LPARD-TDF-2012

Lafayette Programmable Autonomous River Droid

Tracking and Data Fusion

ECE492-Spring 2012
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executive Summary</td>
<td>1</td>
</tr>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>System Overview</td>
<td>4</td>
</tr>
<tr>
<td>Shore Sensors</td>
<td>7</td>
</tr>
<tr>
<td>Boat Sensors</td>
<td>10</td>
</tr>
<tr>
<td>Software</td>
<td>13</td>
</tr>
<tr>
<td>Boat System Requirement Document</td>
<td>16</td>
</tr>
<tr>
<td>Boat Payload Analysis</td>
<td>20</td>
</tr>
<tr>
<td>Boat Position Tracking Independence</td>
<td>21</td>
</tr>
<tr>
<td>Heading Analysis</td>
<td>22</td>
</tr>
<tr>
<td>Manual Control Analysis</td>
<td>25</td>
</tr>
<tr>
<td>EMI Analysis</td>
<td>31</td>
</tr>
<tr>
<td>Environmental Analysis</td>
<td>32</td>
</tr>
<tr>
<td>Hazmat Analysis</td>
<td>38</td>
</tr>
<tr>
<td>Lifetime Capacity Analysis</td>
<td>39</td>
</tr>
<tr>
<td>Power Analysis</td>
<td>41</td>
</tr>
<tr>
<td>Manufacturability Analysis</td>
<td>50</td>
</tr>
<tr>
<td>API Documentation</td>
<td>57</td>
</tr>
<tr>
<td>Coding Verification Documentation</td>
<td>58</td>
</tr>
<tr>
<td>Data Storage Format Analysis</td>
<td>59</td>
</tr>
<tr>
<td>Current and Future Budget Report</td>
<td>60</td>
</tr>
<tr>
<td>Users Manual</td>
<td>69</td>
</tr>
<tr>
<td>Appendices</td>
<td>83</td>
</tr>
<tr>
<td>Omnidirectional Proposal for Ultrasound</td>
<td>84</td>
</tr>
<tr>
<td>Software Reference Manual</td>
<td>86</td>
</tr>
<tr>
<td>QA Tests</td>
<td>189</td>
</tr>
<tr>
<td>Circuit Diagrams</td>
<td>214</td>
</tr>
<tr>
<td>PCB Layouts</td>
<td>227</td>
</tr>
<tr>
<td>References</td>
<td>229</td>
</tr>
</tbody>
</table>
Executive Summary

This report describes the first part of a multi-year capstone design project undertaken by the Electrical and Computer Engineering class of 2012. The goal of the project is the development of a small autonomous boat that will be capable of precision navigation and data collection in a creek or river even in rapidly moving water. This year is focused on developing sensor systems that track the boat hardware and measure its position with an uncertainty approaching 10 cm root mean squared (RMS) at a range up to 25 meters. A software framework for system control and data acquisition is being developed with feedback for end user hydrologists. This report also includes all of the analyses that were done regarding the environment, safety of the project, development of software and hardware and an appendix that includes our quality assurance tests, software documentation, and circuit designs.
Introduction

As urbanization and population around the globe increase, maintaining the quality and quantity of water supplies is becoming an important issue (1). There are a multitude of factors which influence water quality, including agricultural runoff, urban runoff, and sewage from manufacturing and industry. Because this is such a complicated and multidimensional problem, measurements to determine the quality of water are problematic. This complexity is only compounded in local river systems where pollution enters at a single location and is then carried down-stream. Additionally, distributed sources of contamination, such as runoff or absorption of airborne pollutants, can enter the river over a wide area. Determining the source of water pollution from a single measurement can be a difficult if not impossible job.

Ideally, multiple measurements could be taken along a stream and compared to obtain a complete picture of the water quality. Hydrologists must visit many sites on the river and manually take water quality readings, or set up a station at a single site to take continuous readings (2). An autonomous system would be excellent for this repetitive task. For example, taking flow measurements is one of the hydrologist’s most tedious tasks. The researcher must make flow velocity and depth measurements at multiple locations across the width of the river. Presently, this involves wading into the river if it is shallow enough, or stringing a probe from a bridge or other structure over the river. By traversing the river automatically while taking measurements, much of the work is eliminated.

In this paper we present such an autonomous boat system. Although several autonomous boat systems already exist, all of these systems are guided by global positioning systems (GPS), which limits the accuracy of the position measurements (3). For making hydrology measurements such as stream flow, significantly more accurate positioning is necessary. Our system utilizes a fusion of both GPS and multiple other sensors such as ultrasonic rangers, optical trackers, compasses, accelerometers and
gyroscopes to sense position more accurately (4). To achieve a higher accuracy of all incoming position
data is passed through a particle filter. Particle filters have been used successfully in tracking models in a
range of robotics applications over the past several years (5) (6). This filter allows us to combine several
different measurements along with a system model of boat position into a single more accurate
measurement of the boat’s states.

These improvements allow our system to determine the boat location to a significantly higher
degree of accuracy than current autonomous boats. This makes it ideal for use by hydrologists studying
water quality in river environments. This would mean a significant reduction of the quantity of work and
money required to survey a single sight on the river, allowing hydrologist to gather a larger number of
river samples and get a more complete picture of river water quality.
The overall system (Fig. 1) consists of a pair of tripod mounted shore-stationed sensors, a sensor interface box and user interface software, boat based sensors and, boat-beacons. Analysis predicts the unfiltered value of the shore sensors will be less than 20 cm RMS. With a particle filter it is anticipated that the 10 cm RMS accuracy requirement can be met at a range of 25 m.
The software modules act as the primary user interface for controlling the boat in automatic mode (with scripts or simple commands). The two tripod sensors are of the exact same design, containing a motor system and the shore-based portions of the infrared and ultrasonic sensor systems. The shore system (Fig. 2) also contains one end of the radio frequency (RF) datalink.

The boat hardware (Fig. 3) consists of two beacons used by the shore sensors and an ArduPilot Mega (7). An XBee link for telemetry transmission is in use as a shore-to-boat communication device. The team has been adapting the hardware and firmware libraries
provided to enable high-resolution positioning of the drone on water, and to communicate in duplex with the shore station. Our system tracks a variety of boat states to ensure the gathered data is of maximum utility (Fig. 5). These states include: absolute x and y position, absolute heading, absolute rotational velocity, and relative forward and sideways speeds. To allow independent control all of these variables independently of one another, the design is based off of a boat with four bi-directional jet-pump motors (Fig 4). This would allow the boat to be stationary even in moving water, while simultaneously maintaining its heading so sensors that rely on being oriented correctly to operate.

![Fig. 4 Boat motor configuration](image)

![Fig. 5 Boat orientation diagram](image)

The system is designed such that it is easy to deploy. The user must deploy the two
tripods, mount the two sensors and attach the interface box. Then the system automatically calibrates and determines its local coordinate system (Fig. 1).

**Shore Sensors**

**Azimuth-Elevation Positioner**

To track the position of the boat and send sensor data back to the Shore Station, we use an Elevation over Azimuth positioner (AZ/EL). The AZ/EL positioner provides a mounting surface that rotates about two axes. Two interconnected servomotors achieve the desired motion. The servomotors interface with an ATMega328 microcontroller chip. Our team has constructed a custom ultrasonic ranging device and infrared tracker mounted on the head of the AZ/EL positioner (Fig. 6, Fig. 7). In addition a second printed circuit board (PCB) is mounted on the back of the positioner that contains an interface for power and communications, additional amplifiers, and signal processing hardware.

![Fig. 6 Shore sensor tripod mounting and AZ/EL positioner](image)

The ATMega328 microcontroller executes a control loop to keep the positioner pointed at the source of the infrared light on the boat. This also ensures that the co-located ultrasonic
rangers are always pointed at the boat. We use the available Inter-Integrated Circuit (I2C) hardware on the microcontroller to transmit data and commands to and from the shore station. Using this interface, the positioner can report where it is pointing and how far the boat is. The AZ/EL positioner is mounted to the top of a tripod and upside down so the servo motors can make the best use of their available movement range. A cable from the shore station brings power and I2C communications to the sensor.

![Fig. 7 Shore sensor and pre-amplifier board](image)

**Infrared**

The boat-mounted infrared beacon consists of an array of 4 high irradiance, wide-angle, infrared light emitting diodes mounted on top of a mast with a switch modulating driver circuit. We use infrared light with a wavelength of 850nm instead of visible light. There is significantly less ambient infrared light, allowing us to differentiate our signal from other ambient light (8). Additionally, we modulate this beacon at 20KHz, to further isolate signal from ambient light.

The shore mounted optical sensor assembly consists of an infrared-pass optical filter with a lens that focuses incoming light to a quadrant photodiode mounted on the AZ/EL tracker (Fig. 7). To determine where the beacon is, we use a lens to focus the light from the beacon to a small circular area over the photodiode. When this lens is pointed directly at the beacon, the intensity on all 4 channels is
equal. If the boat beacon is skew with the lens the incoming light will land disproportionately on one of the sensors. Using this information we can calculate the magnitude of the skew in both azimuth and elevation directions.

The quadrant photodiode channel is connected to an RLC band pass filter which is then followed by multiple amplification stages. The multiple gain stages allow us to turn on and off gain in order to prevent saturation. The amplification stages are connected to a peak detector circuit transforming the alternating current (AC) output of the photodiode into an analog direct current (DC) voltage. This voltage is converted into a digital measurement, which is then fed into an onboard microcontroller containing the control loop.

**Ultrasonic**

In addition to the Infrared tracking system, the project includes an ultrasonic pinging system to determine the range of the boat from a shore tripod. This range measurement allows the system to calculate an x and y, giving two independent measurements for fusion into a more accurate position. Our system sends an ultrasonic ping to the boat, which then responds with another ping. The time delay from the first ping to the response is combined with a pre-calibrated speed of sound to calculate the boat’s distance.

The ultrasonic shore station circuitry contains two different ultrasonic transducers, one for receiving and one for transmitting, as well as driver and receiver circuitry (Fig. 7). The driver circuit consists of a 4 MHz oscillator which is divided down to a 40 kHz signal and used to drive a dual, half H-bridge circuit running off of 18 volts which produces a 36 V peak to peak, 40 kHz signal across the transmitter.
The receiver is sensitive only to signals in the range of 38 to 42 kHz. These signals are then passed through two separate amplification stages (Fig. 8). The amplified signals are then sent through a comparator which triggers off of a specific threshold. This signal is then processed by a microcontroller to detect incoming pulses.

**Boat Sensors**

![Boat processor, sensors, and RF link](image)

*Fig. 9 Boat processor, sensors, and RF link*

**Accelerometer**

A three-axis accelerometer is integrated on the ArduPilot Inertial Measurement Unit
(IMU) shield, with communication handled by the open source libraries provided for the ArduPilot (Fig. 9). The accelerometer reports acceleration in the range of +/- 3g as meters per second squared. This data is both transmitted in the raw to the shore station for logging and used by the particle filter to help track the change from the boat’s previous position.

**Global Positioning System**

A GPS is sited on a breakout board developed specifically to work with the ArduPilot setup (Fig. 9). This board is connected directly to the front of the ArduPilot board, where it communicates using a dedicated Universal Asynchronous Receiver/Transmitter (UART). Communication with the GPS is handled by another open source library that can be found in the ArduPilot source code package. The GPS itself is, unfortunately, not accurate enough to be used alone. Specifically, it is capable of resolving to within a 3 meter radius of the unit 50% of the time - this means the system can establish a global positioning fix that is sufficiently accurate to be used for locating the operating site. We can also use the course speed and heading information provided by the GPS to help update positions, though it must be given a light weight to avoid the potential for larger errors.

**Gyroscopes**

There are two Gyroscope packages sited on the ArduPilot IMU board: one XY-axis gyroscope and one Z-axis gyroscope (Fig. 9). Their placement enables the tracking of the three different rotational axes on the main board, which allows us to feed the rotation in radians per second to the position filter. The gyroscope data is also used to calculate the system’s pitch and roll, which is used by the magnetometer libraries while calculating a heading when compensating for tilt.

**Magnetometer**

The final sensor provided in the ArduPilot package is an optional triple-axis magnetometer (Fig. 9). This is located on a small breakout board utilizing an I2C connection to
the ArduPilot via the IMU. This magnetometer can be used as a compass for determining the heading of the boat, allowing for simplified navigation that doesn’t rely on guess-and-check movement. Unfortunately the triple-axis nature requires more involved firmware capable of accounting for roll and pitch to ensure only the yaw angle is tracked. The gyroscope data is extremely valuable for this purpose, allowing the library to calculate the corrected heading.

Filtering

Our system uses a particle filter to combine measurements from these sensors into a coherent and accurate picture of the boat’s state. Particle filters are part of a class of artificial intelligence algorithms called sequential Monte Carlo methods (6). In the simplest possible terms a particle filter keeps track of a range of possible system states called “particles” and discards those that are the least probable according to external measurements. Particle filters have been used in a wide variety of autonomous systems in recent years due to their advantages over more traditional methods such as Kalman filters (8). Most importantly, particle filters do not require that the system model be linear. Additionally, they are computationally simple allowing even a low power processor to execute one. Finally, as the number of system states being tracked increases the probability that the filtered system state is the true system state approaches 1.

Our model for the boat is highly nonlinear, as can be seen by an examination of the transfer function (Fig. 10). This non-linearity requires a form of filtering capable of working even with non linear models. This ruled out a standard Kalman filter (although not an extended Kalman filter). Ultimately, we decided to use a particle filter because of its simplicity and ability to work
System Control and Data Acquisition (SCADA) Software

The laptop software is the single point of comprehensive control for our entire system. Every time the software is started it enters an operational calibration state at which point sensors are calibrated for current operating conditions. The boat’s initial position will then be found by scanning the sensors over the operational swath. From this point on the software will use readings from the trackers to determine the position of the boat.

The boat is continuously tracked and sensor information is logged. The graphical user interface (GUI) has two areas for displaying sensor measurements – one section for sensors relevant to the boat’s status, such as location, heading, speed and internal temperature; the other for any relevant payload hydrology sensors such as depth, pH, water temperature, and turbidity (Fig. 11). All of these sensors have their information logged in separate CSV files to allow the ability to graph in external graphing software with no additional overhead.
The software supports three autonomous modes of navigation as well as manual mode. Freeze mode which holds the boat’s current position in the water. Ferry mode where the user can specify a pair of x and y coordinates as a destination for the boat. Scripting mode allows the specification of a combination of ferry and freeze commands. When in autonomous navigation mode, an emergency stop sends the boat into a freeze state to prevent it from being damaged. It also has an emergency ‘return to origin’ mode. Manual control mode overrides all other modes of operation via a separate radio frequency remote control. This is meant as a failsafe as well as a more direct means of control for the boat.

Calibration

A fundamental aspect of the system is the ability to measure the position of the boat using our sensors. To ensure measurement accuracy, the various subsystems must be calibrated. In this
process we take raw measurements from our sensors and compare them to standard systems of measurement. We then decide what modifications to make our measurements so that they match the standard. There are three main aspects to our system calibration: primary calibration, operational calibration, and verification calibration.

Primary calibration takes place before the system is shipped to the end-user. In this stage, the instrument is modified so the measurements are internally consistent and as close as possible to the standard. For our system the AZ/EL is calibrated to determine the angle zeros as well as the angle to pulse width modulation time ratio.

Operational calibration is done directly prior to use. This stage is used to eliminate errors caused by outside factors that will affect the system’s measurements. The range finding technique requires knowing the speed of sound, but the speed of sound can vary greatly depending on temperature and other factors. Therefore, our system must determine the speed of sound empirically before use. It calculates this value by sending pings back and forth over a known distance and calculating the time of flight.

Verification calibration is used during the operation of the system in order to ensure the measurements received from the system are still valid. This can also be called “sanity checking” and usually entails checking to see if the measurements are within possible limits. A simple example is checking to see if a time measurement is negative. Since measured time cannot be negative, the system can clearly see something has gone wrong and can warn the user of erroneous data. The main check for our system is ensuring the baseline between the sensors is always less than the sum of the distances from the sensors to the boat. If the sum is greater than the baseline then the system is reporting a configuration that is not geometrically possible.
**LPARD-BSRD:**

The boat is driven by 4 motors, two sideways thrusters at the boat's bow and stern that point perpendicular to the center line and two forward/backwards thrusters that point parallel to the center line. (Fig. 2)

![Diagram of boat motor configuration](image)

**Suggested Emotion Edge Kayak Parameters:**

- \( a = \) boat major axis in meters,
- \( b = \) boat minor axis in meters,
- \( c = \) boat height minor axis in meters,
- boat is assumed to be an ellipse
- \( m = \) boat mass in kg
- \( r = \) density of water in kg/m³
- \( Tr = \) time for a full revolution in seconds
- \( w_{\text{max}} = \) max rotational velocity
- \( m = \) dynamic viscosity of water at room temp
- \( V_{\text{rotmax}} = \) Maximum rotational velocity at 1/4 length from boat center

\[
a = 1.448 \\
b = 0.3668 \\
c = 1 \\
3 \\
* b \\
m = 34 \\
r = 998.2 \\
Tr = 20 \\
w_{\text{max}} = 2 \pi \frac{20}{m} \\
m = 1 \times 10^{-3} \\
V_{\text{rotmax}} = w_{\text{max}} \times 0.25 \times a \\
V_{\text{max}} = 2
\]
**Motors:**

- The boat model predicted a combined force of 42N for the forward motors to reach top speed.
- Vmax must meet 2 m/s velocity max based on hull velocity analysis
- Approximation estimates about 200W of power given to each motor.
- For the sideways motors we estimate 100W of force each
- Based on preliminary research the inlet diameter will be approximately 0.1312 feet.

**Manual Control:**

- 75 MHz 3-6 Channel RF Link
- RC controller is the only mechanism capable of switching to manual mode
- Restores to Freeze Mode on release

**Use Cases:**

Two possible ways of taking depth measurements: The first is by using a pressure transducer - by lowering this device down into the water the amount of pressure above the device would determine the depth. Option two was to use on-board sonar on the hull of the boat.

Determining pH levels through taking measurements about 6 inches below the surface (Note on shallow waters around Lafayette College: the Bushkill creek is about a little less than a foot deep in the shallow areas up to around 8 feet in the deepest areas).

Turbidity may require optical measuring devices and can be used for a variety of other measurements, such as salinity (Note on basic principle of measuring: take a measurement of conductivity and translate that into specific measurement desired).

For measuring dissolved gases, hydrologists might care about the amount of dissolved oxygen in the water, though different end users may want to see other dissolved gases. (Reference: Megan Rothenberger of the Biology department who works in the area of water quality)

Packaged measurements that do multiple tests (temp, pH, depth), but these packages are very costly.

**For legal issues, there shouldn't be any problems since the vehicle is unmanned; many issues arise when there people aboard a watercraft.**
**LPARD-TDF-2012 On Water Safety Plan:**

The purpose of this safety plan is to present information and rules that will help in the prevention of injury or serious harm while using the LPARD-TDF-2012 system. This safety plan is not all inclusive, but is meant to inform the user so as to prevent the vast majority of injuries. The system designers and manufacturers are by no means responsible for any injuries that occur during the use of the LPARD-TDF-2012 system if the items within the this safety plan and supplemental User’s Manual are not abided by.

**Safety Procedures**

**Electrical:**

1. Keep electrical equipment (laptop, shore power supply, shore sensors, etc.) at least 20 feet away from any water source to avoid possible electrocution and equipment damage.
2. If electrical components smell like they are burning, power down the system as soon as possible and do not touch any electrical components because they might be extremely hot and cause burns.
3. Malfunctioning electrical components run the risk of catching on fire. If the boat or any subsystem catches on fire, do not throw water on any system that is powered up. Proceed with caution when putting out electrical fires and avoid contact with flame.
4. Powering up: If the boat is being powered in maintenance mode, keep the power cord and electrical output free of any possible water sources that could cause electrical shock.
5. Check electric components often for visible problems and have spare fuses ready at all times.

**Environmental:**

1. While the LPARD-TDF-2012 system is designed to operate in suboptimal conditions, it is recommended that the system is not used when in poor weather conditions. Check weather forecasts before planning and trip and consistently keep an eye on the sky to make sure that there are no developing clouds or storms. Lighting storms and extremely windy conditions can cause suboptimal conditions that could result of serious injury.
2. Shore equipment should be setup and operated on a relatively flat surface so that the equipment as well as the user are able to operate on stable ground.
3. Be sure to survey the shore station setup area and do not set up the shore station in an area that at risk of falling rocks or tree limbs.
4. If you must step into the water to position or pick up the boat, make sure that there are not submerged objects that can possible cause physical harm.
5. Take precautions to limit the amount of UV exposure due to the sun. It is recommended to wear sunscreen if planning on operating the system for a prolonged period of time.

**Attire:**

1. Proper footwear should be used when setting up and operating the LPARD-TDF-2012 system. The shores of rivers, creeks and streams are uneven and
often difficult to navigate. Inappropriate footwear will greatly increase the risk of slipping or falling.

Mechanical:
1. At no point should hands or fingers be placed near the propulsion components when the system is powered on. If the propulsion system gets dirty or jammed, be sure to power the system down before attempting to clean or handle the boat.
1. Malfunctions are always possible. If the propulsion system goes down while the boat is on the water, try to avoid getting in the water to retrieve it. It this is absolutely necessary to enter the water, wear a flotation device.
1. Do not operate the boat in manual or autonomous mode in a location where there is a chance of hitting other boats or swimmers. Survey the area both upstream and downstream to make sure that no one has a possibility of getting hit.
1. Do not steer the boat towards protruding rocks or trees in the water. If the objects are visible, use manual mode to ensure that the boat does not accidentally hit an object and cause damage.
1. Do not attach external objects to the boat that the boat is not specifically meant to carry that can add extra weight or cause the boat to become overloaded or unbalanced.

Transportation:
1. While transporting the LPARD-TDF-2012 system, make sure that the boat and all other required equipment are properly fastened to the vehicle. Do not use straps or cables that are partially torn or show excessive wear.
1. Use an appropriate vehicle to transport the entire system. Do not position any system parts in the vehicle that will obstruct the driver’s view.

Other:
1. While the system is designed to have the ability to be operated by a single person, it is recommended to not work alone in isolated areas. Someone should be kept within earshot at all times since cell phone carriers’ are not available at all locations.
1. Even though it is not entirely necessary, it is recommended that a lifeguard/EMT certified individual present in case of an emergency.
1. Do not launch the boat from a dock or launch ramp while other boats are trying to dock / cast off.
1. If planning on operating the system at nighttime, have a flashlight ready so that it is possible to navigate obstacles on the shore safely.
1. Depending on the location of operation, always be aware of dangerous wildlife that may be present.
Boat Payload Analysis: R008

1) Equipment installed on the boat is minimized:

Only necessary boat hardware is used see Table A to see a list of all boat hardware and why it is necessary.

<table>
<thead>
<tr>
<th>Boat hardware system</th>
<th>Reason for inclusion on boat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ardupilot</td>
<td>Acts as main boat microcontroller and is responsible for managing data filtering, RF communication, system control, sensor data acquisition.</td>
</tr>
<tr>
<td>GPS</td>
<td>Aids in obtaining a better estimate of the boat position.</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>Responsible for high accuracy timing of ultrasonic sensors. (Recommend replacement with a smaller and lighter microcontroller unit).</td>
</tr>
<tr>
<td>Xbee</td>
<td>Communicates with shore station.</td>
</tr>
<tr>
<td>Actuator Board</td>
<td>Provides IR beacon for Az/El tracking and ultrasonic actuators for ranging.</td>
</tr>
<tr>
<td>Packaging</td>
<td>Prevents damage to subsystems from turbulent water and keeps sub-systems in place and organized for easy access.</td>
</tr>
<tr>
<td>Batteries</td>
<td>Provides power to run all electronic systems on boat including microcontrollers and motors.</td>
</tr>
<tr>
<td>Payload Sensor Modules</td>
<td>Required for hydrologic data acquisition.</td>
</tr>
<tr>
<td>Motor</td>
<td>Required for moving boat or keeping station in moving water.</td>
</tr>
</tbody>
</table>

2) Equipment on the boat is of a mass and location so as to not reduce the boat's stability and resistance to capsizing:

Unconsidered in the current system as no boat is available for testing and the most significant portion of the boat's weight and internal volume has not yet been implemented (batteries). Future groups may wish to start at the following web-site for this analysis: [http://en.wikipedia.org/wiki/Metacentric_height](http://en.wikipedia.org/wiki/Metacentric_height)

3) Equipment leaves ample room and freeboard for the weight and size of typical end user payload:

Unconsidered in the current system as no boat is available for testing and the most significant portion of the boat's weight has not yet been implemented (batteries).
Introduction

An examination of the system’s planned operating swath of 25 meters against the actual operating swath, as well as capabilities for operating with degraded positioning out to 200 meters.

Analysis

Unfortunately the full system does not operate out to the specified 25 meters. In Figure 1 the individual sensors’ current operational swathes are detailed. Due to an unfortunate bug in the beacon PCB, the range and angle sensors cannot operate simultaneously – this means the full, fused position measurements are unavailable anywhere in the operational swath.

Figure 1. Individual sensor operational swaths. Note that the compass, accelerometers, and gyro are equivalent in swath to the GPS.

While the fused measurement is unavailable in any operating swath, the system is capable of the degraded measurements out to 200 meters. This is due to the boat sensors (GPS, compass, accelerometers, gyro) all operating anywhere on the surface of the Earth while they have power. The primary limitation to that degraded operating range is the RF link. The XBee ZigBee radios being used have been seen to operate at 30 meters during the GPS QA test and are specified to work from 40 meters up to a maximum possible 1.6 kilometers. Seeing as how this is far beyond the 200 meter degraded range, it can be expected that the system is capable of degraded positioning even beyond the specified 200 meters.
Introduction

An examination of possible techniques for use with the triple-axis compass provided in the Ardupilot kit (HMC5883L) to further enhance boat heading to 0.25° root mean square (RMS) accuracy. We will examine the motivation, hardware capabilities, and some proposed solutions for increasing the resolution below.

Motivation

An estimate of the boat’s heading is desirable for several reasons. First is for determining the amount of thrust necessary from each set of motors for movement. Second is for maintaining position against any river current. An example use case is for flow rate measurements: the boat needs to be capable of pointing into the current while both holding position and moving (either ferrying or crabbing sidewise).

Hardware Capabilities

The compass design specification provides for an expected accuracy of 1 to 2 degrees on each axis. By merging values from the X and Y axis we have been to be able to meet 1.5° RMS, which has been verified during systems testing by examining the accuracy when pointing in several known directions. A compass library is currently used in the firmware that is capable of resolving hard iron offsets with the compass values and tilt compensation with the assistance of accelerometers. This does require the three-axis ADXL335 accelerometer that came as a portion of the boat hardware.

Also useful for resolving the heading is the provided gyroscopes. All three gyroscope axes (IDG-500 xy-axis and ISZ-500 z-axis packages) have a high-sensitivity setting of 110°/s and high speed setting of 500°/s, which are both beyond the values handled by the boat simulation (which maximizes at 14.9°/s or 0.26rad/s). By selecting the high-sensitivity value for more accuracy, we find it has a 4.5V range with 9.1mV steps, meaning each step covers approximately 0.22°/s. At our current polling speed of 1 Hz this value is just below the desired heading value, and when combined with the compass and previous states in the particle filter should allow for resolving at the desired heading accuracy. To avoid these numbers skewing due to drift, it is recommended the gyroscopes be reset when the system is motionless (x and y accelerometers read as zero). In addition, the runtime of the system should be limited as the drift error accumulates.

Possible Solutions

There are three additional phases planned for implementation that will bring heading accuracy to the desired amount: hard iron offsets, tilt compensation, and a firmware implemented particle filter. These additions will also have to account for any drift caused by water flow and drift due to hardware errors (such as gyroscope friction).

Hard Iron Offsets

Hard iron offsets allow us to compensate for hard iron, or magnetic objects that remain fixed relative to the sensor. By calculating and compensating for hard iron it is possible to more accurately follow soft iron effects (in this case the Earth’s magnetic field). The library being used for the compass is capable of calculating hard iron offsets, and is currently planned to do so during calibration. At the cost of calibration time our system will then be capable of canceling out some portion of local
magnetic fields, allowing it to more accurately follow the magnetic field of the Earth. For example, we can potentially account for some of the active systems on the boat (such as power connections). Unfortunately this hasn’t been implemented in the current iteration due to debugging time constraints. Once implemented the hard iron offsets and magnetic declination adjustment will ensure the compass reported heading more closely matches True North.

Tilt Compensation

The tilt compensation approach is conceptually simple: use a three axis accelerometer to adjust the reported readings for motion. Using the same library noted above with the onboard compass and accelerometer we can readily account for tilt. Specifically, we can use the accelerometer data to account for the compass not being held flat. This will reduce noise due to environmental factors such as vibration and intentional (or unintentional) system motion, especially adjustments required due to tilt. This behavior is highly desirable, as constant noise was noted during testing, even during stationary phases, and additional noise is expected during system deployment due to movement and environmental factors such as river flow.

Particle Filter

The third technique for refining the heading data is a particle filter. Particle filters are commonly used in Robotics and Artificial Intelligence research, and are statistical model for handling situations in which the variables change continuously but are not sufficiently restricted (such as to a Gaussian distribution where Kalman filters are efficient). By having large numbers of virtual “particles” weighted based on belief, it is possible to establish an estimate of the system state (the position and heading for the drone). The belief can be determined by examining the various sensor measurements (for heading this will be compass and gyroscopes), while particles can be moved to another state in the same way. One advantage of this style of movement is that it takes account of past measurements, which will help smooth out some hardware induced jitter. In addition, a useful feature of particle filters is that as the number of particles approaches infinity the output becomes the actual system state. This allows for fine tuning the filter accuracy at the expense of resources such as computing time and space. Note that this would replace the Kalman filter, which tends to be less accurate in cases such as ours where the system model is nonlinear and noise may not be Gaussian.

Conclusion

The expectation is that a combination of these methods could enable further heading resolution. By eliminating as much influence from local magnetic fields as possible, reducing movement induced noise, and using particle filters to fuse compass and gyroscope data we expect to resolve the specified 0.25° RMS accuracy.

