UAV Landingpad

Senior Design Final Documentation

Team Expeditus

Steven Huerta Christopher Smith Johnathan Dixon Dylan Geyer

April 26, 2016

Contents

List of Figures

Overview Statements

0.1 Mission Statement

Mission statement inserted here.

0.2 Elevator Pitch

Elevator Pitch inserted here.

1

Overview and concept of operations

The UAV Lander project requires the autonomous take-off, waypoint navigation, and landing of a UAV. Most of this can be provided by built-in capabilities of a flight controller. However, the larger problem within this project is to land the UAV within $\pm$.1m of the center of the landing pad and with minimal error in orientation. This section will provide the reader with a broad overview of the technologies relating to the objectives of this project.

1.1 Scope

This document provides the reader with an understanding of the UAV Landing Project to include the purpose of the project, the project's main system components, how these components will function together, and a description of the technologies used to develop this project. This document is limited to the technologies that are/will be implemented on the build of the UAV. Necessarily, this document does not extend to some of the development tools or include details concerning the technologies involved with the landing pad.

1.2 Purpose

The purpose of this project is to develop software that will enable a UAV to autonomously take-off, navigate through some number of user defined waypoints, return to the landing pad, and land with $\pm$.1m of the center of the landing pad with $\pm 15^{\circ}$ of the correct orientation. The platform for testing this software will be a carbon fiber frame hexrotor controlled by a flight controller augmented with input from an ODroid.

1.2.1 Flight Controller

The flight controller unit(FCU) will control the manual or autonomous flight of the UAV. The autonomy, specifically, will address the functions of take-off and waypoint navigation. Autonomous landing using relatively inexpensive flight controllers can typically only provide an accuracy of $\pm$10m, though many users claim that accuracy is half of that.

The team has selected the Pixhawk flight controller, with a GPS peripheral. The GPS unit will allow for waypoint navigation controlled by a ground control station(which is a piece of software to tell the UAV where to go and what to do). The Pixhawk includes:

- Pixhawk autopilot
- Buzzer
- Safety switch button
- 3DR power module with XT60 connectors and 6-position connector cable
- Extra 6-position cable to connect a 3DR GPS+Compass module
- Micro USB cable

- SD card and adapter
- Mounting foam
- 3-wire servo cable
- I2C splitter module with cable

It also includes the features:

- Advanced 32 bit ARM Cortex® M4 Processor running NuttX RTOS
- 14 PWM/servo outputs (8 with failsafe and manual override, 6 auxiliary, high-power compatible)
- Abundant connectivity options for additional peripherals (UART, I2C, CAN)
- Integrated backup system for in-flight recovery and manual override with dedicated processor and stand-alone power supply
- Backup system integrates mixing, providing consistent autopilot and manual override mixing modes
- Redundant power supply inputs and automatic failover
- External safety button for easy motor activation
- Multicolor LED indicator
- High-power, multi-tone piezo audio indicator
- microSD card for long-time high-rate logging

The large reason behind using the Pixhawk is that there is an established API(Mavlink) to send commands and receive commands with the flight controller. Mavros will provide a convenient handle so we can take advantage of the built-in autonomy for take-off and waypoint navigation, yet still have the ability to interrupt the built-in autonomy when reaching the landing zone.

1.2.2 ODroid

The ODroid XU4 is small board computer featuring:

- Samsung Exynos5422 Cortex™-A15 2Ghz and Cortex™-A7 Octa core CPUs
- Mali-T628 MP6(OpenGL ES 3.0/2.0/1.1 and OpenCL 1.1 Full profile)
- 2Gbyte LPDDR3 RAM PoP stacked
- eMMC5.0 HS400 Flash Storage
- 2 x USB 3.0 Host, 1 x USB 2.0 Host
- Gigabit Ethernet port
- HDMI 1.4a for display

This computer, with 8 cores, is enough to run a full Ubuntu 14.04 install, which is needed to run a ROS Indigo/Jade Distro. This computer will also be connected to one or two cameras, and be responsible for processing the image stream, calculating needed landing instructions, and passing those instructions to the Flight Controller.

