Arizona State University

Intel-Cornell Cup

2015

Autonomous Disaster Relief Agents (ADRA)

Supervisor:
Dr. Yinong Chen

Author:
Vageesh Bhasin
David Ingraham
Sami Mian

April 20, 2015

Abstract

In a disaster situation, such as stranded tourists & hikers, identifying & locating them, and delivering essential survival supplies becomes a task of utmost importance for first-responders[1], like EMS[2] personnels. The project aims at developing a tool that will support the current search & rescue operation, as well provide a mechanism for delivering packages in a safe and timely manner.

The solution is to build a network of inter-communicating copters that will be responsible for surveying an area, finding positions of potential targets and dropping packages near them. For demonstration purposes, we will be building a network of 2 copters, one responsible for surveillance and the other for delivery. The 2 copters will be in constant communication with a base camp, that will be doing all the heavy computational processes, such as human detection, flight control & navigational commands, etc. (explained further).

[1]First people to arrive at and assist in a scene of emergency
[2]Emergency Medical Services

Contents

1 Challenge

1.1 Description

First-responders are faced with a challenging task of locating, rescuing and providing necessary aides to victims of a disaster. Even if the responders have necessary resources, delivering them in a safe and timely manner poses a challenge in itself. This is especially true in the case of stranded hikers and visitors of beautiful landscapes and tourist spots in Arizona. Besides this, there is a probability of delay in a rescue operation due to unforeseen and unavoidable circumstances, such as harsh weather and hazardous terrain.

The project aims at tackling the problem in hand, by providing the first-responders with a tool that will allow them to **safely survey an area**, **locate victims** and **deliver essential supplies**. Thus, helping the victim till the time a rescue team reaches them. This will be achieved with a network of quadcopters[3], having two different (inter-changeable) roles. One set of copters will be responsible for surveying an area and identifying potential targets, whereas the second set of copters will be responsible for delivering the supply package.

1.2 Motivation

We are avid nature lovers and frequently hike the scenic Arizona landscapes, including forests, hills and mountains. News about stranded hikers is very common, and given the scorching heat and diverse terrain in Arizona, rescuing these hikers becomes a difficult job for the EMS personnel. Due to the heat, dehydration can come quickly, therefore a quick response is of essence. Sometimes, the only rescue option is via a helicopter, which requires preparation and even longer execution time.

The above reasons made us think about devising a method to help these victims & personnels. Our intended goal is to support the current search & rescue operation, and provide a way of delivering essential supplies to the victims until the EMS personnels reach them.

1.3 Requirements

To support the search operation, we require the tool to be quick to assemble & deploy, and should be able to survey & locate the victims quickly. The other criteria essential for the tool to be successful involves a mechanism to deliver packages safely & quickly. Even if the tool is autonomous[4] or semiautonomous[5], we want human judgment to have significant impact, allowing it to override the default tool behavior. Last, but not the least, we want the solution to leverage available resources for communication and at the same time not be fully dependent on them. The solution should be able to switch between network protocols so as to keep the dependency at it's minimum.

[3]A multi-rotor helicopter, uses four propellers to left and propel

[4]Having the power to govern itself

[5]Largely self governing within an bigger entity

1.4 Importance

In a retrospective cohort study of search & rescue operations (conducted in the state of Oregon), it was observed that out of 1040 searches in the period 1997-2000, involving 1509 victims, 70 (4.6 %) of them died. In the same study, out of 1262 searches in the period 2001-2013, involving 1778 victims, 115 (6.5 %) of them died. On an average, the search & rescue time was 18.5 hours (1997-2000) and 19.75 hours (2001-2003).[2]

The above information is relevant to our cause, because the tool not only allows for quicker response, but also provides wider coverage by deploying multiple copters. The impact of the tool will save countless lives, as well as help in making the SAR operation efficient and less time consuming.

2 Solution

2.1 Concept & Desired Solution

Given the challenge in hand, the solution is required to perform three important tasks:

1. **Survey & Locate victims**: The solution (or a component of it) should be able to survey a designated area and identify potential victims that require rescuing. A good solution should be able to carry out this task in an efficient manner so as to keep the surveillance time as minimum as possible.

2. **Relay victim coordinates**: Once the victims are identified, the solution(or a component of it) should provide the positions of the victim to the base camp[6]. A good solution should then continue surveying the area for locating other targets.

3. **Deliver supply package**: Finally, when the position coordinates of the victims are received by the base camp, the solution(or a component of it) should deliver its payload[7](which may contain communication aides, food packets, medical supplied, etc.). A good solution should deliver this package quickly and at a safe & reachable distance from the victims.

Each of the above tasks are equally valuable towards the mission's success. However, the last 2 tasks rely heavily on the first task. Therefore, a good solution should allow deploying multiple sub-components to boost the first task execution.

2.2 Proposed Solution

Keeping the desired solution in mind, our project leverages aerial capabilities of quadcopters(discussed in XYZ) for surveying an area and delivering packages, and processing capabilities of the Intel-Atom processor for identifying humans from sensory data, as well as providing navigational commands to the quadcopters.

[6]A camp from where SAR activities are coordinated

[7]Package carried by a vehicle

A high-level overview of the system work-flow can be described as follows:

1. **Assembly & Deployment** - The system solution is transported to the disaster location, where it is assembled and a base camp is established.
2. **Surveillance** - A quadcopter is deployed with a target destination. It's main responsibility is to survey within a circular radius from the target position and gather imagery data. It should be able to use pre-planned navigational techniques to completely comb it's designated area. The gathered data should be relayed back to the base camp for further processing.
3. **Image Data Processing** - The imagery data from the quadcopter is processed by the master node(located at the base camp), which uses Image Processing techniques to detect humans from the data. A good solution should be able to detect humans with high accuracy and low computational time.
4. **Delivery** - Once potential targets have been identified and ascertained, a second quadcopter carrying a payload of either communication aides or medical supplies or food packets is deployed. This copter flies directly to the target location and drops off the package near the victims.

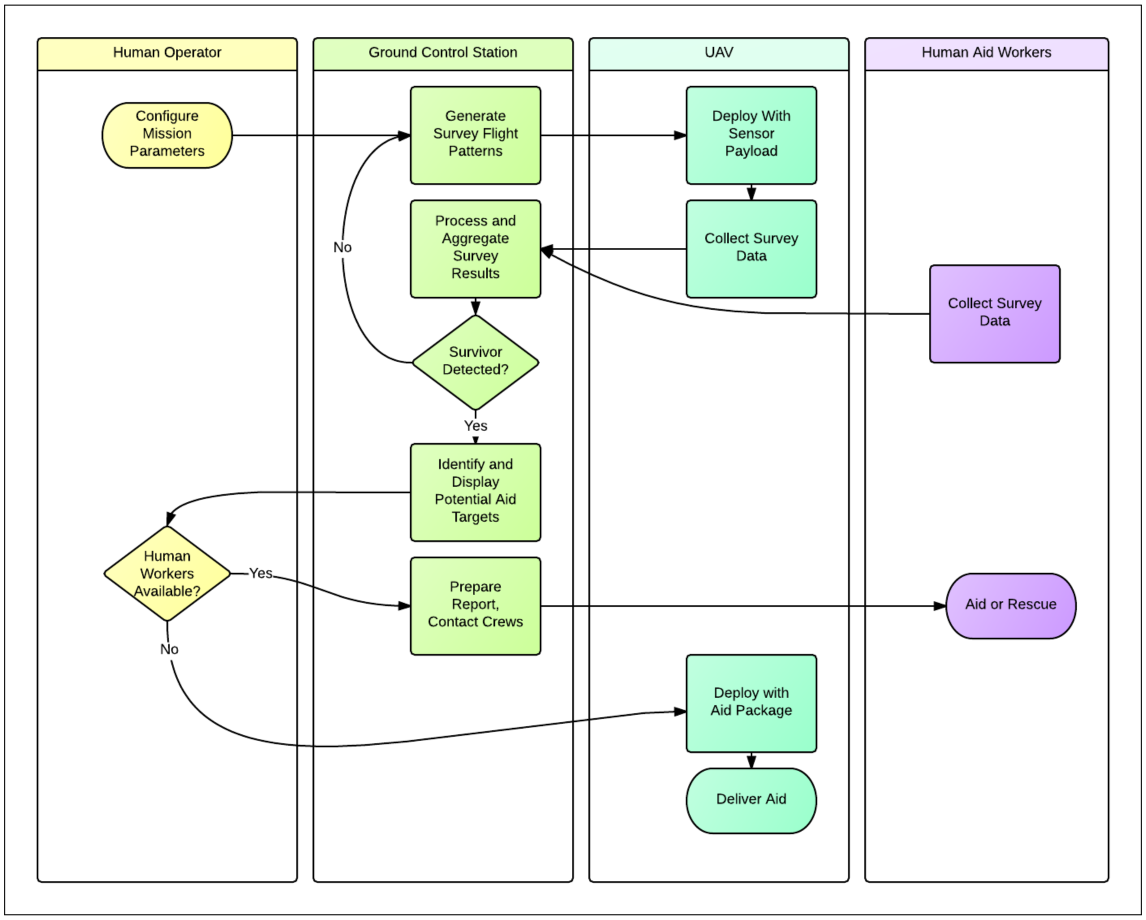


Figure 1: A flow diagram showing the work-flow of the system solution

2.3 Project Components

We've taken a modular approach towards developing our proposed solution, so that it is easier to understand and build. Listed below are the components of the project and their description of role responsibilities:

1. **Quadcopters**:

   Surveillance Copter - This copter will be responsible for surveying an area and gathering images of that area. This data will be sent back to the base camp in real-time for further processing.