---

1 See T003: R000-3 in the official Acceptance Test Report

2 A description of hard and soft iron offsets can be found at http://blogs.freescale.com/2011/03/14/hard-and-soft-iron-magnetic-compensation-explained/

3 An example of tilt compensation with Arduino, our magnetometer, and a similar accelerometer can be found at https://www.loveelectronics.co.uk/Tutorials/13/tilt-compensated-compass-arduino-tutorial

4 A description of a particle filter being used for indoor positioning via sensor fusion can be found at http://galileo-platform.com/handbook/Pages/How-improve-indoor-positioning.aspx

5 Further examples of particle filters being used for aircraft navigation can be found in the following article: Gustafsson, F.; Gunnarsson, F.; Bergman, N.; Forssell, U.; Jansson, J.; Karlsson, R.; Nordlund, P.-J.; “Particle filters for positioning, navigation, and tracking,” Signal Processing, IEEE Transactions on , vol.50, no.2, pp.425-437, Feb 2002 doi: 10.1109/78.978396
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=978396&isnumber=21093
vi A description of the library can be found here: http://n0m1.com/2012/02/27/6doF-arduino-compass-accelerometer/


LPARD Manual Control Analysis per R003

Date: 3/3/2012
Engineers: Charles Thomas, Edward Eppe III

This document attempts to outline how the manual mode and control of the LPARD system will work. It outlines the technical aspects and minimum requirements of the system and attempts to highlight and solve any problems that may be encountered when a future group implements the system, since the control loop will not be closed this year. It is divided into three sections that explain what frequency was picked and why, the technical aspects of the transmitter and receiver and finally how the boat switches to manual mode and how it should behave in this mode.

Frequency Conflicts and F.C.C Regulations

The FCC provides several frequencies for use with remote controlled systems. These frequency bands are outlined below and their advantages and disadvantages discussed.

The 27 MHz frequency band does not require a license and provides room for six channels. There are no usage restrictions except that the 6th channel is shared with CB radios. The effect of this channel sharing would most likely be negligible due to the decreased popularity in CB radio. However it is still a factor that must be taken into account as the purpose of the river droid is to make environmental measurements easier to make in a larger range of locations. Therefore, restricting the locations to areas with no CB radios is undesirable. There is also the possibility of interference between other RC devices on the same frequency.

The 50 MHz frequency band is specifically allocated for amateur radio operators and requires an Amateur Radio License as outlined in the FCC Rules & Regulations Part 97. There are 10 channels available within the frequency band. However, because one of the river droid’s main goals is to increase the ease of river measurements, requiring the operator to register for a license is undesirable and should be avoided if possible. Again, there is also the possibility of interference between other RC devices on the same frequency.

The 72 MHz frequency band has a total of 50 channels available (channels 11-60) generally divided into sub-bands of 10 channels for use with each control device. These however are only available for model aircraft and therefore are not applicable to this application.

The 75MHz band has 30 channels allocated to it(channels 61-90) also sub-divided into sub-bands of 10 channels. This band is allocated only for surface systems. The channels use frequencies ranging from 75.410 –75.990 MHz. Often only even or
odd channels are used to avoid interference between channels. This range does not require any form of licensing. Once again, there is also the possibility of interference between other RC devices on the same frequency.

The 2.4 GHz Spread Spectrum is another frequency allocated by the FCC. This frequency solves many of the interference problems faced by other frequency bands. By using an algorithm to jump along the spectrum the chances of any two signals attempting to use the same frequency at any one time are vastly reduced. The small wavelengths also mean much shorter antennas which have obvious benefits. There are several disadvantages, though. The XBee transceivers run at the same frequency. Although the spread spectrum in theory should handle this, the proximity of the high gain antennas means there would be some spectrum coupling and saturation of the frequency. The short wave lengths of the spectrum also require there to be a line of sight between the transmitter and the receiver. It also means the signal is more likely to be reflected or absorbed by certain objects. The spectrum-coupling problem can be overcome by orienting the receivers in a certain way. The design of the boat hull can also minimize shapes that may reflect the signal. 2.4 GHz systems are generally much more expensive than alternative frequencies which may or may not be a problem.

From the analysis above RC controllers that operate in the 75 MHz band are the most desirable and should be used if possible. The 27 MHz band should be used if for whatever reason the 75 MHz band is unavailable. The 2.4 GHz band can be considered as a viable frequency if the other two cannot be used. Although the spectrum coupling problem can be overcome the line-of-sight issue cannot which is a major disadvantage. Because the main XBee link requires line of sight it is desirable for the backup system to not require line of sight. Therefore the 2.4 GHz frequency should only be used if the others absolutely cannot.

**Transceiver Specifications**

The RC transceiver will have to be capable of transmitting at least 3 channels, for forward and reverse, rotation, and one for turning manual mode on and off. The transmitter therefore requires a toggle switch for manual mode activation. Generally more channels will probably be preferred. Extra channels could be used for strafing left and right, increased throttle control, and possibly a position hold option to allow semi-autonomous control. These options will require up to 6 channels which is generally the upper limit of the number of channels 75 MHz transceivers are capable operating with. 75 MHz receivers require an antenna length of 50 cm which must be taken into account when designing the structure of boat. On average 75 MHz receivers have a range of 150 meters. However receivers with a range of 300 meters are also available for relatively cheap thus meeting our distance requirements. Receivers with a failsafe mechanism are also available and should be used if possible. This allows the receiver to be programmed to output certain values in case of a loss of signal. For instance this could automatically put the
boat into freeze mode if the manual control signal drops out.

To summarize the specifications are:
- 3-6 channels
- Toggle switch on transmitter
- 300 meters range
- 50 cm receiver side antenna
- Built-in failsafe

**Basic Operation of Manual Mode**

Manual mode can only be turned on or off through the use of the RC controller as opposed to using the GUI or both. This is both desirable and necessary, and it is easy to switch between the two. It is desirable as it means the boat can still be controlled in case of a complete system failure or any loss of the main 2.4 GHz wireless link. It is necessary due to the nature of the manual mode activation through the RC controller. Manual mode is activated through a toggle switch on the RC controller. When the toggle is switched on one of the channels on the RC link is activated and remains activated until it is switched off. This notifies the processor on the boat that we are in manual mode instructing it to ignore any navigation related messages from the XBee link until manual mode is switched off. This means its impractical to allow the GUI to change in and out of manual mode. An example of where this method would fail is if the RC controller switched manual mode on and then the GUI attempted to turn manual mode off. The RC controller cannot be informed of this change and therefore the toggle will still be in the on position and the ‘on’ channel still activated. This puts the controller and the GUI and boat out of sync creating both a technical problem to overcome as well as creating confusion for the end user.

The channels from the RC receiver are routed into the Ardupilot which has a port especially designed for RC receivers. When the ‘on’ channel is activated the Arduino is notified and one of the built in interrupts is called. The Arduino ignores the main link’s navigational instructions and instead processes the channel inputs and directs the motors and servos accordingly. It also should notify the shore station that it has been place in manual mode through the XBee link. Only once the ‘on’ channel has been quiet for a certain amount of time (to avoid false off signals incase of brief signal dropouts) does the system again return to automatic mode. The system automatically returns to ‘Freeze Mode’ and should notify the shore side GUI that manual mode has been turned off.
For a possible implementation of manual control, the boat circuitry is powered through the RC receiver channels connected to ArduPilot Mega or through the ESC connected through the throttle out channel on the ArduPilot Mega board. If the boat is not powered through the RC receiver in this way, it could also be powered through a pin on the ArduPilot Mega board itself, in the same way it is being powered in our current implementation. The board also has a resettable fuse between the power input and the rest of the board which trips if the current through it exceeds 500mA. If a future system design implemented RC control, this fail-safe fuse would protect the circuitry from burning out while allowing the use of high current servos. The image below shows the possible RC power connections to the ArduPilot.

For RC setup, first connect the RC receiver to the ArduPilot Mega. Use female-to-female servo extension cables if necessary. An RC controller with enough channels to control each element of movement is necessary for the design. In this case we are concerned with RC boat movement, so there should be a channel for throttle and a channel to control the rudder. As previously described, there should also be a channel for switching manual mode on/off. These channels should each be connected to an input on the Mega board.
A possible controller to use in this design is the FrySky CT-6B 2.4GHz 6 Channel Transmitter Remote Controller (Model: CAM80912 for around $50). Originally intended for the use of RC airplanes, this controller contains six channels which we may possibly need as previously described and operates in the 2.4GHz band as required. One of these channels should also be designated for manual mode on/off. As for the receiver, the design could use the Spektrum AR6110 DSM2 Microlite 6-Channel Park Flyer Rx, Air. This is available for around $45. Images of the transmitter and receiver are shown below.

Software for manual control would simply need to check if the system is in manual or autonomous mode. If it is in autonomous mode, the receiver should then know not to accept any more commands from the transmitter. While in manual mode however, the software will allow the receiver to accept commands from the controller. These modes can be easily changed on the transmitter with the manual mode channel. Finally, below is a possible schematic of the boat design with manual control.
EMI Analysis

The Statement of Work states that our system must abide by the US CFR Title 47 Part 15 subpart B regulations for Class A digital equipment. Going to the FCC documentation, we discover the following information: The field strength of radiated emissions from a Class A digital device, as determined at a distance of 10 meters, shall not exceed the following:

<table>
<thead>
<tr>
<th>Frequency of emission (MHz)</th>
<th>Field strength (microvolts/meter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30–88</td>
<td>90</td>
</tr>
<tr>
<td>88–216</td>
<td>150</td>
</tr>
<tr>
<td>216–960</td>
<td>210</td>
</tr>
<tr>
<td>Above 960</td>
<td>300</td>
</tr>
</tbody>
</table>

For our system, the source that will create the greatest amount of EMI will be our 4 MHz oscillator found on our backend board and boat board. Based on analysis for an ideal square wave, the amplitude of the odd harmonics are given by: \( A_n = \frac{2}{n\pi} \). So, looking at the closest harmonic of our 4 MHz oscillator which falls into the 30MHz or greater FCC category, we find that the 9th harmonic falls at 36 MHz.

\[
A_9 = \frac{2}{9\pi} = 0.07073553
\]

Taking the length from the 5V, 4MHz oscillator to ground to be 1mm, the electric field created by the 9th harmonic of the oscillator is \( \frac{(5 \text{V})(0.07073553)}{5 \text{mm}} = 70.7355 \text{ V/m} \). This field strength decreases as a factor of \( \frac{1}{r} \) as we move away from the oscillator. Using the following equation:

\[
100 \text{ } \frac{\mu V}{m} = \frac{70.73 \text{ V/m}}{2^x}, \text{ where } x \text{ is distance in meters at which we meet the EMI intensity requirement}
\]

We see that we hit the 100 \( \frac{\mu V}{m} \) at 19.43 meters, nearly twice the distance as required by FCC. Therefore, we propose an EMI enclosure that reduces the electric field strength. Our current strength at 10 meters is \( \frac{70.73 \text{V}}{2^{10}} = 0.069077 \text{ V/m} \) and we need to get down to 100 \( \frac{\mu V}{m} \). Therefore, we need shielding on our backend board and boat board of:

\[
20 \log \left( \frac{0.069077}{100 \times 10^{-6}} \right) = 56.79 \text{dB}.
\]

---

1 Equation for harmonics found in: Signal and Power Integrity - Simplified, 2nd Edition by Eric Bogatin
Environmental Analysis per R007 & GPR002

Date: 4/3/12
Engineer: Edward Epplle III

This document analyzes the environmental aspects of the LPARD system and addresses the following design requirements:

- A possible spec for the boat packaging of the LPARD system shall be provided, such that the following aspects could be achieved in future implementation:
  - It may be operated day or night.
  - It may be operated in moderate fog and rain.
  - It may be operated in winds less than 20 km/h
  - It may survive operation when splashed with clear or muddy water.
  - It will survive immersion to IEC IP-67 standards and regain operation immediately after it resurfaces

- The shore portion of the LPARD system shall be designed and packaged such that:
  - It meets GPR002.
  - IP-67 standards can be met by the shore equipment in production.

For reference, the following standards have been described:

IEC IP-67 requires that the system:
- be completely protected against dust.
- be protected from the effects of immersion between 15cm and 1m.

GPR002 requires that the system:
- demonstrate reliable and normal functional operation in ambient lab temperatures of 15 °C to 30 °C, 10% to 80% RH, non-condensing.
- must tolerate a storage environment of 0 °C to +60 °C, 5% to 95% RH, non-condensing.
- should use electronic components rated for commercial temperature range (0 – 70 °C).
Shore Side

The shore side of the LPARD system mainly consists of the A1 Station box and the tripods with their attached sensors. Upon review of the shore design’s components, it has been determined that all parts used in the A1 shore box and all parts in the sensor packages meet the operational and storage environmental requirements of GPR002.

For the A1 Station box, IP-67 standards can be met by placing it inside a Pelican box, where it would be completely protected from dust. Additionally, the Pelican box is waterproof and could be used to survive the immersion requirement of IP-67. Wires will be going to the sensors from this box, so a CAT-6 waterproof cable could be used between these two assemblies. Cable specs: Model: Cat 5e/cat3/cat5/Cat6, Model Number: FTP, Price: US $0.25-$4.5 per Meter

A1 Station Casing:

![A1 Station Diagram]

The other portion of the shore side consists of the sensor/tripod packages. As for these, an acrylic box can be manufactured to fit around the body of the AZ/EL positioner to protect it from water, dust, and any other harmful particles. To completely seal in the body of the positioner, a tight O-ring seal can be made for any cables that lead to the A1 Station Box. For the head of the AZ/EL positioner, a smaller, fitted acrylic case will be created to shield the front end sensor boards from harmful particles. Unfortunately, with the ultrasonic method of range finding, this case will not be completely covering the head of the AZ/EL positioner in order to not interfere with the sensors mounted on its face. However, this case will adequately protect the head of the AZ/EL.

Sensor Casing Diagram:
**Boat Side**

Since we are not implementing the final packaging on the boat side, instead of making the implemented boat hardware waterproof and resistant to environmental damage such as rough weather, a possible boat packaging specification is provided.

Similar to the shore side, a Pelican box could be used to encase the boat circuitry. The Pelican box is specified as watertight, crushproof, and dustproof with an automatic pressure equalization valve and an O-ring seal. These specs easily satisfy the IP-67 requirement and would protect the circuitry against any wind less than 20km/h and splashing of clear or muddy water.

Each individual measurement device should also be considered in a packaging spec. For the accelerometer and the gyroscope, the type of packaging will not influence their measured data. The compass, however, can not be packaged in a material that will have significant current flowing through it, and it cannot be blocked off from the Earth’s magnetic field. The GPS needs to be able to obtain coordinates from satellites, so the packaging must allow these signals to penetrate. The XBee also needs to be able to penetrate the packaging, to receive its wireless signal. As an alternative, a hole in the packaging could be provided for the antenna, though water and dust sealing would then have to be addressed for this hole. Some solutions include using waterproof wire, potting it with epoxy, or adding a gasket to the wire hole. Special encasings could also be built for the electronics. These special encasings could also be used to protect the hardware from deep immersion, should it require it. Gaskets would be the quickest and easiest solution to sealing the hole, though. Waterproof fittings are also an ideal option, as their compression hubs create a waterproof seal around wires.
The system should also be able to operate any time of the day. Operating in the night is not a problem for the system. For the IR sensing it will actually become easier to track the boat, as there are much lower levels of ambient light at this time of day. For the ultrasound sensing, the amount of ambient light does not matter, and thus the time of day of operation is irrelevant to this sensing. Therefore, the system can operate in both the night and the day.

The system should also be able to operate in moderate rain or fog. We know that ultrasonic waves travel at 4 times faster in water than in air. This is because sound is based on the vibration of particles and the particles in gas are much farther than in water. The closer together the particles the better the efficiency. Therefore, if there is fog, the ultrasonic waves will travel farther. Rain is a different issue. The ultrasonic waves will travel more efficiently, but there is an issue of refraction. It could be possible for enough of the sound waves to refract away from the receiver and the receiver not get as high an amplitude. With random reflections, anything is really possible, but we believe that the system would still work with a degraded accuracy. For all the transmitted signal to be reflected and the receiver get none is highly improbable for shorter distances.

Based off of a published article: Environmental Effects on the Speed of Sound by Dennis A. Bohn (http://www.crazy.rane.com/pdf/ranenotes/Environmental_Effects_on_the_Speed_of_Sound.pdf), we can see that temperature and humidity have an effect on the ultrasonic ranging system. As the article states, the speed of sound at a given reference humidity is: $331 \times \sqrt{1+(t/273)}$, where $t$ is the temperature in Celsius. If we consider a temperature operating range of 0°C to 40°C, this results in a speed of sound change from 354.9 m/s to 331.45 m/s. A 23.45 m/s difference in the speed of sound will have a drastic effect on our range measurement. If our speed of sound used for ranging is off by 10 m/s, for example, a 10 meter range measurement will be off by an unacceptable 60cm.

For humidity, the percent change in velocity in the speed of sound with respect to humidity has been measured at different temperatures.
So, as can be seen from the graph, the percentage change in sound velocity changes most at the 40 centigrade mark and falls off from there. So, although the temperature is a much more important factor, both humidity and temperature can throw off our range.

Below is a separate analysis on temperature. The data points are showing how the speed of sound changes based on temperature:

<table>
<thead>
<tr>
<th>Temperature (degree C)</th>
<th>Speed of sound (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>351.96</td>
</tr>
<tr>
<td>30</td>
<td>349.08</td>
</tr>
<tr>
<td>25</td>
<td>346.18</td>
</tr>
<tr>
<td>20</td>
<td>343.26</td>
</tr>
<tr>
<td>15</td>
<td>340.31</td>
</tr>
<tr>
<td>10</td>
<td>337.33</td>
</tr>
<tr>
<td>5</td>
<td>334.33</td>
</tr>
<tr>
<td>0</td>
<td>331.3</td>
</tr>
<tr>
<td>-5</td>
<td>328.24</td>
</tr>
<tr>
<td>-10</td>
<td>325.16</td>
</tr>
<tr>
<td>-15</td>
<td>322.04</td>
</tr>
<tr>
<td>-20</td>
<td>318.89</td>
</tr>
<tr>
<td>-25</td>
<td>315.72</td>
</tr>
</tbody>
</table>

According to this data for every one degree of temperature change in Celsius we will have a change in the speed of sound of .6038 m/s. This means that for every degree in temperature change the position of the boat could change by .02 meters. In order to meet our position accuracy requirements we will have to keep track of temperature and change the speed of sound accordingly in our distance calculation. We are implementing a way of calibrating the system to initially figure out what the speed of sound is to avoid the error in temperature and humidity changes.

As for the IR aspect of the system, assuming there is an adequate packing around the sensors
and that water cannot get on the lens, the IR tracking should work properly with some degradation. Otherwise the IR will refract off the water on the lens and be completely inaccurate.

Components and subsystems have demonstrated to be safely storable in lab temperatures as we have seen these past weeks, with varying ambient temperatures. Similarly, subsystems have been operating in these same lab ambient temperatures. For temperature range ratings, please refer to the manufacturability vendor matrix in the manufacturability analysis. This matrix contains links to the vendors’ websites, which often contain the specifications of their parts.
This document analyzes the existence of possible hazardous materials in the LPARD system design. Hazardous materials should be avoided in the project design according to GPR004. If use of a certain hazardous materials is essential and there is no non-hazardous alternative, the use of hazardous materials must comply with the attached Lafayette College Chemical Hygiene Plan (LCCHP). Please refer to the LCCHP for the list of hazardous substances.

The LPARD system design does not require the use of and does not utilize any hazardous materials; therefore, a Material Data Safety Sheet is not required. After careful analysis of the Bill of Materials, all of the components used in the implementation of the design are RoHS compliant, do not contain any of the materials listed in the Lafayette College Chemical Hygiene Plan, and do not contain any warnings indicating the use of possible hazardous materials. As for this analysis, please refer to the manufacturability vendor matrix in the manufacturability analysis. This matrix contains each component used in our implementation as well as links to the vendors’ websites, which often contain all the specifications of their parts. Here we can see that the parts are RoHS compliant and each part has been checked against the list of hazardous materials. Furthermore, no batteries were used in the design so there is no concern with NiCd or Lead battery usage.
Lifetime Capacity Analysis per R002:

Date: 4/30/12
Engineer: Michael Rupolo

Purpose:
The focus of the document is to ensure that the SCADA has sufficient capacity for retaining data records over the lifetime of the system.

Calculation

The overhead required for an Excel 2007 CSV workbook is 1382 bytes.

\[ initial = 1382 \text{ bytes} \]

We record data at the maximum specified rate of 60 times per minute therefore once per second (this can be configured at a lesser variable rate but it will be done so uniformly across all logs). The workbook will be updated by row with a maximum of 6 columns per second. An example of one such row would be:

<table>
<thead>
<tr>
<th>Time</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Heading</th>
<th>Fault</th>
</tr>
</thead>
</table>

We will also assume that each of these updated fields is represented by a double (64 bits or 8 bytes).

\[ rate_{update} = \frac{data}{min} = 60 \frac{updates}{minute} \times 6 \frac{fields}{update} \times 8 \frac{bytes}{field} = 2880 \text{ bytes/min} \]

Assuming that the lifetime of the system is 5 years, that we conduct one session per day and that we operate for a maximum of 8 hours per session we get:

\[ number_{session} = ceil\left(5 \text{ years} \times 365.25 \frac{days}{year}\right) = ceil(1826.25) = 1827 \text{ sessions} \]

\[ time_{lifetime} = 5 \text{ years} \times 365.25 \frac{days}{year} \times 24 \frac{hours}{day} \times 60 \frac{mins}{hour} = 2629800 \text{ mins} \]

Combining all these calculations we get:

\[ data_{log} = (initial \times number_{session}) + (time_{lifetime} \times rate_{update}) \]

\[ = (1382 \text{ bytes} \times 1827) + \left(2629800 \text{ mins} \times 2880 \frac{bytes}{min}\right) \]

\[ = 7573824000 \text{ bytes} \]
We will assume a maximum of 20 sensor logs required. We require one system log, two AZEL raw sensor logs, and three position logs (two initial and one fused) as well as having support on the ardupilot for 14 sensors. This gives us a total of 20 sensors resulting in:

\[
data_{total} = data_{log} \times 20 = 7573824000 \text{ bytes} \times 20 = 151476480000 \text{ bytes}
\]

\[
= 141.073465 \text{ gigabytes}
\]

The lifetime capacity analysis has calculated the maximum amount of data the system can potentially store to be 141.073465 gigabytes. Commercial laptops are more than capable of maintaining a capacity of that magnitude.

Solution: When taking into account the maximum amount of data to have the potential of 141.073465 gigabytes it is recommended that the operational laptop have an onboard hard drive with a capacity of 500 gigabytes. 250 gigabytes is possible but is not the recommended solution. An alternative recommendation would be to archive the data logs to an external drive every 6 months to avoid an overflow of data storage.

Conclusion:

The requirement specified by R002-10 is confirmed by the calculation and solution in this analysis to be fully supported by the SCADA.
LPARD Power Analysis per R011

Date: 3/13/12  
Engineer: Edward Epple III

This document outlines and analyzes the power requirements of the LPARD system on both the shore and boat sides. Included are power budgets for the system's components and power flow diagrams.

Proposal: AC power mode, or “maintenance mode”, will be designed and implemented, while power input independence will be analyzed and specified for future years’ design teams. It is assumed that the specified boat motors will not be operational during AC power mode.

**Shore Power Budget**  
Note: The shore-side Xbee draws power from the specified shore laptop.

The following chart outlines the power requirements for the subsystems of the shore station and contains maximum or worst-case values.:

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Voltage(V)</th>
<th>Current(A)</th>
<th>Power(watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ/EL Positioner</td>
<td>18</td>
<td>1.00</td>
<td>18.0</td>
</tr>
<tr>
<td>Ultrasonic Range Shore</td>
<td>18</td>
<td>0.01</td>
<td>0.18</td>
</tr>
<tr>
<td>IR Angle Shore</td>
<td>18</td>
<td>0.02</td>
<td>0.36</td>
</tr>
</tbody>
</table>

**Boat Power Budget**  
Note: GPS, magnetometer, Xbee, Xbee shield, and IMU shield all draw power from the ardupilot.

The following chart outlines the power requirements for the subsystems of the boat:

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Voltage(V)</th>
<th>Current(A)</th>
<th>Power(watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ardupilot</td>
<td>5.0</td>
<td>0.2</td>
<td>1.0</td>
</tr>
<tr>
<td>GPS</td>
<td>3.3</td>
<td>0.048</td>
<td>0.1584</td>
</tr>
<tr>
<td>Magnetometer</td>
<td>2.5</td>
<td>0.0001</td>
<td>0.00025</td>
</tr>
<tr>
<td>Visual Beacon</td>
<td>5.0</td>
<td>1.0</td>
<td>5.0</td>
</tr>
<tr>
<td>Audio Beacon</td>
<td>18</td>
<td>0.01</td>
<td>0.18</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>9.0</td>
<td>1.0</td>
<td>9.0</td>
</tr>
</tbody>
</table>
Shore Power Flow
Below is the top-most level diagram for shore power, which covers both the implementation and power input independence specification aspects of the design. Differentiation of implementation and power input independence specification takes place in the A1 station box and will be shown in further diagrams.
Shore Power Flow (Implementation)

For implementation, we will provide power through the AC mains, in a permanent “maintenance mode” state. In this state tracking and data fusion is still possible, though motors will not be functional and the system should not move. Design for possible shore power input independence will be specified in the next section. Below are the diagrams of the power flow diagram and the corresponding A1 Station box based on the design implementation.
Shore Power Flow (Power Input Independence Spec)

This is a possible design for power input independence on the shore side, which will include batteries as well as a jack for an AC adapter. In this way, the system can operate temporarily on battery power. A switch will exist to route either battery or AC power, depending on the situation. It would then require only a flip of a switch to change between modes. Flipping the switch to the AC side will activate "maintenance mode", in which the boat could be operated for extended periods of time without draining the batteries. A possible top level shore system diagram and shore power flow diagram are below.

Ports:
- J1,2: 4pos MATE-N-LOK Cap
- J4: USB cable cut
- J5: AC Main jack

Wire Specs:
- W1: USB_A to USB mini Wire
- W2: USB_A to USB_B
- W3,4: 1" 6wire Ribbon Cable
- W5,6: 1" 4wire Ribbon Cable
- W7: 18V DC
- P1,2: 6pos Micro-Match Connector
- P3,4: 4pos 0.100" Connector
  - Digikey: W2902-ND

U13 2-Port USB Hub:
- J1-2: USB Inputs
- U12 Xbee USB Adapter:
- J1: USBmini jack
- J2: 2 Mounted Pin Headers
- U11 Xbee-Pro ZB:
- P1: 2 Row Pins
- J1: RPSMA Connector

U10 Interface Board:
- J1-4: 4pin 0.100" Header

U15 Packaging: Pelican Box

---

![Diagram Image]
Shore Power Flow
Power Input Independence

Wall Power 120VAC
Wall Adapter (Vout = 18 V; Imax = 2.7 A)

Batteries ~18V
Maintenance Switch 18V
P&D Interface ~1A 18V
Expansion Board 18V
AZ/EL Positioner (18V; 1A)

Sensor Board 18V

5V
**Boat Power Flow (Implementation)**

For implementation, we will provide power through the AC mains. Design for possible boat power input independence will be specified in the next section. Below are the diagrams of the power flow diagram and the corresponding boat top-level diagram based on the design implementation. LM317s are included in the design to provide the additional voltages needed by the boat system.
Boat Power Flow (Implementation) (continued)

Beacon-Power PCB contains voltage regulators, an Arduino Uno, and the audio/visual beacons.
Boat Power Flow (Power Input Independence Spec)
Again, it is assumed that the batteries for the motor are separate, and not included in the design for power input independence of the rest of the boat. A possible top level boat system diagram and boat power flow diagram are below.
The "Power" block contains a maintenance power switch, batteries for the motor, and batteries for the rest of the boat. Again, this is a possible design for future implementation.
Manufacturability Analysis per GPR008

Date: 3/16/12  
Engineer: Edward Eppe III

The LPARD manufacturability analysis aims to prove that the system design is suitable to be manufactured in quantities of at least 1000 units per year. To demonstrate this, it will be shown that each component of the system is available from at least two vendors. In addition, it will be shown that the design considers future industry trends, so that there will not be necessary to significantly change the design because of obsolete components. Furthermore, the tolerances of critical components will be considered such that we can obtain at least a 99% yield based on the 1000 parts per year requirement.

A major trend in industry these days is the reduction of negative impact on the environment. Many industries have begun designing and manufacturing new products which comply with certain standards of eco-friendliness. An important aspect of this trend is the restriction of hazardous substances, which restricts the use of hazardous substances in the manufacture of various electronics. After review of the vendor matrix, it can be seen that all of the parts are RoHS compliant, a standard that does not indicate any significant changes anytime soon, and that a large majority of the vendors’ available parts are also compliant.

Another trend in current electronics is the production of remote-controlled vehicles. Beginning in the 60’s, RC vehicles have been extremely popular with consumers. Later we also saw the development of the RC boat and RC helicopter, along with many variations of each. On the topmost level, the system is an enhanced version of a remote-controlled boat. This boat will be able to move via coordinates entered into the software or by some RC controller. Hydrologists have previously used cableway and tethered boats to take measurements on the water, and have recently started taking a few types of measurements via RC boats. This is a favorable trend to our design. Additionally the main boat component in the design, the Ardupilot, shows trends of technological improvement and increasing popularity. Over the past three years, the Ardupilot has been upgraded with newer technology and continues to be developed for the use of RC helicopters, boats, and cars.

Tolerances of components have also been considered in the design. For the frontend of the sensors, the audio components that need to be tolerant are R26, R23, and R24. These resistors represent the first stage gain of the received signal. If the signal is too high, we will trigger off of noise and if it is too low, we won’t receive our transmitted signal at all. The resistors in the backend board, that serve the same function for the second stage of amplification, need to be tolerant for similar reasons (R23, R24, R26). For our final implementation, our circuit does not deal with noise problems as well as it should. Due to our inability to filter out the 40 kHz received noise, our circuits trigger off of noise at just past 10 meters. An example of this amplified noise problem can be seen below, where, with no input, we get 400 mV spikes after amplification:
To get range out to ten meters, we use trim pots for our second stage amplification resistors. The first stage amplification resistors, therefore, need not be extremely tolerant in our implementation. As long as these resistors are within 5%, we can adjust the overall gain using the second stage trimpot. In order to get our range, we need to finely tune the feedback trimpot so as to achieve a precise value. Therefore, using this adjust-by-one-resistor technique, the trimpots are the only part of our circuit that need to be very tolerant. A tolerance of 0.5% is needed to achieve the 10 meter range without using additional noise canceling filters. With noise canceling filters, we believe a much greater range is possible because there will not be 400 mV swings on the output when nothing should be received.

