

# **Software Design Document for RoboSub**

**Version 1.0 approved**

**Prepared by Olin-Mao Alvarez  
Jonathan Song  
Kevin Tran  
Henry Ho  
Danny Padilla**

**04/15/18**

Table of Contents.....	<pg 1>
Revision History.....	
<pg 4>	
1. Introduction.....	<pg 5>
1.1. Purpose.....	<pg 5>
1.2. Document Conventions.....	<pg 5>
1.3. Intended Audience and Reading Suggestions.....	<pg 5>
1.4. System Overview.....	<pg 5>
2. Design Considerations.....	<pg 6>
2.1. Assumptions and dependencies.....	<pg 6>
2.2. General Constraints.....	<pg 6>
2.3. Goals and Guidelines.....	<pg 6>
2.4. Development Methods.....	<pg 6>
3. Architectural Strategies.....	<pg 8>
4. System Architecture.....	<pg 9>
4.1. Data Flow Diagram.....	<pg 9>
4.2. Level 1 DFD.....	<pg 9>
5. Policies and Tactics.....	<pg 12>
5.1. Specific Products Used.....	<pg 12>
5.2. Requirements traceability.....	<pg 12>
5.3. Testing the software.....	<pg 12>
5.4. Software Maintenance.....	<pg 13>
5.5. Engineering trade-offs.....	<pg 13>
5.6. Guidelines and conventions.....	<pg 13>
5.7. Algorithms or Idioms to implement functionality.....	<pg 13>
5.8. Interfaces.....	<pg 13>
5.9. Hierarchical Organization of Source Code.....	<pg 14>
5.10. System's deliverables.....	<pg 14>
5.11. Abstraction.....	<pg 14>
6. Detailed System Design.....	<pg 15>
6.1 ROS Module.....	<pg 15>
6.1.1 Responsibilities.....	<pg 15>
6.1.2 Constraints.....	<pg 15>
6.1.3 Composition.....	<pg 15>
6.1.4 Uses/Interactions.....	<pg 15>
6.1.5 Resources.....	<pg 15>
6.1.6 Interface/Exports.....	<pg 15>
6.2 Computer Vision Module.....	
6.2.1 Responsibilities.....	<pg 15>
6.2.2 Constraints.....	<pg 15>
6.2.3 Composition.....	<pg 16>
6.2.4 Uses/Interactions.....	<pg 16>
6.2.5 Resources.....	<pg 16>
6.2.6 Interface/Exports.....	<pg 16>

6.3	Arduino Module	
6.3.1	Responsibilities.....	<pg 16>
6.3.2	Constraints.....	<pg 16>
6.3.3	Composition.....	<pg 16>
6.3.4	Uses/Interactions.....	<pg 16>
6.3.5	Resources.....	<pg 16>
6.3.6	Interface/Exports.....	<pg 16>
7.	Detailed Lower level Component Design	
7.1	Detector.py.....	<pg 17>
7.1.1	Classification.....	<pg 17>
7.1.2	Processing Narrative(PSPEC).....	<pg 17>
7.1.3	Interface Description.....	<pg 17>
7.1.4	Processing Detail.....	<pg 17>
7.1.4.1	Design Class Hierarchy.....	<pg 17>
7.1.4.2	Restrictions/Limitations.....	<pg 17>
7.1.4.3	Performance Issues.....	<pg 17>
7.1.4.4	Design Constraints.....	<pg 17>
7.1.4.5	Processing Detail For Each Operation.....	<pg 17>
7.2	CVController.py.....	<pg 17>
7.2.1	Classification.....	<pg 17>
7.2.2	Processing Narrative(PSPEC).....	<pg 17>
7.2.3	Interface Description.....	<pg 16>
7.2.4	Processing Detail.....	<pg 16>
7.2.4.1	Design Class Hierarchy.....	<pg 16>
7.2.4.2	Restrictions/Limitations.....	<pg 16>
7.2.4.3	Performance Issues.....	<pg 16>
7.2.4.4	Design Constraints.....	<pg 16>
7.2.4.5	Processing Detail For Each Operation.....	<pg 16>
7.3	Houston.py.....	<pg 16>
7.3.1	Classification.....	<pg 16>
7.3.2	Processing Narrative(PSPEC).....	<pg 16>
7.3.3	Interface Description.....	<pg 16>
7.3.4	Processing Detail.....	<pg 16>
7.3.4.1	Design Class Hierarchy.....	<pg 16>
7.3.4.2	Restrictions/Limitations.....	<pg 16>
7.3.4.3	Performance Issues.....	<pg 16>
7.3.4.4	Design Constraints.....	<pg 16>
7.3.4.5	Processing Detail For Each Operation.....	<pg 16>
7.4	Navigation.py.....	<pg 16>
7.4.1	Classification.....	<pg 16>
7.4.2	Processing Narrative(PSPEC).....	<pg 16>
7.4.3	Interface Description.....	<pg 16>
7.4.4	Processing Detail.....	<pg 16>
7.4.4.1	Design Class Hierarchy.....	<pg 16>