   Delivery Copter - This copter will be responsible for delivering the payload at the target location.

2. **Base Camp**:

   The base camp will be powered by the Intel-Atom processor, and it's main responsibility includes processing & extracting objects from the relayed imagery data, providing navigational commands, and dispatching the delivery copter to target location.

2.4 Project Sub-Components

Furthermore, we've divided the above mentioned components into 3 sub-components, as described below:

1. **Hardware & Communications** - This sub-component deals with selection of hardware materials, building the copters and setting up communication infrastructure.

   Hardware Selection & Design - Hardware components were selected on the following criteria: availability, reliability, and quality, ease of use and integration, and cost. After the selection process, design analysis is carried out for determining a suitable design in terms of ease of manufacturing/assembling, and stability and reliability.

   Communications - Communication systems are the backbone of creating a network of intercommunicating robots. Since the copters have minimal computing power, for decision making, communication systems become very important. Having continuous communication channel between the copters and the base camp is of utmost importance. An optimal communication system should be able to handle multiple channels and be broad enough to accommodate heavy data transfer.

   Power Systems - The copters are designed to have a power monitoring module. Continuous monitoring of power consumption, as well as decision making in low power scenarios will be taken care of by the base camp.

2. **Software & Algorithms** - This sub-component deals with navigational algorithms, image processing and interface development.

   Attitude Control - Attitude estimation and control deals with determining the orientation of the copter with respect to it's surrounding environment. Because of the unknown nature of the environment, the copter and it's system should be robust enough to not allow external forces to modify it's orientation and position.

Guidance & Navigation - For an aerial vehicle, Guidance [8] & Navigation[9] refers to planning, sensing and controlling the motion of the vehicle to arrive at a target. One of the major challenges is to determine an optimal path for saving time and energy. For the purpose of the competition, the project aims at developing the system to navigate in an open environment[10].

Image Processing - Image Processing refers to the signal processing of images for either obtaining another image or extract a set of characteristics. Visual sensors, such as video cameras, would be mounted on the surveillance copter for acquiring image data for analysis. The data will be processed at the base camp to extract key descriptors for detecting humans.

3. **Modeling & Simulation** - Before actual implementation of all the algorithms, it is very important to have a running simulation of the system for testing and analytical purposes. Using open-source simulation packages, we've created simulation to demonstrate the project functioning.

2.5 Changes from Original Solution

At the time of conception, the solution was aimed at being fully-autonomous. As we progressed further with our concept, we conducted interviews with the EMS personnels who are the main users for our solution. It was evident from their feedback, that a fully-autonomous solution is not desired, and a human operator should act as the administrator of the system solution. Therefore, we have shifted our focus on developing a semi-autonomous solution, where the copters and the base camp will do all the surveying and processing, but a human operator will be responsible for the final decision, which includes, verifying detected humans and delivery copter dispatch.

2.6 Innovation

Using drones for the purpose of surveillance or dispatch is not new. Drones have been used widely by U.S. Military for reconnaissance usage as well as lethal usage[13]. Drones have also been used for commercial purposes, such as Amazon Prime Air[4].

However, the novel aspect of our solution is to combine the two usages and provide an integrated solution for supporting relief activities. Our solution is modular and inter-changeable in nature. In other words, each component is built keeping modularity in mind. The payload module provides multiple options, and can be customized as per the need of the mission. Secondly, the copters are built keeping multi-purpose utility in mind. A delivery copter can be switched into a surveillance copter just by adding the image module.

While developing and working on our initial concept, we've realized that the above mentioned strengths will allow us to use the solution across multiple scenarios. Some of the uses, besides our main target use, include:

- **Agricultural Use** - Using drones to check for incests and pests on a field, and spraying chemicals to counter them.

[8]The determination of the desired path of travel from the vehicle's current location to the designated target[16]

[9]The method of determining the position, course and distance traveled[16]

[10]An environment where density of obstacles is less

- **Media Arrangements** - Using drones to shoot videos across functions, like graduation convocations & wedding arrangements, and deliver important packages, like graduation speech letters & wedding rings or vows, respectively.

3 Performance Evaluation

3.1 Measures to meet challenge needs

Given the challenge requirements, an optimal solution should adhere to the following measure properties and fit the target goal:

Criteria	Description	Target Goal
Assembly	Time taken to assemble the system at target site	The system should be assembled at the target location within XXX minutes.
Deployment & Data Gathering	Time taken for a component to be deployed from base camp & gather input data for processing	The system should be able to deploy a component within XXX minutes and gather image data for processing.
Detection Accuracy	Accuracy of identifying and detecting humans from the image data	The system should be able to detect humans with at least XXX accuracy and within XXX milliseconds.
Delivery	Time taken to drop off the payload and the quality of payload at the target	The system should drop the payload without any damages to it.

Table 1: Challenge Meeting Measures

3.2 Performance Measure Matrix for Sub-systems

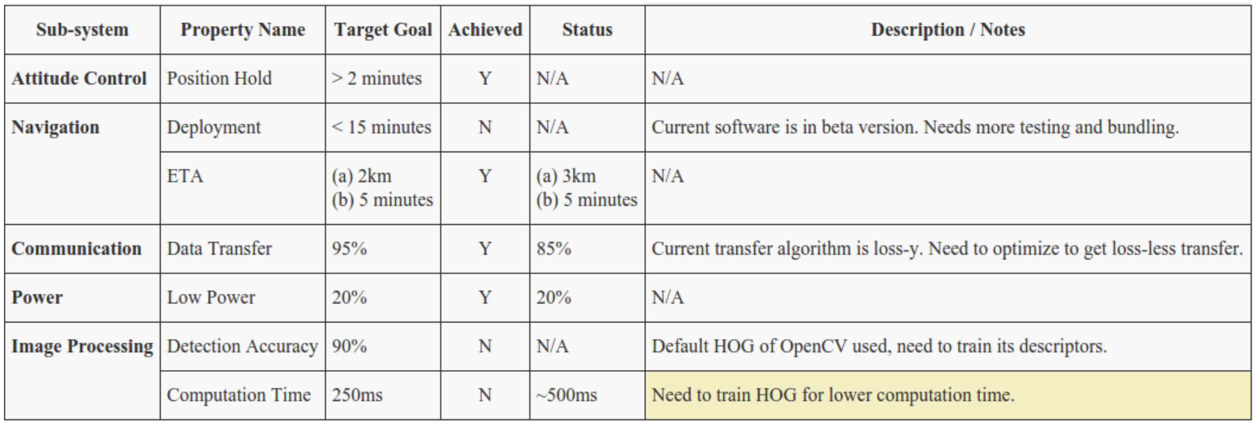
To attain overall goal for an optimal solution, we set out a performance measure matrix for our sub-systems as below:

Sub-system	Property Name	Property Criteria	Target Goal
Attitude Control	Position Hold	Should be able to hold it's position for at least target time	>2 minutes
Navigation	Deployment	Should be able to deploy within target time	<2 minutes
	ETA	Should be able to arrive at it's target location(a) within target time(b)	(a) 500 meters (b) 5 minutes
	Object avoidance	Should be able to avoid obstacles on it's flight path	
Communications	Network Channel	Should be able to communicate constantly on the network	
	Data Transfer	Should be able to transfer data (telemetry & image) with at least target transfer accuracy	95%
Power	Power Monitoring	Should be able to monitor power consumption and estimated remaining power	
	Low Power	Should be able to alert the operator in case of low power than target threshold	20%
Image Processing	Detection Accuracy	Should be able to detect humans in a frame with at least target accuracy	90%
	Computation Time	Should be able to detect humans in a frame within target time	200ms

Table 2: Subsystem Performance Measure Matrix

3.3 Performance Evaluation

After development of the subsystems, we tested them according to the matrix stated in Section 3.2. The below table represents the results:



Sub-system	Property Name	Target Goal	Achieved	Status	Description / Notes
Attitude Control	Position Hold	> 2 minutes	Y	N/A	N/A
Navigation	Deployment	< 15 minutes	N	N/A	Current software is in beta version. Needs more testing and bundling.
	ETA	(a) 2km (b) 5 minutes	Y	(a) 3km (b) 5 minutes	N/A
Communication	Data Transfer	95%	Y	85%	Current transfer algorithm is loss-y. Need to optimize to get loss-less transfer.
Power	Low Power	20%	Y	20%	N/A
Image Processing	Detection Accuracy	90%	N	N/A	Default HOG of OpenCV used, need to train its descriptors.
	Computation Time	250ms	N	~500ms	Need to train HOG for lower computation time.