In terms of measurement accuracy, as long as all the resistors and capacitors are within 5% tolerance, degraded accuracy will not occur due to tolerance. The tolerance of the trimpot affects the overall range and not necessarily accuracy. If the trimpot is set too high, the circuit will always receive and will always trigger off of the minimum range distance of ~1m. The oscillator for the transmitter, if at a 5% tolerance, will be at either 38 or 42 kHz, which is at the boundary condition of what we can receive, so it is very likely that the circuit will not work at all. So, specifying a 2.5% tolerance or less for the oscillator ensures functionality of the circuit.

The accuracy of the ranges for the ultrasound sensors are based on testing at 5 and 10 meters. The data for each of these is shown below:

**Five Meter Range Testing Results**

The accuracy of the ranges for the ultrasound sensors are based on testing at 5 and 10
meters. The data for each of these is shown below:

**Five Meter Range Testing Results**

![5 Meter Test Graph](image)

Standard Deviation = 0.05957

**Ten Meter Range Testing Results**

![10 Meter Test Graph](image)

Standard Deviation = 0.07813

Based on this data we are confident that up to 10 meters we will be able to get a range within our error requirement of 14cm. We tested past 10 meters but were unable to get consistent enough data to call the system working. Thus, we work only up to 10 meters accurately.
These tolerances ensure that the system is reliable and that the system requirements are met with at least a 99% yield in manufacturing. Below is a matrix containing all purchased materials used in the implementation of our design this year with at least two vendors from which these parts are available. Two vendor web links have also been provided for each component for quick and easy purchase should the following years’ groups need to buy components of the design. This table proves that each component is reliably available, which shows that industry trends indicate the preferred use of these parts.
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Quantity</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CAD 2000P</td>
<td>50%</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>CAD 1000P</td>
<td>50%</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IC AUTO PIN-AND GATE</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>IC DRIVER PMD</td>
<td>3.2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ADAP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>IC 12-Pin DIP</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>IC 12-Pin DIP</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>IC 12-Pin DIP</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>IC 12-Pin DIP</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>IC 12-Pin DIP</td>
<td>0.0005</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>IC 12-Pin DIP</td>
<td>0.0001</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>IC 12-Pin DIP</td>
<td>0.00005</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>IC 12-Pin DIP</td>
<td>0.00001</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>IC 12-Pin DIP</td>
<td>0.000005</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>ADP Desktop Class</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>
API Documentation

Date: 5/7/12
Engineer: Michael Rupolo

Purpose:

The focus of the document is to ensure that:

1. LPARD has a fully documented software API and SDK that allows an applications programmer to write software that uses interfaces and functions of SCADA system(R009-1)
2. Software support for LPARD-TDF-2012 uses API to access SCADA functions(R009-2)
3. SDK includes compilers, linkers, libraries, include files, and utilities as well as developer level documentation(R009-3)
4. The complete SDK including API documentation and application source under configuration control shall be delivered to or linked by the project website(R009-4)

Solution:

1. Fully Documented Lpard – LPARD API is fully documented by LPARD_2012_refman.pdf. LPARD_2012_SDK.zip contains Python 2.7, PyQt 4.9.1, PySerial 2.5, and py2exe 0.6.9. Using the SDK package in conjunction with the supported API interfaces our functions it is possible for: added functionality to the software controller, expansion of the current GUI application, or future development of external applications built on top of the existing functionality and interfaces.

2. API to access SCADA – GUI application accesses uses API functions and interfaces to instantiate controller in order to bridge the gap between user input and hardware control. Confirmed by team member David Salter.

3. SDK – LPARD_2012_SDK.zip contains Python 2.7, PyQt 4.9.1, PySerial 2.5, and py2exe 0.6.9. Additionally contains setup installation instructions in README.txt

4. Delivery to website – LPARD_SDK.zip, LPARD_2012-refman.pdf, and LPARD_TDF_2012_SOFTWARE.zip are uploaded to the project website.

Conclusion:

All four requirements specified by R009 are confirmed by the solution and its external references.
Coding Verification Documentation

Date: 5/7/12
Engineer: Michael Rupolo

Purpose:

The focus of the document is to ensure that SCADA software is fully documented with source code, design, and end-user documentation (R002-3), and that SCADA application software was written in conformance with the LPARD-TDF-2012 API using the delivered SDK(R002-5)

Solution:


Design - The design is affirmed as fully documented by class hierarchies shown in reference document LPARD_2012_refman.pdf generated by doxygen.


Conformance with API – The SDK contains Python 2.7, PyQt 4.9.1, PySerial 2.5, and py2exe 0.6.9. All code was written using only libraries supported by these packages. In addition, python is IDE independent and can be edited in a simple text editor so no additional development plugins are required.

Conclusion:

All three requirements specified by R002 -3 are confirmed by the solution as verified by external reference documents.

The requirement specified by R002-5 is confirmed by the solution in this analysis.
Data Storage Format Analysis per R002:

Date: 4/26/12
Engineer: Michael Rupolo

Purpose:

The focus of the document is to ensure that the data logged by our system is stored in a portable, non-proprietary format readily useable by common data analysis tools.

Solution:

Non-proprietary - All of the data logged by our system is stored using CSV (comma separated values) formatted files which is a non-standard conventional format where a comma is used to separate each value.

Portability - CSV is a common, relatively simple file format that is widely supported by consumer, business, and scientific applications. Among its most common uses is moving tabular data between programs that natively operate on incompatible (often proprietary and/or undocumented) formats. This works because so many programs support some variation of CSV at least as an alternative import/export format.

Usability – Open Source Platforms which support the format are OpenOffice.org Calc, Lotus 1-2-3, Gnumeric, KSpread. Proprietary Platforms which support the format are Microsoft Excel and Corel Quattro Pro.

Conclusion:

All three requirements specified by R002 -12 are confirmed by the solution in this analysis to be non-proprietary, portable, and readily usable.
Final Budget Report

Out of the 3,000 dollars that we were allocated to complete our portion of the three year senior design autonomous boat project, we spent 2,453.51. This means that we were under budget by 546.49 dollars. After spring break we had spent 1,468.66 dollars. Once we returned from spring break I reallocated the left over money to the remaining parts of the project that had not been purchased. After allocation I was able to come up with a final estimate of about 750 dollars left at the end of the project. As it turns out we actually spent $203.50 more than my estimate. This excess spending came from a couple of broken parts that had to be replaced and from ordering three extra pcb boards that I had not accounted for. In the end, we were able to stay in budget, though, by 546.49 dollars. Below is the list of purchases and how much of the budget we spent.
## Purchases

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>WRL-10419</td>
<td>Xbee Pro 50mW RPSMA - Series 2</td>
<td>$44.95</td>
<td>$89.90</td>
</tr>
<tr>
<td>1</td>
<td>DEV-11061</td>
<td>Arduino Mega 2560 R3</td>
<td>$58.95</td>
<td>$58.95</td>
</tr>
<tr>
<td>1</td>
<td>RTL-10424</td>
<td>USB Cable-A-to-miniB 6 Foot</td>
<td>$4.95</td>
<td>$4.95</td>
</tr>
</tbody>
</table>

Shipping Fees: $9.80

Grand Total: $163.60

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C-200-WRL-08687</td>
<td>Xbee Explorer USB</td>
<td>$24.95</td>
<td>$24.95</td>
</tr>
</tbody>
</table>

Shipping Fees: $7.99

Grand Total: $32.94

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>WRL-00145</td>
<td>2.4GHz Duck Antenna RP-SMA</td>
<td>$7.95</td>
<td>$15.90</td>
</tr>
</tbody>
</table>

Shipping Fees: $4.03

Grand Total: $19.93

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EXPCK1</td>
<td>Customizable Ardupilot Mega E Set</td>
<td>$354.95</td>
<td>$354.95</td>
</tr>
</tbody>
</table>

Shipping Fees: $19.51

Grand Total: $374.46

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NA</td>
<td>XtreamBee Board (No Xbee included)</td>
<td>$24.95</td>
<td>$24.95</td>
</tr>
<tr>
<td>1</td>
<td>NA</td>
<td>Xbee-Oil Pan connector cable</td>
<td>$1.50</td>
<td>$1.50</td>
</tr>
</tbody>
</table>

Shipping Fees: $9.80

Grand Total: $36.25

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>296-9518-5-ND</td>
<td>IC QUAD HALF-H DRVR 16-DIP</td>
<td>$2.50</td>
<td>$10.00</td>
</tr>
</tbody>
</table>

Shipping Fees: $9.53

Grand Total: $19.53
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>475-2844-1-ND</td>
<td>Emitter IR 850NM SMD</td>
<td>$9.54</td>
<td>$38.16</td>
</tr>
<tr>
<td>10</td>
<td>475-1200-ND</td>
<td>Emitter IR 850NM 5MM Radial</td>
<td>$0.48</td>
<td>$4.84</td>
</tr>
<tr>
<td>1</td>
<td>1050-1024-ND</td>
<td>Arduino Uno</td>
<td>$26.56</td>
<td>$26.56</td>
</tr>
<tr>
<td>25</td>
<td>296-14236-ND</td>
<td>IC DIV-BY-N Counter Preset 16 DIP</td>
<td>$0.41</td>
<td>$10.25</td>
</tr>
<tr>
<td>25</td>
<td>296-2417-1-ND</td>
<td>IC OPAMP GP 10MHZ Quad</td>
<td>$2.32</td>
<td>$58</td>
</tr>
<tr>
<td>25</td>
<td>811-1299-ND</td>
<td>Inductor Radial 2.2MH 0.11A</td>
<td>$0.58</td>
<td>$14.5</td>
</tr>
<tr>
<td>100</td>
<td>P2.0KACT-ND</td>
<td>RES 2.0K OHM 1/8W 5% 0805 SMD</td>
<td>$0.01</td>
<td>$1</td>
</tr>
<tr>
<td>250</td>
<td>P100KACT-ND</td>
<td>RES 100K OHM 1/8W 5% 0805 SMD</td>
<td>$0.01</td>
<td>$2.5</td>
</tr>
<tr>
<td>250</td>
<td>P1.0KACT-ND</td>
<td>RES 1.0K OHM 1/8W 5% 0805 SMD</td>
<td>$0.01</td>
<td>$2.5</td>
</tr>
<tr>
<td>50</td>
<td>490-1619-1-ND</td>
<td>CAP CER 680PF 50V 5% NP0 0805</td>
<td>$0.07</td>
<td>$3.5</td>
</tr>
<tr>
<td>10</td>
<td>497-1576-1-ND</td>
<td>IC Comparator HS Dual 14-SOIC</td>
<td>$0.79</td>
<td>$7.9</td>
</tr>
<tr>
<td>25</td>
<td>296-14095-1-ND</td>
<td>IC DIV-BY-N Counter Preset 16 DIP</td>
<td>$0.4</td>
<td>$10</td>
</tr>
</tbody>
</table>

Shipping Fees: $9.53
Grand Total: $189.24

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>718-QP5.8-6-TO5</td>
<td>Photodiodes Quadrant 1.19mm</td>
<td>$18.4</td>
<td>$73.6</td>
</tr>
<tr>
<td>25</td>
<td>815-ACO-4-EK</td>
<td>Stand Clock Oscillator 4MHz 5V</td>
<td>$1.44</td>
<td>$36</td>
</tr>
<tr>
<td>77-</td>
<td>VJ0805Y273JXACB</td>
<td>Multilayer Ceramic Capacitors .027uF</td>
<td>$0.14</td>
<td>$7</td>
</tr>
<tr>
<td>10</td>
<td>695-CSX750FCC-4</td>
<td>Stand Clock Oscillator 4MHz 5V</td>
<td>$2.6</td>
<td>$26</td>
</tr>
</tbody>
</table>

Shipping Fees: $7.01
Grand Total: $149.61

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>PT-LH003M</td>
<td>Metal M12 Lens Holder, 20mm HS</td>
<td>$4.75</td>
<td>$9.5</td>
</tr>
<tr>
<td>2</td>
<td>PT-1220</td>
<td>12mm, F2.0 Board Lens</td>
<td>$5.27</td>
<td>$10.54</td>
</tr>
<tr>
<td>2</td>
<td>PT-0420</td>
<td>4mm, F2.0 Board Lens</td>
<td>$3.59</td>
<td>$7.18</td>
</tr>
<tr>
<td>2</td>
<td>PT-0820</td>
<td>8mm, F2.0 Board Lens</td>
<td>$3.83</td>
<td>$7.66</td>
</tr>
</tbody>
</table>

Shipping Fees: $13.18
Grand Total: $48.06

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>B004RBX0GO</td>
<td>Ravelli Professional 65&quot; Ball Head Tripod</td>
<td>$43.89</td>
<td>$87.78</td>
</tr>
</tbody>
</table>

Shipping Fees: $0.00
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>LEN922</td>
<td>Varifocal 9-22mm Board Mount CCTV</td>
<td>$19.99</td>
<td>$39.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td>$30.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td>$69.98</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3004</td>
<td>Ultrasonic Transcuer Pair 40KHz</td>
<td>$6.14</td>
<td>$61.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td>$40.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td>$96.26</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I2CKIT1</td>
<td>Starter Kit with cables..I2C board</td>
<td>$89.95</td>
<td>$89.95</td>
</tr>
<tr>
<td>1</td>
<td>I2C2PC</td>
<td>i2C-2-PC RS232 USB Host Adapter</td>
<td>$69.95</td>
<td>$69.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td>$10.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td>$169.90</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1300</td>
<td>USB AVR Programmer</td>
<td>19.95</td>
<td>59.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td>$13.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td>$73.80</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>uos</td>
<td>Board 4in x 1.6in 2 Layers</td>
<td>33</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td>$37.80</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td>$103.80</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>512-3059</td>
<td>Universal Mate-n-lok (Cap)</td>
<td>0.52</td>
<td>5.2</td>
</tr>
<tr>
<td>10</td>
<td>512-3058</td>
<td>Universal Mate-n-lok Plug</td>
<td>0.41</td>
<td>4.1</td>
</tr>
<tr>
<td>40</td>
<td>350550-3</td>
<td>Universal Mate-N-LOK Socket</td>
<td>0.17</td>
<td>6.8</td>
</tr>
<tr>
<td>40</td>
<td>350547-7</td>
<td>Universal Mate-N-LOK 20-14AWG</td>
<td>0.49</td>
<td>19.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td>$6.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td>$42.68</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>445-1330-1-ND</td>
<td>CAP CER 1000PF 50V 5% NP0 0805</td>
<td>$0.06</td>
<td>$6.4</td>
</tr>
<tr>
<td>50</td>
<td>490-1627-1-ND</td>
<td>CAP CER 2000PF 50V 5% NP0 0805</td>
<td>$0.07</td>
<td>$3.26</td>
</tr>
<tr>
<td>2</td>
<td>62-1175-ND</td>
<td>Adapter Desktop Class I 50W 18V</td>
<td>$26.22</td>
<td>$52.44</td>
</tr>
<tr>
<td>10</td>
<td>A25216-ND</td>
<td>Conn Header 4Pos Rtag .250 Tin</td>
<td>$2.91</td>
<td>$29.09</td>
</tr>
<tr>
<td>25</td>
<td>296-1452-5-ND</td>
<td>IC QUAD 2IN POS-NAND GATE 14</td>
<td>$0.42</td>
<td>$10.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Shipping Fees</strong></td>
<td></td>
<td>$9.53</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Grand Total:</strong></td>
<td></td>
<td>$111.22</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>497-2937-1-ND</td>
<td>IC DRVR P/P 4CH W/DIODES 20-SOIC</td>
<td>3.251</td>
<td>32.51</td>
</tr>
<tr>
<td>100</td>
<td>P240ACT-ND</td>
<td>RES 240 OHM 1/8W 5%</td>
<td>0.0123</td>
<td>1.23</td>
</tr>
<tr>
<td>100</td>
<td>P1.5KACT-ND</td>
<td>RES 1.5K OHM 1/8W 5%</td>
<td>0.0123</td>
<td>1.23</td>
</tr>
<tr>
<td>100</td>
<td>P715CCT-ND</td>
<td>RES 715 OHM 1/8W 1% 0805</td>
<td>0.021</td>
<td>2.1</td>
</tr>
<tr>
<td>100</td>
<td>RHM100KCRCT-ND</td>
<td>RES 100K OHM 1/8W 1%</td>
<td>0.0245</td>
<td>2.45</td>
</tr>
<tr>
<td>100</td>
<td>RHM2.00KCCT-ND</td>
<td>RES 2.00K OHM 1/8W 1%</td>
<td>0.03</td>
<td>2.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Shipping Fees</strong></td>
<td></td>
<td>$9.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Discounts:</strong></td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Grand Total:</strong></td>
<td></td>
<td>$51.71</td>
</tr>
</tbody>
</table>

### Table 3

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NA</td>
<td>LOWES Bubble Level</td>
<td>3.47</td>
<td>6.94</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TAX</strong></td>
<td></td>
<td>$0.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Grand Total:</strong></td>
<td></td>
<td>$7.43</td>
</tr>
</tbody>
</table>

### Table 4

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>P2.0KDACT-ND</td>
<td>RES 2.0K OHM 1/8W .1% 0805 SMD</td>
<td>0.2</td>
<td>9.98</td>
</tr>
<tr>
<td>50</td>
<td>P100DACT-ND</td>
<td>RES 100K OHM 1/8W .1% 0805 SMD</td>
<td>0.2</td>
<td>9.98</td>
</tr>
<tr>
<td>100</td>
<td>P5.1ACT-ND</td>
<td>RES 5.1K OHM 1/8W 5% SMD</td>
<td>0.01</td>
<td>1.23</td>
</tr>
<tr>
<td>50</td>
<td>P1.2KACT-ND</td>
<td>RES 1.2K OHM 5%</td>
<td>0.02</td>
<td>0.81</td>
</tr>
<tr>
<td>50</td>
<td>P1.6KACT-ND</td>
<td>RES 1.6K OHM 5%</td>
<td>0.02</td>
<td>0.81</td>
</tr>
<tr>
<td>50</td>
<td>P12.4KCCT-ND</td>
<td>RES 12.4K OHM 1%</td>
<td>0.03</td>
<td>1.38</td>
</tr>
<tr>
<td>25</td>
<td>811-1299-ND</td>
<td>Inductor Radial 2.2MH .11A</td>
<td>0.58</td>
<td>14.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Shipping Fees</strong></td>
<td></td>
<td>$9.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Grand Total:</strong></td>
<td></td>
<td>$48.25</td>
</tr>
<tr>
<td>Quantity</td>
<td>Vendor Part</td>
<td>Description</td>
<td>Unit Price</td>
<td>Total Price</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------------------------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>NA</td>
<td>IEEE Membership</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAX</td>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td></td>
<td>$16.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>NA</td>
<td>Distance Sensors - Maxbotix UT MB1100</td>
<td>4.95</td>
<td>39.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAX</td>
<td></td>
<td>9.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td></td>
<td>$49.56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>uos</td>
<td>Board 4in x 1.6in 2 Layers</td>
<td>33</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td></td>
<td>18.53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td></td>
<td>$117.53</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>uos</td>
<td>Backend AZEL Board</td>
<td>33</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td></td>
<td>38.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td></td>
<td>$137.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>93505A102</td>
<td>Aluminum Hex Standoff 3/16&quot;, 3/8&quot;</td>
<td>0.36</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>91772A108</td>
<td>18-8 SS Pan Head Phillips Screw 4-40</td>
<td>4.3</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td></td>
<td>$22.30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>boat</td>
<td>Boat Board</td>
<td>33</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shipping Fees</td>
<td></td>
<td>38.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grand Total:</td>
<td></td>
<td>$104.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Vendor Part</th>
<th>Description</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>718-QP5.8-6-TO5</td>
<td>Photodiodes Quadrant 1.19x1.19mm</td>
<td>18.4</td>
<td>73.6</td>
</tr>
<tr>
<td>50</td>
<td>667-ERA-6AEB202V</td>
<td>Thin Film Resistors - SMD 0805 1/8W 2K</td>
<td>0.21</td>
<td>10.5</td>
</tr>
<tr>
<td></td>
<td>Part Number</td>
<td>Description</td>
<td>Quantity</td>
<td>Price</td>
</tr>
<tr>
<td>----</td>
<td>-----------------</td>
<td>--------------------------------------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>50</td>
<td>667-ERA-6AEB104V</td>
<td>Thin Film Resistors - SMD 0805 1/8W 100K</td>
<td></td>
<td>0.21</td>
</tr>
<tr>
<td>5</td>
<td>667-ERX-2FJS1R0E</td>
<td>Through Hole Metal Film Resistor 2W 1Ohm</td>
<td></td>
<td>0.34</td>
</tr>
<tr>
<td>5</td>
<td>667-ERX-2FJS2R0E</td>
<td>Through Hole Metal Film Resistor 2W 2Ohm</td>
<td></td>
<td>0.34</td>
</tr>
<tr>
<td>5</td>
<td>262-3.0-RC</td>
<td>Metal Oxide Resistors 3Ohms 5%</td>
<td></td>
<td>0.22</td>
</tr>
<tr>
<td>5</td>
<td>71-CPF2-F-5</td>
<td>Through Hole 2watts 5Ohms 1%</td>
<td></td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Shipping Fees</strong></td>
<td></td>
<td>$8.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Grand Total</strong></td>
<td></td>
<td>$110.61</td>
</tr>
</tbody>
</table>
Future Budget Analysis

In order for future groups to finish this project, in the worst case, a total of 5000 dollars should be allocated. This is based on the following purchases and failures:

1. 4 motors are needed; purchasing 6 would be a good choice. Based on purchasing motors from Cornwall Model Boats this would cost 216.36 dollars
2. Extra parts for the motors will have to be purchased, such as mounts and cables. These parts were estimated to cost 200 dollars
3. 1 Emotion Edge Kayak was researched and deemed a good boat for this project. This costs 400 dollars
4. Based on expensive boat batteries, the cost for batteries could be up to 624 dollars
5. Water proofing methods, such as water proof transducers, materials, and pelican cases, are estimated to cost up to 600 dollars
6. Based on the number of parts we had to replace this year and how much that cost, we predict future groups spending about 450 on replacing parts

This total comes up to 2490.36 dollars. If for some reason the system we built this year were to completely fail and have to be replace the total will be 2490.36 plus what we paid this year. This will bring the total to 4943.87 dollars or 5000 dollars rounded up. Keep in mind this total is based on worse case scenarios for each part and this year’s system failing.
User’s Manual

LPARD-TDF-2012
ECE492 - Spring 2012
Latest Revision Date: 5/7/2012
# Table of Contents

**Contents**

Table of Contents ....................................................................................................................... 2  
Introduction ................................................................................................................................. 3  
Overview................................................................................................................................... 4  
Package Contents...................................................................................................................... 4  
Getting Started........................................................................................................................... 5  
  Setup...................................................................................................................................... 5  
    1. Setup base station ....................................................................................................... 5  
    2. Starting a new Session .............................................................................................. 6  
    3. Using the GUI .............................................................................................................. 7  
Sensors .................................................................................................................................. 9  
  Wiring Connections: ...........................................................................................................10  
  Ultrasound Calibration Setup: .........................................................................................10  
  IR Calibration Setup: ......................................................................................................10  
  Origin Setup and Ranging: .............................................................................................10  
Safety Guide .............................................................................................................................12  
  Purpose.................................................................................................................................12  
  Safety Procedures ................................................................................................................12  
    Electrical: .......................................................................................................................12  
    Environmental: ..............................................................................................................12  
    Attire: .............................................................................................................................13  
    Mechanical: .....................................................................................................................13  
    Transportation: ..............................................................................................................13  
    Other: ..............................................................................................................................13  
Troubleshooting ........................................................................................................................14  
Glossary of Terms.....................................................................................................................15  
Service & Warranty Information .............................................................................................16
Introduction

The freshwater resources of the planet Earth are essential to quality of life. Accurate and comprehensive measurements of these resources give us insight into the sustainability of water, as they help us manage and track pollution. Portable and agile wide-area monitoring of water quality in hard to reach areas will aid the enforcement of environmental laws as well as the study of water conservation. The Lafayette Programmable Autonomous River Droid (LPARD) is an aquatic robotic system for facilitating hydrology measurements in small rivers and creeks.

LPARD is an autonomous, floating platform that stabilizes itself in fast moving, shallow rivers, while carrying a hydrology and environmental measurement payload achieving absolute positional and heading motion control, automatically compensating for river current, wind, and other disturbances. This system is designed to allow two people located on land to quickly and remotely perform accurate and comprehensive hydrology and environmental measurements that otherwise would take two or more people to perform.

Measurements will be able to be made day or night, at precise, fixed locations on the surface of fast moving shallow water without requiring alteration of the river, nor any anchor to the river bottom, nor a taut line across the river, nor a suspended platform, nor a tether to a fixed structure, nor wading, nor human occupied boat. Supported measurements include:

- Depth Measurements
- Turbidity
- Imaging
- Dissolved gases
- pH
- Temperature
- Acoustic Doppler Current Profiling (ADCP)
Overview

The LPARD system consists of a stationary shore station and a remote controlled boat platform. The shore station is divided into a controller laptop running Windows 7 with two separate sensor arrays for triangulation and navigation, an RC controller is also provided for manual override control. Each tower consists of a sound based triangulation system and a light based triangulation system for added precision.

Package Contents

The following items are included in the LPARD package:

- 2 x Tripod Mounted Sensor Arrays
- 1 x LPARD Autonomous River Droid
- 1 x Shore Interface Box
- 2 x 75ft CAT6 cables
- 1 x Folding table
- 1 x 10m or greater tape measure with open reel
- 1 x LPARD.exe
Getting Started

Setup

1. Setup base station
   a. Setup up table in a clear area near the water
   b. Put laptop and shore hardware on table
   c. Connect Shore Hardware into USB port of laptop
   d. Setup tripods equidistant from the shore station with a clear view of the shore at least 1m away from shore
   e. Connect CAT6 cable between the shore hardware and the sensors
   f. Measure distance between tripods. Should be as close to 10m as possible.
   g. Place the boat on a stable place near the shore. This will be your origin point.

h. Turn on laptop
i. Double click executable icon on desktop. See software installation for more info.
2. **Starting a new Session**

   a. Click File->New Session

   ![New Session GUI](image)

   b. Enter user's name and location.

   c. Choose speed of sound calibration setting
      i. Default - will use speed of sound at room temperature and 1 atm
      ii. Previous Session - will use most recently calibrated constant
      iii. New - will require a baseline input to automatically calibrate

   d. Choose mode of operation (Real-Time, Boat Sim, Sensor Sim)
      i. If running in Real-Time: Turn boat on and manually turn tripods to point the sensors toward the boat.

   e. Main GUI window will return.
3. Using the GUI

a. Reading Sensors
   i. Boat - Designated area for all sensors which represent status of the boat such as position, velocity, mode, and internal temperature.
   ii. Payload - Designated area for all sensors which represent status of anything related to taking hydrology measurements such as depth, pH and temperature.

b. Operating Boat
   i. Autonomous
      1. Freeze - This is the default mode of operation. This will maintain the current position of the boat in the water.
      2. Ferry - This mode can specify an absolute coordinate (x, y) for the boat to drive itself to - boat will continue until it reaches its location. This can be specified by text input of the coordinates in the upper right hand corner followed by clicking ‘GO’. The desired path will then appear on the situational display. If there is an obstacle, manual mode will be required to circumvent this obstacle (see manual mode below).
      3. Scripting - This mode allows the specification of a combination of ferry and freeze commands. Ferry will require an input of x and y, freeze will require a time. Additionally, recording can be specified for each command. All scripts are saveable as '.script' files. These
scripts can be loaded for use at a later date. *Note: Do not open ‘.script’ files outside the program or they may become corrupt*

(Script window can be seen below)

ii. Manual - This mode will override all other modes of operation via a remote control. This is meant as a failsafe as well as a more direct means of control for the boat.

iii. Reading the situational display -

1. Display will keep track of previous boat movements
2. It will also display projected boat movements
3. All coordinate commands will be logged in the System Output Log
4. The ability to clear all path lines is under Settings->Graph
   Settings->Clear Graph
c. Recording Data - To record data click the record button on the GUI. The GUI will begin recording data of all the sensors for as long as you wish. When you are satisfied with the duration of data collected click stop recording. Another dialog box will open with an area for you to decide to insert notes to correspond with your saved data.

![Record Data - GUI_record.ui](Image)

d. Logging - The recorded data of each new session of logging (each time the program starts) creates a new file of the format:
"Logs/<date time>/<log element name> - <date> <time>"
All logs can be found in the Logs directory. Each log element also has its own directory within the Logs directory i.e. the system log is found in the directory "Logs/system/" and the gps log is found in "Logs/gps/" etc.

Sensors

The ultrasound and IR setup consists of six transducers, an IR LED, and two quad photodetectors. There are two transducers and one photodetector on each of the two tripods as well as two transducers and one LED located on the boat’s circuitry. The basic theory behind the sensors is to set up the two tripods on the shore that will form the other two vertices of a triangle (with the boat’s transducers and LED being the third). Based on the amount of time that the ultrasonic signal takes to get from a shore transducer to the boat and back, we can calculate two ranges: one from each tripod to the boat. Assuming a 2D system, we can therefore
extrapolate the (x,y) coordinates of the boat relative to the tripods. We can also use the azimuth and elevation angles that are found via the IR system along with the ultrasound ranges in order to extrapolate the (x,y) coordinates from each of the tripods. In order to do either of these, we need to calibrate the sensors to determine the speed of sound on the specific day of use and to define an origin point for the system to work off of.

**Wiring Connections:**
The first thing that needs to be done is that the tripods have to be connected to the laptop and given power.

1. Connect the ultrasound sensor circuitry to the top of each tripod if they are not already attached.
2. Run the two power line cables that connect the 18V power supply to each of the ultrasonic tripods.
3. Connect one end of the I2C line to interface J4 on the backend board of the AZ/EL positioner and the other end to the I2C to USB converter. Then, plug the I2C to USB converter into laptop.

