Steering servo power enable, on pin 2

### Execution Order

The execution of the code shall be divided into frames. Frames may be executed on a clock or may be free-running, as long as all timing constraints are observed.

Each frame shall consist of the following steps:

* Read all local inputs
* Check for incoming requests on Control Serial
* Process any incoming requests
* Perform operations required by the current state
* Select the state for the next frame
* Write to all outputs, including the Control and Log Serial channels.
* On each frame, the Arduino shall output its current state as a JSON string with no end of line terminations on its Control Serial channel.

### Non Volatile Memory

On the output phase of each frame in states other than Self-Test, Fault, and Power-Up, the Arduino code shall record the current state and the last state in three suitable locations in NVM.

### REST Handling

The Arduino shall use aREST to process data coming over the Control Serial and to prepare responses for that interface.

All REST requests directed at the Arduino shall take the form /a/<request>. The leading /a shall be stripped by the Beaglebone before passing the request to the Arduino. All requests to the Arduino shall be performed via GET.

The aREST library has four types of request -- digital pin (both read and write), analog pin (both read and write), variable read, and function. The Arduino shall not respond to digital or analog pin requests except in ArmedTest mode. All variable writes shall occur via function call and shall be buffered such that all variable writes are checked for validity prior to changing the state of the boat in any way.

The variables and functions exposed via REST are enumerated in the [Arduino REST](http://drive.google.com/open?id=1RWsXm9O9lPdw0rC2VZZq4Y6lQdgeENAV9jq4zoIyAd4) file.

### State Machine

See the [state table](http://drive.google.com/open?id=1Mn0NGXtgshYRYwtLzPwd0pfDcAN9FWZ69MgP0L4LUjE) for a complete description of each state and the transitions. The Arduino state machine shall have three classes of states:

* Startup -- The Arduino has just started and it is currently in self-test mode.
* Safe -- No single state transition and/or command may result in activation of the propeller.
* Not Safe -- The propeller is either turning or may be started with a single command or change of state. Prior to transitioning from a startup or safe state to an unsafe one, the boat will sound the horn for a minimum of two seconds.

The startup states are as follows:

* Power-Up: The Arduino shall start in this phase. It shall immediately transition to the next phase, Self-Test.
* Self-Test: The Arduino shall check its inputs for proper function. If a fault is detected, the next state will be Fault. If they are working correctly, it will check the NVM for the last state. If the state is Armed, Active, or Self-Recovery, it will transition to those states. Otherwise, it will transition to Disarmed.

The safe states are as follows:

* Disarmed: The Arduino is quiescent and waiting for command. The steering servo is powered off and the motor drive relays are also powered off. The default heading target is set equal to the current heading on each frame.
* Fault: The Arduino has entered a fault state. It transmits a fault string in response to an appropriate query.

The unsafe states are as follows:

* Armed: The Arduino is ready to transition to an active state, but is not yet active.
* Active: The Arduino is active and shall attempt to steer to the given heading. The steering PID loop shall be active only when the throttle is on.
* ActiveRudder: The Arduino is active and the rudder position shall be set by external command rather than the PID loop.
* SelfRecovery: The Arduino is active but has lost all contact with the Beaglebone and the shore. It shall steer a preset recovery course (notionally, to the nearest land) until contact is re-established.
* LowBattery: The boat’s battery is low and it is waiting for the solar panels to charge it up again. This state is implemented even without the solar panels present, because it is better in a low battery situation for the boat to wait for rescue and continue to transmit its position rather than discharge the batteries completely and lose communications as well.
* ArmedTest: This mode allows a full systems checkout prior to launch using the regular operational software. It may only be entered from SelfTest if the Beaglebone is also in ArmedTest. When entering this state, the Arduino shall sound the horn for a minimum of two seconds prior to accepting commands.

### Required Built-in Libraries

* Wire.h
* Servo.h

### Required External Libraries

* PID\_v1.h
* Adafruit\_Sensor.h
* Adafruit\_LSM303\_U.h
* Adafruit\_L3GD20\_U.h
* Adafruit\_9DOF.h
* Adafruit\_NeoPixel.h
* aREST.h (as modified for Hackerboat)

## Beaglebone Software

### Hardware

The Beaglebone shall be a Beaglebone White version A3 or later. It may be replaced by a Beaglebone Black at a later date.

### Software Components

The Beaglebone shall run Debian Linux 8.2 or later. All interprocess communication shall be accomplished with one or more sqlite3 database files. The Beaglebone shall have the following software components running:

* A lightweight web server capable of handling incoming REST requests with SSL and authentication turned on. Lighhtpd shall be the webserver software.[[2]](#footnote-2) The REST handling component shall be implemented as a CGI or FastCGI program.
* A logging component that records the debug output of the Arduino on Serial. This may be as simple as cat /dev/tty01 > arduino.log.
* A state machine component that maintains the state of the Beaglebone.
* A navigation process that calculates the course to the next waypoint and sums in all other navigation influences (terrain, collision avoidance, AIS, etc).
* An AIS receiver process
* A terrain avoidance process based on downloaded map data
* A GPS parsing process
* An obstacle detection process

### Interfaces & Sensors

#### WiFi Interface

The Beaglebone shall be connected to the radio network via a USB WiFi dongle inserted into the USB host port. This interface shall be read only by the webserver components.

#### Arduino Interfaces

The Beaglebone shall communicates with the Arduino over two serial links and shall have one GPIO link that allows the Beaglebone to reset the Arduino if necessary.

The Beaglebone shall receive the Log Serial output from the Arduino on UART1.