7.1.4.2	Restrictions/Limitations.....	<pg 16>
7.1.4.3	Performance Issues.....	<pg 16>
7.1.4.4	Design Constraints.....	<pg 16>
7.1.4.5	Processing Detail For Each Operation.....	<pg 16>
7.4	HardwareInterface.ino.....	<pg 16>
7.1.1	Classification.....	<pg 16>
7.1.2	Processing Narrative(PSPEC).....	<pg 16>
7.1.3	Interface Description.....	<pg 16>
7.1.4	Processing Detail.....	<pg 16>
7.1.4.1	Design Class Hierarchy.....	<pg 16>
7.1.4.2	Restrictions/Limitations.....	<pg 16>
7.1.4.3	Performance Issues.....	<pg 16>
7.1.4.4	Design Constraints.....	<pg 16>
7.1.4.5	Processing Detail For Each Operation.....	<pg 16>
8.	Database Design.....	<pg 21>
9.	User Interface	
9.1.	Overview of User Interface.....	<pg 22>
9.2.	Screen Frameworks or Images.....	<pg 22>
9.3.	User Interface Flow Model.....	<pg 22>
10.	Requirements Validation and Verification.....	<pg 23>
11.	Glossary.....	<pg 26>
12.	References.....	<pg 26>

## Revision History

Name	Date	Reason For Changes	Version
Eagle 1	04/17	Initial Software for AUV Competition	1.0
Osprey	07/17	Updated software to go with new AUV hardware	2.0

# **1. Introduction**

## **1.1 Purpose**

This Document outlines the software architecture and design for the Osprey Software controlling the RoboSub senior design project. It goes in depth for the Computer Vision module, The ROS module, and the Arduino module. It also describes the functionality of the ROS, Arduino and Computer Vision modules.

## **1.2 Document Conventions**

This document has very basic formatting, points of emphasis are either highlighted or bolded

## **1.3 Intended Audience and Reading Suggestions**

This document is intended for future developers of the RoboSub projects or developers looking to understand the software architecture for Osprey. It is also intended for the California State University - Los Angeles Computer Science department, including students and faculty.

## **1.4 System Overview**

The Osprey software controls an Autonomous Underwater Vehicle (AUV) which is being developed for an international robotics competition. It is responsible for handling all of the AUV's basic functions. The Osprey software is comprised of three main components; the first being the ROS module which acts a central hub for all signals taken in from hardware, the second is the Arduino module which takes in commands from the ROS module and guides motors and other hardware needed to complete each task, and lastly the Computer Vision module which combines different image filtering and machine learning algorithms for object detection to provide necessary information to the AUV.

## **2. Design Considerations**

### **2.1 Assumptions and Dependencies**

- Osprey will be written with a combination of Python and Arduino language
- Osprey will run on Ubuntu as made necessary by ROS.
- The Computer Vision will be written in Python and uses OpenCV for image processing
- The Arduino portion of the software shall be written in Arduino language

### **2.2 General Constraints**

- The Osprey software must run on Ubuntu
- The Batteries running the AUV can run for approximately 30 mins
- Must have base C++ libraries installed
- The Computer Vision will be run using OpenCV, Numpy and Scikit Learn these all must be installed for the cv to function
- Osprey must run on ROS Lunar Loggerhead

### **2.3 Goals and Guidelines**

- Osprey has a delivery date of 30 July 2018 (the beginning of the competition)
- Osprey code must be easily read and modified
- Osprey modules and submodules must be object oriented in order to facilitate easy modification for future teams using the code base.
- The software will emphasize efficiency given that we have limited computational capacity
- The software must support full autonomy for the AUV.

### **2.4 Development Methods**

The Osprey software has been developed using Agile methods. Most of the code that was inherited from the previous year's competition was completely useless hence the entire thing had to be redone. The Team mostly focused on modularize existing code and updating methods to be more efficient and more legible. Furthermore, there was a more object oriented approach such that further teams will not have to do this as well. The Computer Vision was developed using

various IDEs. The Computer Vision implements the strategy pattern of object oriented design. The Computer Vision was created using OpenCV as well as a variety of different machine learning libraries such as keras, and scikit learn. Additionally, the machine learning algorithms were done by testing different image preprocessing techniques as well as implementing different processing protocols in order to minimize the computational workload used by CV.