**Ultrasound Calibration Setup:**
1. Place the two tripods approximately 5 feet from the end of the shore line and a distance of 2 meters apart from one another. Tripod 2 should be on the right and tripod 1 should be on the left. Set the height of each of the tripods so that they are approximately the same height as one another.
2. Measure the exact distance from sensor face to sensor face using a tape measure.
3. Open the SCADA program located on the laptop’s desktop and plug the measured distance into the “baseline distance” field by creating a new session and clicking on “new speed of sound.” Press the “Calibrate” button, which will cause the motors to turn the ultrasound sensors towards each other and then make them communicate and calculate the speed of sound.

**IR Calibration Setup:**
1. After calibrating ultrasound, press the “Calibrate IR” button, which will give the user manual control of each AZ/EL positioner. BE SURE TO PLACE THE BOAT AT DESIRED ORIGIN AND POWER ON BOAT IR BEACON!
2. Use the control stick on each AZ/EL to manually aim the photodetector assembly towards the beacon, the lock LED on the photodetector assembly will light up when it is within range of the beacon at which point you can stop adjusting. REMEMBER TO DO THIS FOR THE OTHER AZ/EL!
3. Once both photodetector assemblies are locked onto the beacon, press the “Done IR” button which will switch the AZ/EL positioners to control loop mode where a control loop on the AZ/EL’s microcontroller will automatically track the beacon.

**Origin Setup and Ranging:**
1. Twist the sensors so that they are now facing perpendicular to the river and parallel to one another without changing the baseline distance between.
2. Place the boat at the shoreline at the position that you want the origin to be, and then press the “Start Ranging” button on the laptop.

3. From this point out, the laptop will record ranging from the ultrasound sensors at the specified rate. There are no other commands that the user must send via laptop. The user only has to make sure the boat is in the water at a depth safe enough to start maneuvering and begin moving the boat via manual or autonomous control.

**Has yet to include error messages from the software during calibration and ranging.**
Safety Guide

This section will contain a detailed safety guide, including all necessary preparations and procedures. This will include both on-the-water and off-the-water safety. Necessary safety equipment will also be detailed.

Purpose

The purpose of this safety plan is to present information and rules that will help in the prevention of injury or serious harm while using the LPARD-TDF-2012 system. This safety plan is not all inclusive, but is meant to inform the user so as to prevent the vast majority of injuries. The system designers and manufacturers are by no means responsible for any injuries that occur during the use of the LPARD-TDF-2012 system if the items within the this safety plan and supplemental User’s Manual are not abided by.

Safety Procedures

Electrical:

- Keep electrical equipment (laptop, shore power supply, shore sensors, etc. at least 20 feet away from any water source to avoid possible electrocution and equipment damage.
- If electrical components smell like they are burning, power down the system as soon as possible and do not touch any electrical components because they might be extremely hot and cause burns.
- Malfunctioning electrical components run the risk of catching on fire. If the boat or any subsystem catches on fire, do not throw water on any system that is powered up. Proceed with caution when putting out electrical fires and avoid contact with flame.
- Powering up: If the boat is being powered in maintenance mode, keep the power cord and electrical output free of any possible water sources that could cause electrical shock.
- Check electric components often for visible problems and have spare fuses ready at all times.

Environmental:

- While the LPARD-TDF-2012 system is designed to operate in suboptimal conditions, it is recommended that the system is not used when in poor weather conditions. Check weather forecasts before planning and trip and consistently keep an eye on the sky to make sure that there are no developing clouds or storms. Lighting storms and extremely windy conditions can cause suboptimal conditions that could result of serious injury.
- Shore equipment should be setup and operated on a relatively flat surface so that the equipment as well as the user are able to operate on stable ground.
- Be sure to survey the shore station setup area and do not set up the shore station in an area that at risk of falling rocks or tree limbs.
- If you must step into the water to position or pick up the boat, make sure that there are not submerged objects that can possible cause physical harm.
Take precautions to limit the amount of UV exposure due to the sun. It is recommended to wear sunscreen if planning on operating the system for a prolonged period of time.

**Attire:**
- Proper footwear should be used when setting up and operating the LPARD-TDF-2012 system. The shores of rivers, creeks and streams are uneven and often difficult to navigate. Inappropriate footwear will greatly increase the risk of slipping or falling.

**Mechanical:**
- At no point should hands or fingers be placed near the propulsion components when the system is powered on. If the propulsion system gets dirty or jammed, be sure to power the system down before attempting to clean or handle the boat.
- Malfunctions are always possible. If the propulsion system goes down while the boat is on the water, try to avoid getting in the water to retrieve it. It this is absolutely necessary to enter the water, wear a flotation device.
- Do not operate the boat in manual or autonomous mode in a location where there is a chance of hitting other boats or swimmers. Survey the area both upstream and downstream to make sure that no one has a possibility of getting hit.
- Do not steer the boat towards protruding rocks or trees in the water. If the objects are visible, use manual mode to ensure that the boat does not accidentally hit an object and cause damage.
- Do not attach external objects to the boat that the boat is not specifically meant to carry that can add extra weight or cause the boat to become overloaded or unbalanced.

**Transportation:**
- While transporting the LPARD-TDF-2012 system, make sure that the boat and all other required equipment are properly fastened to the vehicle. Do not use straps or cables that are partially torn or show excessive wear.
- Use an appropriate vehicle to transport the entire system. Do not position any system parts in the vehicle that will obstruct the driver’s view.

**Other:**
- While the system is designed to have the ability to be operated by a single person, it is recommended to not work alone in isolated areas. Someone should be kept within earshot at all times since cell phone carriers’ are not available at all locations.
- Even though it is not entirely necessary, it is recommended that a lifeguard/EMT certified individual present in case of an emergency.
- Do not launch the boat from a dock or launch ramp while other boats are trying to dock / cast off.
- If planning on operating the system at nighttime, have a flashlight ready so that it is possible to navigate obstacles on the shore safely.
- Depending on the location of operation, always be aware of dangerous wildlife that may be present.
Troubleshooting

This section will contain frequently asked questions and answers, as well as detailed descriptions of predicted common issues with the system. Walkthroughs will be provided for the common top-level issues, while more technical issues will refer to the maintenance manual.

- If the AZ/ELs do not respond during calibration, first check the green LED on the front end board of the setup (board with the ultrasonic transducers). If this LED is not green, then check all the power connections to make sure the system is power up correctly.
  - 18 volts is supplied to the shore station.
  - The 7 meter long I2C cable is correctly plugged into the tripod stand
  - The 4 pin connector from the tripod stand is correctly plugged into the backend board with the white pin going to pin one of the backend board (pin 1 is the pin with an arrow pointing at it).

If the LED was already green, then the system is already powered up correctly and the I2C is not working properly. To fix this, unplug the 18 volt power cord from the shore station and plug it back in. This will reset I2C connection.

- If the range measurement always produces a value that is much shorter than expected (approximately 1 meter every time), the ultrasound system is likely triggering off of an unknown source. Check the “receive” LED on the malfunctioning tripod labeled D6. If this LED is flickering, then rotate the trimpot clockwise until the LED is no longer blinking. If the LED flickering intensifies, then the trimpot should be turned in the opposite direction. Turn the trimpot just far enough so that the LED turns off and no further since turning the trimpot farther than necessary will reduce the range of the ultrasonic ranging system.

- If the ranges recorded are consistently reported as large numbers (68 meters), then the boat is out of range for the ultrasonic ranging system and there is nothing the user can do to fix this. The boat will rely on GPS at this point now.
APPENDICES
Author: Gregory Busillo

Date: February 27, 2012

Proposal to create an omnidirectional system for the ultrasound sensors

*New parts required: 1 motor on the boat, some packaging for the motor, and the hardware to physically attach the motor to the boat*

Currently we plan on implementing two sensors on the boat and each tripod. These sensors on the tripods will be attached to the AZ/EL positioner, but do not require being on there to operate, and the sensors on the boat will be in a fixed position. This reflects what we have written in our ATP in order to complete our tests. In the future, however, the boat and shore sensors need to be able to communicate to each other in any direction. In order to achieve this omnidirectivity, we believe the AZ/EL positioner can be programmed to use the ultrasound sensors to determine the correct angle it should be turned at in order to ping towards the boat, and the sensors on the boat will have to be attached to a motor and programmed to do the exact same thing. If this is done, then the sensors will always be facing in the general direction of their pinging destination. The AZ/EL positioner has approximate limits of +/- 80 degrees so we will not have a true 180 degrees of directivity on the shore looking towards the boat. These are the steps that will have to happen in order for this setup to work.

1. The sensors on the boat need to be attached to a motor which will have to be purchased.
2. The sensors on each tripod will have to be attached to the AZ/EL motor already implemented on each tripod.
3. Each motor will have to be programmed with a control algorithm using a microcontroller that calculates the angle the motor needs to be turned at that will point the sensors on the shore towards the boat and the sensors on the boat towards the shore. In order for these calculations to be done, the following steps need to happen:
   a. The tripods need to be separated at a set distance and the sensors on the shore and boat will have to be facing each other in such a way that both tripods can send and receive pings to and from the boat.
   b. The motors will then initially lock on to the boat position during a calibration sequence. The system we build this year will take care of calculating the range to the boat from each tripod during this calibration sequence.
   c. Once the calibration sequence is finished the xy position of the boat (which will be the origin) and the xy positions of the two shore tripods (with respect to the boat origin) will be calculated by the laptop. These positions will have to be sent to each microcontroller that is being used to control the motors. The ArduPilot on the boat will receive the xy positions over the RF link and then send them to the microcontroller for the boat sensor motor. The laptop on the shore will send the xy positions to the microcontroller on each tripod.
   d. Using trigonometric functions ($\tan(\theta) = \text{opp/adj}$) and the xy positions, each motor microcontroller on the shore can calculate the angle at which it should be turned to so
that it will be pointed towards the boat (It is also possible that these calculations be done in the laptop and sent to their respective microcontroller, in which case the xy positions do not need to be sent to each microcontroller as stated in c). The boat will also be able to calculate what angle it should be turned to in order to be pointed towards each of the shore tripods.

**e.** These xy positions will continue to be updated (using the ranging technique we are implementing this year) and communicated to the microcontrollers by the ArduPilot and Laptop.

**f.** Since we are planning on pinging from one tripod to the boat at a time, then the motor on the boat will have to keep swinging back and forth between the two tripods. In order for the boat to know which tripod it should be looking at we will simply code the boat microcontroller to always start with which ever angle it has stored first. The boat can then follow the pattern of going back and forth from there. The microcontroller on the boat will have to be programmed to tell the motor to switch back and forth between the two tripods. Simply put, once it receives and sends back out a signal, the motor can be delayed some small amount and then told to switch to the other tripod angle. This amount of delay will have to be calculated in testing.

**g.** The final result will be an ultrasound ranging system on top of motors using one sensor on each motor where each motor will be able to point towards its intended pining destination. The motor microcontrollers will be continuously updated with boat positions so the motors on the shore will be tracking the boat as it moves.

4. The control algorithm already present on the AZ/EL for the IR sensor system will be used to position the motors on the shore to aim towards the boat initially (the microcontrollers will still be given the xy positions to follow the control method stated above). If the IR sensor malfunctions, then we can have the xy position scheme stated above to take care of the position of the shore motors instead.
## Contents

1 LPARD - Lafayette Programmable Autonomous River Drone .......................... 1
   1.1 Introduction .................................................................................. 1
   1.2 Code Breakdown .......................................................................... 1
      1.2.1 Shore Software ..................................................................... 1
      1.2.2 Boat Firmware ..................................................................... 1
   1.3 Other ............................................................................................. 2

2 Bug List .................................................................................................. 3

3 Namespace Index ....................................................................................... 5
   3.1 Namespace List ................................................................................ 5

4 Class Index .................................................................................................. 7
   4.1 Class Hierarchy ................................................................................ 7

5 Class Index .................................................................................................. 9
   5.1 Class List .......................................................................................... 9

6 File Index ..................................................................................................... 11
   6.1 File List ............................................................................................ 11

7 Namespace Documentation ........................................................................... 13
   7.1 BoatSim Namespace Reference ...................................................... 13
      7.1.1 Detailed Description ................................................................. 13
   7.2 guiMainPage Namespace Reference ............................................. 13
      7.2.1 Detailed Description ................................................................. 14
   7.3 guiManualPositioning Namespace Reference .................................. 14
      7.3.1 Detailed Description ................................................................. 14
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.2.3</td>
<td>getValue . . . . . . . . . . . . . . . . . . . . . . . . 22</td>
</tr>
<tr>
<td>8.3.2.4</td>
<td>getValueLength . . . . . . . . . . . . . . . . . . . . . . 22</td>
</tr>
<tr>
<td>8.3.2.5</td>
<td>isOk . . . . . . . . . . . . . . . . . . . . . . . . . . . 22</td>
</tr>
<tr>
<td>8.4</td>
<td>lpard::interface::lpardXBee::Boat Class Reference . . . . . . . . . . . . 22</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Constructor &amp; Destructor Documentation . . . . . . . . . . . . 23</td>
</tr>
<tr>
<td>8.4.1.1</td>
<td><strong>init</strong> . . . . . . . . . . . . . . . . . . . . . . . . . 23</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Member Function Documentation . . . . . . . . . . . . . . . . . 23</td>
</tr>
<tr>
<td>8.4.2.1</td>
<td>run . . . . . . . . . . . . . . . . . . . . . . . . . . . 23</td>
</tr>
<tr>
<td>8.5</td>
<td>lpard::interface::BoatSim::BoatSim Class Reference . . . . . . . . . . . . 23</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Constructor &amp; Destructor Documentation . . . . . . . . . . . . 25</td>
</tr>
<tr>
<td>8.5.1.1</td>
<td><strong>init</strong> . . . . . . . . . . . . . . . . . . . . . . . . . 25</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Member Function Documentation . . . . . . . . . . . . . . . . . 25</td>
</tr>
<tr>
<td>8.5.2.1</td>
<td>boatCommand . . . . . . . . . . . . . . . . . . . . . . . . 25</td>
</tr>
<tr>
<td>8.5.2.2</td>
<td>boatCommandResponse . . . . . . . . . . . . . . . . . . . 25</td>
</tr>
<tr>
<td>8.5.2.3</td>
<td>boatStatusMessage . . . . . . . . . . . . . . . . . . . . . 25</td>
</tr>
<tr>
<td>8.5.2.4</td>
<td>command . . . . . . . . . . . . . . . . . . . . . . . . . 26</td>
</tr>
<tr>
<td>8.5.2.5</td>
<td>getMessages . . . . . . . . . . . . . . . . . . . . . . . . 26</td>
</tr>
<tr>
<td>8.5.2.6</td>
<td>run . . . . . . . . . . . . . . . . . . . . . . . . . . . 26</td>
</tr>
<tr>
<td>8.5.2.7</td>
<td>update . . . . . . . . . . . . . . . . . . . . . . . . . . . 26</td>
</tr>
<tr>
<td>8.6</td>
<td>lpard::interface::BoatSimThreader::BoatSimThreader Class Reference . 27</td>
</tr>
<tr>
<td>8.7</td>
<td>Comp6DOF_n0m1 Class Reference . . . . . . . . . . . . . . . . . . 27</td>
</tr>
<tr>
<td>8.8</td>
<td>lpard::controller::lpardController::Controller Class Reference . . . . . . . . . . . . 28</td>
</tr>
<tr>
<td>8.8.1</td>
<td>Constructor &amp; Destructor Documentation . . . . . . . . . . . . 29</td>
</tr>
<tr>
<td>8.8.1.1</td>
<td><strong>init</strong> . . . . . . . . . . . . . . . . . . . . . . . . . 29</td>
</tr>
<tr>
<td>8.8.2</td>
<td>Member Function Documentation . . . . . . . . . . . . . . . . . 29</td>
</tr>
<tr>
<td>8.8.2.1</td>
<td>getDegrees . . . . . . . . . . . . . . . . . . . . . . . . . 29</td>
</tr>
<tr>
<td>8.8.2.2</td>
<td>getDegreesPerPwm . . . . . . . . . . . . . . . . . . . . . . 30</td>
</tr>
<tr>
<td>8.9</td>
<td>lpard::controller::lpardControllerThread::ControllerThread Class Reference 30</td>
</tr>
<tr>
<td>8.10</td>
<td>FrmeldResponse Class Reference . . . . . . . . . . . . . . . . . . 30</td>
</tr>
<tr>
<td>8.10.1</td>
<td>Detailed Description . . . . . . . . . . . . . . . . . . . . . 31</td>
</tr>
<tr>
<td>8.11</td>
<td>GPS_NMEA_Class Class Reference . . . . . . . . . . . . . . . . . . 31</td>
</tr>
<tr>
<td>8.12</td>
<td>guiAbout::GuiAbout Class Reference . . . . . . . . . . . . . . . . . 32</td>
</tr>
<tr>
<td>8.12.1</td>
<td>Detailed Description . . . . . . . . . . . . . . . . . . . . . 32</td>
</tr>
<tr>
<td>8.12.2</td>
<td>Constructor &amp; Destructor Documentation . . . . . . . . . . . . 32</td>
</tr>
</tbody>
</table>

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.12.2.1</td>
<td><strong>init</strong></td>
<td>32</td>
</tr>
<tr>
<td>8.12.3</td>
<td>Member Function Documentation</td>
<td>33</td>
</tr>
<tr>
<td>8.12.3.1</td>
<td>showGui</td>
<td>33</td>
</tr>
<tr>
<td>8.12.4</td>
<td>Member Data Documentation</td>
<td>33</td>
</tr>
<tr>
<td>8.12.4.1</td>
<td>ui</td>
<td>33</td>
</tr>
<tr>
<td>8.13</td>
<td>guiMainPage::GuiMainPage Class Reference</td>
<td>33</td>
</tr>
<tr>
<td>8.15</td>
<td>guiNewSession::GuiNewSession Class Reference</td>
<td>35</td>
</tr>
<tr>
<td>8.16</td>
<td>guiRecord::GuiRecord Class Reference</td>
<td>35</td>
</tr>
<tr>
<td>8.17</td>
<td>guiScript::GuiScript Class Reference</td>
<td>35</td>
</tr>
<tr>
<td>8.17.1</td>
<td>Detailed Description</td>
<td>36</td>
</tr>
<tr>
<td>8.17.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>36</td>
</tr>
<tr>
<td>8.17.2.1</td>
<td><strong>init</strong></td>
<td>36</td>
</tr>
<tr>
<td>8.17.3</td>
<td>Member Function Documentation</td>
<td>36</td>
</tr>
<tr>
<td>8.17.3.1</td>
<td>keyPressEvent</td>
<td>36</td>
</tr>
<tr>
<td>8.17.3.2</td>
<td>showGui</td>
<td>37</td>
</tr>
<tr>
<td>8.17.4</td>
<td>Member Data Documentation</td>
<td>37</td>
</tr>
<tr>
<td>8.17.4.1</td>
<td>controller</td>
<td>37</td>
</tr>
<tr>
<td>8.17.4.2</td>
<td>scriptList</td>
<td>37</td>
</tr>
<tr>
<td>8.18</td>
<td>HMC5883L Class Reference</td>
<td>37</td>
</tr>
<tr>
<td>8.19</td>
<td>lpard::interface::lpardI2C::I2C2PC_MockObj Class Reference</td>
<td>38</td>
</tr>
<tr>
<td>8.20</td>
<td>lpard::interface::lpardI2C::I2CInterface Class Reference</td>
<td>38</td>
</tr>
<tr>
<td>8.20.1</td>
<td>Detailed Description</td>
<td>39</td>
</tr>
<tr>
<td>8.20.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>39</td>
</tr>
<tr>
<td>8.20.2.1</td>
<td><strong>init</strong></td>
<td>39</td>
</tr>
<tr>
<td>8.20.3</td>
<td>Member Function Documentation</td>
<td>39</td>
</tr>
<tr>
<td>8.20.3.1</td>
<td>AZEL_Mode</td>
<td>40</td>
</tr>
<tr>
<td>8.20.3.2</td>
<td>changeAZ</td>
<td>40</td>
</tr>
<tr>
<td>8.20.3.3</td>
<td>changeEL</td>
<td>40</td>
</tr>
<tr>
<td>8.20.3.4</td>
<td>data</td>
<td>40</td>
</tr>
<tr>
<td>8.20.3.5</td>
<td>status</td>
<td>40</td>
</tr>
<tr>
<td>8.20.3.6</td>
<td>US_Mode</td>
<td>41</td>
</tr>
<tr>
<td>8.21</td>
<td>lpard::logger::lpardLogger::Logger Class Reference</td>
<td>41</td>
</tr>
<tr>
<td>8.21.1</td>
<td>Detailed Description</td>
<td>41</td>
</tr>
</tbody>
</table>
CONTENTS

8.22 MagnetometerRaw Struct Reference .......................... 42
8.23 MagnetometerScaled Struct Reference ....................... 42
8.24 guiScript::Mode Class Reference ............................. 42
  8.24.1 Constructor & Destructor Documentation ................ 43
  8.24.1.1 __init__ ........................................ 43
8.24.2 Member Function Documentation ................................ 43
  8.24.2.1 __repr__ ....................................... 43
8.24.3 Member Data Documentation .................................. 43
  8.24.3.1 mode ........................................... 43
  8.24.3.2 record .......................................... 43
  8.24.3.3 x ............................................... 43
  8.24.3.4 y ............................................... 43
8.25 ModemStatusResponse Class Reference ....................... 44
  8.25.1 Detailed Description ................................ 44
8.26 nameValueObj::NameValueObj Class Reference ................. 44
8.27 PayloadRequest Class Reference .............................. 45
  8.27.1 Detailed Description ................................ 45
8.27.2 Member Function Documentation ............................ 45
  8.27.2.1 getPayload ..................................... 45
  8.27.2.2 getPayloadLength ............................... 45
  8.27.2.3 setPayload .................................... 45
  8.27.2.4 setPayloadLength ............................... 46
8.28 RemoteAtCommandRequest Class Reference .................... 46
  8.28.1 Detailed Description ................................ 47
8.28.2 Constructor & Destructor Documentation ................... 47
  8.28.2.1 RemoteAtCommandRequest .......................... 47
  8.28.2.2 RemoteAtCommandRequest .......................... 47
  8.28.2.3 RemoteAtCommandRequest .......................... 47
  8.28.2.4 RemoteAtCommandRequest .......................... 47
8.28.3 Member Function Documentation ............................ 47
  8.28.3.1 getFrameData ................................... 47
  8.28.3.2 getFrameDataLength .............................. 48
8.29 RemoteAtCommandResponse Class Reference ................... 48
  8.29.1 Detailed Description ................................ 48
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.37</td>
<td>Tx16Request Class Reference</td>
<td>56</td>
</tr>
<tr>
<td>8.37.1</td>
<td>Detailed Description</td>
<td>57</td>
</tr>
<tr>
<td>8.37.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>57</td>
</tr>
<tr>
<td>8.37.2.1</td>
<td>Tx16Request</td>
<td>57</td>
</tr>
<tr>
<td>8.37.2.2</td>
<td>Tx16Request</td>
<td>57</td>
</tr>
<tr>
<td>8.37.3</td>
<td>Member Function Documentation</td>
<td>57</td>
</tr>
<tr>
<td>8.37.3.1</td>
<td>getFrameData</td>
<td>57</td>
</tr>
<tr>
<td>8.37.3.2</td>
<td>getFrameDataLength</td>
<td>57</td>
</tr>
<tr>
<td>8.38</td>
<td>Tx64Request Class Reference</td>
<td>58</td>
</tr>
<tr>
<td>8.38.1</td>
<td>Detailed Description</td>
<td>58</td>
</tr>
<tr>
<td>8.38.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>58</td>
</tr>
<tr>
<td>8.38.2.1</td>
<td>Tx64Request</td>
<td>59</td>
</tr>
<tr>
<td>8.38.2.2</td>
<td>Tx64Request</td>
<td>59</td>
</tr>
<tr>
<td>8.38.3</td>
<td>Member Function Documentation</td>
<td>59</td>
</tr>
<tr>
<td>8.38.3.1</td>
<td>getFrameData</td>
<td>59</td>
</tr>
<tr>
<td>8.38.3.2</td>
<td>getFrameDataLength</td>
<td>59</td>
</tr>
<tr>
<td>8.39</td>
<td>TxStatusResponse Class Reference</td>
<td>59</td>
</tr>
<tr>
<td>8.39.1</td>
<td>Detailed Description</td>
<td>60</td>
</tr>
<tr>
<td>8.40</td>
<td>GUI_about_ui::Ui_about Class Reference</td>
<td>60</td>
</tr>
<tr>
<td>8.41</td>
<td>GUI_mainPageReturn_ui::Ui_guiMain Class Reference</td>
<td>61</td>
</tr>
<tr>
<td>8.42</td>
<td>GUI_manualPositioning_ui::Ui_manualPositioning Class Reference</td>
<td>62</td>
</tr>
<tr>
<td>8.43</td>
<td>GUI_newSession_ui::Ui_newSession Class Reference</td>
<td>62</td>
</tr>
<tr>
<td>8.44</td>
<td>GUI_record_ui::Ui_recordData Class Reference</td>
<td>63</td>
</tr>
<tr>
<td>8.45</td>
<td>GUI_script_ui::Ui_Script Class Reference</td>
<td>64</td>
</tr>
<tr>
<td>8.46</td>
<td>XBee Class Reference</td>
<td>65</td>
</tr>
<tr>
<td>8.46.1</td>
<td>Detailed Description</td>
<td>65</td>
</tr>
<tr>
<td>8.46.2</td>
<td>Member Function Documentation</td>
<td>66</td>
</tr>
<tr>
<td>8.46.2.1</td>
<td>begin</td>
<td>66</td>
</tr>
<tr>
<td>8.46.2.2</td>
<td>getNextFrameId</td>
<td>66</td>
</tr>
<tr>
<td>8.46.2.3</td>
<td>getResponse</td>
<td>66</td>
</tr>
<tr>
<td>8.46.2.4</td>
<td>readPacket</td>
<td>66</td>
</tr>
<tr>
<td>8.46.2.5</td>
<td>readPacket</td>
<td>66</td>
</tr>
<tr>
<td>8.46.2.6</td>
<td>readPacketUntilAvailable</td>
<td>66</td>
</tr>
<tr>
<td>8.46.2.7</td>
<td>send</td>
<td>66</td>
</tr>
</tbody>
</table>

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
8.46.2.8 setSerial ........................................... 67
8.47 XBeetAddress Class Reference .................................. 67
8.48 XBeetAddress64 Class Reference .................................. 67
8.48.1 Detailed Description ......................................... 68
8.49 lpard::interface::lpardXBee::XbeeInterface Class Reference .......................... 68
8.49.1 Detailed Description ......................................... 68
8.49.2 Constructor & Destructor Documentation ......................... 68
8.49.2.1 __init__ ........................................... 68
8.49.3 Member Function Documentation ................................ 69
8.49.3.1 closePort .......................................... 69
8.49.3.2 getMessage ........................................ 69
8.49.3.3 sendMessage ........................................ 69
8.50 XBeerequest Class Reference ................................... 69
8.50.1 Detailed Description ......................................... 70
8.50.2 Constructor & Destructor Documentation ......................... 70
8.50.2.1 XBeerequest ....................................... 70
8.50.3 Member Function Documentation ................................ 70
8.50.3.1 getApid ........................................... 70
8.50.3.2 getFrameData ....................................... 71
8.50.3.3 getFrameDataLength ................................... 71
8.50.3.4 getFrameld ......................................... 71
8.50.3.5 setFrameld ........................................ 71
8.51 XBeereponse Class Reference .................................. 71
8.51.1 Detailed Description ......................................... 72
8.51.2 Constructor & Destructor Documentation ......................... 73
8.51.2.1 XBeereponse ....................................... 73
8.51.3 Member Function Documentation ................................ 73
8.51.3.1 getApid ........................................... 73
8.51.3.2 getAtCommandResponse ................................ 73
8.51.3.3 getChecksum ....................................... 73
8.51.3.4 getErrorCode ...................................... 73
8.51.3.5 getFrameData ....................................... 73
8.51.3.6 getFrameDataLength ................................... 74
8.51.3.7 getLsbLength ....................................... 74
8.51.3.8 getModemStatusResponse .......................... 74
8.51.3.9 getMsbLength ................................... 74
8.51.3.10 getPacketLength ................................. 74
8.51.3.11 getRemoteAtCommandResponse ...................... 74
8.51.3.12 getRx16IoSampleResponse ......................... 74
8.51.3.13 getRx16Response .................................. 74
8.51.3.14 getRx16IoSampleResponse ......................... 75
8.51.3.15 getRx64Response .................................. 75
8.51.3.16 getTxStatusResponse ............................ 75
8.51.3.17 getZBRxIoSampleResponse ......................... 75
8.51.3.18 getZBRxResponse .................................. 75
8.51.3.19 getZBTxStatusResponse ......................... 75
8.51.3.20 init .............................................. 75
8.51.3.21 isAvailable ..................................... 75
8.51.3.22 isError .......................................... 75
8.51.3.23 reset .............................................. 76

8.52 ZBRxIoSampleResponse Class Reference ............... 76
8.52.1 Detailed Description ................................ 76
8.52.2 Member Function Documentation .................... 77
  8.52.2.1 getAnalog ....................................... 77
  8.52.2.2 isAnalogEnabled ................................ 77
  8.52.2.3 isDigitalEnabled ............................... 77
  8.52.2.4 isDigitalOn .................................... 77

8.53 ZBRxResponse Class Reference ......................... 77
8.53.1 Detailed Description ................................ 78
8.53.2 Member Function Documentation .................... 78
  8.53.2.1 getDataLength .................................. 78
  8.53.2.2 getDataOffset .................................. 78

8.54 ZBTxRequest Class Reference ......................... 78
8.54.1 Detailed Description ................................ 79
8.54.2 Constructor & Destructor Documentation ............ 79
  8.54.2.1 ZBTxRequest .................................... 79
  8.54.2.2 ZBTxRequest .................................... 79
8.54.3 Member Function Documentation .................... 79
Chapter 1

LPARD - Lafayette Programmable Autonomous River Drone

1.1 Introduction

For the ECE492 Senior Design project, the ECE Class of 2012 is creating the Tracking and Data Fusion component of the three-year Lafayette Programmable Autonomous River Droid project. This is the API for the software and firmware.