Figure 2: Subsystem performance evaluation

4 Technical Documentation

The technical documentation section has been designed to outline the components used in the solution. A description of the component and it's sub-components have been provided, in an attempt to guide a reader towards replicating the solution.

4.1 Software Architecture

4.1.1 ROS: Overview

The project uses ROS[11] as its underlying architecture for creating a network of nodes, where each node is responsible for a certain task. ROS is an operating system (not a traditional OS) that provides a structured communications layer for a heterogeneous[12] compute cluster[13][17].

It provides hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers

4.1.2 ROS Nomenclature

A system built using ROS consists of a number of processes connected at runtime in a peer-to-peer topology. These processes are analogous to system modules, and are called **nodes**. A system is typically comprised of many nodes.[17]

[11]Robot Operating System
[12]Diverse in character
[13]Loosely or tightly coupled computers performing a specific task

Nodes communicate via passing a data structure, known as **messages**. A message is a a strictly typed data structure. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types and constants. Messages can be composed of other messages, and arrays of other messages.[17]

These nodes communicate with each other using special communication channels known as **topics**. A node that wants to send a message *"publishes"* it on a specific topic. A node that is interested in a certain kind of data will *"subscribe"* to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics.

Services are another way that nodes can communicate with each other. Services allow nodes to send a request and receive a response.

Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms[6].

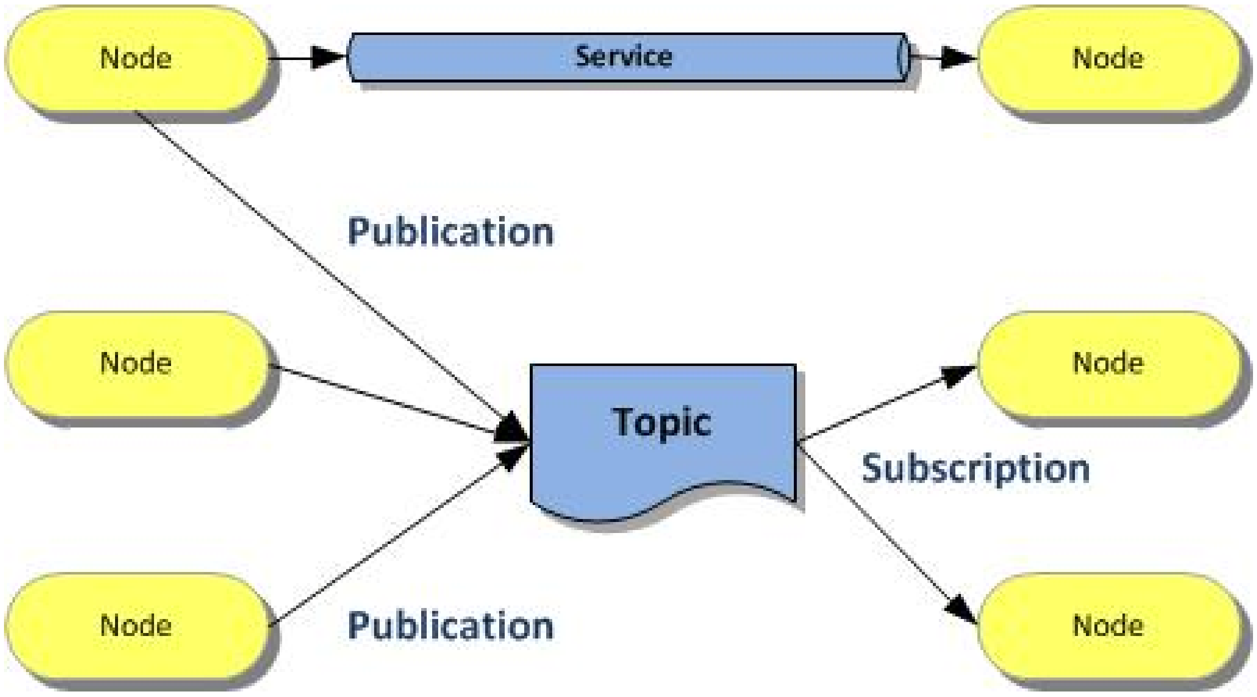


Figure 3: High level overview of ROS Architecture[10]

4.1.3 Why ROS?

ROS is open-source, easily deployable on embedded systems platforms and has a big community. Besides these, the below technical specifications are some of the reasons that convinced us to select ROS over other platforms (such as Microsoft Robotics Developer Studio, Naoqi and URBI) [17]:

- *Peer-to-Peer* - ROS allows to visualize and use components in a system as individual nodes, that communicate with each other via a communication layer provided by ROS.
- *Multi-Lingual* - ROS is language-neutral. ROS currently supports the following different languages: C++, Python, JAVA, Lua, and LISP.
- *Tool-based* - A large number of small tools are used to build and run the various ROS components, rather than constructing a monolithic development and runtime environment. These tools perform various tasks, e.g., navigate the source code tree, get and set configuration parameters, visualize the peer-to-peer connection topology, measure bandwidth utilization, graphically plot message data, auto-generate documentation, etc.
- *Thin* - The ROS build system performs modular builds inside the source code tree, and it's use of CMake makes it comparatively easy to follow this “thin” ideology.
- *Free & Open-Source* - The full source code of ROS is publicly available. ROS is distributed under the terms of the BSD license, which allows the development of both non-commercial and commercial projects.

4.1.4 Using ROS

4.1.4.1 ROS File-system

The file-system describe the hierarchy of components within the ROS system, and are described below[6]:

- **Packages** - Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most atomic build item and release item in ROS. Meaning that the most granular thing you can build and release is a package.
- **Metapackages** - Metapackages are specialized Packages which only serve to represent a group of related other packages.
- **Package Manifests** - Manifests (package.xml) provide metadata about a package, including it's name, version, description, license information, dependencies, and other meta information like exported packages.

- **Repositories** - A collection of packages which share a common VCS[14] system. Packages which share a VCS share the same version and can be released together using the catkin[15] release automation tool.
- **Message (msg) types** - Message descriptions define the data structures for messages sent in ROS.
- **Service (srv) types** - Service descriptions define the request and response data structures for services in ROS.

4.1.4.2 ROS Configuration & Management

ROS (current stable version *Indigo*) can easily be downloaded and installed for it's supported platforms from the following download link.

After installing ROS, we need to setup environment variables to be used by ROS. This can be accomplished by issuing the following command:

```
# source /opt/ros/<distro>/setup.bash

$ source /opt/ros/indigo/setup.bash
```

Snippet 1: Sourcing ROS Environment variables

ROS uses *catkin* as it's workspace manager and build tool. To create a catkin workspace, use the following command:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Snippet 2: Initializing catking workspace

Once the catkin workspace is initialized, you can download repositories and build them using *catkin_make* command.

4.1.4.3 Building our ROS Package

Bundling everything within a package is advantageous in many aspects. ROS packages draw it's strengths from the modular approach used by it's creators. Modularity, in this case, comes in the form of building specific functionality into libraries which can be used by other packages[5]. As a good habit, all use packages should be built within the *src* folder of catkin workspace.

A typical catkin package consists of the following items:

- **package.xml** - Contains the package meta information.
- **CMakeLists.txt** - Input file for the CMake build system for building software packages.

[14]Version Control System

[15]A build system, responsible for generating 'targets' from raw source code that can be used by an end user

Catkin allows a shorthand command for building the boilerplate[16] of a ROS package. To create a package, you can use the following command:

```
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
$ catkin_create_pkg adra std_msgs rospy roscpp
```

Snippet 3: Creating a ROS Package

The above command will create a package named *adra*, which will have dependencies on the following packages *std_msgs*, *rospy* and *roscpp*. After creating our package, we can write our custom library code within this directory, and build it using *catkin_make*.

4.1.4.4 Creating our Nodes

A node a simple executable file within the ROS package. They use ROS client libraries to publish messages on topics, or subscribe to receive messages from topics. Nodes can also be used as service providers or can invoke services.

Before creating and running our nodes, ROS requires us to run a couple of prerequisite processes, which can be done using the *roscore* command. This command invokes and starts the *ROS Master*[17], *ROS Parameter Server*[18] and *rosout logging node*[19].