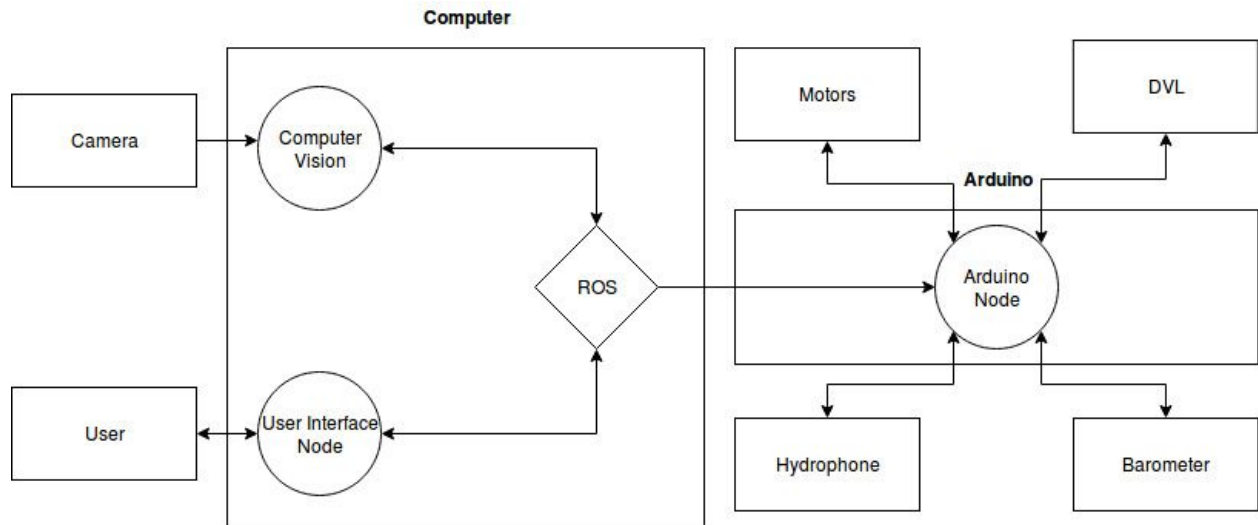


### 3. Architectural Strategies

- The vast majority of the project was written in Python to provide uniformity between the ROS side of the project and the Computer Vision side.
- The machine learning algorithms were created with various Python libraries including SciKit Learn, Keras, and OpenCV
- The Computer Vision module relied heavily on OpenCV for image processing and filtering.
- ROS was used to interface with the Arduino Module (which controls the hardware) because it provides a robust set of functions that would otherwise have to be written from scratch.
- Data will be collected by the Arduinos and the Cameras and interpreted by the ROS module.
- The list of tasks completed and to be complete will be handled by the ROS module
- The hardware used for signal processing includes Hydrophones, three cameras, a doppler velocity log and an Inertial Measurement Unit