1.2.3 Hexrotor

The hexrotor is the platform our team will using to test and demonstrate our autonomous landing implementation(take-off and waypoint will be handled by the flight controller's built-in autonomy). The hexrotor was decided upon as being more stable, and therefore more desirable, than a quadrotor design. The hexrotor specifically includes:

- Turnigy Talon Hexcopter (V1.0) Carbon Fiber Frame(625mm diameter)
- 6 AeroSky Performance Brushless Multi-Rotor Motor MC3525 850KV
- 6 Exceed RC Proton 30A Brushless ESC Speed Controller
- Hexacopter Power Distribution Board
- APM Power Module with XT60 Connectors Kit
- 3 Each Carbon Fiber Propeller 10x4.7 Black (CW/CCW)
- Battery

A custom cage will be designed and constructed to house the Pixhawk, GPS, and ODroid XU4.

1.2.4 Robot Operating System(ROS)

The Robot Operating System is a pseudo-operating system that provides libraries and tools allowing a fast integration of hardware. ROS creates a network to allow the passing of information between nodes that either push or pull information as needed. Our implementation will use ROS to create a network of nodes that will push or pull information, such as pulling images to calculate direction and distance needed to reach the landing pad. A node will be responsible for pushing that information to the flight controller via MavROS.

1.2.5 Mavlink

Mavlink(Micro Air Vehicle link) is a message protocol for communicating with small unmanned vehicles including ground vehicles, planes, and helicopters. Mavlink is often used for many flight controllers, providing a good interface for sending commands, as well as receiving information from the flight controller. The team will be using MavROS in implementation, which is Mavlink, but with a ROS wrapper. This will allow us to keep our ODroid side implementation in a ROS environment.

1.3 Systems Goals

The goals of this project:

1. Autonomously take-off from landing pad.
2. Autonomously navigate through a series of waypoints.
3. Autonomously return to the landing pad.
4. Autonomously land within $\pm$.1m of the center of the pad.
5. Autonomoulsy land with the correct orientation $\pm 15^{\circ}$.

1.4 System Overview and Diagram

The flight controller will be responsible for take-off and waypoint navigation. As described earlier, the Pixhawk flight controller is capable of receiving instructions through the use of a GUI, a ground control station, to establish take-off, waypoint navigation, including navigation back to the landing pad. The ODroid will be receiving and listening to the instructions(or missions) being executed by the flight controller. Reaching the last waypoint(the landing pad), the ODroid will interrupt the autonomous landing on the flight controller. The ODroid will use a

camera and the stream of images to estimate movements to reach the landing pad, on center and with correct orientation. These instructions will be sent to the flight controller. The instructions will continuously be sent to the flight controller until the UAV has landed(Fig. 1.1).

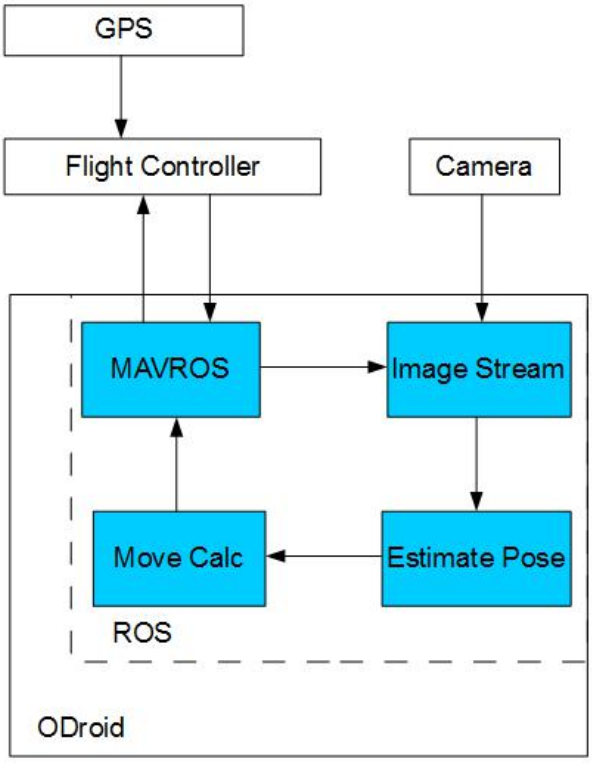


Figure 1.1: Communication between Flight Controller and ODroid

1.5 Technologies Overview

HARDWARE DEPENDENCIES:

- 3DR Pixhawk Flight Controller - The flight controller will provide the ability to fly the hexrotor craft manually and autonomously. More information here.
- 3DR GPS - Provides global coordinates, which is essential for gps defined waypoint navigation. More information here.
- ODroid XU4 - This will serve as the host for the ROS framework for processing image information, calculating move instructions, and sending those instructions to the flight controller. More information here.
- Camera - Currently a camera has not been selected. The camera will need the capability of clearly recognizing lights and discriminating color at a distance of up to 10m at at least 30fps.