ROS provides some commands which can help in viewing running nodes[21]:

- **rosnode list** - Displays a list of active node

```
$ rosnode list
# Output will be like:
# /rosout
```

Snippet 4: Listing ROS Nodes

- **rosnode info <node_name>** - Displays the information about a specific node

```
$ rosnode info /rosout
# Output will be like:
# ---------------------------------------------------------
# Node [/rosout]
# Publications:
#   * /rosout_agg [rosgraph_msgs/Log]

# Subscriptions:
#   * /rosout [unknown type]

# Services:
#   * /rosout/set_logger_level
#   * /rosout/get_loggers
```

Snippet 5: ROS Node Information

[16]Piece of code that can be reused in new applications as a template
[17]Provides naming and registration services for nodes
[18]A shared, multi-variate dictionary that is accessible via network APIs
[19]A system wide logging node

- **rosrun [package_name] [node_name]** - Runs a specific node within a ROS package

```
$ rosrun adra adra_master
# Runs the node "adra_master" within the package "adra"

# Running node list command now will list a new node
$ rosnode list
# Output will be like:
# /rosout
# /adra_master
```

Snippet 6: Running a ROS Node

After learning how to view ROS nodes, it's time to create a sample node:

```cpp
// Include the ROS C++ APIs
#include <ros/ros.h>

// Include file declarations
#include <adra/sample_node.h>

int main(int argc, char** argv) {

  // Initialize ROS node named "adra_sample_node"
  ros::init(argc, argv, "adra_sample_node");
  ros::NodeHandle nh;

  // DO WORK

  // Wait for SIGINT/Ctrl-C
  ros::spin();
  return 0;
}
```

Snippet 7: Sample ROS Node Code in C++

Once you save the code for the node, we need to build it and run it.

```
# Don't forget to update the CMakeLists.txt to include our newly created file
$ catkin_make

$ rosrun adra adra_sample_node
```

Snippet 8: Building and Running our node

4.1.4.5 ROS Tools

One of most important driving factor behind choosing ROS is it's rich package repositories and community. We use ***rqt*** extensively since it's a software framework that implements and integrates various available GUI tools in the form of plugins.

Some of the plugins that we find valuable are:

- **Node Graph [rqt_graph]** - A plugin to visualize the ROS Computation graph[20].

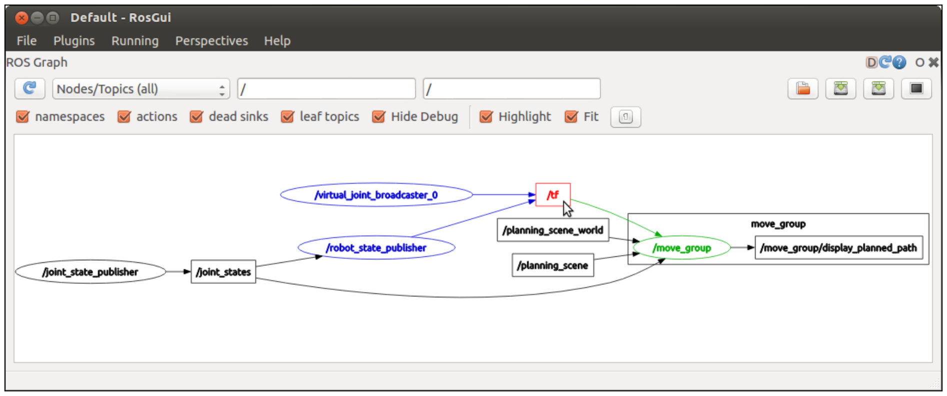
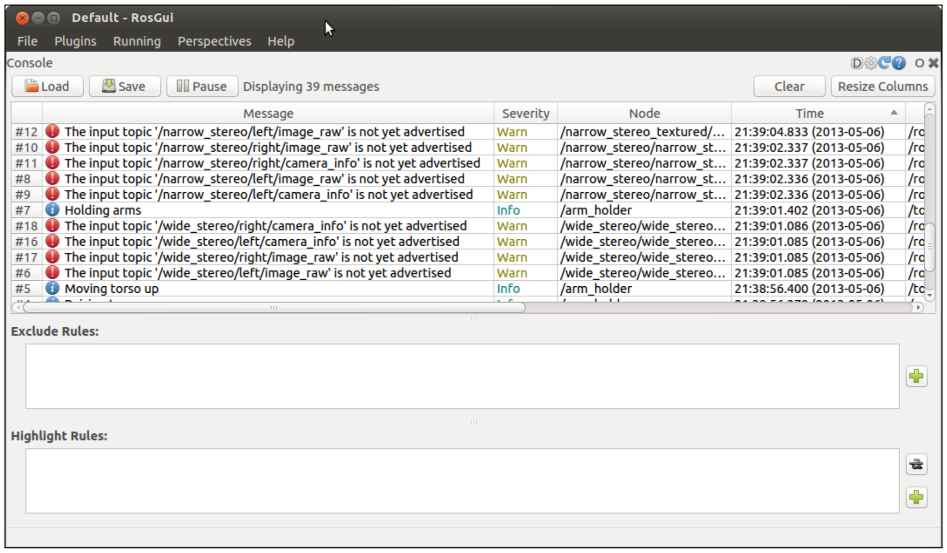


Figure 4: GUI for rqt_graph[19]

- **Process Monitor [rqt_top]** - A plugin to monitor ROS processes.
- **Console [rqt_console]** - A plugin that displays and allows filtering of ROS messages.



	Message	Severity	Node	Time	
#12	The input topic '/narrow_stereo/left/image_raw' is not yet advertised	Warn	/narrow_stereo_textured/...	21:39:04.833 (2013-05-06)	/ro
#10	The input topic '/narrow_stereo/right/image_raw' is not yet advertised	Warn	/narrow_stereo/narrow_st...	21:39:02.337 (2013-05-06)	/ro
#11	The input topic '/narrow_stereo/right/camera_info' is not yet advertised	Warn	/narrow_stereo/narrow_st...	21:39:02.337 (2013-05-06)	/ro
#8	The input topic '/narrow_stereo/left/image_raw' is not yet advertised	Warn	/narrow_stereo/narrow_st...	21:39:02.336 (2013-05-06)	/ro
#9	The input topic '/narrow_stereo/left/camera_info' is not yet advertised	Warn	/narrow_stereo/narrow_st...	21:39:02.336 (2013-05-06)	/ro
#7	Holding arms	Info	/arm_holder	21:39:01.402 (2013-05-06)	/to
#18	The input topic '/wide_stereo/right/camera_info' is not yet advertised	Warn	/wide_stereo/wide_stereo...	21:39:01.086 (2013-05-06)	/ro
#16	The input topic '/wide_stereo/left/camera_info' is not yet advertised	Warn	/wide_stereo/wide_stereo...	21:39:01.085 (2013-05-06)	/ro
#17	The input topic '/wide_stereo/right/image_raw' is not yet advertised	Warn	/wide_stereo/wide_stereo...	21:39:01.085 (2013-05-06)	/ro
#6	The input topic '/wide_stereo/left/image_raw' is not yet advertised	Warn	/wide_stereo/wide_stereo...	21:39:01.085 (2013-05-06)	/ro
#5	Moving torso up	Info	/arm_holder	21:38:56.400 (2013-05-06)	/to

Figure 5: GUI for rqt_console[18]

- **Message Type Browser [rqt_msg]** - A plugin for introspecting available ROS message type.

[20] Peer-to-peer network of ROS processes that are processing data together

- **Topic Monitor [rqt_topic]** - A plugin for displaying debug information about ROS topics including publishers, subscribers, publishing rate, and ROS Messages.

- **Plot [rqt_plot]** - A plugin for visualizing data values in a 2D plot.

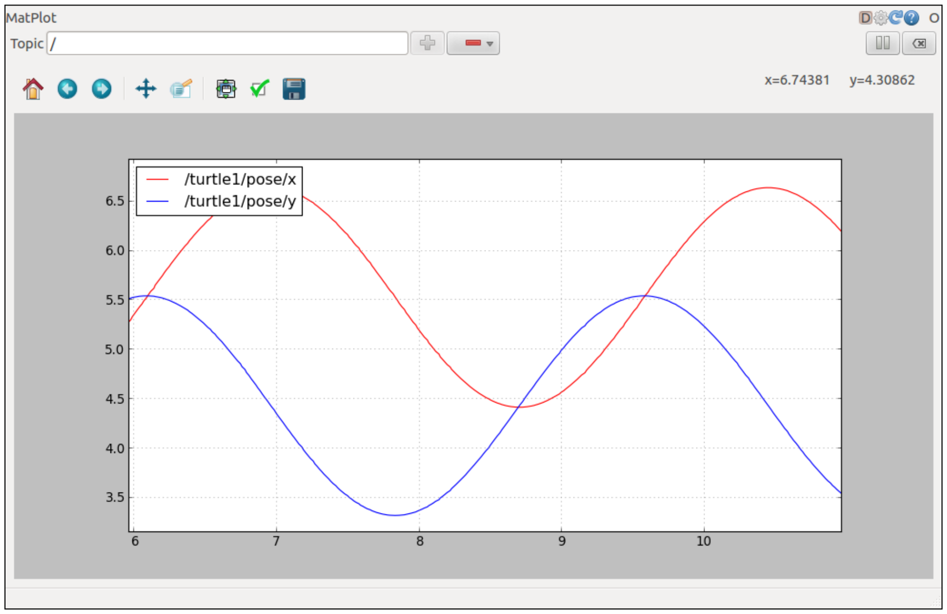


Figure 6: GUI for rqt_plot[20]

Besides the above mentioned plugins, we also use **Gazebo** for our simulation purpose. Gazebo allows us to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. ROS allows integration with Gazebo using the ***gazebo_ros_pkgs***, which is a set of packages that provide wrapper classes around Gazebo.