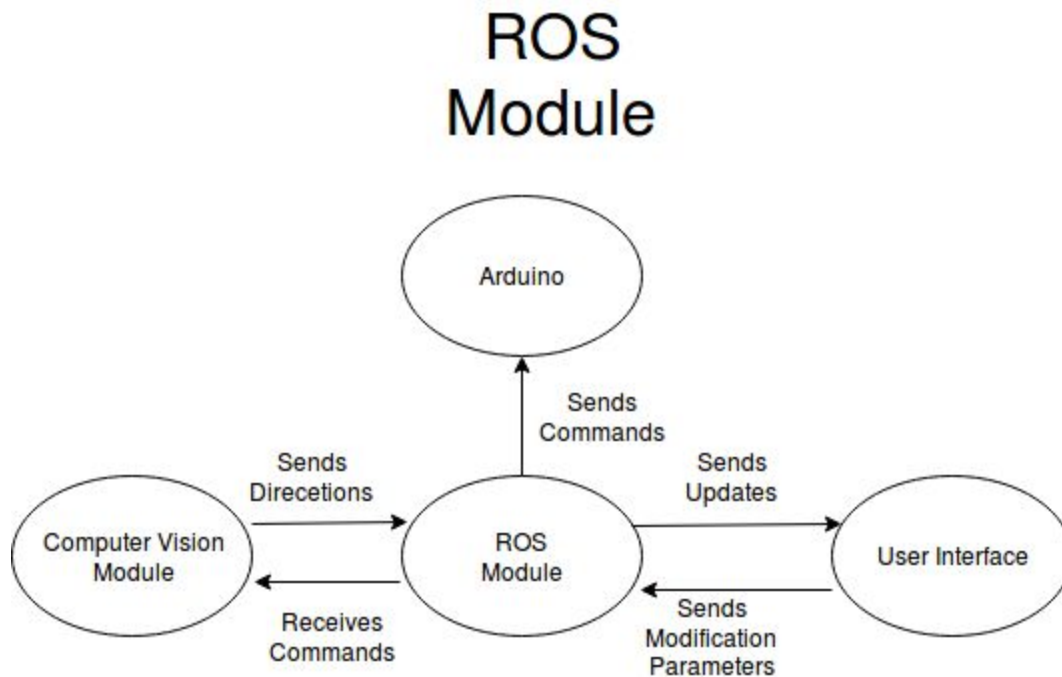
## 4. System Architecture

### 4.1 Data Flow Diagram



### 4.2 Level 1 DFDs

#### 4.2.1 ROS Module

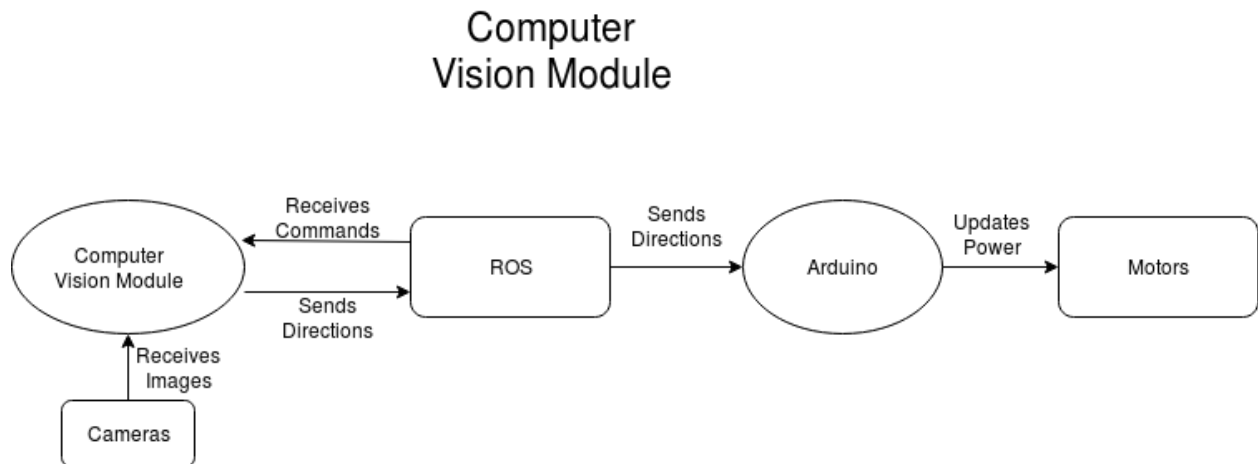


The ROS module will be used as an interface between the Main Computer and the Arduinos. ROS has 2 main functions. One of the ROS functions is to publish data that is formulated once it receives values from the Computer Vision Module. The purpose of publishing the formulated data is so the data can be retrieved by the Arduinos which will then send commands to the corresponding piece of hardware. The 2nd function of ROS is to subscribe to the data which is broadcasted onto ROS by the Arduinos. In other words, it is the opposite of publish, which will retrieve data that is sent by the Arduinos.

#### 4.2.2 Arduino Module

The Arduino Module is used for the main computer to communicate with each piece of hardware. Each Arduino has different functions which as receive commands from the main computer, send commands to other Arduinos, receive data from hardware and send data to hardware. The barometer, and hydrophone will send data to their corresponding Arduino, while the motos will receive instructions from the Arduino.

#### 4.2.3 Computer Vision Module



The Computer Vision works as follows. First the CV receives a command from ROS indicating what type of object or task it is looking for. Next the Computer Vision selects a corresponding Detector Class. Each Detector is designed for a specific task. The Detector implements a Preprocessor interface and a Classifier interface. The preprocessor interface implementation determines what type of filtering will be done with each image and it also returns general regions of interest that are likely to include the target object. Then the CV sends each of these regions to a classifier. Classifiers used include, but are not limited to, Support Vector Machines and Convolutional Neural Networks. The classifiers then return positive matches. The cv takes the

regions with a positive match and provides directions to ROS to guide the AUV towards it. ROS relays this information to the Arduinos which then trigger the motors to navigate towards the direction of the detected object.