1.2 Code Breakdown

1.2.1 Shore Software

Shore software consists of:

- The GUI
- Boat simulation model
- Freeze script to make executable
- Controller for I2C/XBee/GUI interface
- The AZEL control and sensor software

1.2.2 Boat Firmware

Boat firmware consists of:

- The main drone firmware
- Libraries for:
2 LPARD - Lafayette Programmable Autonomous River Drone

- XBee communication
- GPS communication
- Compass access and tilt compensation

1.3 Other

Note that compiling the PDF requires the package texlive-fonts-recommended.
Chapter 2

Bug List

File **AZELCode.h**  Main loop does not have fixed timing.
  Hard coded limits.

File **drone_firmware.pde**  Compass not tilt compensated.
  Compass iron offsets infinite loop, and have been disabled.
  Doesn’t receive XBee packets properly. Could be a mode issue (needs API=2).
  Cannot set declination angle externally.
Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

- **BoatSim** (This package contains a single class that acts as a real time simulation of a boat) ........................................ 13
- **guiMainPage** ................................................................. 13
- **guiManualPositioning** .................................................. 14
- **guiNewSession** ............................................................ 14
- **guiRecord** ........................................................................ 14
- **guiScript** (This package contains two classes responsible for the creating, importing, and saving of automated script files) ............. 15
- **lpard** ............................................................................. 15
- **lpardI2C** (This package contains a class that handles the interface to an AZEL Positioner through the I2C 2 PC Adapter) ............ 16
- **lpardXbee** (This package contains a class that communicates in real time with a boat using Xbee rf tranceivers) .................. 16
- **nameValueObj** ............................................................... 16
- **setup** ........................................................................... 17
- **utils** (This package contains all utilities for general use in the software) ................................................................. 17
Chapter 4

Class Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- ADXL335 .................................................. 19
- lpard::interface::lpardXBee::Boat .......................... 22
- lpard::interface::BoatSim::BoatSim ......................... 23
- lpard::interface::BoatSimThreader::BoatSimThreader ...... 27
- Comp6DOF_n0m1 ............................................. 27
- lpard::controller::lpardController::Controller .......... 28
- lpard::controller::lpardControllerThread::ControllerThread 30
- GPS_NMEA_Class ............................................. 31
- guiAbout::GuiAbout ........................................ 32
- guiMainPage::GuiMainPage .................................. 33
- guiNewSession::GuiNewSession ................................ 35
- guiRecord::GuiRecord ....................................... 35
- guiScript::GuiScript ........................................ 35
- HMC5883L ..................................................... 37
- lpard::interface::lpardI2C::I2C2PC_MockObj .................. 38
- lpard::interface::lpardI2C::I2CInterface ................... 38
- lpard::logger::lpardLogger::Logger ......................... 41
- MagnetometerRaw ............................................. 42
- MagnetometerScaled .......................................... 42
- guiScript::Mode .............................................. 42
- nameValueObj::NameValueObj ................................ 44
- GUI_about_ui::Ui_about ..................................... 60
- GUI_mainPageReturn_ui::Ui_guiMain ......................... 61
- GUI_manualPositioning_ui::Ui_manualPositioning .......... 62
- GUI_newSession_ui::Ui_newSession .......................... 62
- GUI_record_ui::Ui_recordData ............................... 63
- GUI_script_ui::Ui_Script ..................................... 64
- XBee .......................................................... 65
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>XBeeAddress</td>
<td>67</td>
</tr>
<tr>
<td>XBeeAddress64</td>
<td>67</td>
</tr>
<tr>
<td>lpard::interface::lpardXBe::XbeeInterface</td>
<td>68</td>
</tr>
<tr>
<td>XBeeRequest</td>
<td>69</td>
</tr>
<tr>
<td>AtCommandRequest</td>
<td>19</td>
</tr>
<tr>
<td>RemoteAtCommandRequest</td>
<td>46</td>
</tr>
<tr>
<td>PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td>Tx16Request</td>
<td>56</td>
</tr>
<tr>
<td>Tx64Request</td>
<td>58</td>
</tr>
<tr>
<td>ZBTxRequest</td>
<td>78</td>
</tr>
<tr>
<td>XBeeResponse</td>
<td>71</td>
</tr>
<tr>
<td>FrameIdResponse</td>
<td>30</td>
</tr>
<tr>
<td>AtCommandResponse</td>
<td>21</td>
</tr>
<tr>
<td>RemoteAtCommandResponse</td>
<td>48</td>
</tr>
<tr>
<td>TxStatusResponse</td>
<td>59</td>
</tr>
<tr>
<td>ZBTxStatusResponse</td>
<td>80</td>
</tr>
<tr>
<td>ModemStatusResponse</td>
<td>44</td>
</tr>
<tr>
<td>RxDataResponse</td>
<td>52</td>
</tr>
<tr>
<td>RxResponse</td>
<td>55</td>
</tr>
<tr>
<td>Rx16Response</td>
<td>50</td>
</tr>
<tr>
<td>Rx64Response</td>
<td>52</td>
</tr>
<tr>
<td>RxIoSampleBaseResponse</td>
<td>54</td>
</tr>
<tr>
<td>Rx16IoSampleResponse</td>
<td>50</td>
</tr>
<tr>
<td>Rx64IoSampleResponse</td>
<td>51</td>
</tr>
<tr>
<td>ZBRxResponse</td>
<td>77</td>
</tr>
<tr>
<td>ZBRxIoSampleResponse</td>
<td>76</td>
</tr>
</tbody>
</table>
Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- ADXL335 ................................................................. 19
- AtCommandRequest ................................................. 19
- AtCommandResponse ............................................... 19
- lpard::interface::lpardXBee::Boat ................................ 21
- lpard::interface::BoatSim::BoatSim ............................ 22
- lpard::interface::BoatSimThreader::BoatSimThreader ....... 27
- Comp6DOF_n0m1 ....................................................... 27
- lpard::controller::lpardController::Controller ............... 28
- lpard::controller::lpardControllerThread::ControllerThread .. 30
- FrameIdResponse ..................................................... 30
- GPS_NMEA_Class ..................................................... 31
- guiAbout::GuiAbout (Controlling class for the About dialog ) .. 32
- guiMainPage::GuiMainPage ......................................... 33
- guiNewSession::GuiNewSession .................................... 35
- guiRecord::GuiRecord .............................................. 35
- guiScript::GuiScript (Responsible for the creating, reading and writing of navigation scripts ) ..................... 35
- HMC5883L ............................................................... 37
- lpard::interface::lpardI2C::I2C2PC_MockObj ..................... 38
- lpard::interface::lpardI2C::I2CInterface ......................... 38
- lpard::logger::lpardLogger::Logger ............................... 41
- MagnetometerRaw ..................................................... 41
- MagnetometerScaled ............................................... 42
- guiScript::Mode ...................................................... 42
- ModemStatusResponse .............................................. 44
- nameValueObj::NameValueObj ....................................... 44
- PayloadRequest ..................................................... 45
- RemoteAtCommandRequest ......................................... 46
<table>
<thead>
<tr>
<th>Class Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoteAtCommandResponse ........................................ 48</td>
</tr>
<tr>
<td>Rx16IoSampleResponse ........................................... 50</td>
</tr>
<tr>
<td>Rx16Response ....................................................... 50</td>
</tr>
<tr>
<td>Rx64IoSampleResponse ............................................ 51</td>
</tr>
<tr>
<td>Rx64Response ....................................................... 52</td>
</tr>
<tr>
<td>RxDataResponse ..................................................... 52</td>
</tr>
<tr>
<td>RxIoSampleBaseResponse ........................................... 54</td>
</tr>
<tr>
<td>RxResponse ........................................................... 55</td>
</tr>
<tr>
<td>Tx16Request .......................................................... 56</td>
</tr>
<tr>
<td>Tx64Request .......................................................... 58</td>
</tr>
<tr>
<td>TxStatusResponse .................................................... 59</td>
</tr>
<tr>
<td>GUI_about_ui::Ui_about ............................................. 60</td>
</tr>
<tr>
<td>GUI_mainPageReturn_ui::Ui_guiMain ................................ 61</td>
</tr>
<tr>
<td>GUI_manualPositioning_ui::Ui_manualPositioning .................. 62</td>
</tr>
<tr>
<td>GUI_newSession_ui::Ui_newSession ................................ 62</td>
</tr>
<tr>
<td>GUI_record_ui::Ui_recordData ..................................... 63</td>
</tr>
<tr>
<td>GUI_script_ui::Ui_Script .......................................... 64</td>
</tr>
<tr>
<td>XBee ................................................................. 65</td>
</tr>
<tr>
<td>XBeeAddress .......................................................... 67</td>
</tr>
<tr>
<td>XBeeAddress64 .......................................................... 67</td>
</tr>
<tr>
<td>lpard::interface::lpardXBee::XbeeInterface ......................... 68</td>
</tr>
<tr>
<td>XBeeRequest ........................................................... 69</td>
</tr>
<tr>
<td>XBeeResponse .......................................................... 71</td>
</tr>
<tr>
<td>ZBRxIoSampleResponse .............................................. 76</td>
</tr>
<tr>
<td>ZBRxResponse ........................................................... 77</td>
</tr>
<tr>
<td>ZBTxRequest ............................................................ 78</td>
</tr>
<tr>
<td>ZBTxStatusResponse .................................................. 80</td>
</tr>
</tbody>
</table>
Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

- boat-systems/drone_firmware/drone_firmware.pde (Main Arduino code file for boat firmware) ........................................ 81
- boat-systems/drone_firmware/libraries/ADXL335/ADXL335.h ....... ??
- boat-systems/drone_firmware/libraries/Comp6DOF/Comp6DOF_n0m1.h ...... ??
- boat-systems/drone_firmware/libraries/GPS_NMEA/GPS_NMEA.h ........ ??
- boat-systems/drone_firmware/libraries/HMC5883L/HMC5883L.h ........ ??
- boat-systems/drone_firmware/libraries/lpard_RF/lpard_rf.h ............. ??
- boat-systems/drone_firmware/libraries/XBee/XBee.h .................. ??
- shore-systems/AZEL/AZELCode/AZELCode.h (Firmware for the AZ/EL Positioner) ....................................................... 85
- shore-systems/AZEL/AZELCode/i2c.h ..................................... ??
- shore-systems/AZEL/AZELCode/pwm.h .................................. ??
- shore-systems/AZEL/AZELCode/tracking.h .............................. ??
- shore-systems/AZEL/AZELCode/ultrasound.h ............................ ??
Chapter 7

Namespace Documentation

7.1 BoatSim Namespace Reference

This package contains a single class that acts as a real time simulation of a boat.

7.1.1 Detailed Description

This package contains a single class that acts as a real time simulation of a boat. It extends the Thread class so the program can be run as a separate thread from the main GUI thread.

Author

Aaron Springut

Date

2012

7.2 guiMainPage Namespace Reference

Classes

• class GuiMainPage

Functions

• def main
7.2.1 Detailed Description

Created on Mar 8, 2012
@author: This PC

7.3 guiManualPositioning Namespace Reference

Classes

- class GuiManualPositioning

Functions

- def main

7.3.1 Detailed Description

Created on Apr 12, 2012
@author: Charles Thomas

7.4 guiNewSession Namespace Reference

Classes

- class GuiNewSession

Functions

- def main

7.4.1 Detailed Description

Created on Apr 4, 2012
@author: Lauren Elizabeth

7.5 guiRecord Namespace Reference

Classes

- class GuiRecord
7.6 guiScript Namespace Reference

This package contains two classes responsible for the creating, importing, and saving of automated script files.

Classes

- class GuiScript
  The GuiScript class is responsible for the creating, reading and writing of navigation scripts.
- class Mode

Functions

- def main

7.6.1 Detailed Description

This package contains two classes responsible for the creating, importing, and saving of automated script files. The classes are GuiScript and Mode.

Date

: Created on Mar 22, 2012

Author

: Michael Rupolo

7.7 lpard Namespace Reference

7.7.1 Detailed Description

print "** Loading Packages **
Loading logger:"
import lpard.logger
print "Loading gui:"
import lpard.gui

### 7.8 lpardI2C Namespace Reference

This package contains a class that handles the interface to an AZEL Positioner through the I2C 2 PC Adapter.

#### 7.8.1 Detailed Description

This package contains a class that handles the interface to an AZEL Positioner through the I2C 2 PC Adapter. This class uses a configuration file to take commands and formulate I2C transmissions for them. There is also a commandline interface utility that can be called directly from this package as well as a mock object for the I2C 2 Pc adapter for testing.

### 7.9 lpardXbee Namespace Reference

This package contains a class that communicates in real time with a boat using Xbee rf tranceivers.

#### 7.9.1 Detailed Description

This package contains a class that communicates in real time with a boat using Xbee rf tranceivers. The Boat class is a thread which shuffles data from the Xbee to two different queues, received packets and packets to send. The XbeeInterface class then used to read and write the packets to the queues.

```
---- > TxQueue ---- > Software <--- > XbeeInterface Boat <--- > Xbee <---- RxQueue <----
```

### 7.10 nameValueObj Namespace Reference

#### Classes

- class NameValueObj

#### 7.10.1 Detailed Description

Created on Apr 5, 2012

@author: Lauren Elizabeth
7.11 setup Namespace Reference

Variables

- list `data_files` = []
- list `f1` = sys.path[0]
- string `f2` = './config'
- list `dirpath` = sys.path[0]
- dictionary `opts`
  - string `description` = "LPARD-TDF-2012 Software"
  - string `long_description` = ""
  - string `version` = "0.1"
  - list `windows` = [{'script': "lpard_main.py", "dest_base": "lpard"}]
- `options` = `opts`,
- tuple `zf` = zipfile.ZipFile('LPARD-TDF-2012 Software.zip', mode='w')

7.11.1 Detailed Description

7.11.2 Variable Documentation

7.11.2.1 dictionary setup::`opts`

**Initial value:**

```python
{'py2exe': {
  # 'bundle_files': 2,
  'includes': ['sip'],
  'dist_dir': dirpath,
  'excludes': ['pdb', 'unittest'],
  'dll_excludes': ['w9xpopen.exe']
}}
```

7.12 utils Namespace Reference

This package contains all utilities for general use in the software.

7.12.1 Detailed Description

This package contains all utilities for general use in the software.
Chapter 8

Class Documentation

8.1 ADXL335 Class Reference

Public Member Functions

- `ADXL335` (int pin_x, int pin_y, int pin_z, float aref)
- void `setThreshold` (float deadzone)
- boolean `getFreefall` ()
- float `getX` ()
- float `getY` ()
- float `getZ` ()
- float `getRho` ()
- float `getPhi` ()
- float `getTheta` ()
- void `update` ()

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/ADXL335/ADXL335.h
- boat-systems/drone_firmware/libraries/ADXL335/ADXL335.cpp

8.2 AtCommandRequest Class Reference

```
#include <XBee.h>
```

Inheritance diagram for AtCommandRequest:
Public Member Functions

• **AtCommandRequest** (uint8_t *command)
  • **AtCommandRequest** (uint8_t *command, uint8_t *commandValue, uint8_t commandValueLength)
  • uint8_t **getFrameData** (uint8_t pos)
  • uint8_t **getFrameDataLength** ()
  • uint8_t * **getCommand** ()
  • void **setCommand** (uint8_t *command)
  • uint8_t * **getCommandValue** ()
  • void **setCommandValue** (uint8_t *command)
  • uint8_t **getCommandValueLength** ()
  • void **setCommandValueLength** (uint8_t length)
  • void **clearCommandValue** ()

8.2.1 Detailed Description

Represents an AT Command TX packet. The command is used to configure the serially connected XBee radio.

8.2.2 Member Function Documentation

8.2.2.2 void AtCommandRequest::clearCommandValue ()

Clears the optional commandValue and commandValueLength so that a query may be sent.

8.2.2.2 uint8_t AtCommandRequest::getFrameData ( uint8_t pos ) [virtual]

Starting after the frame id (pos = 0) and up to but not including the checksum. Note: Unlike Digi's definition of the frame data, this does not start with the API ID. The reason for this is the API ID and Frame ID are common to all requests, whereas my definition of frame data is only the API specific data.

Implements XBeeRequest.

Reimplemented in RemoteAtCommandRequest.
8.3 AtCommandResponse Class Reference

8.2.2.3 uint8_t AtCommandRequest::getFrameDataLength ( ) [virtual]

Returns the size of the api frame (not including frame id or api id or checksum).
Implements XBeeRequest.
Reimplemented in RemoteAtCommandRequest.
The documentation for this class was generated from the following files:
- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.3 AtCommandResponse Class Reference

#include <XBee.h>

Inheritance diagram for AtCommandResponse:

```
XBeeResponse
    
FrameIdResponse
    
AtCommandResponse
    
RemoteAtCommandResponse
```

Public Member Functions

- uint8_t * getCommand ()
- uint8_t getStatus ()
- uint8_t * getValue ()
- uint8_t getValueLength ()
- bool isOk ()

8.3.1 Detailed Description

Represents an AT Command RX packet

8.3.2 Member Function Documentation

8.3.2.1 uint8_t * AtCommandResponse::getCommand ( )

Returns an array containing the two character command

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
Reimplemented in RemoteAtCommandResponse.

8.3.2.2 uint8_t AtCommandResponse::getStatus ( )

Returns the command status code. Zero represents a successful command
Reimplemented in RemoteAtCommandResponse.

8.3.2.3 uint8_t * AtCommandResponse::getValue ( )

Returns an array containing the command value. This is only applicable to query com-
mands.
Reimplemented in RemoteAtCommandResponse.

8.3.2.4 uint8_t AtCommandResponse::getValueLength ( )

Returns the length of the command value array.
Reimplemented in RemoteAtCommandResponse.

8.3.2.5 bool AtCommandResponse::isOk ( )

Returns true if status equals AT_OK
Reimplemented in RemoteAtCommandResponse.

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.4 Ipard::interface::IpardXBee::Boat Class Reference

Inherits Thread.

Public Member Functions

- def __init__

  Constructor for the Boat thread.

- def run

  This method runs the thread.
Public Attributes

- inQ
- outQ
- zigbee
- alive

8.4.1 Constructor & Destructor Documentation

8.4.1.1 def lpard::interface::lpardXBee::Boat::init( self, inQ, outQ, zigbee )

Constructor for the Boat thread.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The pointer to the class instance.</td>
</tr>
<tr>
<td>inQ</td>
<td>A synchronized queue for passing commands to the simulation.</td>
</tr>
<tr>
<td>outQ</td>
<td>A synchronized queue for receiving commands from the GUI.</td>
</tr>
<tr>
<td>zigbee</td>
<td>A xbee.ZigBee object loaded with a PySerial object to an XBee</td>
</tr>
</tbody>
</table>

8.4.2 Member Function Documentation

8.4.2.1 def lpard::interface::lpardXBee::Boat::run( self )

This method runs the thread.

This method overrides the Thread.run() method. When the thread is started this method is called. In a loop, checks inQ for new messages to send and puts new messages from the xbee into outQ.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The pointer to the class.</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following file:

- shore-systems/lpard/interface/lpardXBee.py

8.5 lpard::interface::BoatSim::BoatSim Class Reference

Public Member Functions

- def __init__
  Constructor for the simulation.
- def run
  This method runs the simulation.
- def command
This method processes any input strings and then call the correct method to deal with them, depending on the header.

• def boatCommand
  This method processes all methods with a "BC" or boat command header.

• def update
  This method updates the boat model states through the transfer function described in the transition matrix.

• def getMessages
  This method checks if any messages have been put into the input queue and calls the command() if a message is present.

• def boatStatusMessage
  This method places the boat status message in the output queue.

• def boatCommandResponse
  This method send a command response.

Public Attributes

• inQ
• outQ
• startTime
• xPos
• yPos
• zPos
• theta
• fMov
• sMov
• oMov
• fMax
• sMax
• oMax
• forwardGain
• sideGain
• thetaGain
• xDest
• yDest
• mode
• calibrate
• fault
• timeStep
8.5 lpard::interface::BoatSim::BoatSim Class Reference

8.5.1 Constructor & Destructor Documentation

8.5.1.1 def lpard::interface::BoatSim::BoatSim::init( self, inQ, outQ )

Constructor for the simulation.

Parameters

| self | The pointer to the class instance. |
| inQ  | A synchronized queue for passing commands to the simulation. |
| outQ | A synchronized queue for receiving commands from the GUI. |

8.5.2 Member Function Documentation

8.5.2.1 def lpard::interface::BoatSim::BoatSim::boatCommand( self, commandString )

This method processes all methods with a "BC" or boat command header.

Parameters

| self | The pointer to the class. |
| commandString | The message that has been sent to the boat. |

8.5.2.2 def lpard::interface::BoatSim::BoatSim::boatCommandResponse( self, queriedVariable )

This method send a command response.

It is normally called when the boatCommand() method determines that a response is necessary.

Parameters

| self | The pointer to the class. |
| queriedVariable | The boat state that is being queried. This can be any one of the variables for command response specified in the boat protocol document. |

8.5.2.3 def lpard::interface::BoatSim::BoatSim::boatStatusMessage( self )

This method places the boat status message in the output queue.

This message gets called every 1000 timesteps in the run method. This corresponds to about 1 second.

Parameters

| self | The pointer to the class. |
8.5.2.4  def lpard::interface::BoatSim::BoatSim::command ( self, commandString )

This method processes any input strings and then call the correct method to deal with them, depending on the header.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| self        | The pointer to the class.
| commandString | The message that has been sent to the boat.

8.5.2.5  def lpard::interface::BoatSim::BoatSim::getMessages ( self )

This method checks if any messages have been put into the input queue and calls the command() if a message is present.

This method should be called consistantly during the run() method.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| self      | The pointer to the class.

8.5.2.6  def lpard::interface::BoatSim::BoatSim::run ( self )

This method runs the simulation.

This method overrides the Thread.run() method. When the thread is started this method is called. It continually loops and updates the transfer model and deals with any input messages.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| self      | The pointer to the class.

8.5.2.7  def lpard::interface::BoatSim::BoatSim::update ( self )

This method updates the boat model states through the transfer function described in the transition matrix.

This matrix can be found in the boat model analysis. Currently this is not implemented correctly and is instead replaced with a simplified version.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| self      | The pointer to the class.

The documentation for this class was generated from the following file:

- shore-systems/lpard/interface/BoatSim.py

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
8.6 Ipard::interface::BoatSimThreader::BoatSimThreader Class Reference

Public Member Functions

- def __init__
- def sendMessage
- def getMessage

Public Attributes

- inQ
- outQ

The documentation for this class was generated from the following file:

- shore-systems/Ipard/interface/BoatSimThreader.py

8.7 Comp6DOF_n0m1 Class Reference

Public Member Functions

- void compCompass (int xMagAxis, int yMagAxis, int zMagAxis, int xAccel, int yAccel, int zAccel, boolean lowpass)
- int roll ()
- int pitch ()
- int yaw ()
- float rollf ()
- float pitchf ()
- float yawf ()
- int xAxisComp ()
- int yAxisComp ()
- int zAxisComp ()
- int xHardOff ()
- int yHardOff ()
- int zHardOff ()
- boolean deviantSpread (int XAxis, int YAxis, int ZAxis)
- boolean calOffsets ()
- int atan2Int (int y, int x)

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/Comp6DOF/Comp6DOF_n0m1.h
- boat-systems/drone_firmware/libraries/Comp6DOF/Comp6DOF_n0m1.cpp
8.8 lpard::controller::lpardController::Controller Class Reference

Public Member Functions

- `def __init__`
  Constructor, initializes the logger, also populates default sensor values and default "listOfValues" values.
- `def getDegreesPerPwm`
  Calculate values required for degree calculations.
- `def getDegrees`
  Returns the azimuth angle and elevation angle in degrees based off of the passed PWM values.
- `def getPWM`
  Returns the azimuth angle and elevation angle in PWM based off of the passed degree values.
- `def initSerialInterface`
  Ensures the AZEL(s) are connected via USB - Finds the AZEL(s) and start the USB communications.
- `def manualAZELPositioning`
- `def autoAZELPositioning`
- `def receiveUltrasonic`
- `def transmiteUltrasonic`
- `def logCurrentAZELStatus`
- `def processRawAZEL`
- `def processAZELData`
- `def startSearch`
- `def contSearch`
- `def moveSensorsLeft`
- `def moveSensorsRight`
- `def moveSensorsUp`
- `def moveSensorsDown`
- `def initCalibrate`
- `def updateData`
- `def translateSensorPos`
- `def setSensorPos`
- `def updateSensorPos`
- `def calSOS`
- `def shutdown`

Public Attributes

- `prev_sensor`
- `xValue`
- `yValue`
- `thetaValue`
8.8 lpard::controller::lpardController::Controller Class Reference

- `I2C_port`
- `r1`
- `r2`
- `baseline`
- `sos`

Static Public Attributes

- `logger` = None
- dictionary `I2C_port` = {}
- `ser_itrfc` = None
- `xbee_itrfc` = None
- dictionary `system_values` = {}
- list `listOfValues` = []
- `i2c_config` = None
- `xbee_config` = None
- `mode` = REAL_TIME
- float `r1` = 1.0
- float `r2` = 1.0
- int `baseline` = 1
- float `sos` = 0.344

8.8.1 Constructor & Destructor Documentation

8.8.1.1 def lpard::controller:: lpardController::Controller:: __init__ ( self, sensors = [1 ])

Constructor, initializes the logger, also populates default sensor values and default "listOfValues" values.

Parameters

<table>
<thead>
<tr>
<th>num_sensors</th>
<th>List of AZEL sensors (Defaults to [1,2])</th>
</tr>
</thead>
</table>

8.8.2 Member Function Documentation

8.8.2.1 def lpard::controller::lpardController::Controller::getDegrees ( self, index, azimuth, elevation )

Returns the azimuth angle and elevation angle in degrees based off of the passed PWM values.

Parameters

<table>
<thead>
<tr>
<th>index</th>
<th>The index of the sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>azimuth</td>
<td>The PWM value for the horizontal axis</td>
</tr>
<tr>
<td>elevation</td>
<td>The PWM value for the vertical axis</td>
</tr>
</tbody>
</table>

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
Calculate values required for degree calculations.
Data is stored in AZEL_dgr_per_pwm

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_left</td>
<td>The PWM value for soft left limit</td>
</tr>
<tr>
<td>pwm_right</td>
<td>The PWM value for soft right limit</td>
</tr>
<tr>
<td>pwm_up</td>
<td>The PWM value for soft up limit</td>
</tr>
<tr>
<td>pwm_down</td>
<td>The PWM value for soft down limit</td>
</tr>
<tr>
<td>deg_left</td>
<td>The degree value for soft left limit</td>
</tr>
<tr>
<td>deg_right</td>
<td>The degree value for soft right limit</td>
</tr>
<tr>
<td>deg_up</td>
<td>The degree value for soft up limit</td>
</tr>
<tr>
<td>deg_down</td>
<td>The degree value for soft down limit</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following file:

- shore-systems/lpard/controller/lpardController.py

8.9 lpard::controller::lpardControllerThread::ControllerThread Class Reference

Public Member Functions

- def run

Public Attributes

- controller

The documentation for this class was generated from the following file:

- shore-systems/lpard/controller/lpardControllerThread.py

8.10 FrameldResponse Class Reference

#include <XBee.h>

Inheritance diagram for FrameldResponse:
8.11 GPS_NMEA_Class Class Reference

Public Member Functions

- uint8_t getFrameId ()

8.10.1 Detailed Description

This class is extended by all Responses that include a frame id

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.11 GPS_NMEA_Class Class Reference

Public Member Functions

- void Init ()
- void Read ()

Public Attributes

- long Time
- long Latitude
- long Longitude
- long Altitude
- long Ground_Speed
- long Speed_3d
- long Ground_Course
- uint8_t Type
- uint8_t NumSats
- uint8_t Fix
- uint8_t Quality
The documentation for this class was generated from the following files:

- `boat-systems/drone_firmware/libraries/GPS_NMEA/GPS_NMEA.h`
- `boat-systems/drone_firmware/libraries/GPS_NMEA/GPS_NMEA.cpp`

8.12 guiAbout::GuiAbout Class Reference

Controlling class for the About dialog.

Public Member Functions

- `def __init__`
  
  Constructor for the about dialog.

- `def showGui`
  
  Make the dialog visible.

Public Attributes

- `ui`
  
  The UI object for the about dialog.

8.12.1 Detailed Description

Controlling class for the About dialog.

Author

Mike Rupolo

Date

22 Mar 2012

8.12.2 Constructor & Destructor Documentation

8.12.2.1 `def guiAbout::GuiAbout::__init__ ( self )`

Constructor for the about dialog.
8.12.3 Member Function Documentation

8.12.3.1 def guiAbout::GuiAbout::showGui ( self )

Make the dialog visible.

8.12.4 Member Data Documentation

8.12.4.1 guiAbout::GuiAbout::ui

The UI object for the about dialog.

The documentation for this class was generated from the following file:

- shore-systems/lpard/gui/guiAbout.py

8.13 guiMainPage::GuiMainPage Class Reference

Public Member Functions

- def __init__
- def showGui

Public Attributes

- ui
- searchTimer
- updateTimer
- controller
- record
- script
- newSession
- about
- boatHeight
- boatWidth
- boat
- scene
- frameHeight
- frameWidth
- divisionX
- gridX
- divisionY
- gridY
- x
- y
• xl
• yl
• xb
• yb
• var
• aVar
• positionsX
• positionsY
• messageQueue
• ready
• value6

Testing Setting up Tables ####.
• value7
• value8
• value9
• payloadList
• xcorr
• ycorr
• pen

The documentation for this class was generated from the following file:

• shore-systems/lpard/gui/guiMainPage.py


Public Member Functions

• def __init__
• def showGui

Public Attributes

• controller
• dTimer
• ui

The documentation for this class was generated from the following file:

• shore-systems/lpard/gui/guiManualPositioning.py
8.15 guiNewSession::GuiNewSession Class Reference

Public Member Functions

- def __init__
- def showGui

Public Attributes

- controller
- dTimer
- uTimer
- ui
- manual

The documentation for this class was generated from the following file:

- shore-systems/lpard/gui/guiNewSession.py

8.16 guiRecord::GuiRecord Class Reference

Public Member Functions

- def __init__
- def showGui

Public Attributes

- ui
- controller

The documentation for this class was generated from the following file:

- shore-systems/lpard/gui/guiRecord.py

8.17 guiScript::GuiScript Class Reference

The GuiScript class is responsible for the creating, reading and writing of navigation scripts.
Public Member Functions

- **def __init__**
  Constructor for the scripting class.