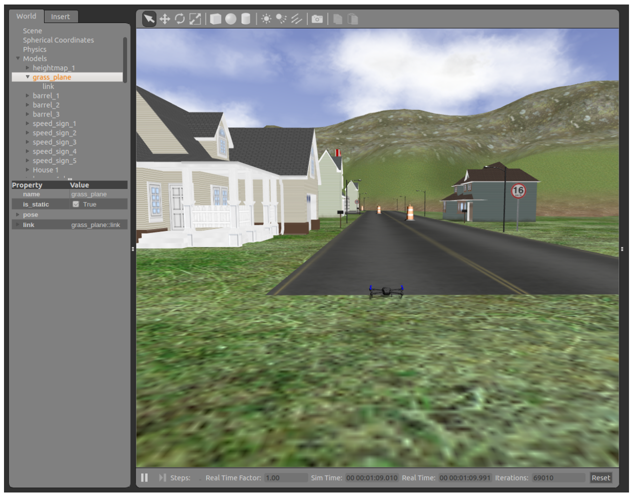


Figure 7: GUI for Gazebo[3]

4.2 Network Infrastructure & Setup

Establishing communication network for data transfer is one of the key areas for a successful solution. An optimal solution for this component should be able to maintain the network for long distance usage. The data channel should be broad enough for sending/receiving image data. While developing the project, we had 2 broad implementation ideas/concepts, which we have discussed below. Due to time & resource limitations, we could only implement a communication network over a one-to-one WiFi network.

4.2.1 One-to-One WiFi Network

One of the original ideas for connecting the quadcopters together was using WiFi to link each robot with the monitoring station. The WiFi connection is important because it will allow the quads to transmit large bandwidth video and images (preferable HD) back to the station for video processing.

The underlying protocol for communication used in this kind of network was MAVLink[21]. MAVLink is a very lightweight, header-only message marshaling library for micro air vehicles. It can pack C-structs over serial channels with high efficiency and send these packets to the ground control station. It is extensively tested and used on multiple drone platforms, such as the PX4, PIXHAWK, APM and Parrot AR-Drone. It serves as their communication backbone for the MCU/IMU communication as well as for Linux interprocess and ground link communication[14].

Messages are transmitted over MAVLink as XML data structures, which are then converted into C/C++, Python or C# Code using generators. A definition of a simple MAVLink message is given below:

```xml
<message id="0" name="HEARTBEAT">
  <description></description>
  <field type="uint8_t" name="type"></field>
  <field type="uint8_t" name="autopilot"></field>
  <field type="uint8_t" name="base_mode"></field>
  <field type="uint32_t" name="custom_mode"></field>
  <field type="uint8_t" name="system_status"></field>
  <field type="uint8_t_mavlink_version" name="mavlink_version"></field>
</message>
```

Snippet 9: Sample MAVLink Message

Description of the message fields are as follows:

- **Description** - The heartbeat message shows that a system is present and responding. The type of the MAV and Autopilot hardware allow the receiving system to treat further messages from this system appropriate (e.g. by laying out the user interface based on the autopilot).
- **Type** - Type of the MAV (quadrotor, helicopter, etc., up to 15 types).
- **Autopilot** - Autopilot Type / Class.
- **Base Mode** - System Mode.

[21] Micro Air Vehicle Communication Protocol

- **Custom Mode** - Navigation Mode.
- **System Status** - System Status flag.
- **Mavlink Version** - Version of the MAVLink protocol

MAVLink also allows us to define our own custom message fields. It is advisable to organize our custom fields into a single message definition file. An example of custom fields is given below:

```
<messages>
  <message id="150" name="QUAD_RAW">
    <description>This message encodes all of the custom data for our UAV.</
    description>
    <field type="uint16_t" name="position"></field>
    <field type="uint8_t" name="port_limit"></field>
    <field type="uint8_t" name="center_limit"></field>
    <field type="uint8_t" name="starboard_limit"></field>
  </message>
</messages>
```

Snippet 10: Custom MAVLink Message

MAVLink also has bridge for ROS developed by the ETH PIXHAWK team, known as MAVROS. The bridge translates between UART/serial ports and UDP links transporting MAVLink and a ROS instance on the computer.

Integrating MAVLink(via MAVROS) is non-intrusive, i.e., it does not need to become a central part of the on-board architecture on our quadcopter. It provides parameter and way-point handles, which are read by the autopilot from the appropriate data structures. Since it is a header-only library, we do not have to compile it. A typical MAVLink flow is described below:

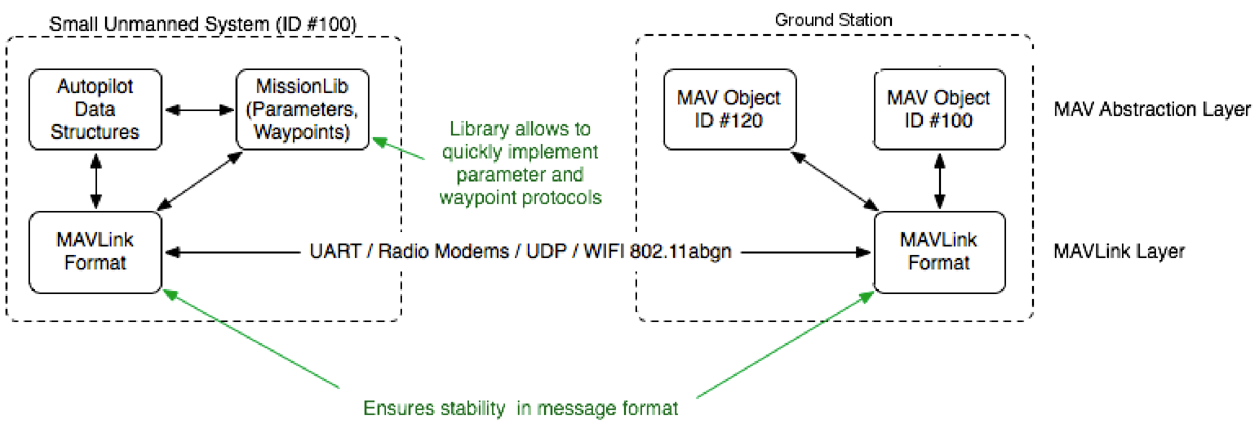

Small Unmanned System (ID #100)
Autopilot Data Structures
MissionLib (Parameters, Waypoints)
MAVLink Format
Library allows to quickly implement parameter and waypoint protocols
UART / Radio Modems / UDP / WIFI 802.11abgn
Ground Station
MAV Object ID #120
MAV Object ID #100
MAVLink Format
MAV Abstraction Layer
MAVLink Layer
Ensures stability in message format

Figure 8: MAVLink Communication Flow

4.2.2 WiFi Mesh Network

One-to-One WiFi network is easy to setup and integrate with our ROS environment. However, there are a few limitations with this method, the biggest being the strength of the WiFi signal and interference from other sources limiting the distance the quads can travel while sustaining the connection.

Another approach towards countering the range problem is to utilize a WiFi mesh network. In a WiFi mesh design, each quadcopter would act as an access point within a network, and each quadcopter would be connected to the others via WiFi and each other. For extremely large swarms of quads, one or two quads can be included that just act as large access points with strong antennas, just like network hubs in a LAN network.

The WiFi mesh network will be implemented using the IEEE 802.11s protocol. This protocol is an 802.11 amendment specifically created for mesh networking. It defines how wireless devices are able to connect together, creating a Wide Local Area Mesh Network (WLAN Mesh), which in this case would be a large ad-hoc network.

In this particular mesh network, each quadcopter will be identified by a pre-assigned static IP address, which will be managed by a router connected to the base station, run by the Intel Altera board provided by the competition. Each quad has a 802.11s capable WiFi adapter, which will allow each quad to become a WiFi access point. On-board software will manage the quadcopters connections to the other robots, actively searching for others which are in range and trying to connect to them. Each quadcopter will have its own object in the Robot Operating System (ROS) project, and all communication between the quads will be handled by a separate program, which will receive inputs from the ROS algorithms.