SOFTWARE DEPENDENCIES:

- Ubuntu 14.04 - ROS necessitates the use of this OS. This will be the environment found on the ODroid. More information here.
- OpenCV - This open source library provides built-in image processing that will allow the team to process images from the camera to compute move instructions. More information here.

- Robot Operating System(ROS) - ROS will serve as the structure for the program so that we can construct a framework to process an image, derive some decision from the image, and send the instruction to the flight controller. More information here.
- Mavlink - A protocol to communicate with the flight controller, such as to send instructions or receive some information about the flight. More information here.

2

Project Overview

This section provides an overview of project management and a brief overview of the phases of the project.

2.1 Team Members and Roles

Team members and assigned roles:

- **Steven Huerta** - Team Lead, Simulation, AI Landing
- **Christopher Smith** - Simulation, AI Landing
- **Jonathan Dixon** - Visual Homography Landing
- **Julian Brackins** - Visual Homography Landing
- **Dylan Geyer** - UAV Build, Visual Homography Landing

Role assignments:

- **UAV Build** - Responsible for the physical assembly of the hexrotor, including wiring and mounting of hardware on the UAV.
- **Visual Homography Landing** - Responsible for the development of an algorithm to calculate the distance and direction to the center of the landing pad.
- **AI Landing** - Responsible for the development of a landing algorithm utilizing an AI approach such as a neural net.
- **Simulation** - Responsible for the development of a test environment wherein a simulated hexrotor utilizes the developed landing algorithms to evaluate the fitness of the landing algorithm.

2.2 Project Management Approach

This project is managed through the Agile framework. The sprints are three weeks with a one week pause between for evaluation and plan adaptation before the next sprint. Sprints 1, 2, and 3 comprise Phase 1. Sprints 4, 5, and 6 comprise Phase 2. Requirements for the project were collected by interviewing the client to form a set of user stories. These user stories provide a backlog managed on Trello. The user stories for this project represent composition of several tasks. Tasks are factored from the user stories and assigned to team members as that task relates to the project. These task assignments and their progress are managed on Trello. Issues and troubleshooting is handled through email and team meetings. The team division of work is to limit the dependencies between team members, so that issues encountered by one team member does not impact the progress of other team members. Tasks are sometimes added, altered, or removed so as to reflect a better size-up of the backlog item.

2.3 Phase Overview

Phase 1

- **O-1**: As an owner, I want the UAV to autonomously take-off from the landing pad.
- **O-2**: As an owner, I want the UAV to autonomously navigate through a series of waypoints.

Phase 2

- **U-1**: As a user, I want to communicate the waypoints to the UAV.
- **O-3**: As an owner, I want the UAV to autonomously return to the location of the landing pad.
- **O-4**: As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft.
- **O-5**: As an owner, I want the UAV to autonomously land on the landing pad with the correct orientation.

2.4 Terminology and Acronyms

- FCU - Flight Control Unit
- ROS - Robot Operating System
- GCS - Ground Conrol Station
- UAV - Unmanned Aerial Vehicle

3

User Stories, Backlog and Requirements

3.1 Overview

This section details the requirements(defined by the user stories) that set the goals, and subsequent tasks, of the project. This document will provide a more detailed view of the user stories. Additionally, this document will describe the client and developers more thoroughly.

3.1.1 Scope

This section includes the client information, user stories, and requirements.

3.1.2 Purpose of the System

The purpose of this project is to provide a prototype of implementing autonomous navigation and landing on a UAV platform. This project is a proof of concept, concentrating on landing algorithm approaches that will provide minimal error in terms of distance and orientation when landing. Ultimately, this technology will be integrated into the larger project where the UAV is capable of launching from an Unmanned Ground Vehicle(UVG), navigate in mild weather conditions, return, and land to recharge.

3.2 Stakeholder Information

The Math and Computer Science Department of South Dakota School of Mines and Technology, in addi- tion to providing ABET certified education to students, conducts software-side robotics research including autonomy, navigation, and computer vision. Dr. Pyeatt is representing the school as the project sponsor, advisor, and project owner.

3.2.1 Customer or End User (Product Owner)

Dr. Pyeatt, representing the school, has articulated the project requirements. The team has used these requirements to create user stories as a means to factor the project into work products that can be more easily tracked. Dr. Pyeatt will provide feedback, as the project owner, to the team.