- **def keyPressEvent**
  This method handles key press events.

- **def showGui**
  Method to show the window. The method will initially show the GuiScript dialog box.

Public Attributes

- **controller**
  The I2C controller.

- **ui**
  The UI object being controlled.

- **scriptList**
  List holding the script modes.

- **i**
  ?

- **file_opt**
  Holds options for opening or saving a file.

### 8.17.1 Detailed Description

The GuiScript class is responsible for the creating, reading and writing of navigation scripts.

### 8.17.2 Constructor & Destructor Documentation

#### 8.17.2.1 def guiScript::GuiScript::__init__ ( self, controller )

Constructor for the scripting class.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The pointer to the class instance.</td>
</tr>
<tr>
<td>controller</td>
<td>Underlying handler for commands external to lpardGUI</td>
</tr>
</tbody>
</table>

### 8.17.3 Member Function Documentation

#### 8.17.3.1 def guiScript::GuiScript::keyPressEvent ( self, e )

This method handles key press events.
This method handles any key press events while the window is open. As of now its only function is to check for the ‘delete’ key. Can be expanded using ‘if statements’ to encompass other keys if desired

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The pointer to the class.</td>
</tr>
<tr>
<td>e</td>
<td>The key event which occurred</td>
</tr>
</tbody>
</table>

8.17.3.2

```python
def guiScript::GuiScript::showGui ( self )
```

Method to show the window The method will initially show the GuiScript dialog box.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The pointer to the class.</td>
</tr>
</tbody>
</table>

8.17.4 Member Data Documentation

8.17.4.1

**guiScript::GuiScript::controller**

The I2C controller.

8.17.4.2

**guiScript::GuiScript::scriptList**

List holding the script modes.

The documentation for this class was generated from the following file:

- shore-systems/lpard/gui/guiScript.py

8.18 HMC5883L Class Reference

**Public Member Functions**

- **MagnetometerRaw**
  - **ReadRawAxis**
- **MagnetometerScaled**
  - **ReadScaledAxis**
- int **SetMeasurementMode** (uint8_t mode)
- int **SetScale** (float gauss)
- char ∗ **GetErrorText** (int errorCode)

**Protected Member Functions**

- void **Write** (int address, int byte)
- uint8_t ∗ **Read** (int address, int length)
The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/HMC5883L/HMC5883L.h
- boat-systems/drone_firmware/libraries/HMC5883L/HMC5883L.cpp

8.19 lpard::interface::lpardI2C::I2C2PC MockObj Class Reference

Public Member Functions

- def __init__
- def write
- def read
- def close

Public Attributes

- verbose
- cmds
- last_cmd
- timeout

The documentation for this class was generated from the following file:

- shore-systems/lpard/interface/lpardI2C.py

8.20 lpard::interface::lpardI2C::I2CInterface Class Reference

Public Member Functions

- def __init__
  Constructor.
- def __del__
- def status
  Gets sensor mode info.
- def data
  Tells sensor to ping and then gets relevent control data.
- def AZEL_Mode
  Toggles AZ/EL Positioner mode between Manual and Tracking.
- def US_Mode
  Toggles Ultrasonic sensor mode between Receive and Transmit.
- def changeAZ
  Changes the Azimuth positon to the value of pwmVal.
- def changeEL
  Changes the Elevation positon to the value of pwmVal.
Public Attributes

- i2c2pc
- config
- verbose

Static Public Attributes

- sensorData = data
- sensorAZEL_Mode = AZEL_Mode
- sensorUS_Mode = US_Mode
- sensorStatus = status

8.20.1 Detailed Description

I2CInterface handles the interface to an AZEL Positioner through the I2C 2 PC Adapter. This class uses a configuration file to take commands and formulate I2C transmissions for them.

Compatible with firmware version 1.02

Bugs:
- No firmware checks for written PWM values (?)

8.20.2 Constructor & Destructor Documentation

8.20.2.1 def lpard::interface::lpardI2C::I2CInterface::init ( self, I2C_port, protocolConfig, verbose = False )

Constructor.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The pointer to the class instance.</td>
</tr>
<tr>
<td>I2C_port</td>
<td>A PySerial object which interfaces with the I2C 2 PC adapter. Default baud rate should be 57600. Use utils.find_ports to get the PySerial object for the I2C port.</td>
</tr>
<tr>
<td>protocolConfig</td>
<td>A ConfigParser object which has been loaded with the appropriate configuration file.</td>
</tr>
</tbody>
</table>

Params:
- I2C_port - a PySerial object which interfaces with the I2C 2 PC adapter. Default baud rate should be 57600. Use utils.find_ports to get the PySerial object for the I2C port.
- protocolConfig - a ConfigParser object which has been loaded with the appropriate configuration file.

8.20.3 Member Function Documentation
8.20.3.1  def lpard::interface::lpardI2C::I2CInterface::AZEL_Mode ( self, sensorID )

Toggles AZ/EL Positioner mode between Manual and Tracking.

8.20.3.2  def lpard::interface::lpardI2C::I2CInterface::changeAZ ( self, sensorID, pwmVal )

Changes the Azimuth position to the value of pwmVal.
Limited on firmware from 800-2200.

8.20.3.3  def lpard::interface::lpardI2C::I2CInterface::changeEL ( self, sensorID, pwmVal )

Changes the Elevation position to the value of pwmVal.
Limited on firmware from 600-1600.

8.20.3.4  def lpard::interface::lpardI2C::I2CInterface::data ( self, sensorID )

Tells sensor to ping and then gets relevant control data.
Data format is specified in config.

Data format is as follows:
- Azimuth: PWM freq. in us
- Elevation: PWM freq. in us
- Time: Round trip ping travel time in us

8.20.3.5  def lpard::interface::lpardI2C::I2CInterface::status ( self, sensorID )

Gets sensor mode info.
Returns a dict based on config.
8.21 lpard::logger::lpardLogger::Logger Class Reference

8.20.3.6 def lpard::interface::lpardI2C::I2CInterface::USMode ( self, sensorID )

Toggles Ultrasonic sensor mode between Receive and Transmit.

The documentation for this class was generated from the following file:

• shore-systems/lpard/interface/lpardI2C.py

8.21 lpard::logger::lpardLogger::Logger Class Reference

Public Member Functions

• def __init__
• def printTime
• def printDate
• def ensureDir
• def addLog
• def write

Public Attributes

• ifile
• dictFiles
• echo_out
• log_echo
• session_time

8.21.1 Detailed Description

The Logger class is used by the LPARD system to log various system and sensor output.

All logs can be found in the Logs directory. Each log element also has its own directory within the Logs directory i.e. the system log is found in the directory "Logs/system/"

Finally each new session of logging (each time the program starts) creates a new file of the format: "Logs/<log element name>/<log element name> - <date> <time>"

Methods:

addLog(name, fields) - Adds a new object to be logged
name - Takes the name of the new log element
fields - The name of the separate fields to be logged (e.g for logging position the fields would be x, y)

write(name, data) - Writes given data to the log element
name - The name of the log element to log to
data - The data to be written (e.g for logging position: write('position', 120, 30) NOTE: Can add as many data parameters as necessary

The documentation for this class was generated from the following file:

• shore-systems/lpard/logger/lpardLogger.py

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
8.22 MagnetometerRaw Struct Reference

Public Attributes

- int XAxis
- int YAxis
- int ZAxis

The documentation for this struct was generated from the following file:

- boat-systems/drone_firmware/libraries/HMC5883L/HMC5883L.h

8.23 MagnetometerScaled Struct Reference

Public Attributes

- float XAxis
- float YAxis
- float ZAxis

The documentation for this struct was generated from the following file:

- boat-systems/drone_firmware/libraries/HMC5883L/HMC5883L.h

8.24 guiScript::Mode Class Reference

Public Member Functions

- def __init__
  Constructor for the Mode class.
- def __repr__
  Overridden toString() equivalent.

Public Attributes

- mode
  Identifier for the command mode.
- x
  Either the x-component for NAVIGATE or the HOLD duration.
- y
  Either the y-component for NAVIGATE or None if HOLD.
- record
  The indicator for whether we're recording command execution data.
8.24 guiScript::Mode Class Reference

8.24.1 Constructor & Destructor Documentation

8.24.1.1 def guiScript::Mode::__init__( self, mode, x, y, record )

Constructor for the Mode class.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The pointer to the class instance.</td>
</tr>
<tr>
<td>mode</td>
<td>The identifier of the command mode.</td>
</tr>
<tr>
<td>x</td>
<td>The desired x-component of navigation in NAVIGATE mode or the desired duration of holding in HOLD mode.</td>
</tr>
<tr>
<td>y</td>
<td>The desired y-component of navigation in NAVIGATE mode or NULL in HOLD mode.</td>
</tr>
<tr>
<td>record</td>
<td>The indicator for whether or not to record all data during the execution of the desired command.</td>
</tr>
</tbody>
</table>

8.24.2 Member Function Documentation

8.24.2.1 def guiScript::Mode::__repr__( self )

Overridden toString() equivalent.

Uses the internal _toString method.

8.24.3 Member Data Documentation

8.24.3.1 guiScript::Mode::mode

Identifier for the command mode.

8.24.3.2 guiScript::Mode::record

The indicator for whether we're recording command execution data.

8.24.3.3 guiScript::Mode::x

Either the x-component for NAVIGATE or the HOLD duration.

8.24.3.4 guiScript::Mode::y

Either the y-component for NAVIGATE or None if HOLD.

The documentation for this class was generated from the following file:

- shore-systems/lpard/gui/guiScript.py

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
8.25 ModemStatusResponse Class Reference

#include <XBee.h>

Inheritance diagram for ModemStatusResponse:

![](inheritance_diagram.png)

Public Member Functions

- `uint8_t getStatus()`

8.25.1 Detailed Description

Represents a Modem Status RX packet

The documentation for this class was generated from the following files:

- `boat-systems/drone_firmware/libraries/XBee/XBee.h`
- `boat-systems/drone_firmware/libraries/XBee/XBee.cpp`

8.26 nameValueObj::NameValueObj Class Reference

Public Member Functions

- `def __init__`
- `def returnName`
- `def returnValue`

Public Attributes

- `name`
- `value`

The documentation for this class was generated from the following file:

- `shore-systems/lpard/gui/nameValueObj.py`
# PayloadRequest Class Reference

```
#include <XBee.h>
```

Inheritance diagram for PayloadRequest:

```
XBeeRequest
```

```
PayloadRequest
```

```
Tx16Request  Tx64Request  ZBTxRequest
```

Public Member Functions

- `PayloadRequest` (uint8_t apiId, uint8_t frameId, uint8_t *payload, uint8_t payloadLength)
- `uint8_t *getPayload ()`
- `void setPayload (uint8_t *payloadPtr)`
- `uint8_t getPayloadLength ()`
- `void setPayloadLength (uint8_t payloadLength)`

8.27.1 Detailed Description

All TX packets that support payloads extend this class.

8.27.2 Member Function Documentation

8.27.2.1 `uint8_t PayloadRequest::getPayload ( )`

Returns the payload of the packet, if not null.

8.27.2.2 `uint8_t PayloadRequest::getPayloadLength ( )`

Returns the length of the payload array, as specified by the user.

8.27.2.3 `void PayloadRequest::setPayload ( uint8_t * payloadPtr )`

Sets the payload array.

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
8.27.2.4 void PayloadRequest::setPayloadLength ( uint8_t payloadLength )

Sets the length of the payload to include in the request. For example if the payload array is 50 bytes and you only want the first 10 to be included in the packet, set the length to 10. Length must be <= to the array length.

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.28 RemoteAtCommandRequest Class Reference

#include <XBee.h>

Inheritance diagram for RemoteAtCommandRequest:

```
  XBeeRequest
    AtCommandRequest
      RemoteAtCommandRequest
```

Public Member Functions

- RemoteAtCommandRequest (uint16_t remoteAddress16, uint8_t *command, uint8_t *commandValue, uint8_t commandValueLength)
- RemoteAtCommandRequest (uint16_t remoteAddress16, uint8_t *command)
- RemoteAtCommandRequest (XBeeAddress64 &remoteAddress64, uint8_t *command, uint8_t *commandValue, uint8_t commandValueLength)
- RemoteAtCommandRequest (XBeeAddress64 &remoteAddress64, uint8_t *command)
- uint16_t getRemoteAddress16 ()
- void setRemoteAddress16 (uint16_t remoteAddress16)
- XBeeAddress64 & getRemoteAddress64 ()
- void setRemoteAddress64 (XBeeAddress64 &remoteAddress64)
- bool getApplyChanges ()
- void setApplyChanges (bool applyChanges)
- uint8_t getFrameData (uint8_t pos)
- uint8_t getFrameDataLength ()

Static Public Attributes

- static XBeeAddress64 broadcastAddress64 = XBeeAddress64(0x0, BROADCAST_ADDRESS)
8.28 RemoteAtCommandRequest Class Reference

8.28.1 Detailed Description

Represents an Remote AT Command TX packet. The command is used to configure a remote XBee radio.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 RemoteAtCommandRequest::RemoteAtCommandRequest ( uint16_t remoteAddress16, uint8_t *command, uint8_t *commandValue, uint8_t commandValueLength )

Creates a RemoteAtCommandRequest with 16-bit address to set a command. 64-bit address defaults to broadcast and applyChanges is true.

8.28.2.2 RemoteAtCommandRequest::RemoteAtCommandRequest ( uint16_t remoteAddress16, uint8_t *command )

Creates a RemoteAtCommandRequest with 16-bit address to query a command. 64-bit address defaults to broadcast and applyChanges is true.

8.28.2.3 RemoteAtCommandRequest::RemoteAtCommandRequest ( XBeeAddress64 &remoteAddress64, uint8_t *command, uint8_t *commandValue, uint8_t commandValueLength )

Creates a RemoteAtCommandRequest with 64-bit address to set a command. 16-bit address defaults to broadcast and applyChanges is true.

8.28.2.4 RemoteAtCommandRequest::RemoteAtCommandRequest ( XBeeAddress64 &remoteAddress64, uint8_t *command )

Creates a RemoteAtCommandRequest with 16-bit address to query a command. 16-bit address defaults to broadcast and applyChanges is true.

8.28.3 Member Function Documentation

8.28.3.1 uint8_t RemoteAtCommandRequest::getFrameData ( uint8_t pos ) [virtual]

Starting after the frame id (pos = 0) and up to but not including the checksum. Note: Unlike Digi’s definition of the frame data, this does not start with the API ID. The reason for this is the API ID and Frame ID are common to all requests, whereas my definition of frame data is only the API specific data.

Reimplemented from AtCommandRequest.
8.28.3.2  uint8_t RemoteAtCommandRequest::getFrameDataLength ( ) [virtual]

Returns the size of the api frame (not including frame id or api id or checksum).
Reimplemented from AtCommandRequest.

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.29  RemoteAtCommandResponse Class Reference

#include <XBee.h>

Inheritance diagram for RemoteAtCommandResponse:

```
XBeeResponse

FrameIdResponse

AtCommandResponse

RemoteAtCommandResponse
```

Public Member Functions

- uint8_t ∗ getCommand ( )
- uint8_t getStatus ( )
- uint8_t ∗ getValue ( )
- uint8_t getValueLength ( )
- uint16_t getRemoteAddress16 ( )
- XBeeAddress64 & getRemoteAddress64 ( )
- bool isOk ( )

8.29.1  Detailed Description

Represents a Remote AT Command RX packet

8.29.2  Member Function Documentation
8.29 RemoteAtCommandResponse Class Reference

8.29.1 uint8_t *RemoteAtCommandResponse::getCommand ( )

Returns an array containing the two character command
Reimplemented from AtCommandResponse.

8.29.2.2 uint16_t RemoteAtCommandResponse::getRemoteAddress16 ( )

Returns the 16-bit address of the remote radio

8.29.2.3 XBeeAddress64 & RemoteAtCommandResponse::getRemoteAddress64 ( )

Returns the 64-bit address of the remote radio

8.29.2.4 uint8_t RemoteAtCommandResponse::getStatus ( )

Returns the command status code. Zero represents a successful command
Reimplemented from AtCommandResponse.

8.29.2.5 uint8_t *RemoteAtCommandResponse::getValue ( )

Returns an array containing the command value. This is only applicable to query com-
mands.
Reimplemented from AtCommandResponse.

8.29.2.6 uint8_t RemoteAtCommandResponse::getValueLength ( )

Returns the length of the command value array.
Reimplemented from AtCommandResponse.

8.29.2.7 bool RemoteAtCommandResponse::isOk ( )

Returns true if command was successful
Reimplemented from AtCommandResponse.

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp
8.30 Rx16IoSampleResponse Class Reference

Inheritance diagram for Rx16IoSampleResponse:

```
    XBeeResponse
       |                  
    RxDataResponse
       |                  
    RxResponse
       |        
  RxIoSampleBaseResponse
       |        
Rx16IoSampleResponse
```

Public Member Functions

- `uint16_t getRemoteAddress16 ()`
- `uint8_t getRssiOffset ()`

The documentation for this class was generated from the following files:

- `boat-systems/drone_firmware/libraries/XBee/XBee.h`
- `boat-systems/drone_firmware/libraries/XBee/XBee.cpp`

8.31 Rx16Response Class Reference

```cpp
#include <XBee.h>
```

Inheritance diagram for Rx16Response:

```
    XBeeResponse
       |                  
    RxDataResponse
       |                  
    RxResponse
       |                    
Rx16Response
```

The documentation for this class was generated from the following files:

- `boat-systems/drone_firmware/libraries/XBee/XBee.h`
- `boat-systems/drone_firmware/libraries/XBee/XBee.cpp`
8.32 Rx64IoSampleResponse Class Reference

Public Member Functions

• uint8_t getRssiOffset ()
• uint16_t getRemoteAddress16 ()

Protected Attributes

• uint16_t _remoteAddress

8.31.1 Detailed Description

Represents a Series 1 16-bit address RX packet

The documentation for this class was generated from the following files:

• boat-systems/drone_firmware/libraries/XBee/XBee.h
• boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.32 Rx64IoSampleResponse Class Reference

Inheritance diagram for Rx64IoSampleResponse:

```
XBeResponse
   ↓
RxDataResponse
   ↓
RxResponse
   ↓
RxIoSampleBaseResponse
   ↓
Rx64IoSampleResponse
```

Public Member Functions

• XBeeAddress64 & getRemoteAddress64 ()
• uint8_t getRssiOffset ()

The documentation for this class was generated from the following files:

• boat-systems/drone_firmware/libraries/XBee/XBee.h
• boat-systems/drone_firmware/libraries/XBee/XBee.cpp
8.33  Rx64Response Class Reference

#include <XBee.h>

Inheritance diagram for Rx64Response:

```
#include <XBee.h>

Inheritance diagram for Rx64Response:

Class Diagram:
- Rx64Response
  - RxResponse
  - RxDataResponse
  - XBeeResponse

Public Member Functions

- uint8_t getRssiOffset ()
- XBeeAddress64 & getRemoteAddress64 ()

8.33.1  Detailed Description

Represents a Series 1 64-bit address RX packet

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.34  RxDataResponse Class Reference

#include <XBee.h>

Inheritance diagram for RxDataResponse:

Class Diagram:
- RxDataResponse
  - RxResponse
  - XBeeResponse

Public Member Functions

- XBeeAddress64 & getRemoteAddress64 ()

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp
8.34 RxDataResponse Class Reference

Public Member Functions

• uint8_t getData (int index)
• uint8_t * getData ()
• virtual uint8_t getDataLength ()=0
• virtual uint8_t getDataOffset ()=0

8.34.1 Detailed Description

Common functionality for both Series 1 and 2 data RX data packets

8.34.2 Member Function Documentation

8.34.2.1 uint8_t RxDataResponse::getData ( int index )

Returns the specified index of the payload. The index may be 0 to getDataLength() - 1
This method is deprecated; use uint8_t * getData()

8.34.2.2 uint8_t * RxDataResponse::getData ( )

Returns the payload array. This may be accessed from index 0 to getDataLength() - 1

8.34.2.3 virtual uint8_t RxDataResponse::getDataLength ( ) [pure virtual]

Returns the length of the payload
Implemented in ZBRxResponse, and RxResponse.

8.34.2.4 virtual uint8_t RxDataResponse::getDataOffset ( ) [pure virtual]

Returns the position in the frame data where the data begins
Implemented in ZBRxResponse, and RxResponse.

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

## 8.35 RxIoSampleBaseResponse Class Reference

```
#include <XBee.h>
```

Inheritance diagram for RxIoSampleBaseResponse:

```
+-----------------------------+   +-----------------------------+
| RxIoSampleBaseResponse      | -->| RxResponse                  |
|                             |     | RxDataResponse              |
|                             |     | RXResponse                  |
|                             | +---+--------------------------+
|                             |     | Rx16IoSampleResponse        |
|                             |     | Rx64IoSampleResponse        |
```

### Public Member Functions

- `uint8_t getSampleSize ()`
- `bool containsAnalog ()`
- `bool containsDigital ()`
- `bool isAnalogEnabled (uint8_t pin)`
- `bool isDigitalEnabled (uint8_t pin)`
- `uint16_t getAnalog (uint8_t pin, uint8_t sample)`
- `bool isDigitalOn (uint8_t pin, uint8_t sample)`
- `uint8_t getSampleOffset ()`

### 8.35.1 Detailed Description

Represents a Series 1 RX I/O Sample packet

### 8.35.2 Member Function Documentation

#### 8.35.2.1 uint16_t RxIoSampleBaseResponse::getAnalog ( uint8_t pin, uint8_t sample )

Returns the 10-bit analog reading of the specified pin. Valid pins include ADC:0-5. Sample index starts at 0
8.36 RxResponse Class Reference

8.35.2.2 uint8_t RxIoSampleBaseResponse::getSampleSize()

Returns the number of samples in this packet

8.35.2.3 bool RxIoSampleBaseResponse::isAnalogEnabled(uint8_t pin)

Returns true if the specified analog pin is enabled

8.35.2.4 bool RxIoSampleBaseResponse::isDigitalEnabled(uint8_t pin)

Returns true if the specified digital pin is enabled

8.35.2.5 bool RxIoSampleBaseResponse::isDigitalOn(uint8_t pin, uint8_t sample)

Returns true if the specified pin is high/on. Valid pins include DIO:0-8. Sample index starts at 0

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.36 RxResponse Class Reference

#include <XBee.h>

Inheritance diagram for RxResponse:

```
<table>
<thead>
<tr>
<th>XBeResponse</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxDataResponse</td>
</tr>
<tr>
<td>RxResponse</td>
</tr>
<tr>
<td>Rx16Response</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Rx16IoSampleResponse</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- uint8_t getRssi()
- uint8_t getOption()
- bool isAddressBroadcast()
- bool isPanBroadcast()
- uint8_t getDataLength()
- uint8_t getDataOffset()
- virtual uint8_t getRssiOffset() = 0

8.36.1 Detailed Description

Represents a Series 1 RX packet

8.36.2 Member Function Documentation

8.36.2.1 uint8_t RxResponse::getDataLength() [virtual]

Returns the length of the payload
Implements RxDataResponse.

8.36.2.2 uint8_t RxResponse::getDataOffset() [virtual]

Returns the position in the frame data where the data begins
Implements RxDataResponse.

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.37 Tx16Request Class Reference

#include <XBee.h>

Inheritance diagram for Tx16Request:

```
XBeeRequest

PayloadRequest

Tx16Request
```

Public Member Functions

- Tx16Request(uint16_t addr16, uint8_t option, uint8_t *payload, uint8_t payloadLength, uint8_t frameld)

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
8.37 Tx16Request Class Reference

- `Tx16Request (uint16_t addr16, uint8_t *payload, uint8_t payloadLength)`
- `Tx16Request ()`
- `uint16_t getAddress16 ()`
- `void setAddress16 (uint16_t addr16)`
- `uint8_t getOption ()`
- `void setOption (uint8_t option)`
- `uint8_t getFrameData (uint8_t pos)`
- `uint8_t getFrameDataLength ()`

8.37.1 Detailed Description

Represents a Series 1 TX packet that corresponds to Api id: TX_16_REQUEST

Be careful not to send a data array larger than the max packet size of your radio. This class does not perform any validation of packet size and there will be no indication if the packet is too large, other than you will not get a TX Status response. The datasheet says 100 bytes is the maximum, although that could change in future firmware.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 `Tx16Request::Tx16Request ( uint16_t addr16, uint8_t *payload, uint8_t payloadLength )`

Creates a Unicast `Tx16Request` with the ACK option and DEFAULT_FRAME_ID

8.37.2.2 `Tx16Request::Tx16Request ( )`

Creates a default instance of this class. At a minimum you must specify a payload, payload length and a destination address before sending this request.

8.37.3 Member Function Documentation

8.37.3.1 `uint8_t Tx16Request::getFrameData ( uint8_t pos ) [virtual]`

Starting after the frame id (pos = 0) and up to but not including the checksum Note: Unlike Digi's definition of the frame data, this does not start with the API ID. The reason for this is the API ID and Frame ID are common to all requests, whereas my definition of frame data is only the API specific data.

Implements `XBeeRequest`.

8.37.3.2 `uint8_t Tx16Request::getFrameDataLength ( ) [virtual]`

Returns the size of the api frame (not including frame id or api id or checksum).

Implements `XBeeRequest`.

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.38  Tx64Request Class Reference

#include <XBee.h>

Inheritance diagram for Tx64Request:

```
   XBeeRequest
      |          |
      |          |
      PayloadRequest
      |          |
         Tx64Request
```

Public Member Functions

- **Tx64Request** (XBeeAddress64 &addr64, uint8_t option, uint8_t *payload, uint8_t payloadLength, uint8_t frameId)
- **Tx64Request** (XBeeAddress64 &addr64, uint8_t *payload, uint8_t payloadLength)
- **Tx64Request** ()
- XBeeAddress64 & getAddress64 ()
- void setAddress64 (XBeeAddress64 &addr64)
- uint8_t &getOption ()
- void setOption (uint8_t option)
- uint8_t getFrameData (uint8_t pos)
- uint8_t getFrameDataLength ()

8.38.1  Detailed Description

Represents a Series 1 TX packet that corresponds to Api Id: TX_64_REQUEST

Be careful not to send a data array larger than the max packet size of your radio. This class does not perform any validation of packet size and there will be no indication if the packet is too large, other than you will not get a TX Status response. The datasheet says 100 bytes is the maximum, although that could change in future firmware.

8.38.2  Constructor & Destructor Documentation
8.38.2.1  Tx64Request::Tx64Request ( XBeeAddress64 & addr64, uint8_t * payload, uint8_t payloadLength )

Creates a unicast Tx64Request with the ACK option and DEFAULT_FRAME_ID

8.38.2.2  Tx64Request::Tx64Request ( )

Creates a default instance of this class. At a minimum you must specify a payload, payload length and a destination address before sending this request.

8.38.3  Member Function Documentation

8.38.3.1  uint8_t Tx64Request::getFrameData ( uint8_t pos ) [virtual]

Starting after the frame id (pos = 0) and up to but not including the checksum  
Note: Unlike Digi's definition of the frame data, this does not start with the API ID. The reason for this is the API ID and Frame ID are common to all requests, whereas my definition of frame data is only the API specific data.

Implements XBeeRequest.

8.38.3.2  uint8_t Tx64Request::getFrameDataLength ( ) [virtual]

Returns the size of the api frame (not including frame id or api id or checksum).

Implements XBeeRequest.

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.39  TxStatusResponse Class Reference

#include <XBee.h>

Inheritance diagram for TxStatusResponse:

```
XBeeResponse
  |
  FrameIdResponse
  |
TxStatusResponse
```

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
Public Member Functions

- uint8_t getStatus ()
- bool isSuccess ()

8.39.1 Detailed Description

Represents a Series 1 TX Status packet

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.40 GUI::Ui about Class Reference

Public Member Functions

- def setupUi
- def retranslateUi

Public Attributes

- frame_6
- line_4
discardbutton
notes
frame_4
lafayette
created
engineers
software
team
recordUserName
class_name
members
recordUserName_4
date
frame_5
team
line_3
about_lpard
graphicsView

The documentation for this class was generated from the following file:

- shore-systems/lpard/gui/GUI_about_ui.py
Public Member Functions

- def setupUi
- def retranslateUi

Public Attributes

- centralwidget
- frame
- groupBox
- scriptButton
- recordButton
- label_4
- yUpdate
- label_7
- stopButton
- label_6
- line_2
- returnToOrigin
- line_3
- doubleSpinBox_x
- doubleSpinBox_y
- scriptButton_2
- groupBox_4
- PayloadSensorTable
- groupBox_5
- boatStatTable
- label_5
- boatStatusLabel
- frame_2
- groupBox_3
- systemLog
- graph
- zoom
- label_8
- line
- menubar
- menuFile
- menuSettings
- menuGraph_Settings
- menuHelp
- menuCalibrate
- actionBaseline
- actionNew_Session
• actionNew_Origin
• actionAbout
• actionUser_Manual
• actionClear_Graph

The documentation for this class was generated from the following file:

• shore-systems/lpard/gui/GUI_mainPageReturn_ui.py

8.42 GUI_manualPositioning_ui::Ui_manualPositioning Class Reference

Public Member Functions

• def setupUi
• def retranslateUi

Public Attributes

• frame_5
• doneButton
• cancelButton
• buttonUp
• buttonDown
• buttonRight
• buttonLeft
• frame
• line_3
• label_8

The documentation for this class was generated from the following file:

• shore-systems/lpard/gui/GUI_manualPositioning_ui.py

8.43 GUI_newSession_ui::Ui_newSession Class Reference

Public Member Functions

• def setupUi
• def retranslateUi
Public Attributes

- frame_6
- calibrateButton
- closeButton
- groupBox
- sosNew
- sosDefault
- sosPrevious
- distanceLabel
- baselineEdit
- groupBox_2
- label_3
- label_2
- userName
- location
- simulationModeBox
- label_4
- frame_5
- line_3
- label_7

The documentation for this class was generated from the following file:

- shore-systems/lpard/gui/GUI_newSession_ui.py

8.44 GUI_record_ui::Ui_recordData Class Reference

Public Member Functions

- def setupUi
- def retranslateUi

Public Attributes

- frame_6
- line_4
- discardbutton
- saveButton
- label_11
- observationNotes
- frame_4
- label_5
- label_6
- label_9
• label_10
• recordLength
• recordSensors
• recordTime
• recordDate
• label_12
• label_13
• recordUserName
• recordLocation
• frame_5
• line_3
• label_7

The documentation for this class was generated from the following file:

• shore-systems/lpard/gui/GUI_record_ui.py

8.45  GUI_script_ui::Ui_Script Class Reference

Public Member Functions

• def setupUi
• def retranslateUi

Public Attributes

• frame_5
• line_3
• label_7
• frame_6
• verticalLayoutWidget
• verticalLayout
• horizontalLayout_2
• edit1_mode
• edit2_x
• edit3_y
• edit4
• edit5
• horizontalLayout
• comboBox_mode
• doubleSpinBox_x
• doubleSpinBox_y
• checkBox_record
• pushButton_insert
• listWidget
The documentation for this class was generated from the following file:

- shore-systems/lpard/gui/GUI_script_ui.py

### 8.46 XBee Class Reference

```cpp
#include <XBee.h>
```

#### Public Member Functions

- void readPacket()
- bool readPacket(int timeout)
- void readPacketUntilAvailable()
- void begin(long baud)
- void getResponse(XBeeResponse &response)
- XBeeResponse &getResponse()
- void send(XBeeRequest &request)
- uint8_t getNextFrameId()
- void setSerial(HardwareSerial &serial)

### 8.46.1 Detailed Description

Primary interface for communicating with an XBee Radio. This class provides methods for sending and receiving packets with an XBee radio via the serial port. The XBee radio must be configured in API (packet) mode (AP=2) in order to use this software.