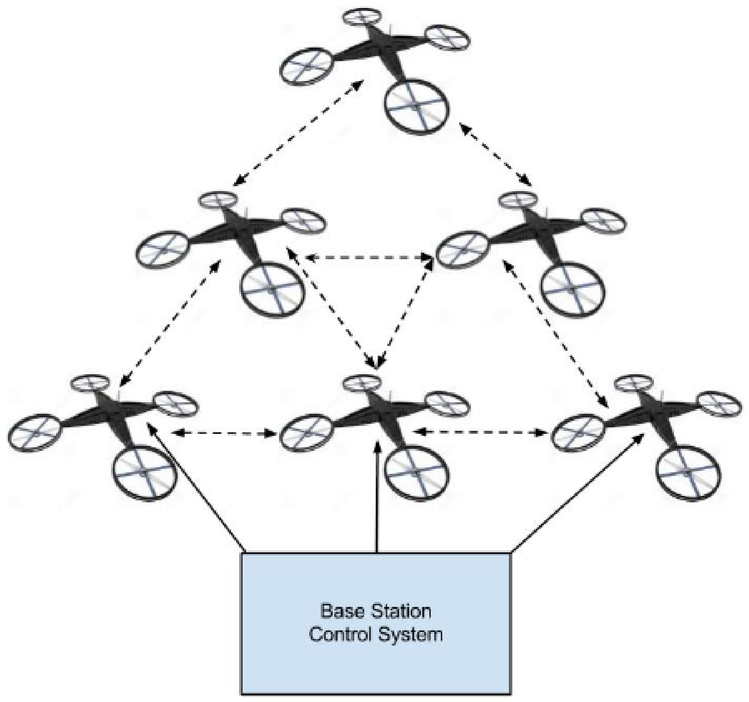


Figure 9: WiFi Mesh Network

As stated before, the primary purpose for this implementation will be in order to extend the range of the quadcopters. The base station will be able to decide which quads will be able to relay back video data. The quadcopters which are out of range from the base station will send their video data through a string of access points on closer quads until it reaches the base station. The data connection path for each of the quadcopters will be determined by the base station every time a new access point is added to the network, just as is done in a DHCP network by a router.

A diagram depicting the WiFi Mesh network is given in fig. 8. The solid arrows represent direct connections between quadcopters and the base station. The dotted lines represent the mesh connections between the access points on the quadcopters.

4.3 Flight Controllers

The flight controller is the nerve center of a drone. Flight control systems can be GPS enabled autopilot systems flown via two way telemetry links or basic stabilization systems using hobby grade radio control hardware[8].

Flight control systems have many sensors available to them – GPS, barometric pressure sensors, airspeed sensors, among others. The major contributors to the flight calculations are still the gyros[22], coupled with accelerometers[23].

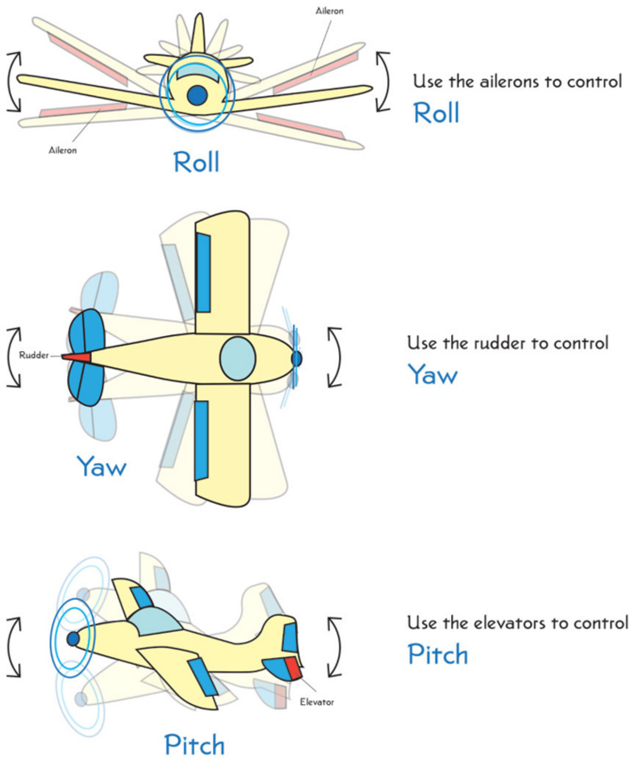

Aileron
Use the ailerons to control
Roll
Aileron
Roll
Rudder
Use the rudder to control
Yaw
Yaw
Use the elevators to control
Pitch
Elevator
Pitch

Figure 10: Flight Dynamic Parameters

[22]Measure rate of rotation about an axis
[23]Measure acceleration

Together, these sensors can help measure the 3 critical flight dynamic parameters (refer fig. 9 [9]):

- **Roll** - Rotation around the front-to-back axis
- **Pitch** - Rotation around the side-to-side axis
- **Yaw** - Rotation around the vertical axis

Special devices, called Inertial Measurement Units (IMU), are used to read the sensors and use sensor fusing algorithms to determine current position and attitude. These readings are then made available to the autopilot for use or transmission.

4.3.1 PixHawk PX4

Our choice over several other autopilot projects (such as ARDUPilot, OpenPilot, etc.) was PX4. The main reasons behind choosing PX4 was it's easy to understand tutorials, integration with ROS using the MAVROS package and a provision of simulating the PX4 stack on ROS+Gazebo.

PX4 is an independent, open-source, open-hardware project aiming at providing a high-end autopilot to the academic, hobby and industrial communities, licensed under BSD license. It is a high-performance autopilot-on-module suitable for fixed wing, multi rotors, helicopters, cars, boats and any other robotic platform that can move[12].

Figure 11: Pixhawk PX4 Flight Controller

4.3.1.1 Technical Specifications[12]

- **Processor** - 32-bit STM32F427 Cortex M4 core with FPU
- **RAM** - 256 KB
- **Flash Memory** - 2 MB
- **Co-Processor** - 32 bit STM32F103

4.3.1.2 Software

The open-source software suite used with PX4 contains everything to let airborne system fly including the QGroundControl[15] and 2/3D aerial maps (with Google Earth support) and drag-and-drop waypoints.

4.3.1.3 Integrating with ROS

A bridge connection needs to be setup with an on-board system, for trading MAVLink messages. A companion computer[24] can be connected using a UART[25] cable, or a UART adapter or a USB[26] cable.

After setting up the bridge connection, we can run a ROS node on the on-board computer (e.g. Odroid Board), which can then be connected to our base camp using a WiFi network.

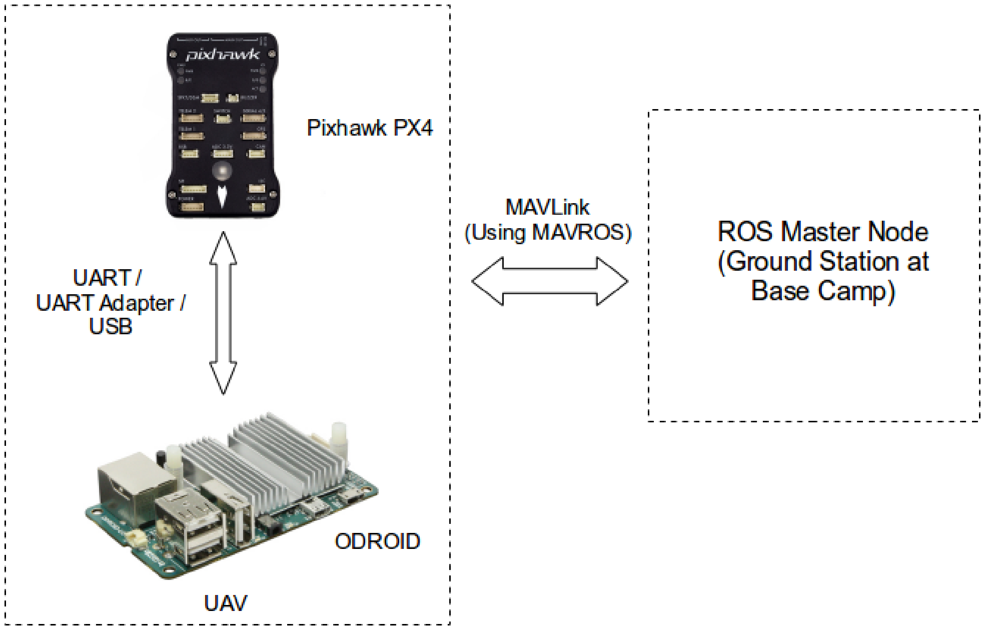

Pixhawk PX4
UART / UART Adapter / USB
ODROID
UAV
MAVLink (Using MAVROS)
ROS Master Node (Ground Station at Base Camp)

Figure 12: PX4 & RSOS Integration

[24]On-board computer connected with bridge connection
[25]Universal Asynchronous Receiver / Transmitter
[26]Universal Serial Bus

The ROS node on our on-board computer can be run by using the sample launch file below:

```
<launch>

  <arg name="fcu_url" default="serial:///dev/ttyACM0:921600" />
  <arg name="gcs_url" default="udp://:14556@192.168.150.2:14550" />
  <arg name="tgt_system" default="1" />
  <arg name="tgt_component" default="50" />

  <node name="mavros" pkg="mavros" type="mavros_node" output="screen">
    <param name="fcu_url" value="$(arg fcu_url)" />
    <param name="gcs_url" value="$(arg gcs_url)" />
    <param name="target_system_id" value="$(arg tgt_system)" />
    <param name="target_component_id" value="$(arg tgt_component)" />

    <rosparam command="load" file="$(find mavros)/launch/px4_blacklist.yaml"/>

    <!-- enable heartbeat send and reduce timeout -->
    <param name="conn_heartbeat" value="5.0" />
    <param name="conn_timeout" value="5.0" />

    <!-- automatically start mavlink on USB -->
    <param name="startup_px4_usb_quirk" value="true" />

  </node>

  <node name="camera" pkg="usb_cam" type="usb_cam_node">
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="800" />
    <param name="image_height" value="600" />
    <param name="pixel_format" value="mjpeg" />
    <param name="framerate" value="30" />
    <param name="camera_frame_id" value="webcam" />
  </node>

</launch>
```

Snippet 11: PX4 ROS Launch File Sample

The above launch file launches 2 nodes, a **MAVROS** node and a **Camera** node. The node publishes data to a topic named *camera/image_raw*. We can then visualize the image data sent via MAVLink using *rqt* plugin for image viewing.