3.2.2 Management or Instructor (Scrum Master)

Dr. Pyeatt, as previously mentioned, is also the team's advisor. Dr. Pyeatt will provide the team with experience and expertise to assist the team with project development. Dr. Pyeatt assistance also includes providing feedback

on approaches taken by the team, as well as providing recommendations. Dr. Pyeatt regularly attends weekly meeting, and has provided very liberal access for consultation outside of meetings.

3.2.3 Investors

The South Dakota School of Mines and Technology are the sole investors in this project.

3.2.4 Developers –Testers

All team members will serve as developers, designers, and testers. Team members will initially be divided up to concentrate in areas of the project that are required. As the project progresses and development needs change, team members will be reassigned to other project areas. Team member assignments are:

- **Christopher Smith:** Simulation, AI Landing
- **Steven Huerta:** Simulation, AI Landing
- **Dylan Geyer:** UAV Build
- **Jonathan Dixon:** Visual Homography Landing

The team

3.3 Requirements and Design Constraints

The requirements for this project are:

- Ability to communicate waypoints to UAV.
- UAV can autonomously take-off.
- UAV can autonomously navigate through waypoints.
- UAV can autonomously navigate back to landing pad.
- UAV can autonomously land safely and with the correct orientation.

The only client defined constraint on this project is funding. We have limited funds, around $1,000, to complete this project. However, more funds may become available depending on need and funding sources. Otherwise, this project does not have hard constraints set by the client, but rather are informed as a consequence of our design decisions.

3.3.1 System Requirements

The Pixhawk was selected as the flight controller for this project because of its built-in autonomy and open-source communication protocol, Mavlink, that will allow the team to use for message passing.

The ODroid was selected as the single board processor because of its processing power. The team will be running a ROS environment on Ubuntu 14.04. This eliminated a great number of boards from being candidates. The ODroid XU4 provides the power needed to run the environment without greatly impacting our project budget.

3.3.2 Network Requirements

There are no network requirements for this project in regards to implementation.

3.3.3 Development Environment Requirements

Development environment is Ubuntu 14.04. This is required because of our use of ROS. ROS distros Indigo/Jade require Ubuntu 14.04. The system will not be further developed to be cross-platform. As the purpose of this project is to provide a proof of concept, and there is not reliable Windows or Mac support for ROS, it would not be a responsible or worthwhile effort in producing cross-platform compatibility.

3.3.4 Project Management Methodology

The team has elected to use Trello to keep track of backlogs and tasks. All team members have access to the trello web application. The Trello board is populated with a backlog representing the user stories. Each of these user stories will factor into a series of tasks that will be assigned at the beginning of each sprint.

This project will encompass a total of six sprints. Each sprint will span three weeks. Each semester will correspond with a phase, made up of three sprints. There will typically be a period of a week between each sprint, where a sprint report and prototypes will be made available for review.

The code and documentation for the project is maintained on github within a project repository. All team members, sponsor/advisor, and course instructor will have access to the repository. The agreed upon use of the repository is for team members to branch the repo to make changes as they are working on their functional area. When that team member is ready to integrate that branch back into the master, the team will conduct a code review. Each team member must be able to provide feedback before a merge can occur. The exception to this is documentation. Documentation may be merged as needed, so long as the team member has taken appropriate steps to ensure that the documentation is still in a function state after merging.

3.4 User Stories

3.4.1 User Story #1

User-1: As a user, I want to communicate the waypoints to the UAV.

3.4.1.a User Story #1 Breakdown

The user will require some means of supplying waypoints and other mission parameters to the UAV. The team needs to explore possible approaches, as well as look at last year's attempt.

3.4.2 User Story #2

Owner-1: As an owner, I want the UAV to autonomously take-off from the landing pad.

3.4.2.a User Story #2 Breakdown

The UAV will need to operate autonomously for the duration of the demonstration, to include take-off. The team will need to explore possible approaches, and refer to last year's attempt if possible. After some research, the team will need to implement this functionality. The team understands that this functionality will be gained through the use of the autonomy supplied with the flight controller, however the team needs to interface with the flight controller to enable that autonomy.

3.4.3 User Story #3

Owner-2: As an owner, I want the UAV to autonomously navigate through a series of waypoints.