Since this code is designed to run on a microcontroller, with only one thread, you are responsible for reading the data off the serial buffer in a timely manner. This involves a call to a variant of readPacket(...). If your serial port is receiving data faster than you are reading, you can expect to lose packets. Arduino only has a 128 byte serial buffer so it can easily overflow if two or more packets arrive without a call to readPacket(...).

In order to conserve resources, this class only supports storing one response packet in memory at a time. This means that you must fully consume the packet prior to calling readPacket(...), because calling readPacket(...) overwrites the previous response.

This class creates an array of size MAX_FRAME_DATA_SIZE for storing the response packet. You may want to adjust this value to conserve memory.

### Author

Andrew Rapp

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
8.46.2  Member Function Documentation

8.46.2.1  void XBee::begin ( long baud )

Starts the serial connection at the supplied baud rate

8.46.2.2  uint8_t XBee::getNextFrameId ( )

Returns a sequential frame id between 1 and 255

8.46.2.3  XBeeResponse & XBee::getResponse ( )

Returns a reference to the current response Note: once readPacket is called again this
response will be overwritten!

8.46.2.4  bool XBee::readPacket ( int timeout )

Waits a maximum of timeout milliseconds for a response packet before timing out; re-
turns true if packet is read. Returns false if timeout or error occurs.

8.46.2.5  void XBee::readPacket ( )

Reads all available serial bytes until a packet is parsed, an error occurs, or the buffer
is empty. You may call xbee.getResponse().isAvailable() after calling this method to
determine if a packet is ready, or xbee.getResponse().isError() to determine if a error
occurred.

This method should always return quickly since it does not wait for serial data to arrive.
You will want to use this method if you are doing other timely stuff in your loop, where
a delay would cause problems. NOTE: calling this method resets the current response,
so make sure you first consume the current response

8.46.2.6  void XBee::readPacketUntilAvailable ( )

Reads until a packet is received or an error occurs. Caution: use this carefully since
if you don’t get a response, your Arduino code will hang on this call forever!! often it’s
better to use a timeout: readPacket(int)

8.46.2.7  void XBee::send ( XBeeRequest & request )

Sends a XBeeRequest (TX packet) out the serial port
8.46.2.8 void XBee::setSerial ( HardwareSerial & serial )

Specify the serial port. Only relevant for Arduinos that support multiple serial ports (e.g. Mega)

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.47 XBeeAddress Class Reference

Inheritance diagram for XBeeAddress:

```
XBeeAddress
   
XBeeAddress64
```

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.48 XBeeAddress64 Class Reference

```c
#include <XBee.h>
```

Inheritance diagram for XBeeAddress64:

```
XBeeAddress
   
XBeeAddress64
```

Public Member Functions

- XBeeAddress64 (uint32_t msb, uint32_t lsb)
- uint32_t getMsb ()
- uint32_t getLsb ()
- void setMsb (uint32_t msb)
- void setLsb (uint32_t lsb)
8.48.1 Detailed Description

Represents a 64-bit XBee Address

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.49 lpard::interface::lpardXBee::XbeeInterface Class Reference

Public Member Functions

- def __init__
  Constructor for XbeeInterface.
- def sendMessage
  Sends a packet with the specified message.
- def getMessage
  Gets the message from the next available packet.
- def closePort
  Closes the serial port to the XBee.
- def getPacket

Public Attributes

- config
- inQ
- outQ
- boat

8.49.1 Detailed Description

The XbeeInterface class is used by the LPARD system to connect to the boat system through the XBee RF transceivers.

8.49.2 Constructor & Destructor Documentation

8.49.2.1 def lpard::interface::lpardXBee::XbeeInterface::__init__( self, zigbeePort, protocolConfig, simulate = False )

Constructor for XbeeInterface.

Parameters
8.50 XBeeRequest Class Reference

self | The pointer to the class instance.
zigbeePort | A PySerial object which interfaces with the XBee. Default baud rate should be 9600. Use utils.find_ports to get the PySerial object for the XBee port.
protocolConfig | A ConfigParser object which has been loaded with the appropriate configuration file.
simulate | True if BoatSim should be used instead of boat. Default is False

8.49.3 Member Function Documentation

8.49.3.1 def lpard::interface::lpardXBee::XbeeInterface::closePort ( self )
Closes the serial port to the XBee.
This must be called when the program is being shutdown.

Parameters

self | The pointer to the class instance.

8.49.3.2 def lpard::interface::lpardXBee::XbeeInterface::getMessage ( self )
Gets the message from the next available packet.

Parameters

self | The pointer to the class instance.

8.49.3.3 def lpard::interface::lpardXBee::XbeeInterface::sendMessage ( self, message )
Sends a packet with the specified message.

Parameters

self | The pointer to the class instance.
message | A string which contains the packet data to send.

The documentation for this class was generated from the following file:

- shore-systems/lpard/interface/lpardXBee.py

8.50 XBeeRequest Class Reference

#include <XBee.h>

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
Inheritance diagram for XBeeRequest:

```
XBeeRequest
    AtCommandRequest
    PayloadRequest
        RemoteAtCommandRequest
        Tx16Request
        Tx64Request
            ZBTxRequest
```

Public Member Functions

- `XBeeRequest (uint8_t apiId, uint8_t frameId)`
- `void setFrameId (uint8_t frameId)`
- `uint8_t getFrameId ()`
- `uint8_t getApiId ()`
- `virtual uint8_t getFrameData (uint8_t pos)=0`
- `virtual uint8_t getFrameDataLength ()=0`

Protected Member Functions

- `void setApiId (uint8_t apilId)`

8.50.1 Detailed Description

Super class of all XBee requests (TX packets) Users should never create an instance of this class; instead use an subclass of this class It is recommended to reuse Subclasses of the class to conserve memory

This class allocates a buffer to

8.50.2 Constructor & Destructor Documentation

8.50.2.1 `XBeeRequest::XBeeRequest ( uint8_t apilId, uint8_t frameId )`

Constructor TODO make protected

8.50.3 Member Function Documentation

8.50.3.1 `uint8_t XBeeRequest::getApiId ( )`

Returns the API id
8.51 XBeeResponse Class Reference

8.50.3.2 virtual uint8_t XBeeRequest::getFrameData ( uint8_t pos ) [pure virtual]

Starting after the frame id (pos = 0) and up to but not including the checksum Note:
Unlike Digi's definition of the frame data, this does not start with the API ID. The reason
for this is the API ID and Frame ID are common to all requests, whereas my definition
of frame data is only the API specific data.
Implemented in Tx16Request, Tx64Request, ZBTxRequest, AtCommandRequest, and
RemoteAtCommandRequest.

8.50.3.3 virtual uint8_t XBeeRequest::getFrameDataLength ( ) [pure virtual]

Returns the size of the api frame (not including frame id or api id or checksum).
Implemented in Tx16Request, Tx64Request, ZBTxRequest, AtCommandRequest, and
RemoteAtCommandRequest.

8.50.3.4 uint8_t XBeeRequest::getFrameId ( )

Returns the frame id

8.50.3.5 void XBeeRequest::setFrameId ( uint8_t frameId )

Sets the frame id. Must be between 1 and 255 inclusive to get a TX status response.

The documentation for this class was generated from the following files:

• boat-systems/drone_firmware/libraries/XBee/XBee.h
• boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.51 XBeeResponse Class Reference

#include <XBee.h>

Inheritance diagram for XBeeResponse:

Public Member Functions

• XBeeResponse ()

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
Class Documentation

• uint8_t getApiId ()
  • void setApiId (uint8_t apilid)
• uint8_t getMsbLength ()
  • void setMsbLength (uint8_t msbLength)
• uint8_t getLsbLength ()
  • void setLsbLength (uint8_t lsbLength)
• uint8_t getChecksum ()
  • void setChecksum (uint8_t checksum)
• uint8_t getFrameDataLength ()
  • void setFrameData (uint8_t *frameDataPtr)
• uint8_t getFrameLength ()
  • void setFrameLength (uint8_t frameLength)
• uint16_t getPacketLength ()
  • void reset ()
  • void init ()
  • void getZBTxStatusResponse (XBeeResponse &response)
  • void getZBRxResponse (XBeeResponse &response)
  • void getZBRxIoSampleResponse (XBeeResponse &response)
  • void getTxStatusResponse (XBeeResponse &response)
  • void getRx16Response (XBeeResponse &response)
  • void getRx64Response (XBeeResponse &response)
  • void getRx16IoSampleResponse (XBeeResponse &response)
  • void getRx64IoSampleResponse (XBeeResponse &response)
  • void getAtCommandResponse (XBeeResponse &responses)
  • void getRemoteAtCommandResponse (XBeeResponse &response)
  • void getModemStatusResponse (XBeeResponse &response)
  • bool isAvailable ()
  • void setAvailable (bool complete)
  • bool isError ()
  • uint8_t getErrorCode ()
  • void setErrorCode (uint8_t errorCode)

Protected Attributes

• uint8_t * _frameDataPtr

8.51.1 Detailed Description

The super class of all XBee responses (RX packets) Users should never attempt to create an instance of this class; instead create an instance of a subclass It is recommended to reuse subclasses to conserve memory
8.51 XBeeResponse Class Reference

8.51.2 Constructor & Destructor Documentation

8.51.2.1 XBeeResponse::XBeeResponse ( )

Default constructor

Copyright (c) 2009 Andrew Rapp. All rights reserved.

This file is part of XBee-Arduino.

XBee-Arduino is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

XBee-Arduino is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with XBee-Arduino. If not, see <http://www.gnu.org/licenses/>.

8.51.3 Member Function Documentation

8.51.3.1 uint8_t XBeeResponse::getApiId ( )

Returns Api Id of the response

8.51.3.2 void XBeeResponse::getAsCommandResponse ( XBeeResponse & responses )

Call with instance of AtCommandResponse only if getApId() == AT_COMMAND_RESPONSE

8.51.3.3 uint8_t XBeeResponse::getChecksum ( )

Returns the packet checksum

8.51.3.4 uint8_t XBeeResponse::getErrorCode ( )

Returns an error code, or zero, if successful. Error codes include: CHECKSUM_- FAILURE, PACKET_EXCEEDS_BYTE_ARRAY_LENGTH, UNEXPECTED_START_BYTE

8.51.3.5 uint8_t XBeeResponse::getFrameData ( )

Returns the buffer that contains the response. Starts with byte that follows API ID and includes all bytes prior to the checksum Length is specified by getFrameDataLength() Note: Unlike Digi's definition of the frame data, this does not start with the API ID. The
reason for this is all responses include an API ID, whereas my frame data includes only
the API specific data.

8.51.3.6 uint8_t XBeeResponse::getFrameDataLength ( )

Returns the length of the frame data: all bytes after the api id, and prior to the checksum
Note up to release 0.1.2, this was incorrectly including the checksum in the length.

8.51.3.7 uint8_t XBeeResponse::getLsbLength ( )

Returns the LSB length of the packet

8.51.3.8 void XBeeResponse::getModemStatusResponse ( XBeeResponse & response )

Call with instance of ModemStatusResponse only if getApiId() == MODEM_STATUS_- RESPONSE

8.51.3.9 uint8_t XBeeResponse::getMsbLength ( )

Returns the MSB length of the packet

8.51.3.10 uint16_t XBeeResponse::getPacketLength ( )

Returns the length of the packet

8.51.3.11 void XBeeResponse::getRemoteAtCommandResponse ( XBeeResponse & response )

Call with instance of RemoteAtCommandResponse only if getApiId() == REMOTE_AT_- COMMAND_RESPONSE

8.51.3.12 void XBeeResponse::getRx16IoSampleResponse ( XBeeResponse & response )

Call with instance of Rx16IoSampleResponse only if getApiId() == RX_16_IO_RESPONSE

8.51.3.13 void XBeeResponse::getRx16Response ( XBeeResponse & response )

Call with instance of Rx16Response only if getApiId() == RX_16_RESPONSE
8.51 XBeeResponse Class Reference

8.51.3.14 void XBeeResponse::getRx64IoSampleResponse ( XBeeResponse & response )

Call with instance of Rx64IoSampleResponse only if getApild() == RX_64_IO_RESPONSE

8.51.3.15 void XBeeResponse::getRx64Response ( XBeeResponse & response )

Call with instance of Rx64Response only if getApild() == RX_64_RESPONSE

8.51.3.16 void XBeeResponse::getTxStatusResponse ( XBeeResponse & response )

Call with instance of TxStatusResponse only if getApild() == TX_STATUS_RESPONSE

8.51.3.17 void XBeeResponse::getZBRxIoSampleResponse ( XBeeResponse & response )

Call with instance of ZBRxIoSampleResponse class only if getApild() == ZB_IO_SAMPLE_- RESPONSE to populate response

8.51.3.18 void XBeeResponse::getZBRxResponse ( XBeeResponse & response )

Call with instance of ZBRxResponse class only if getApild() == ZB_RX_RESPONSE to populate response

8.51.3.19 void XBeeResponse::getZBTxStatusResponse ( XBeeResponse & response )

Call with instance of ZBTxStatusResponse class only if getApild() == ZB_TX_STATUS_- RESPONSE to populate response

8.51.3.20 void XBeeResponse::init ( )

Initializes the response

8.51.3.21 bool XBeeResponse::isAvailable ( )

Returns true if the response has been successfully parsed and is complete and ready for use

8.51.3.22 bool XBeeResponse::isError ( )

Returns true if the response contains errors
8.51.3.23 void XBeeResponse::reset()

Resets the response to default values

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.52 ZBRxIoSampleResponse Class Reference

#include <XBee.h>

Inheritance diagram for ZBRxIoSampleResponse:

```
ZXbeeResponse
    `-- RxDataResponse
        `-- ZBRxResponse
            `-- ZBRxIoSampleResponse
```

Public Member Functions

- bool containsAnalog()
- bool containsDigital()
- bool isAnalogEnabled(uint8_t pin)
- bool isDigitalEnabled(uint8_t pin)
- uint16_t getAnalog(uint8_t pin)
- bool isDigitalOn(uint8_t pin)
- uint8_t getDigitalMaskMsb()
- uint8_t getDigitalMaskLsb()
- uint8_t getAnalogMask()

8.52.1 Detailed Description

Represents a Series 2 RX I/O Sample packet
8.52.2 Member Function Documentation

8.52.2.1 uint16_t ZBRxIoSampleResponse::getAnalog ( uint8_t pin )

Returns the 10-bit analog reading of the specified pin. Valid pins include ADC:xxx.

8.52.2.2 bool ZBRxIoSampleResponse::isAnalogEnabled ( uint8_t pin )

Returns true if the pin is enabled.

8.52.2.3 bool ZBRxIoSampleResponse::isDigitalEnabled ( uint8_t pin )

Returns true if the pin is enabled.

8.52.2.4 bool ZBRxIoSampleResponse::isDigitalOn ( uint8_t pin )

Returns true if the specified pin is high/on. Valid pins include DIO:xxx.

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.53 ZBRxResponse Class Reference

#include <XBee.h>

Inheritance diagram for ZBRxResponse:

```
XBeeResponse
  |
  v
RxDataResponse
  |
  v
ZBRxResponse
  |
  v
ZBRxIoSampleResponse
```

Public Member Functions

- XBeeAddress64 & getRemoteAddress64 ()
- uint16_t getRemoteAddress16 ()
- uint8_t getOption ()

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
8.53.1 Detailed Description

Represents a Series 2 RX packet.

8.53.2 Member Function Documentation

8.53.2.1 `uint8_t ZBRxResponse::getDataLength ( ) [virtual]`

Returns the length of the payload.
Implements `RxDataResponse`.

8.53.2.2 `uint8_t ZBRxResponse::getDataOffset ( ) [virtual]`

Returns the position in the frame data where the data begins.
Implements `RxDataResponse`.

The documentation for this class was generated from the following files:

- `boat-systems/drone_firmware/libraries/XBee/XBee.h`
- `boat-systems/drone_firmware/libraries/XBee/XBee.cpp`

8.54 ZBTxRequest Class Reference

```cpp
#include <XBee.h>
```

Inheritance diagram for `ZBTxRequest`:

```
XBeeRequest
    |
    |
 PayloadRequest
    |
    |
 ZBTxRequest
```

Public Member Functions

- `ZBTxRequest (XBeeAddress64 &addr64, uint8_t *payload, uint8_t payloadLength)`
- `ZBTxRequest (XBeeAddress64 &addr64, uint16_t addr16, uint8_t broadcastRadius, uint8_t option, uint8_t *payload, uint8_t payloadLength, uint8_t frameId)`
8.54 ZBTxRequest Class Reference

- ZBTxRequest()
- XBeeAddress64 & getAddress64()
- uint16_t getAddress16()
- uint8_t getBroadcastRadius()
- uint8_t getOption()
- void setAddress64(XBeeAddress64 &addr64)
- void setAddress16(uint16_t addr16)
- void setBroadcastRadius(uint8_t broadcastRadius)
- void setOption(uint8_t option)

Protected Member Functions

- uint8_t getFrameData(uint8_t pos)
- uint8_t getFrameDataLength()

8.54.1 Detailed Description

Represents a Series 2 TX packet that corresponds to Api Id: ZB_TX_REQUEST

Be careful not to send a data array larger than the max packet size of your radio. This class does not perform any validation of packet size and there will be no indication if the packet is too large, other than you will not get a TX Status response. The datasheet says 72 bytes is the maximum for ZNet firmware and ZB Pro firmware provides the ATNP command to get the max supported payload size. This command is useful since the maximum payload size varies according to certain settings, such as encryption. ZB Pro firmware provides a PAYLOAD_TOO_LARGE that is returned if payload size exceeds the maximum.

8.54.2 Constructor & Destructor Documentation

8.54.2.1 ZBTxRequest::ZBTxRequest(XBeeAddress64 &addr64, uint8_t *payload, uint8_t payloadLength)

Creates a unicast ZBTxRequest with the ACK option and DEFAULT_FRAME_ID

8.54.2.2 ZBTxRequest::ZBTxRequest()

Creates a default instance of this class. At a minimum you must specify a payload, payload length and a destination address before sending this request.

8.54.3 Member Function Documentation

8.54.3.1 uint8_t ZBTxRequest::getFrameData(uint8_t pos) [protected, virtual]

Starting after the frame id (pos = 0) and up to but not including the checksum Note: Unlike Digi’s definition of the frame data, this does not start with the API ID. The reason
for this is the API ID and Frame ID are common to all requests, whereas my definition of frame data is only the API specific data.

Implements XBeeRequest.

8.54.3.2 uint8_t ZBTxRequest::getFrameDataLength() [protected, virtual]

Returns the size of the api frame (not including frame id or api id or checksum).

Implements XBeeRequest.

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp

8.55 ZBTxStatusResponse Class Reference

#include <XBee.h>

Inheritance diagram for ZBTxStatusResponse:

```
XBeeResponse

FrameIdResponse

ZBTxStatusResponse
```

Public Member Functions

- uint16_t getRemoteAddress()
- uint8_t getTxRetryCount()
- uint8_t getDeliveryStatus()
- uint8_t getDiscoveryStatus()
- bool isSuccess()

8.55.1 Detailed Description

Represents a Series 2 TX status packet

The documentation for this class was generated from the following files:

- boat-systems/drone_firmware/libraries/XBee/XBee.h
- boat-systems/drone_firmware/libraries/XBee/XBee.cpp
Chapter 9

File Documentation

9.1 boat-systems/drone_firmware/drone_firmware.pde File Reference

Main Arduino code file for boat firmware.

```c
#include <Wire.h>
#include <XBee.h>
#include <GPS_NMEA.h>
#include <Comp6DOF_n0m1.h>
#include <HMC5883L.h>
```

Functions

- void setup ()
- void loop ()
- void calibrate ()
- boolean init_compass ()
- void transmit (String str)
- void receivePacket (int timeout)
- void parseIncoming (uint8_t *data)
- void send_status ()
- void check_gps ()
- void check_compass ()
- float nonCompHeading (MagnetometerRaw raw)

Variables

- long timer
  
  Used to track the start time of the most recent loop.
- long diff
Used to determine the time between loops and limit to 1 Hz.

- `const char* DELIM = " "`
  Packet delimiter.
- `XBee xbee = XBee()`
  Our `XBee` object.
- `ZBTxRequest zbTx`
  Our transmit request.
- `XBeeResponse response = XBeeResponse()`
  For incoming packets.
- `ZBRxResponse rx = ZBRxResponse()`
  For incoming ZB packets.
- `ModemStatusResponse msr = ModemStatusResponse()`
  For modem status response.
- `XBeeAddress64 addr64 = XBeeAddress64(0, 0)`
  Coordinator address.
- `Comp6DOF_n0m1 sixDOF`
  Object used for tilt compensation.
- `HMC5883L compass`
  The actual compass.
- `float declination`
  Declination angle for the operating area.
- `int pos [3]`
  Boat positions in the order X, Y, Z in cm.
- `int head`
  Heading in 100ths of degrees.
- `long comp_read`
  Most recent compass reading in 100ths of degrees.
- `int accel_read [3]`
  Most recent accelerometer readings going X, Y, Z.
- `int gyro_read [3]`
  Most recent gyroscope readings going X, Y, Z.
- `int volt_read`
  Subsystem voltage readings.
- `int amp_read`
  Subsystem current readings.
- `int power_read`
  Subsystem power readings.
- `int temp_read`
  Subsystem temperature readings.
- `char ctrl_mode = 'A'`
  Control state where A->auto and M->manual.
- `long gps_time`
  Time from epoch (from the GPS) in ms.
9.1 boat-systems/drone_firmware/drone_firmware.pde File Reference

- long gps_lat
  Latitude multiplied by $10^{-7}$ to clear the decimal.
- long gps_long
  Longitude multiplied by $10^{-7}$ to clear the decimal.
- long gps_speed
  GPS speed over ground field. In m/s * 100?
- long gps_angle
  GPS track angle. Degrees * 100?

9.1.1 Detailed Description

Main Arduino code file for boat firmware. Contains both the setup and loop Arduino functions, as well as convenience methods for XBee communication and boat sensor controls. Note that this is the second code branch - the first has more sensors operating but does not have working XBee communication.

Author
Sam Courtney <courtnes@lafayette.edu>

Version
1.0

Date
2012

Bug
Compass not tilt compensated.
Compass iron offsets infinite loop, and have been disabled.
Doesn’t receive XBee packets properly. Could be a mode issue (needs API=2).
Cannot set declination angle externally.

9.1.2 Function Documentation

9.1.2.1 void calibrate ( )
Method handling calibration.

9.1.2.2 void check_compass ( )
Convenience method that updates compass data and the relevant boat state fields.

9.1.2.3 void check_gps ( )
Convenience method that checks for new GPS data and updates the relevant boat state fields.

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
9.1.2.4 boolean init_compass ( )
Method handling compass initialization. Note that this also handles the calculation of hard iron offsets.

Returns
   true if no error was detected

9.1.2.5 void loop ( )
Main loop for the boat firmware. The ArduPilot automatically loops through this code.

9.1.2.6 float nonCompHeading ( MagnetometerRaw raw )
Convenience method that updates calculates the non-compensated heading value using the HMC5883L compass. In other words, we're assuming the compass is level.

Returns
   the heading value

9.1.2.7 void parseIncoming ( uint8_t * data )
Convenience method that handles parsing of our protocol's packets.

Parameters
   data  Pointer to the frame data that is to be parsed.

9.1.2.8 void receivePacket ( int timeout )
Convenience method allowing reception of an XBee packet.

Parameters
   timeout  The time to wait for an incoming packet

9.1.2.9 void send_status ( )
Convenience method that handles construction of a boat status packet. Also calls the send method.
9.2 shore-systems/AZEL/AZELCode/AZELCode.h File Reference

Parameters

| data | Pointer to the frame data that is to be parsed. |

9.1.2.10 void setup ( )

Setup loop for firmware. Automatically run any time the ArduPilot receives power.

9.1.2.11 void transmit ( String str )

Convenience method allowing transmission to the stored address.

Parameters

| str | The character string to send |

9.2 shore-systems/AZEL/AZELCode/AZELCode.h File Reference

Firmware for the AZ/EL Positioner.