## **5. Policies and Tactics**

### **5.1 Choice of which specific products used**

Python, ROS and Arduino IDE were used to develop the Osprey. The Computer Vision was written completely in Python and used a variety of different libraries. Most significant was the use of OpenCV for image processing, colorspace shifting and feature extraction. Additionally, SciKit-Learn, Numpy, Pandas and Keras were used as well. SciKit-Learn and Keras were used to apply deep learning algorithms for the object detection. The CV was developed using the Python Libraries in the Pycharm IDE. Additionally Matplotlib was used to provide key visual representations of data which were useful during the development of the image processing protocols. For testing, Ubuntu computers were used

### **5.2 Plans for ensuring requirements traceability**

Traceability for the for the software requirements will be provided in an attached software requirement document. This document provides detailed information as to what all of the requirements, which requirements were met and how those requirements were met.

### **5.3 Plans for testing the software**

There are a variety of testing methodologies that Osprey will use to ensure that all requirements are met and that updates to the source code do not cause any errors. There are testing requirements for each of the individual components of Osprey, as well integration testings to assure that all separate components connect correctly.

Some of ROS tests include but are not limited to;

- Directing the AUV to drop a fixed distance
- Directing the AUV to complete a circle
- Directing the AUV to navigate forward a fixed distance then rotate and return to its starting position
- Having the AUV fire torpedoes
- Having the AUV release chips used for tasks

Some of the CV tests include but are not limited to;

- Finding the correct object through as series of images
- Intersection over union for each classifier to gauge accuracy of classifiers
- Testing that the CV updates directions properly.
- Testing that the proper methods are implemented in each of the Classifiers

Additionally integration testing includes but is not limited to;

- Having the AUV navigate towards a buoy detected from the cameras.
- Having the AUV navigate through a gate detected from by the cameras
- Having the AUV find a correct numbered die in a set of multiple dice

Additionally, any methods, modules or functions that maneuver the AUV, modify hardware signal data, process images, or affect direction shall be subject to testing to ensure that

data flows consistently throughout the software and that the proper implementations of each module are applied.

## **5.4 Software Maintenance**

The Osprey software shall be stored on a variety of devices, and using version control which is accessible to all members of the RoboSub team, including Computer Science and Engineering. Furthermore, documentation shall be maintained such that future developers of the RoboSub project can easily understand and modify existing code.

## **5.5 Engineering Trade-Offs**

Osprey will use Python for as its language, though it a high level (slow) language, as it is easier to implement and modify source code in accordance to the Agile software development methodology used by the Computer Science subdivision of the RoboSub team.

The Arduino portion of the software will be written in Arduino language as Python cannot be used to control the Arduinos.

Drivers will be written for the Doppler Velocity Log as the model which is on the AUV (Teledyne Pathfinder) only has drivers for Windows and the ROS portion of project can run exclusively on Ubuntu.

## **5.6 Coding Guidelines and Conventions**

All portions of the source code that are written in Python must follow conventions outlined in the Pep 8 style guidelines.

Additionally the CV follows Object Oriented Design principles and implements the “Strategy Pattern” which will enable Osprey to switch implementations of CV Detectors and Classifiers at runtime.

Coordinates shall be handled as follows:

Horizontal : -1 Left, 0 Steady, 1 Right

Vertical: -1 Down, 0 Steady, 1 Up

Y direction: -1 Back, 0 Steady, 1 Forward

## **5.7 Choices of Algorithms or Idioms to implement functionality**

Osprey’s CV follows the Strategy Pattern design idiom in order to facilitate runtime transitioning between different classifiers and preprocessor that will be needed for each task’s unique requirements. The remaining portion of the project is implemented using object oriented design principles for easy modification and legibility. The Strategy Pattern is not necessary of the non-CV portions of the project as they will not require large changes during runtime.

## **5.8 Interfaces for End-Users**

Osprey will come with a command line interface to enable easy modifications of values during competition. The CLI is not meant for large changes in the Osprey software, rather it is mostly used for modifications of simple variables.

## **5.9 Hierarchical Organization of Source Code**

Osprey is started by launching the CLI which will set up the AUV class which will oversee all the communication and functionality of the AUV through the use of ROS. The AUV class manages all the modules for controls, sensors, and tasks such as Navigation and Computer Vision. The Arduino module will then receive information that is sent by the Navigation module through ROS and commands the individual hardware components based off of the directions given by the Computer Vision module.

## **5.10 How to build and/or generate the system's deliverables (how to compile, link, load, etc.)**

To start Osprey you must clone the Robosub repository from GitHub on a computer running Ubuntu 17.04. By running `robosub.py` with python it will then automatically install all the dependencies needed to run Osprey with a series of scripts. After the installation and a restart of the system, running the python code again with all of the Arduinos properly connected will allow the user to operate Osprey.