3.4.3.a User Story #3 Breakdown

The UAV will need to be able to navigate through a series of waypoints defined ahead of time by the user. The team will need to explore possible approaches and refer to last year's attempt. The team will then need to implement this functionality. The team understands that this functionality is supplied with the flight controller. However, the team will still need to find a way to enable this functionality.

3.4.4 User Story #4

Owner-3: As an owner, I want the UAV to autonomously return to the location of the landing pad.

3.4.4.a User Story #4 Breakdown

The UAV will need to return the landing pad. Naively, this will occur after visiting the waypoints, however, there are other conditions that may necessitate the need to return to the landing pad such as finding the object of interest or running low on power. The team plans to address the first case (return after visiting all waypoints). Extension to this item may be addressed if the team completes the project with sufficient time remaining.

3.4.5 User Story #5

Owner-4: As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft

3.4.5.a User Story #5 Breakdown

The UAV will need to land on the landing pad with $\pm$.1m distance error. The team will explore visual homography and AI approaches. Each approach will have the goal of moving the UAV to the landing pad.

3.4.6 User Story #6

Owner-5: As an owner, I want the UAV to autonomously land on the landing pad with the correct orienta- tion

3.4.6.a User Story #6 Breakdown

Building from the previous user story, the UAV will need to land with the correct orientation. Each of the previously mentioned algorithms will require that the UAV lands with $\pm 15^{\circ}$ of correct orientation. However, the final project may see the fusion of different approaches to achieve the desired result.

4

Design and Implementation

This section is used to describe the design details for each of the major components in the system. The autonomous systems of teh UAV can be broken into individual components and the following sections will be divided by such. The first section will cover our simulation environment, the second will cover the autonomous takeoff and waypoint navigation, the third section will discuss our two autonomous landing approaches, and finally the fourth section will cover the UAV.

4.1 Simulation Environment

4.1.1 Technologies Used

The simulation environment will be implemented using Gazebo as the actual environment itself and will use ROS so we can use the message passing services to communicate with different modules of code that will be written. This environment will handle all testing so nothing is tested on the physical UAV until it is verified working in a simulated environment that utilizes a pixhawk.

4.1.2 Component Overview

- Ubuntu 14.04 LTS
- Gazebo
- Mavlink
- python
- c++
- R.O.S.
- Mavros (Ros wrapper around Mavlink)

4.1.3 Phase Overview

The simulation environment is still a work in progress on setting up communication with a simulated pixhawk with software in the loop (SITL), however as soon as the pixhawk is received hardware in the loop (HITL) will be attempted because of issues simulating the device. Issues as of now are loading a waypoint file and sending commands. A file checksum appears to be invalid for the waypoint file and a invalid flight controller unit is returned after sending commands to the simulated pixhawk with the mavros cmd node.

4.1.4 Design Details

Currently the Design for the simulation is relying on outside sources for installing the environment and setting up packages. These details are discussed in the Prototype section within sprint report #3.

4.2 Autonomous Takeoff and Waypoint Navigation

4.2.1 Technologies Used

The autonomous takeoff and waypoint navigation system is separate from the simulation environment and landing approaches because these will be contained in a mission that will be uploaded into the pixhawk. There will be a waypoint publisher (wpt_pub) node that will receive input from a user and create a mission file out of that input. Another module within the node will start publishing waypoints to the node so that it nodes what path to take what to do using mavros messages.

4.2.2 Component Overview

wpt_pub dependencies:

- Ubuntu 14.04 LTS
- python
- c++
- R.O.S.
- Mavros
- Mavros msgs

4.2.3 Phase Overview

The wpt_pub node can currently publish mavros messages that contain take off, land, and arm commands successfully, however the simulated pixhawk reports back that it is an invalid flight controller unit most times. It does receive waypoints successfully through mavros messages.