Using ROS command for topic information, *rostopic hz /camera/image_raw*, we can see the data transfer information like below:

```
subscribed to [/camera/image_raw]
average rate: 30.026
  min: 0.027s max: 0.037s std dev: 0.00237s window: 28
average rate: 30.016
  min: 0.027s max: 0.041s std dev: 0.00250s window: 58
average rate: 30.057
  min: 0.027s max: 0.041s std dev: 0.00243s window: 88
```

Snippet 12: ROS Topic Information

4.4 Multirotor Airframe & Power Systems

This project relies on small unmanned aerial vehicles (SUAS) or surveying and package delivery. There are four main categories of SUAS used today: Multicopters, Helicopters, Fixed Wing, and VTOL Fixed Wing. For the purposes of this project, we chose to implement an initial concept using multicopters due to their mechanical simplicity and ease of deployment.

Due to recent advancements in microprocessor and inertial sensing technology, multicopters are now very capable and affordable for the hobbyist market. We chose to construct our quadcopters from components designed for hobbyists due to their low cost and commercial availability. By using components that are available off the shelf we are able to create an affordable system and devote more development time to our innovative software functionality

Our primary objectives for the multirotor design were to maximize range and flight time.

4.4.1 Power Systems

A multirotor power system consists of several components that are all interdependent. When designing a multirotor, one must consider the requirements of their system and select propellers, motors, speed controllers, and power sources accordingly. In order to have an efficient system, care must be taken to pair each of these components.

4.4.2 Propellers

Generally, a slower moving propeller with a large disc area is more efficient than a faster moving propeller of smaller disc area producing an equal amount of static thrust. Because of this, we decided to use large 17” propellers to produce the thrust for our system.

In order to most effectively select motors and the power source, potential component sets were simulated using the ecalc xcoptercalc application maintained by Markus Muller. Twelve potential sets of commercially available components were compared using ecalc. From these simulations, the following set of components was selected.

4.4.3 Motors

Turnigy Multistar Elite 3508 motors were selected. These motors have a stator that is 35mm in diameter by 8mm tall and a KV of 268RPM/Volt. Because of the large stator and slow KV, these motors are capable of producing a large amount of torque very efficiently at low RPM. These characteristics pair well with the 17” propellers we selected.

4.4.4 Power Supply

A Turnigy Multistar 6s (24v) 8Amp hour LiPo was selected for the power supply. Lithium Polymer (LiPo) batteries are popular for multicopters due to their high energy density, high current output and their relatively low cost. The motors we selected produce their maximum rated power output at the 24v voltage of a 6 cell LiPo. By using 6 cell as opposed to the more common 4 cell, we are able to deliver more power to the motors with less current. This allows the usage of a lighter battery with a lower current rating. Additionally, this allows us to use smaller gauge wire and connectors for power distribution, contributing additional weight savings.

4.5 Image Processing

Image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it[1].

For our image processing in the project, we have heavily utilized OpenCV[27]. It is a library of programming functions mainly aimed at real-time computer vision. OpenCV is written in C++ and its primary interface is in C++. It also has full interfaces in Python, Java and MATLAB/OCTAVE. It implements both infrastructure operations and image-processing and vision functions[11].

As an introduction to Image-processing (since it is one of the core areas of our solution), we would like to share some basic tutorials on image-processing for human detection using OpenCV and HOG algorithm[7].

NOTE: We've uploaded sample OpenCV code on our blog and github.

4.5.1 Basic Image Processing: Capturing video from camera

Reading image data from a capture object, like a USB Camera or built-in camera, is really simple. We use the *VideoCapture* object of OpenCV, and read the data frame-by-frame. Sample code and in-line comment explanation will help you understand video capturing better:

```cpp
// Include the OpenCV Header Files
#include "opencv2/highgui/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
  // Create a VideoCapture Object and Open camera "0"
  VideoCapture cap(0);

  // Check if camera is opened
  if (!cap.isOpened())
  {
    cout << "Cannot open the video cam" << endl;
    return -1;
  }

  // Get Frame Size (Width and Height)
  double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH);
  double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT);

  // Create a window to show image
  namedWindow("MyVideo",CV_WINDOW_AUTOSIZE);



```

[27]Open Source Computer Vision

```

  while (1)
  {
    Mat frame;

    // Read new frame into matrix
    bool bSuccess = cap.read(frame);

    if (!bSuccess)
    {
      cout << "Cannot read a frame from video stream" << endl;
      break;
    }

    // Show the frame in the window
    imshow("MyVideo", frame);

    // Quit if user presses 'esc' key
    if (waitKey(30) == 27)
    {
      cout << "esc key is pressed by user" << endl;
      break;
    }
  }
  return 0;
}
```

Snippet 13: Video Capture Sample Code

4.5.2 Human Detection: Histograms of Oriented Gradients

Histograms of Oriented Gradients(HOG) are feature descriptors that are used in the field of image processing for human detection. It evaluates and counts the occurrences of gradient orientations in a localized portion of an image. In other terms, an object appearing in an image can be described by its *distribution of intensity gradients*.

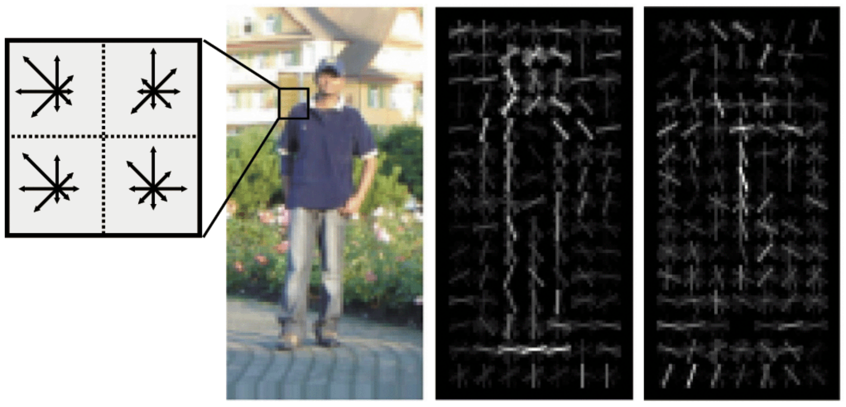


Figure 13: Overview of how HOG works

A sample implementation of HOG algorithm in OpenCV can be done as below:

```cpp
#include <iostream>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main (int argc, const char * argv[])
{
    VideoCapture cap(CV_CAP_ANY);
    cap.set(CV_CAP_PROP_FRAME_WIDTH, 320);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT, 240);
    if (!cap.isOpened())
        return -1;

    Mat img;
    HOGDescriptor hog;
    hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());
    namedWindow("video capture", CV_WINDOW_AUTOSIZE);
    while (true)
    {
        cap >> img;
        if (!img.data)
            continue;
        vector<Rect> found, found_filtered;
        hog.detectMultiScale(img, found, 0, Size(8,8), Size(32,32), 1.05, 2);

        size_t i, j;
        for (i=0; i<found.size(); i++)
        {
            Rect r = found[i];
            for (j=0; j<found.size(); j++)
                if (j!=i && (r & found[j])==r)
                    break;
            if (j==found.size())
                found_filtered.push_back(r);
        }
        for (i=0; i<found_filtered.size(); i++)
        {
        Rect r = found_filtered[i];
          r.x += cvRound(r.width*0.1);
        r.width = cvRound(r.width*0.8);
        r.y += cvRound(r.height*0.06);
        r.height = cvRound(r.height*0.9);
        rectangle(img, r.tl(), r.br(), cv::Scalar(0,255,0), 2);
    }
        imshow("video capture", img);
        if (waitKey(20) >= 0)
            break;
    }
    return 0;
}
```

Snippet 14: HOG Implementation using OpenCV

5 Project Execution

5.1 Challenges in Project

Some of the challenges faced during execution of the project are:

1. **Power Consumption Dilemma**:
   - **Problem**: Power consumption overheads for communication and image processing tasks.
   - **Solution**: Running optimization tests to figure out what level of data compression is suitable for keeping overheads minimum.
2. **Low Power Plan of Action**:
   - **Problem**: In case of low power, what should the plan of action be? Should we continue surveying and gather more data causing a lost in copters, or should we recall the copters causing a loss in data and mission time?
   - **Solution**: By default, we will be alerting the operator when power is below threshold. The operator's decision and judgement will be followed by the system.
3. **Network Configuration**:
   - **Problem**: The current network setup (Peer-to-Peer) has a major issue of low range for mission. We want the system to be used for longer mission radius.
   - **Solution**: We plan to implement the WiFi Mesh network as discussed in section 4.2.2.
4. **Object Detection**:
   - **Problem**: The default HOG algorithm is designed for smaller images(1/3rd the size of current captured image). Using this algorithm for this scale of image causes heavy processing, leading to slow computation time.
   - **Solution**: Overriding the default descriptors of HOG by training it will help in increasing accuracy and decrease computation time.