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include "pwm.h"
#include "i2c.h"
#include "ultrasound.h"
#include "tracking.h"
#include <util/delay.h>
```

Defines

- `#define F_CPU 8000000UL`

Enumerations

- `enum clock_div_t {
  clock_div_1 = 0, clock_div_2 = 1, clock_div_4 = 2, clock_div_8 = 3,
  clock_div_16 = 4, clock_div_32 = 5, clock_div_64 = 6, clock_div_128 = 7,
  clock_div_256 = 8, clock_div_1 = 0, clock_div_2 = 1, clock_div_4 = 2,
  clock_div_8 = 3, clock_div_16 = 4, clock_div_32 = 5, clock_div_64 = 6,
  clock_div_128 = 7, clock_div_256 = 8, clock_div_1 = 0, clock_div_2 = 1,
  clock_div_4 = 2, clock_div_8 = 3, clock_div_16 = 4, clock_div_32 = 5,
};`
\texttt{clock\_div\_64 = 6, clock\_div\_128 = 7, clock\_div\_256 = 8 }

Various Clock Divider settings.

9.2.1 Detailed Description

Firmware for the AZ/EL Positioner. This program runs in the ATmega 328 of the AZ/EL Positioner to manage operations. Current features include

1. PWM Drive of the AZ and EL servos
2. Manual joystick mode
3. Automatic analog position input mode
4. Soft limits (Hard Coded)
5. I2C Interface (incomplete)

Author

Chris Nadovich <nadovicc@lafayette.edu>
David Salter <salterd@lafayette.edu>

Version

1.02

Date

31 March 2012

Bug

Main loop does not have fixed timing.
Hard coded limits.
Index

__init__
  guiAbout::GuiAbout, 32
  guiScript::GuiScript, 36
  guiScript::Mode, 43
  lpard::controller::lpardController::Controller,
    29
  lpard::interface::BoatSim::BoatSim, 25
  lpard::interface::lpardI2C::I2CInterface,
    39
  lpard::interface::lpardXBee::Boat, 23
  lpard::interface::lpardXBee::XbeeInterface,
    68
__repr__
  guiScript::Mode, 43

ADXL335, 19
AtCommandRequest, 19
  clearCommandValue, 20
  getFrameData, 20
  getFrameDataLength, 20
AtCommandResponse, 21
  getCommand, 21
  getStatus, 22
  getValue, 22
  getValueLength, 22
  isOk, 22
AZEL_Mode
  lpard::interface::lpardI2C::I2CInterface,
    39

begin
  XBee, 66
boat-systems/drone_firmware/drone_firmware.pde,
  81
boatCommand
  lpard::interface::BoatSim::BoatSim, 25
boatCommandResponse
  lpard::interface::BoatSim::BoatSim, 25
BoatSim, 13
boatStatusMessage
  lpard::interface::BoatSim::BoatSim, 25
  getAnalog
  getFrameDataLength
  getFrameData
  isOk
  getValue
  getValueLength
  getCommand
  getFrameDataLength
  getFrameData
  getStatus
  isOk

FrameIdResponse, 30

calibrate
  drone_firmware.pde, 83
changeAZ
  lpard::interface::lpardI2C::I2CInterface,
    40
changeEL
  lpard::interface::lpardI2C::I2CInterface,
    40
check_compass
  drone_firmware.pde, 83
clearCommandValue
  AtCommandRequest, 20
closePort
  lpard::interface::lpardXBee::XbeeInterface,
    69
command
  lpard::interface::BoatSim::BoatSim, 25
  Comp6DOF_n0m1, 27
component
  guiScript::GuiScript, 37
data
  lpard::interface::lpardI2C::I2CInterface,
    40
drone_firmware.pde
  calibrate, 83
  check_compass, 83
  check_gps, 83
  init_compass, 83
  loop, 84
  nonCompHeading, 84
  parseIncoming, 84
  receivePacket, 84
  send_status, 84
  setup, 85
  transmit, 85
<table>
<thead>
<tr>
<th>Method</th>
<th>Class/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxIoSampleBaseResponse</td>
<td>getApiId XBeeRequest XBeeResponse getAtCommandResponse XBeeResponse getChecksum XBeeResponse getCommand AtCommandResponse RemoteAtCommandResponse</td>
<td>54</td>
</tr>
<tr>
<td>ZBRxIoSampleResponse</td>
<td>getMessage lpard::interface::lpardXBee::XbeeInterface</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getMessages lpard::interface::BoatSim::BoatSim</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getModemStatusResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMsbLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getNextFrameId XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPacketLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPayload PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getPayloadLength PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress16 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress64 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAtCommandResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMessage lpard::interface::lpardXBee::XbeeInterface</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getMessages lpard::interface::BoatSim::BoatSim</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getModemStatusResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMsbLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getNextFrameId XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPacketLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPayload PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getPayloadLength PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress16 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress64 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAtCommandResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMessage lpard::interface::lpardXBee::XbeeInterface</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getMessages lpard::interface::BoatSim::BoatSim</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getModemStatusResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMsbLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getNextFrameId XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPacketLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPayload PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getPayloadLength PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress16 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress64 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAtCommandResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMessage lpard::interface::lpardXBee::XbeeInterface</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getMessages lpard::interface::BoatSim::BoatSim</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getModemStatusResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMsbLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getNextFrameId XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPacketLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPayload PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getPayloadLength PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress16 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress64 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAtCommandResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMessage lpard::interface::lpardXBee::XbeeInterface</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getMessages lpard::interface::BoatSim::BoatSim</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getModemStatusResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMsbLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getNextFrameId XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPacketLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPayload PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getPayloadLength PayloadRequest</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress16 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAddress64 RemoteAtCommandResponse</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>getRemoteAtCommandResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMessage lpard::interface::lpardXBee::XbeeInterface</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getMessages lpard::interface::BoatSim::BoatSim</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>getModemStatusResponse XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getMsbLength XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getNextFrameId XBeeResponse</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>getPacketLength XBeeResponse</td>
<td>74</td>
</tr>
</tbody>
</table>
getIndex(3)

getZBTxStatusResponse
XBeeResponse, 75
isDigitalEnabled
isAvailable

error

GUI_about::Ui::Ui_about, 31
GUI_mainPage::Ui::Ui_guiMain, 31
GUI_manualPositioning::Ui::Ui_manualPositioning, 31
GUI_newSession::Ui::Ui_newSession, 31
GUI_script::Ui::Ui_Script, 31

_gui::Ui::Ui_about, 32
showGui, 32
ui, 32

guiMainPage, 13

guiNewSession, 13

guiNewSession::GuiNewSession, 35
guiRecord, 15
guiScript, 15
guiScript::GuiScript, 35

__init__, 36
controller, 36
keyPressEvent, 36
scriptList, 37
showGui, 37

guiScript::Mode, 42

__init__, 43
__repr__, 43
mode, 43
record, 43
x, 43
y, 43

HMC5883L, 37

init
XBeeResponse, 75
init_compass
drone_firmware.pde, 83
isAnalogEnabled

RxIoSampleBaseResponse, 55
ZBRxIoSampleResponse, 77

isDigitalEnabled

XBeeResponse, 75

isOk

AtCommandResponse, 22
RemoteAtCommandResponse, 49

keypressEvent

guiScript::GuiScript, 36

loop

drone_firmware.pde, 84
lpard, 15

lpard::controller::lpardController::Controller, 28

__init__, 29
getDegrees, 29
getDegreesPerPwm, 30

lpard::controller::lpardControllerThread::ControllerThread, 30

lpard::interface::BoatSim::BoatSim, 23

__init__, 25
boatCommand, 25
boatCommandResponse, 25
boatStatusMessage, 25
command, 25
getMessages, 26
run, 26
update, 26

lpard::interface::BoatSimThreader::BoatSimThreader, 27

lpard::interface::lpardI2C::I2C2PC_MockObj, 38

lpard::interface::lpardI2C::I2CInterface, 38

__init__, 39
AZEL_Mode, 39
changeAZ, 40
changeEL, 40
data, 40
status, 40
US_Mode, 40

lpard::interface::lpardXBee::Boat, 22

__init__, 23

run, 23

lpard::interface::lpardXBee::XbeeInterface, 68
__init__, 68
closePort, 69
getMessage, 69
sendMessage, 69
lpard::loggers::logger::Logger, 41
lpardI2C, 16
lpardXbee, 16
MagnetometerRaw, 42
MagnetometerScaled, 42
mode
guiScript::Mode, 43
ModemStatusResponse, 44
nameValueObj, 16
nameValueObj::NameValueObj, 44
nonCompHeading
drone_firmware.pde, 84
opts
setup, 17

parseIncoming
drone_firmware.pde, 84
PayloadRequest, 45
getPayload, 45
getPayloadLength, 45
setPayload, 45
setPayloadLength, 45
readPacket
Xbee, 66
readPacketUntilAvailable
Xbee, 66
receivePacket
drone_firmware.pde, 84
record
guiScript::Mode, 43
RemoteAtCommandRequest, 46
getFrameData, 47
getFrameDataLength, 47
RemoteAtCommandRequest, 47
RemoteAtCommandResponse, 48
getCommand, 48
getRemoteAddress16, 49
getRemoteAddress64, 49
getStatus, 49
getValue, 49
getValueLength, 49
isOk, 49
reset

XbeeResponse, 75
run
lpard::interface::bootsim::BoatSim, 26
lpard::interface::lpardXbee::Boat, 23
Rx16IoSampleResponse, 50
Rx16Response, 50
Rx64IoSampleResponse, 51
Rx64Response, 52
RxDataResponse, 52
ggetData, 53
ggetDataLength, 53
ggetDataOffset, 53
RxIoSampleBaseResponse, 54
ggetAnalog, 54
ggetSampleSize, 54
isAnalogEnabled, 55
isDigitalEnabled, 55
isDigitalOn, 55
RxResponse, 55
ggetDataLength, 56
ggetDataOffset, 56

scriptList
guiScript::GuiScript, 37
send
Xbee, 66
sendStatus
drone_firmware.pde, 84
sendMessage
lpard::interface::lpardXbee::xbeeInterface, 69
setFrameId
Xbeerequest, 71
setPayload
PayloadRequest, 45
setPayloadLength
PayloadRequest, 45
setSerial
Xbee, 66
setup, 17
drone_firmware.pde, 85
opts, 17
shore-systems/azel/azelCode/azelCode.h, 85
showGui
guiAbout::GuiAbout, 33
guiScript::GuiScript, 37
status
lpard::interface::lpardI2C::I2CInterface, 40

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
INDEX 91

transmit
  drone_firmware.pde, 85
Tx16Request, 56
  getFrameData, 57
  getFrameDataLength, 57
  Tx16Request, 57
Tx64Request, 58
  getFrameData, 59
  getFrameDataLength, 59
  Tx64Request, 58, 59
TxStatusResponse, 59

ui
  guiAbout::GuiAbout, 33
update
  lpard::interface::BoatSim::BoatSim, 26
US_Mode
  lpard::interface::lpardI2C::I2CInterface, 40
util, 17

x
  guiScript::Mode, 43
XBe, 65
  begin, 66
  getNextFrameId, 66
  getResponse, 66
  readPacket, 66
  readPacketUntilAvailable, 66
  send, 66
  setSerial, 66
XBeAddress, 67
XBeAddress64, 67
XBeRequest, 69
  getApId, 70
  getFrameData, 70
  getFrameDataLength, 71
  getFrameld, 71
  setFrameld, 71
XBeResponse, 70
XBeResponse, 71
  getApId, 73
  getAtCommandResponse, 73
  getChecksum, 73
  getErrorCode, 73
  getFrameData, 73
  getFrameDataLength, 74
  getLsbLength, 74
  getModemStatusResponse, 74
  getMsbLength, 74
  getPacketLength, 74
  getRemoteAtCommandResponse, 74
  getRx16IoSampleResponse, 74
  getRx16Response, 74
  getRx64IoSampleResponse, 74
  getRx64Response, 75
  getTxStatusResponse, 75
  getZBRxIoSampleResponse, 75
  getZBRxResponse, 75
  getZBTxStatusResponse, 75
  init, 75
  isAvailable, 75
  isError, 75
  reset, 75
  XBeeResponse, 73

y
  guiScript::Mode, 43
ZBRxIoSampleResponse, 76
  getAnalog, 77
  isAnalogEnabled, 77
  isDigitalEnabled, 77
  isDigitalOn, 77
ZBRxResponse, 77
  getDataLength, 78
  getDataOffset, 78
ZBTxRequest, 78
  getFrameData, 79
  getFrameDataLength, 80
  ZBTxRequest, 78
ZBTxStatusResponse, 80

Generated on Tue May 8 2012 13:27:22 for LPARD-TDF-2012 by Doxygen
4/14/12  All Station Interface Boards

![Diagram of connections between interface boards with labels for SCL, GND, VDD, and SDA]

Test Procedure For QA:

- Used Digital Multimeter to verify that all connections were good and complete.
- Plugged in 18V power source to verify that 18V was available at each Vdd pin.
- Verified that all connector cables snugly connected to the interface board.

Technician: Mike Olahia  4/14/12
QA
All holes 4-40 threaded
measured all hole locations
Received 4/11/12

Technician: Mike DiPietro 4/11/12

Mike DiPietro
ECE 492 -- Senior Design
4/5/2012
Contact: dipietrm@lafayette.edu

Notes:
- All holes 4-40
- 1/4 inch thick Aluminum
- Qty: 1
Received Bracket on 4/20

Through holes 4-40 clearance
Holes correct distances apart
Fits well on the boat station
Technician: Mike DiPietro 4/20/12

Mike DiPietro
ECE 492 -- Senior Design
4/19/2012
Contact: dipietrm@lafayette.edu

Notes:
- All holes 4-40
- 1/32 inch thick Aluminum
- Qty: 1
Mike DiPietro  
ECE 492 -- Senior Design  
4/19/2012  
Contact: dipietrm@lafayette.edu

Notes:
- All holes 4-40  
- 1/4 inch thick Aluminum  
- Qty: 1

<table>
<thead>
<tr>
<th>HOLE</th>
<th>XDIM</th>
<th>YDIM</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>0.8075</td>
<td>-6.6025</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A2</td>
<td>4.50</td>
<td>6.50</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A3</td>
<td>2.85</td>
<td>-6.40</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A4</td>
<td>2.85</td>
<td>-5.2950</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A5</td>
<td>.8625</td>
<td>-4.6975</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A6</td>
<td>2.6225</td>
<td>-3.1275</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A7</td>
<td>4.50</td>
<td>-3.0</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A8</td>
<td>2.6225</td>
<td>-2.4375</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A9</td>
<td>.7100</td>
<td>-2.3125</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A10</td>
<td>.3750</td>
<td>-7.5</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
<tr>
<td>A11</td>
<td>1.4500</td>
<td>-7.5</td>
<td>Ø.13 ( \pm ) .025</td>
</tr>
</tbody>
</table>

QA  

Technician: [Signature]  4/26/12

Received Plate on 4/20

Measured all hole distances

Threaded 4-40 holes
Front End Board & Back End Board QA Testing:

The front end board consists of two separate sections, an ultrasound section and an infrared section. For the transmission part of the ultrasound, we just needed to test the connection was correct from the back end source since only the transducer is on the front end board and none of the driving circuitry (Test01). Next we needed to test the front end receiver circuitry to ensure the gains and frequency response was acceptable (Test02).

Test01:

Procedure: Using pins 5 and 7 on Interface A, which feed to the two terminals of the transmitter, we attached a 20Vpp square wave at 40 kHz using a function generator. In order to pass the test, we simply needed to ensure that the terminal transmitter receives the provided signal. Figure 1 shows that the backend signal correctly gets to the transmitter. Figure 2 shows a received signal that is directly in front of the transmitter, proving that the transmitter transmits.

Specifications: Provided transmission signal from the backend board correctly gets to the transmitter and is transmitted.

Results:

Board #1,2,3: Boards 1-3 all connected correctly and produced the following results. No differences between them, so same picture used as reference.

20Vpp input correctly shows up on input of transmitter (shown at 5V/div)
Board #1,2,3: Boards 1-3 all transmitted the provided signal, and a receiver placed directly in front of the transmitter produced the provided signal as shown in Figure 2.

Nearly 20 Vpp signal shown on receiver directly in front of transmitter at 40 kHz.

![Figure 2](image)

Pass/Fail: PASS (all three boards)

Signature: [Signature]

Date/Time: 4-27-2012, 12:10 AM

Test02:

Procedure: We placed a transmitter hooked up to a function generator applying 20 Vpp directly against the board’s receiver. With an oscilloscope measuring the voltage across the receiver, we adjusted the transmission frequency from 37 kHz to 43 kHz in increments on 500 Hz. The results of this are shown in figures 3a, 3b, and 3c. We then applied 250 mVpp across the transmitter and measured the amplified signal that is sent to the backend board. The results of this are shown in figures 4a, 4b, and 4c.

Specifications: Frontend board is able to amplify the input signal at a 40 kHz input range and an understanding of the frequency response of the receiver is gained.
Results:

Board #1:

Figure 3a

Received Signal vs. Frequency

Figure 4a

Amplified Signal vs. Frequency
Board #2:

**Figure 3b**

 Received Signal vs. Frequency

![Graph of Received Signal vs. Frequency]

**Figure 4b**

 Amplified Signal vs. Frequency

![Graph of Amplified Signal vs. Frequency]

Board #3:
Figure 3c

Received Signal vs. Frequency

Figure 4c

Amplified Signal vs. Frequency

Boat Board:
Pass/Fail: PASS (all 3 backend boards and boat board #1)

Signature:  

Backend Board Date/Time: 4-27-2012, 2:51 AM

Boat Board Date/Time: 5-7-2012, 3:09 PM

Test03:

Procedure: Ensure that the input pins on the UOS are correct by using the multi-meter and power down AZ/EL. Then plug UOS into AZ/EL on the turret and power on AZ/EL. Set multi-meter to voltage measurement mode and measure zero input bias at OUTA, OUTB, OUTC, OUTD, and AMP1OUT. Then inject square wave signal on the Quad Detector input channels A, B, C, D, and AMP1IN, measure voltage at input node, and output node of the respective op amps, to calculate gain. Repeat with oscilloscope measuring OUTA, OUTB, OUTC, and OUTD, while varying input signal frequency to determine corner frequencies, center frequency, and quality factor.

Specifications: Listed as actual under results.

Results:
Board #1:

Zero Input Bias (V)

<table>
<thead>
<tr>
<th></th>
<th>OUTA</th>
<th>OUTB</th>
<th>OUTC</th>
<th>OUTD</th>
<th>AMP1OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>Actual</td>
<td>2.558</td>
<td>2.555</td>
<td>2.452</td>
<td>2.434</td>
<td>2.519</td>
</tr>
</tbody>
</table>

Gain (V/V)

<table>
<thead>
<tr>
<th></th>
<th>OUTA</th>
<th>OUTB</th>
<th>OUTC</th>
<th>OUTD</th>
<th>AMP1OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Vin (Vpp)</td>
<td>0.004</td>
<td>0.004</td>
<td>0.004</td>
<td>0.004</td>
<td>0.1</td>
</tr>
<tr>
<td>Vout (Vpp)</td>
<td>0.79</td>
<td>0.79</td>
<td>0.78</td>
<td>0.79</td>
<td>2</td>
</tr>
<tr>
<td>Actual</td>
<td>197.5</td>
<td>197.5</td>
<td>195</td>
<td>197.5</td>
<td>20</td>
</tr>
</tbody>
</table>
Signal Generator @ 10Vpp w/ 100 ohm resistor 100mA
Light Output at 400mW/Sr

Resonant Freq. (kHz)

<table>
<thead>
<tr>
<th></th>
<th>OUTA</th>
<th>OUTB</th>
<th>OUTC</th>
<th>OUTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Actual</td>
<td>22.6</td>
<td>22.3</td>
<td>22.3</td>
<td>22.3</td>
</tr>
</tbody>
</table>

Corner Freq. (kHz)

<table>
<thead>
<tr>
<th></th>
<th>OUTA</th>
<th>OUTB</th>
<th>OUTC</th>
<th>OUTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hi</td>
<td>23.7</td>
<td>23.8</td>
<td>23.4</td>
<td>23.3</td>
</tr>
<tr>
<td>Lo</td>
<td>21.4</td>
<td>21.2</td>
<td>21.5</td>
<td>21.4</td>
</tr>
</tbody>
</table>

Q        | 9.826087 | 8.576923 | 11.73684 | 11.73684 | 308.0968 | 0.000009 | 05 1.6129E-05

Pass/Fail: Pass for UOS v0.2 SN 1 & SN

Signature: 何錫泉 (Andrew S. C. Ho) Date/Time: 4-24-2012, 5:30 PM
Test04:

Procedure: Used voltage source to apply 18 volts to the Vss of the backend board and used 5 volts for the enable and backend Vs input. This enabled the transmitter constantly. Oscilloscope hooked up to pins 5 and 7 to measure the signal across the transmission terminals (shown in Figure 5).

Specifications: The signal sent out pins 5 and 7 of the backend board of Interface A, which represent the two terminals of the transmitter have an approximately 36Vpp, 40 kHz signal applied across them.

Results:

Backend Board #1,2 (3 yet to be soldered), and Boat Board #1:

Shows signals across transmission terminals generated by back boards 1 and 2 (shown as one image because results were the same).

![Figure 5](image)

Pass/Fail: Pass (for board #1 and #2)

Signature: 

Backend Board Date/Time: 4-27-2012, 3:21 AM

Boat Board Date/Time: 5-1-2012, 5:34 PM
Test05:

**Procedure:** Used voltage source to provide 5 volts to the backend board’s receiver side. Used a 40 kHz, 225 mVpp signal as from the function generator as the output from the frontend board to see the frequency and gain response of the second stage of amplification.

**Specifications:** The second stage provides an amplification in the range of 1-2 V/V that can be tuned via a trimpot to get the accuracy needed in testing.

**Results:**

**Backend Board #1,2 (3 yet to be soldered) & Boat Board #1:** Both of these boards amplified the 225mVpp input signal to 400 mV, a gain of approximately 1.75 V/V. This shows that both of these boards are possible of amplification in the second stage. This gain represents the gain that we used in order to achieve our 10 meter results.

**Pass/Fail:** Pass (for backend board #1 and #2 and boat board #1)

Signature: [Signature]

**Backend Board Date/Time:** 4-29-2012, 11:16 PM

**Boat Board Date/Time:** 5-7-2012, 3:09 PM

Test06:

**Procedure:** power the boat board and hook up boat board IR to Arduino’s 22kHz PWM output.

**Specifications:** If the Ultrasound receiver does not trigger off of the IR driver circuit, then the test passes.

**Results:**

**Boat Board #1 (2 not soldered):** LED from the ultrasound receiver circuit triggers when the IR driver is powered up, so the two subsystems are incorrectly interacting with one another.

**Pass/Fail:** Fail (boat board #1)

Signature: [Signature]

**Date/Time:** 5-2-2012, 3:11 PM
NOTE: Averages in Blue

(0.344m/ms)

<table>
<thead>
<tr>
<th>Actual Distance in Meters</th>
<th>Raw Distance Estimate in Meters</th>
<th>Error in measurements (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.829</td>
<td>1.990</td>
<td>0.161</td>
</tr>
<tr>
<td>1.829</td>
<td>2.000</td>
<td>0.171</td>
</tr>
<tr>
<td>1.829</td>
<td>1.990</td>
<td>0.161</td>
</tr>
<tr>
<td>1.829</td>
<td>1.990</td>
<td>0.161</td>
</tr>
<tr>
<td>1.829</td>
<td>1.990</td>
<td>0.161</td>
</tr>
<tr>
<td><strong>1.829</strong></td>
<td><strong>0.163</strong></td>
<td></td>
</tr>
<tr>
<td>2.286</td>
<td>2.490</td>
<td>0.204</td>
</tr>
<tr>
<td>2.286</td>
<td>2.490</td>
<td>0.204</td>
</tr>
<tr>
<td>2.286</td>
<td>2.507</td>
<td>0.221</td>
</tr>
<tr>
<td>2.286</td>
<td>2.500</td>
<td>0.214</td>
</tr>
<tr>
<td>2.286</td>
<td>2.502</td>
<td>0.216</td>
</tr>
<tr>
<td><strong>2.286</strong></td>
<td><strong>0.212</strong></td>
<td></td>
</tr>
<tr>
<td>2.743</td>
<td>2.978</td>
<td>0.235</td>
</tr>
<tr>
<td>2.743</td>
<td>2.980</td>
<td>0.237</td>
</tr>
<tr>
<td><strong>2.743</strong></td>
<td><strong>0.236</strong></td>
<td></td>
</tr>
<tr>
<td>3.200</td>
<td>3.400</td>
<td>0.200</td>
</tr>
<tr>
<td>3.200</td>
<td>3.410</td>
<td>0.210</td>
</tr>
<tr>
<td><strong>3.200</strong></td>
<td><strong>0.205</strong></td>
<td></td>
</tr>
<tr>
<td>3.658</td>
<td>3.919</td>
<td>0.262</td>
</tr>
<tr>
<td>3.658</td>
<td>3.930</td>
<td>0.273</td>
</tr>
<tr>
<td>3.658</td>
<td>3.925</td>
<td>0.268</td>
</tr>
<tr>
<td><strong>3.658</strong></td>
<td><strong>0.267</strong></td>
<td></td>
</tr>
<tr>
<td>4.115</td>
<td>4.390</td>
<td>0.275</td>
</tr>
<tr>
<td>4.115</td>
<td>4.400</td>
<td>0.285</td>
</tr>
<tr>
<td><strong>4.115</strong></td>
<td><strong>0.280</strong></td>
<td></td>
</tr>
<tr>
<td>5.029</td>
<td>5.290</td>
<td>0.261</td>
</tr>
<tr>
<td>5.029</td>
<td>5.280</td>
<td>0.251</td>
</tr>
<tr>
<td><strong>5.029</strong></td>
<td><strong>0.256</strong></td>
<td></td>
</tr>
<tr>
<td>6.409</td>
<td>6.630</td>
<td>0.221</td>
</tr>
<tr>
<td>6.409</td>
<td>6.630</td>
<td>0.221</td>
</tr>
<tr>
<td><strong>6.409</strong></td>
<td><strong>0.221</strong></td>
<td></td>
</tr>
<tr>
<td>8.077</td>
<td>8.360</td>
<td>0.283</td>
</tr>
<tr>
<td>8.077</td>
<td>8.370</td>
<td>0.293</td>
</tr>
<tr>
<td><strong>8.077</strong></td>
<td><strong>0.288</strong></td>
<td></td>
</tr>
<tr>
<td>9.772</td>
<td>10.070</td>
<td>0.298</td>
</tr>
<tr>
<td>10.077</td>
<td>10.430</td>
<td>0.353</td>
</tr>
<tr>
<td>10.077</td>
<td>10.410</td>
<td>0.333</td>
</tr>
<tr>
<td>10.077</td>
<td>10.450</td>
<td>0.373</td>
</tr>
<tr>
<td>10.077</td>
<td>10.407</td>
<td>0.330</td>
</tr>
<tr>
<td>10.077</td>
<td>10.455</td>
<td>0.378</td>
</tr>
</tbody>
</table>
$y = -0.0004x^4 + 0.0099x^3 - 0.0945x^2 + 0.3702x - 0.2563$
<table>
<thead>
<tr>
<th>Date</th>
<th>Type</th>
<th>Meaured</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>26-Apr</td>
<td>tape</td>
<td>5.17</td>
<td>5.17</td>
</tr>
<tr>
<td></td>
<td>meaured</td>
<td>5.177</td>
<td>5.189</td>
</tr>
<tr>
<td></td>
<td>experimental</td>
<td>5.184</td>
<td>5.199</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.2</td>
<td></td>
</tr>
</tbody>
</table>
Aluminum Plate

4-40

\( \frac{1}{4}'' - 20 \)

fx: 5.470

fx: 1.603

308

.125

.308

.125

4-40

.125

.308

.125

3.205

8.000

16.000

QA Test
Traped Plate (Qty: 2)

Received → 3/9/12
All holes 4-40 threaded
Middle hole for tripod mount fits
Measured for hole distances

Tech: Mike DiPietro 3/9/12
Tripod 4-pin Ribbon Cables

Testing Procedure:
- Tested connection from end to end using a DMM.

Technician: Mike O'Reilly 4/3/12
Resistor Value (KΩ):
98.828

<table>
<thead>
<tr>
<th>Channel A: 10x expected gain</th>
<th>3db(kHz)</th>
<th>peak(kHz)</th>
<th>3db(kHz)</th>
<th>b/w(kHz)</th>
<th>Q</th>
<th>I(mA)</th>
<th>Vin(V apm)</th>
<th>Vout(V amp)</th>
<th>Gain(V/V)</th>
<th>Offset(V)</th>
<th>SatHi(V)</th>
<th>SatLo(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.07</td>
<td>22.22</td>
<td>23.17</td>
<td>2.1</td>
<td>10.581</td>
<td>0.012648</td>
<td>0.09934</td>
<td>1</td>
<td>10.06644</td>
<td>2.5812</td>
<td>4.224</td>
<td>0.223</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Channel B: 100x expected gain</th>
<th>3db(kHz)</th>
<th>peak(kHz)</th>
<th>3db(kHz)</th>
<th>b/w(kHz)</th>
<th>Q</th>
<th>I(mA)</th>
<th>Vin(V apm)</th>
<th>Vout(V amp)</th>
<th>Gain(V/V)</th>
<th>Offset(V)</th>
<th>SatHi(V)</th>
<th>SatLo(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.3</td>
<td>22.6</td>
<td>23.57</td>
<td>2.27</td>
<td>9.9559</td>
<td>0.001255</td>
<td>0.0097</td>
<td>1</td>
<td>103.0928</td>
<td>2.873</td>
<td>4.284</td>
<td>0.22</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Channel C: 100x expected gain</th>
<th>3db(kHz)</th>
<th>peak(kHz)</th>
<th>3db(kHz)</th>
<th>b/w(kHz)</th>
<th>Q</th>
<th>I(mA)</th>
<th>Vin(V apm)</th>
<th>Vout(V amp)</th>
<th>Gain(V/V)</th>
<th>Offset(V)</th>
<th>SatHi(V)</th>
<th>SatLo(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.08</td>
<td>21.9</td>
<td>22.96</td>
<td>1.88</td>
<td>11.649</td>
<td>0.001143</td>
<td>0.0101</td>
<td>1</td>
<td>99.0099</td>
<td>2.5864</td>
<td>4.285</td>
<td>0.218</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Channel D: 100x expected gain</th>
<th>3db(kHz)</th>
<th>peak(kHz)</th>
<th>3db(kHz)</th>
<th>b/w(kHz)</th>
<th>Q</th>
<th>I(mA)</th>
<th>Vin(V apm)</th>
<th>Vout(V amp)</th>
<th>Gain(V/V)</th>
<th>Offset(V)</th>
<th>SatHi(V)</th>
<th>SatLo(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.16</td>
<td>22.15</td>
<td>22.78</td>
<td>1.62</td>
<td>13.673</td>
<td>0.001022</td>
<td>0.0095</td>
<td>1</td>
<td>105.2632</td>
<td>2.2074</td>
<td>4.284</td>
<td>0.215</td>
<td></td>
</tr>
</tbody>
</table>
Resistor Value ($\text{K\Omega}$): 98.828

Channel A: 100x expected gain

<table>
<thead>
<tr>
<th></th>
<th>3db(kHz)</th>
<th>peak(kHz)</th>
<th>3db(kHz)</th>
<th>b/w(kHz)</th>
<th>Q</th>
<th>I(mA)</th>
<th>Vin(V apm)</th>
<th>Vout(V amp)</th>
<th>Gain(V/V)</th>
<th>Offset(V)</th>
<th>SatHi(V)</th>
<th>SatLo(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21.25</td>
<td>22.3</td>
<td>23.3</td>
<td>2.05</td>
<td>10.878</td>
<td>0.001245</td>
<td>0.0094</td>
<td>1</td>
<td>106.383</td>
<td>3.324</td>
<td>4.254</td>
<td>0.255</td>
</tr>
</tbody>
</table>

Channel B: 100x expected gain

<table>
<thead>
<tr>
<th></th>
<th>3db(kHz)</th>
<th>peak(kHz)</th>
<th>3db(kHz)</th>
<th>b/w(kHz)</th>
<th>Q</th>
<th>I(mA)</th>
<th>Vin(V apm)</th>
<th>Vout(V amp)</th>
<th>Gain(V/V)</th>
<th>Offset(V)</th>
<th>SatHi(V)</th>
<th>SatLo(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21.3</td>
<td>22.6</td>
<td>23.57</td>
<td>2.27</td>
<td>9.9559</td>
<td>0.001255</td>
<td>0.0097</td>
<td>1</td>
<td>103.0928</td>
<td>2.873</td>
<td>4.284</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Channel C: 100x expected gain

<table>
<thead>
<tr>
<th></th>
<th>3db(kHz)</th>
<th>peak(kHz)</th>
<th>3db(kHz)</th>
<th>b/w(kHz)</th>
<th>Q</th>
<th>I(mA)</th>
<th>Vin(V apm)</th>
<th>Vout(V amp)</th>
<th>Gain(V/V)</th>
<th>Offset(V)</th>
<th>SatHi(V)</th>
<th>SatLo(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21.08</td>
<td>21.9</td>
<td>22.96</td>
<td>1.88</td>
<td>11.649</td>
<td>0.001143</td>
<td>0.0101</td>
<td>1</td>
<td>99.0099</td>
<td>2.5864</td>
<td>4.285</td>
<td>0.218</td>
</tr>
</tbody>
</table>

Channel D: 100x expected gain

<table>
<thead>
<tr>
<th></th>
<th>3db(kHz)</th>
<th>peak(kHz)</th>
<th>3db(kHz)</th>
<th>b/w(kHz)</th>
<th>Q</th>
<th>I(mA)</th>
<th>Vin(V apm)</th>
<th>Vout(V amp)</th>
<th>Gain(V/V)</th>
<th>Offset(V)</th>
<th>SatHi(V)</th>
<th>SatLo(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21.16</td>
<td>22.15</td>
<td>22.78</td>
<td>1.62</td>
<td>13.673</td>
<td>0.001022</td>
<td>0.0095</td>
<td>1</td>
<td>105.2632</td>
<td>2.2074</td>
<td>4.284</td>
<td>0.215</td>
</tr>
</tbody>
</table>
QA Test: UOS-2 Version 0.1

Setup:

- Connected the board via jumpers to a DC power supply, oscilloscope, and digital multi-meter referencing the appropriate output leads
- Injected current by wiring the signal generator in series with the PCB and an appropriately sized resistor to act as a current source

- Instrument View:
- Board View:

Data can be found in: UOS 2 - 0.1 Data.pdf
Frontend Board Sensor Layout

Backend Board Sensor and Communication Layout
Boat Board Power Management and Sensor Layout
References