4.2.4 Design Details

```
/*****************************************************************************/
    /**
* \file wpt_pub.hpp
*
* \brief ROS Implementation of a waypoint publisher (header)
* \author Christopher J Smith
* \date February 08, 2013
*
* Waypoint publisher for the landing pad project
*
*****************************************************************************
    */

#ifndef _wpt_pub_hpp_
#define _wpt_pub_hpp_

#include <ros/ros.h>
#include <ros/rate.h>
#include <tf/tf.h>
#include <string>

#include <boost/thread.hpp>
```

```
#include <geometry_msgs/Pose.h>

//waypoint.cpp header files in mavros file
#include <mavros_msgs/WaypointList.h>
#include <mavros_msgs/WaypointSetCurrent.h>
#include <mavros_msgs/WaypointClear.h>
#include <mavros_msgs/WaypointPull.h>
#include <mavros_msgs/WaypointPush.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/CommandTOL.h>
#include <mavros_msgs/SetMode.h>

namespace wpt_pub
{
   class WPTPUB
   {
   public:
      WPTPUB( const ros::NodeHandle &_nh,
            const ros::NodeHandle &_nh_priv);

      ~WPTPUB();
   private:
      void spin();
      void spinOnce();
      void deg_to_min();
      void read_file();
      void startpos(const geometry_msgs::PoseStamped::ConstPtr& msg);

      ros::NodeHandle nh;
      ros::NodeHandle nh_priv;
      std::string frame_id;
      std::string wpt_file;
      boost::mutex cmd_lock;


      geometry_msgs::Pose pose;

      ros::Subscriber sub;

      ros::Rate spin_rate;
      boost::thread spin_thread;

      mavros_msgs::CommandTOL cmd;
      std::vector<mavros_msgs::CommandTOL> cmds;

      mavros_msgs::Waypoint waypt;
      std::vector<mavros_msgs::Waypoint> wpts;

      bool first_spin;

      ros::Publisher wpt_pub;
      ros::Subscriber pos_sub;

   };
```

```
}
#endif
```

```
#include "wpt_pub/wpt_pub.hpp"
#include <string>
#include <ros/time.h>
#include <tf/tf.h>
#include <fstream>

namespace wpt_pub
{
   WPTPUB::WPTPUB(  const ros::NodeHandle &_nh,
            const ros::NodeHandle &_nh_priv ):
      //Parameters set at initialization of class
      nh(_nh),
      nh_priv(_nh_priv),
      frame_id("landingpad/uav/wpt_pub"),
      wpt_file("wpt_file"),
      spin_rate( 100 ),
      spin_thread( &WPTPUB::spin, this ),
      first_spin(true)
   {
      cmd_lock.unlock();
      nh_priv.param("frame_id", frame_id, (std::string)"landingpad/uav/wpt_pub
          ");
      sub = nh.subscribe("/iris/ground/truth/pose", 1000, &WPTPUB::startpos,
          this);
      read_file();
   }

   WPTPUB::~WPTPUB()
   {
      spin_thread.interrupt();
   }

   void WPTPUB::startpos( const geometry_msgs::PoseStamped::ConstPtr& msg)
   {
      static int count = 0;
      if(count++%50 != 0)
         return;

      pose = msg->pose;
      ROS_INFO("Received pose information from simulation");
   }

   void WPTPUB::read_file()
   {
      int seq, is_curr, frame, command, auto_cont;
      double params[4], lat, lon, alt;

      std::ifstream fin;
      //fin.open(wpt_file.c_str());

      while( fin >> seq >> is_curr >> frame >> command >>
            params[0] >> params[1] >> params[2] >> params[3] >>
```

```
            lat >> lon >> alt >> auto_cont)
    {
        command = 1;
        if( command == 16)
        {
            waypt.is_current = bool(is_curr);
            waypt.frame = frame;
            waypt.command = command;
            waypt.param1 = params[0];
            waypt.param2 = params[1];
            waypt.param3 = params[2];
            waypt.param4 = params[3];
            waypt.x_lat = lat;
            waypt.y_long = lon;
            waypt.z_alt = alt;
            waypt.autocontinue = bool(auto_cont);
            wpts.push_back(waypt);
        }
        else //Assuming command column is either Takeoff or Land command
        {
            cmd.request.min_pitch = 0;
            cmd.request.yaw = 0;
            cmd.request.latitude = pose.position.x;
            cmd.request.longitude = pose.position.y;
            cmd.request.altitude = 1.5;
            cmds.push_back(cmd);
            ROS_ERROR("COMMAND PUSHED INTO VECTOR");
        }

    }
    cmd.request.min_pitch = 0;
            cmd.request.yaw = 0;
            cmd.request.latitude = pose.position.x;
            cmd.request.longitude = pose.position.y;
            cmd.request.altitude = 1.5;
            cmds.push_back(cmd);
            ROS_ERROR("COMMAND PUSHED INTO VECTOR");
}
void WPTPUB::spin()
{
    while( ros::ok())
    {
        boost::this_thread::interruption_point();
        spinOnce();
        spin_rate.sleep();
    }
}
void WPTPUB::spinOnce()
{
    cmd_lock.lock();
    if(first_spin)
    {
        first_spin = false;

        //Set Guided Mode
```

```
            ros::ServiceClient cl = nh.serviceClient<mavros_msgs::SetMode>("/
                mavros/set_mode");
            mavros_msgs::SetMode srv_setMode;
            srv_setMode.request.base_mode = 0;
            srv_setMode.request.custom_mode = "GUIDED";
            if(cl.call(srv_setMode))
               ROS_ERROR("setmode send ok %d value:", srv_setMode.response.
                   success);
            else
            {
               ROS_ERROR("Failed SetMode");
               first_spin = true;
            }

            //Arm Device in simulation
            ros::ServiceClient arming_cl = nh.serviceClient<mavros_msgs::
                CommandBool>("/mavros/cmd/arming");
            mavros_msgs::CommandBool srv;
            srv.request.value = true;
            if(arming_cl.call(srv))
               ROS_ERROR("ARM send ok %d", srv.response.success);
            else
            {
                  ROS_ERROR("Failed arming or disarming");
               first_spin = true;
            }

            //Request Takeoff
            ros::ServiceClient takeoff_cl = nh.serviceClient<mavros_msgs::
                CommandTOL>("/mavros/cmd/takeoff");
            if(takeoff_cl.call(cmds[0]))
               ROS_ERROR("srv_takeoff send ok %d", cmds[0].response.success);
            else
            {
               ROS_ERROR("Failed Takeoff");
               first_spin = true;

            }
         }
         cmd_lock.unlock( );
      }
}
```

```
#include <wpt_pub/wpt_pub.hpp>

#include <ros/ros.h>
#include <cstdlib>

/*!
* \brief Main Function
*
* \author Chris Smithn
*
* Initializes ROS, instantiates the node handle for the waypoint publisher to
    use and
```

```
* instantiates the wpt_pub class.
*
* \param argc Number of command line arguments
* \param argv 2D character array of command line arguments
*
* \returns EXIT_SUCCESS, or an error state
*/
int main( int argc, char *argv[] )
{
   ros::init( argc, argv, "wpt_pub_node" );

   ros::NodeHandle nh;
   ros::NodeHandle nh_priv( "~" );


   wpt_pub::WPTPUB wptpub( nh, nh_priv);

   ros::spin( );

   std::exit( EXIT_SUCCESS );
}
```