## **5.11 Abstract Classes and Interfaces Used**

Osprey uses interfaces for the Computer Vision. Specifically it uses an Preprocessor interface and a Classifier interface. These interfaces are used to ensure that the proper methods are used for every component of the Computer Vision given that classifiers and preprocessors will be changed during runtime.

Additionally the task handling portion of the ROS module uses a Task interface to ensure consistency of implementation throughout each task despite each task having different needs.

## **6. Detailed System Design**

### **6.1 ROS Module (Module)**

#### **6.1.1 Responsibilities**

The ROS module is responsible for communication between each node such as the main computer node and Arduino node. ROS will enable the main computer and the Arduinos to publish and subscribe to data.

#### **6.1.2 Constraints**

The constraints of the ROS module requires older versions of operating systems. ROS is also limited to only being able to run on Ubuntu.

#### **6.1.3 Composition**

ROS is composed of a network of ROS nodes that send and receive data with publishers and subscribers through the use of streaming topics.

#### **6.1.4 Uses/Interactions**

The ROS module is used by all the other modules to send and receive information with each other.

#### **6.1.5 Resources**

The ROS module has access to the data that is published to it, which come from the main computer and Arduinos

#### **6.1.6 Interface/Exports**

Provided in source code.

### **6.2 Computer Vision (Module)**

#### **6.2.1 Responsibilities**

The Computer Vision is responsible for detecting objects throughout the course, confirming that those objects are the ones associated with the task at hand and providing information to the other modules to help navigate throughout the course or to a specific task.

#### **6.2.2 Constraints**

The CV is constrained by the three cameras the AUV has at its disposal two front facing one bottom facing, additionally it is constrained by having limited CPU power. This means that techniques must take into CPU usage into account. Additionally, CV is



constrained by the nature of the competition. Each run has a time limit which means that each CV algorithm must be accurate, efficient and simple.

### **6.2.3 Composition**

The CV is composed of a series of different Detectors, which implement different Classifiers and Preprocessor in order to find their intended targets. Additionally each specific detector has differing built-in methods that will help it fulfill its intended purpose.

### **6.2.4 Uses/Interactions**

The CV interacts with the ROS module to provide navigation information to the Arduino module.

### **6.2.5 Resources**

The CV has access to an Intel i7 Processor as well as an Nvidia GTX 1060 GPU

### **6.2.6 Interface/Exports**

Provided in Source Code.

## **6.3 Arduino (Module)**

### **6.3.1 Responsibilities**

The Arduino Module is responsible for the communication between the main computer and the hardware.

### **6.3.2 Constraints**

A constraint and limitation of the Arduinos include its limited memory space. Another constraint and limitations of the Arduinos are the limited pin spaces on the Uno Arduinos

### **6.3.3 Composition**

The Arduino Module is composed of all of the Arduinos and hardware

### **6.3.4 Uses/Interactions**

The Arduino Module interacts with the ROS module to provide or receive information to and from the main computer as well as the hardware

### **6.3.5 Resources**

The Arduino Module have access to 1 Arduino Mega and several Arduino Unos

### **6.3.6 Interface/Exports**

Provided in Source Code.

# **7. Detailed Lower level Component Design**

## **7.1 Detector.py**

### **7.1.1 Classification**

This file is an interface which describes the methods all detectors must implement, it is a component of CV

### **7.1.2 Processing Narrative (PSPEC)**

It is a python file that is loaded onto the AUV and makes sure that all Detector implementations contain the proper methods needed for runtime.

### **7.1.3 Interface Description**

The Detector interface has preprocessor and classifier required fields, as well as a detect abstract method.

### **7.1.4 Processing Detail**

#### **7.1.4.1 Design Class Hierarchy**

All detectors inherit from Detector.py

#### **7.1.4.2 Restrictions/Limitations**

Must use as little CPU/GPU resources as possible.

#### **7.1.4.3 Performance Issues**

Some detectors may be slower or faster depending on the classification method which is implemented for that particular detector

#### **7.1.4.4 Design Constraints**

#### **7.1.4.5 Processing Detail For Each Operation**

## **7.2 CVController.py**

### **7.2.1 Classification**

This file is an interface which lets Houston.py and the Camera interact

### **7.2.2 Processing Narrative (PSPEC)**

It is a python file that is loaded onto the AUV which will help the coordinates of the AUV be sent to Houston.py

### **7.2.3 Interface Description**

### **7.2.4 Processing Detail**

#### **7.2.4.1 Design Class Hierarchy**

#### **7.2.4.2 Restrictions/Limitations**

Must use as little CPU/GPU resources as possible.