The Beaglebone shall transmit REST requests to and receives data from the Arduino’s Control Serial interface on UART2.

#### GPS

The Beaglebone is connected to the GPS via a serial link transmitting NMEA sentences. The GPS uses UART4.

### Non Volatile Memory

The state machine component shall maintain its last state continuously in NVM in order to permit return to that state after a power interruption. It shall also maintain the last launch site.

### Web Server Configuration

In order to keep memory, processing, and power requirements down, the Beaglebone shall run a lightweight httpd suitable for embedded use, such as lighttpd. The web server shall have the following properties:

* It shall be capable of handling REST requests as details below in [REST Handling](#h.matrzxshjbh6).
* It shall be configured to use SSL.
* It shall be capable of being configured with password authentication.

### REST Handling

Any incoming GET REST request whose first term is /a/ shall:

* Have the leading /a stripped
* Be transmitted to the Arduino via the Control Serial link
* Re-transmit the response received over the Control Serial link to the requester.

All other requests shall be processed locally by the Beaglebone.

Beaglebone REST requests, unlike the Arduino, are governed by the hierarchical system described in the [Beaglebone REST](http://drive.google.com/open?id=1or1shiEhdwQCfQ2rXg1q7574qdFHAn6TxFtjcdgavA8) table.

### State Machine

The state machine shall be handled by a single process running continuously. It shall accept input from all of the inputs detailed in [Interfaces & Sensors](#h.j7yw3q6edu7e) with the exception of the WiFi interface. See the [state table](http://drive.google.com/open?id=1gbiKb4Y342_lMnKd3KMx3NkdM5Uo42mgYAabHWJFXdk) for a more complete description of each state and its transition.

The Beaglebone shall use the boneStateSet REST request to inform the Arduino of its state at least once per second and on every state transition.

Like the Arduino, the Beaglebone states shall be divided into StartUp, Safe, and Unsafe, paralleling the similarly named categories for the Arduino.

The StartUp states shall be as follows:

* Start: The Beaglebone shall start in this state. It will immediately transition to SelfTest.
* SelfTest: The Beaglebone shall check its inputs for proper function. If a fault is detected, the next state will be Fault. If they are working correctly, it will check the NVM for the last state. If the state is WaypointNavigation, LossOfSignal, or ReturnToLaunch, it will transition to those states. Otherwise, it will transition to Disarmed.

The Safe states shall be as follows:

* Disarmed: The Beaglebone is quiescent and awaiting command. It shall continuously updates the launch site while in this state.
* Fault: The Arduino has entered a fault state. It transmits a fault string in response to an appropriate query.

The Unsafe states shall be as follows:

* Armed: The Beaglebone is ready to transition to an active state, but is not yet active.
* Manual: In this state, the navigation of the boat shall be under remote manual control. If the Beaglebone does not receive a request from the shore station for five seconds in this state, it shall go to LossOfSignal.
* WaypointNavigation: The Beaglebone shall navigate per its internal list of waypoints as described in the Navigation section.
* LossOfSignal: The Beaglebone has lost contact with the shore station. If it reaches this state from WaypointNavigation, it shall proceed with navigation as previously commanded. If it reaches this state from Manual, it will proceed directly to ReturnToLaunch.
* ReturnToLaunch: The Beaglebone shall attempt to return to the location it was launched from.
* ArmedTest: The Beaglebone shall accept all commands that would be valid in any other Unsafe state.

### Navigation

Navigation shall only be computed in modes WaypointNavigation and ReturnToLaunch. In all other modes, the boat is either not powered or is under manual command from the shore station.

#### General

Under all navigation modes, the Beaglebone will prepare a flow field based on its current position and the position of its current target. This flow field may then be summed with flow fields from other sources (for example, shore or obstacle avoidance) to produce a commanded course and speed.[[3]](#footnote-3) In the case that the flow field produces a zero throttle command, a random dither shall be added in order to prevent deadlock.

The strength of the flow field shall be the distance to the waypoint multiplied by waypointStrength. It shall be less then or equal to waypointStrengthMax, and it shall be sufficient to produce forward motion at all points.

#### Waypoint Structure

Each waypoint shall have three fields:

* Target latitude
* Target longitude
* Stop: This shall be an enum with the following possible values:
  + End: If this is the last waypoint, the Beaglebone shall transition to Armed.
  + Return: If this is the last waypoint, the Beaglebone shall transition to ReturnToLaunch
  + Repeat: If this is the last waypoint, the Beaglebone shall navigate to the first waypoint in the list and start over.

#### WaypointNavigation

The Beaglebone shall maintain an ordered list of waypoints. When in WaypointNavigation mode, it shall navigate to each waypoint in list order. When it reaches the last waypoint in the list, it shall obey the the Stop condition described above.

#### ReturnToLaunch

The Beaglebone shall navigate to the stored launch location as if it was an ordinary waypoint.

### Logging

The Beaglebone shall record the following log files:

* A log of all logging output from the Arduino
* A log of all REST requests and responses, with IP addresses and all identifying information
* A navigation log that records all navigation inputs and outputs

1. This should be https in the future, but let’s not complicate things quite yet. [↑](#footnote-ref-1)
2. This is subject to change if there’s a good reason. Otherwise, lighttpd seems to be a common and well-supported choice. [↑](#footnote-ref-2)
3. This approach is described at length in Chapter 3 of Arkin’s “Behavior Based Robotics”. It is here to allow future additions, such as obstacle detection and autonomous use of maps. [↑](#footnote-ref-3)