4.3 Landing Approaches

4.3.1 Visual Homography

One of the approaches we will be using for the task of landing the UAV will be using visual homography. The basic goal here is to take an image that will contain the landing pad, and, based on the how the target looks on the pad, determine range to pad, orientation with respect to the pad, and the angle above the horizon with respect to the pad. This information will then be used to determine how to actually fly the UAV so that it will be able to safely land on the pad.

4.3.1.a Technologies Used

In order to accomplish this, we will use a number of tools. First and foremost, we will be using OpenCV as our image recognition library. OpenCV is a powerful open-source toolkit that will be able to handle any image processing that we need to do. The algorithm will be developed in Python, as this provides an easy-to-understand syntax, and will be able to deliver the performance that we need.

4.3.1.b Component Overview

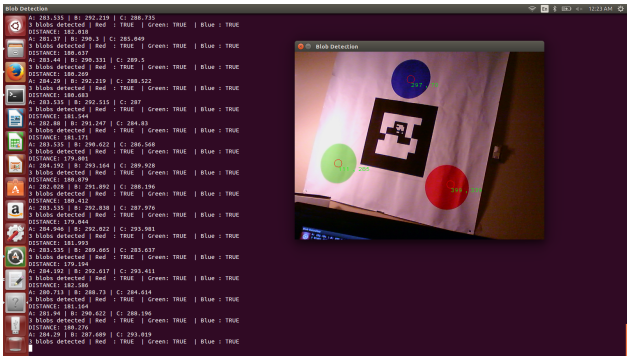
- Ubuntu 14.04 LTS
- Python
- OpenCV

4.3.1.c Phase Overview

The algorithm is still a work in progress, but steps have been made. We have been able to compile and run last year's code, and have a working blob detection system up and running. This system is able to detect a pad with three colored circles by looking for the colors of the circles. The circles are red, green, and blue. It is then able to calculate the distance to the target, since it knows what size the target should be. This should be very helpful moving forward, as we are able to find the pad. In the coming sprints, we will attempt to move toward a pad that has four circles, as we believe this will give us more information to use for our homography computations.