#### **7.2.4.3 Performance Issues**

#### **7.2.4.4 Design Constraints**

#### **7.2.4.5 Processing Detail For Each Operation**

## **7.3 Houston.py**

### **7.3.1 Classification**

This file is an interface which will receive data from Computer Vision to send out instructions to the hardware

### **7.3.2 Processing Narrative (PSPEC)**

It is a python file that is loaded onto the AUV which helps all the components of the AUV to work together

### **7.3.3 Interface Description**

The Houston interface utilizes task manager, navigation and computer vision classes.

### **7.3.4 Processing Detail**

#### **7.3.4.1 Design Class Hierarchy**

#### **7.3.4.2 Restrictions/Limitations**

Must use as little CPU/GPU resources as possible.

#### **7.3.4.3 Performance Issues**

Some tasks may be slower or faster depending on how complicated the task may be

#### **7.3.4.4 Design Constraints**

#### **7.3.4.5 Processing Detail For Each Operation**

## **7.4 Navigation.py**

### **7.4.1 Classification**

This is an interface file which will publish directions onto ROS for the motor to navigate to

#### **7.4.2 Processing Narrative (PSPEC)**

It is a python file that is loaded onto the AUV which will help with the navigation of the AUV

#### **7.4.3 Interface Description**

The Navigation interface utilizes ROS

#### **7.4.4 Processing Detail**

##### **7.4.4.1 Design Class Hierarchy**

##### **7.4.4.2 Restrictions/Limitations**

Must use as little CPU/GPU resources as possible.

##### **7.4.4.3 Performance Issues**

Navigation is dependant on task manager and houston, so will only work as fast as those classes do

##### **7.4.4.4 Design Constraints**

##### **7.4.4.5 Processing Detail For Each Operation**

### **7.5 HardwareInterface.ino**

#### **7.5.1 Classification**

The kind of component, such as a subsystem, class, package, function, file, etc.

#### **7.5.2 Processing Narrative (PSPEC)**

A process specification (PSPEC) can be used to specify the processing details

#### **7.5.3 Interface Description**

#### **7.5.4 Processing Detail**

##### **7.5.4.1 Design Class Hierarchy**

Class inheritance: parent or child classes.

##### **7.5.4.2 Restrictions/Limitations**

##### **7.5.4.3 Performance Issues**

##### **7.5.4.4 Design Constraints**

#### **7.5.4.5 Processing Detail For Each Operation**

## **8. Database Design**

This project did not require any database.

## 9. User Interface

### 9.1 Overview of User Interface

The user interface is in the form of a command line interface. It will allow users to adjust the values of some of the variables used in Osprey and will display logs and the status information of Osprey using the built in commands.

### 9.2 Screen Frameworks or Images

```
Type help or ? to list commands.
auv> help

Documented commands (type help <topic>):
=====
exit help logging model motor navigation tasks test

auv> help tasks

[view] to view tasks
[set] to set tasks list
[reset] to reset tasks list
auv> tasks view
['gate', 'path', 'dice', 'chip', 'path', 'chip', 'slots', 'pinger b', 'roulette', 'pinger a', 'cash in']
auv>
```

### 9.3 User Interface Flow Model

help - logging, model, motor, navigation, tasks, test  
logging - on, off, toggle, state  
model - (model number), view  
motor - on, off, toggle, state  
navigation - cv, keyboard  
tasks - view, set, reset

# 10. Requirements Validation and Verification

The methodologies and testing strategies identified at this point include four major approaches: TESTING, DEMONSTRATION, INSPECTION, and ANALYSIS with various variations to adapt to the characteristics:

- **Testing** using additional ad-hoc created software including a correlation testing unit.
- **Demonstration** of the specified capability
- **Inspection** of the software code possibly using additional inspection techniques
- **Analysis** of the specific code operation/algorithm to prove functionality.

Requirements Related to: <b>1. Robot Operating System</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.1-1	Robot Operating System shall be installed on Ubuntu 17.04.	Inspection
4.1.1-2	Robot Operating System shall have ROS Lunar Loggerhead installed.	Inspection
4.1.1-3	Robot Operating System shall receive data from IMU.	Inspection
4.1.1-4	Robot Operating System shall receive data from Cameras.	Inspection
4.1.1-5	Robot Operating System shall receive data from ATmega2560.	Inspection
4.1.1-6	Robot Operating System shall send data to ATmega2560.	Inspection
4.1.1-7	Robot Operating System shall be run on the Computer.	Inspection

Requirements Related to: <b>2. User Interface</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.2-1	The User Interface shall be a CLI.	Demonstration
4.1.2-2	The User Interface shall be used with ROS.	Inspection
4.1.2-3	The User Interface shall provide methods of changing AUV parameters.	Demonstration
4.1.2-4	The User Interface shall provide AUV testing capabilities.	Demonstration