5.2 Risks & Mitigations

Every project has risks, either identified before-hand or encountered while executing. The below list describes the risks we have faced while executing the process, as well as mitigation steps taken:

1. **Damages**:
   - **Self Damage**: While deployment, the copter might damage itself or it's surroundings if sufficient space is not provided. A solution for this risk is to use a custom 3m X 3m launchpad.

- **Environment Damages**: Due to sharp propellers rotating at high speeds, there might be a possibility of accidental damages if not handled properly. Our solution is to fabricate a brace/ring around the propeller so as to keep it from causing damages.
- **Hardware Damages**: Many individual components are used to build the whole system together. Protecting these components, while keeping the weight to a reasonable limit is a challenge. We have fabricated the design of the copter in such a way that it's main components(flight controller, battery and on-board computer) are protected.

2. **Obstacles**:

- **Airspace Disturbance**: There might be a risk of other aerial vehicles sharing the same airspace. In such a scenario, we need to plan our route or take an action to avoid collision.
- **Environment Disturbance**: Sometimes, the weather or environment may not be friendly for flying the copters. In such cases, safety overrides should be provided to the operator for safely landing and retrieving the copter.

5.3 Peer Reviews

While executing the project, we have tried to conduct reviews with peers, target audience and industry experts. Some of the feedback given by them have been listed below:

1. **Target Audience (EMS Personnels)**

   (a) **Feedback** More control for the operator.
   Impact Change of scope of project, from autonomous to semi-autonomous.

   (b) **Feedback** Different payload modules.
   Impact 3 different payload *cartridges* housing packages for varying needs (like communication aides, food packets and medical supplies).

2. **Industry Expert (Network Configuration)**

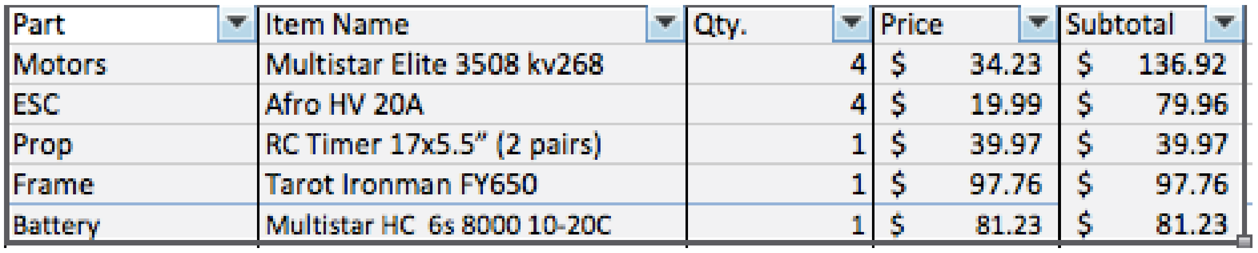
   (a) **Feedback** Less reliance on Peer-to-peer network due to its inability to function over long range.
   Impact Learning about WiFi mesh network using IEEE 802.11s protocol, and implementing it as a future task.

3. **Peers**

   (a) **Feedback** Allowing multiple UAV configurations
   Impact UAV configurations like multicopter(besides quadcopter) and fixed-wings are being researched on.

5.4 Component List and Budget

The below list describes the components used in building a copter and their prices. Besides this, the copter also uses a **Pixhawk PX4** flight controller which costs around 200 $.



Part	Item Name	Qty.	Price	Subtotal
Motors	Multistar Elite 3508 kv268	4	$ 34.23	$ 136.92
ESC	Afro HV 20A	4	$ 19.99	$ 79.96
Prop	RC Timer 17x5.5" (2 pairs)	1	$ 39.97	$ 39.97
Frame	Tarot Ironman FY650	1	$ 97.76	$ 97.76
Battery	Multistar HC 6s 8000 10-20C	1	$ 81.23	$ 81.23

Figure 14: Components used and their prices (per copter)

6 Recommendations & Next Steps

There are a few important additions to this project that need to be made in the immediate future.

1. The first change that needs to be implemented is upgrading the communication systems of the quad copters. The major change that needs to be done for this is implementing the WiFi mesh network with a group of the quadcopters. This is important because it will allow for easy control of larger swarms of multicopters, as well as allowing quads far away from the base station to send back high bandwidth video data without a decline in quality. The other communication system that can be added to the quads is a system that allows the first responders to communicate with the located victims, such as an audio connection.

2. The next major change is creating replaceable payload components for multi-domain projects. This includes adding payloads consisting of more advanced sensors for detection and environment analysis. The main sensors currently being considered are a thermal camera, and LIDAR. The thermal camera would make it much easier to detect humans in large environments (especially at night in the desert). The IDAR would allow for the quad to implement Simultaneous Localization and Mapping (SLAM), which would allow it to crease three dimensional models of the surrounding environments; these maps would make it easier for autonomous route planning for the rest of the quads as well as for the first responders. The other payload change would be modifying a delivery payload so it can also contain basic detection sensors, like HD cameras.

3. The last of the important additions to this project that should be done is upgrading and optimizing several of the software components. The two big pieces of the software that need to be optimized are the path planning algorithms and the user interface for the ground station. The path planning algorithms needs to be upgraded to use more data about the surrounding environment. This can be aided with new data from the upgraded payloads, such as SLAM data. The ground control station needs to be upgraded so the system is easy for first responders to use and requires very little training on their part. If possible, the use of the on-board FPGAs in the Intel embedded board can also be utilized to improve the image processing performance and return more accurate results.

Appendix: Figures

Appendix: Code Snippets

References

[1] Acharya, T., and Ray, A. *Image Processing: Principles and Applications*. Wiley, 2005.

[2] Adams, A., Schmidt, T., Newgard, C., Federiuk, C., Christie, M., Scorvo, S., and DeFreest, M. Search is a time-critical event: when search and rescue missions may become futile. *Wilderness & Environmental Medicine 18* (2007), 95--101.

[3] Baranov, I. *Clearpath Robotics: ROS 101*, Mar. 2014. `http://www.clearpathrobotics.com/blog/ros-101-drive-husky/`.

[4] Barr, A. Amazon testing delivery by drone, ceo bezos says. *Retrieved April 22* (2013), 2014.

[5] Bohren, J. *Building Modular ROS Packages*, Feb. 2014. `http://jbohren.com/articles/modular-ros-packages/`.

[6] Conley, K. *ROS Documentation: Concepts*, Sept. 2009. `http://wiki.ros.org/ROS/Concepts`.

[7] Dalal, N., and Triggs, B. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* (2005), vol. 1, IEEE, pp. 886--893.

[8] Fabio, A. *Droning On: Choosing a Flight Controller*, June 2014. `http://hackaday.com/2014/06/06/droning-on-flight-controller-round-up/`.

[9] How Things Fly. *Roll, Pitch, and Yaw*. `https://howthingsfly.si.edu/flight-dynamics/roll-pitch-and-yaw`.

[10] Laplace, J. *Robot Operating System Overview*, Nov. 2011. `http://generationrobots.developpez.com/tutoriels/presentation-robot-operating-system/`.

[11] Parker, J. *Algorithms for Image Processing and Computer Vision*. IT Pro. Wiley, 2010.

[12] Pixhawk. *Pixhawk PX4 Specifications*. `https://pixhawk.org/modules/pixhawk`.

[13] Posen, B. R. Command of the commons: the military foundation of us hegemony. *International Security 28*, 1 (2003), 5--46.

[14] QGroundControl. *Creating a MAVLink Message*. `http://qgroundcontrol.org/mavlink/create_new_mavlink_message`.

[15] QGroundControl. *Ground Control Station for Small Air|Water|Land autonomous unmanned vehicles*. `http://qgroundcontrol.org/`.

[16] Quadrelli, M., McHencry, M., Wilcox, B., Hall, J., Volpe, R., Nesnas, I., Nayar, H., Backes, P., Mukherjee, R., Matthies, L., Zimmerman, W., and Mittman, D. Guidance, navigation, and control technology assessment for future planetary science missions. *Planetary Science Program Support Task, National Aeronautics and Space Administration* (2013), i--10.

[17] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. Ros: an open-source robot operating system. In *ICRA workshop on open source software* (2009), vol. 3, p. 5.

[18] Saito, I. *ROS Documentation: RQT Console*, Sept. 2012. `http://wiki.ros.org/rqt_console`.

[19] Saito, I. *ROS Documentation: RQT Graph*, Sept. 2012. `http://wiki.ros.org/rqt_graph`.

[20] Saito, I. *ROS Documentation: RQT Plot*, Sept. 2012. `http://wiki.ros.org/rqt_plot`.

[21] Wise, M. *ROS Documentation: Understanding Nodes*, Aug. 2009. `http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes`.