Requirements Related to: <b>3. AUV Tasks</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.3-1	The AUV Tasks shall be to Find the Casino.	Demonstration
4.1.3-2	The AUV Tasks shall be to Enter the Casino (Gate).	Demonstration
4.1.3-3	The AUV Tasks may be to Shoot Craps.	Demonstration
4.1.3-4	The AUV Tasks may be to Buy a Gold Chip.	Demonstration
4.1.3-5	The AUV Tasks may be to Try the Slots.	Demonstration
4.1.3-6	The AUV Tasks may be to Hit the Jackpot.	Demonstration
4.1.3-7	The AUV Tasks may be to Play Roulette.	Demonstration
4.1.3-8	The AUV Tasks may be to Cash in Your Chips.	Demonstration

Requirements Related to: <b>4. AUV AI</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.4-1	The AUV AI shall be able to Find the Casino.	Analysis
4.1.4-2	The AUV AI shall be able to Enter the Casino (Gate).	Analysis
4.1.4-3	The AUV AI may be able to Shoot Craps.	Analysis
4.1.4-4	The AUV AI may be able to Buy a Gold Chip.	Analysis
4.1.4-5	The AUV AI may be able to Try the Slots.	Analysis
4.1.4-6	The AUV AI may be able to Hit the Jackpot.	Analysis
4.1.4-7	The AUV AI may be able to Play Roulette.	Analysis
4.1.4-8	The AUV AI may be able to Cash in Your Chips.	Analysis

Requirements Related to: <b>5. Arduino</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.5-1	The Arduino shall receive data from the Sensors.	Inspection
4.1.5-2	The Arduino shall send commands to the Pneumatic Control.	Inspection
4.1.5-3	The Arduino shall send commands to the Motor Control.	Inspection
4.1.5-4	The Arduino shall receive data from the Computer.	Inspection
4.1.5-5	The Arduino shall send data from the Computer.	Inspection
4.1.5-6	The Arduino shall be connected to ROS.	Inspection

Requirements Related to: <b>6. Computer Vision</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.6-1	The Computer Vision shall receive data from the Cameras.	Inspection
4.1.6-2	The Computer Vision shall output processed data.	Inspection
4.1.6-3	The Computer Vision shall provide coordinates	Inspection
4.1.6-4	The Computer Vision shall detect objects associated with tasks	Inspection
4.1.6-5	The Computer Vision shall detect tasks	Inspection
4.1.6-6	The Computer Vision shall shall provide depth perception	Inspection
4.1.6-7	The Computer Vision shall use classifiers for object detection	Inspection
4.1.6-8	The Computer Vision shall provide distances from objects	Inspection
4.1.6-9	The Computer Vision shall provide stereo-vision	Inspection
4.1.6-10	The Computer Vision shall provide directions to ROS	Inspection

Requirements Related to: <b>7. Sensors</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.7-1	The Sensor data shall be published to ROS.	Inspection

Requirements Related to: <b>8. Pneumatic Control</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.8-1	Pneumatic Control shall be capable of receiving commands from ROS.	Inspection

Requirements Related to: <b>9. Motor Control</b>		
Requirement No.	Requirement Description	Method for Testing
4.1.9-1	Motor Control shall receive commands from the Arduino.	Inspection
4.1.9-2	Motor Control shall operate at command level.	Inspection
4.1.9-3	Motor Control shall allow the AUV full horizontal and vertical movement.	Demonstration

## 11. Glossary

ROS	-	Robot Operating System
CV	-	Computer Vision
AUV	-	Autonomous Underwater Vehicle
CLI	-	Command Line Interface
Hz	-	Hertz
IMU	-	Inertial Measurement Unit
USB	-	Universal Serial Bus
IDE	-	Integrated Development Environment

## 12. References

### Website address referenced

1. Robosub Competition
  - <http://www.robonation.org/competition/robosub>
2. Operating System
  - <http://wiki.ros.org/lunar/Installation/Ubuntu>
3. OpenCV Library Documentation
  - <https://docs.opencv.org/trunk/index.html>
4. Robot Operating System Documentation
  - <http://wiki.ros.org/Documentation>
5. Arduino
  - <https://www.arduino.cc/en/Guide/Introduction>
  - <https://www.arduino.cc/en/Reference/Wire>

### Documents Referenced

1. Robosub Draft Tasks 2018
  - a. [http://www.robonation.org/sites/default/files/2018%20RoboSub%20Tasks\\_DRAFT%2011%2001%2017.pdf](http://www.robonation.org/sites/default/files/2018%20RoboSub%20Tasks_DRAFT%2011%2001%2017.pdf)