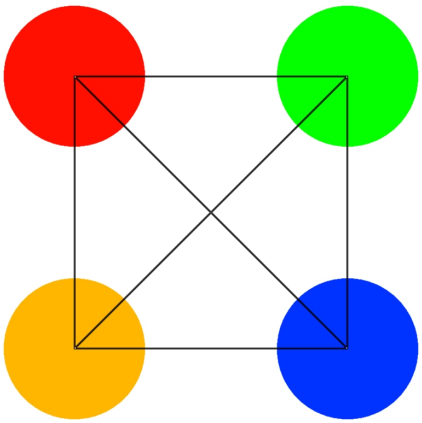
4.3.1.d Design Details

In the following image, you can see results from last year's code running and detecting the colored circles on a large target with three circles.

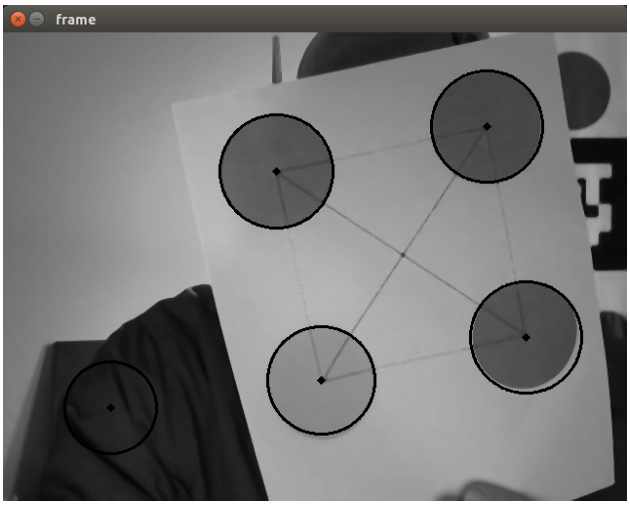


It may be hard to notice, but the text that is currently being displayed in the terminal window shows that there are three blobs detected, exactly one for each color. It is also displaying the range to target, which is correct as far as we can tell.

The next image is what we are going to attempt to use for the next target. It has four colored circles, so we should be able to compute homography more accurately.



The final image is just another possible method that we have briefly explored where instead of blob detection we use hough transforms to detect only circles in the images. We have seen papers where others have had successes using targets consisting of concentric circles, so we think that we may be able to have some amount of success with this, in the event that the above methods don't end up panning out.



4.3.2 Reinforcement Learning

4.3.2.a Technologies Used

To accomplish the reinforcement learning approach we will be using gazebo and ROS so that we can visually see that the agent is learning the states and actions required to land the UAV.

4.3.2.b Component Overview

- Ubuntu 14.04 LTS
- Python
- C++
- ROS pose messages
- sarsa_alg:

  The class that instantiates the agent and performs the appropriate calculations to pick an action to move on to the next state and recieves a reward. Each action will send the appropriate messages to gazebo so that the UAV moves accordingly and allows the user to visually see what the agen is doing.

- sarsa_node:

  The ROS nod that creates the sarsa_alg class and handles initial topics to publish to and receive from.

4.3.2.c Phase Overview

The Linear, gradient-descent Sarsa($lambda$) with binary features and ϵ-greedy policy will be the algorithm of reinforcement learning that will be implemented. Currently ideas for using a 2-D state space are in the works with a vector field graient to be the actions to funnel the UAV into the center of a cone at a given height and radius. Because of the setbacks of the simulation no coding as been accomplished, but will be as soon as an environment is set up.

4.3.2.d Design Details

Algorithm 1 Linear, gradient-descent Sarsa(λ) with binary features and ϵ-greedy policy

Initialize $\vec{\theta}$ arbitrily and $\vec{e} = \vec{0}$
for Each Episode **do**
  $s \leftarrow$ initial state of episode
  for all $a \in A(s)$ **do**
    $F_a \leftarrow$ set of features present in s, a
    $Q_a \leftarrow \sum_{i \in F_a} \theta(i)$
  end for
  $a \leftarrow$ arg $max_a Q_a$
  With probability ϵ : $a \leftarrow$ a random action $\in A(s)$
  for Each Step of Episode **do**
    $\vec{e} \leftarrow \gamma\lambda\vec{e}$
    for All $\bar{a} \neq a$ **do**
      for All $i \in F_{\bar{a}}$ **do**
        $e(i) \leftarrow 0$
      end for
    end for
    for All $i \in F_a$ **do**
      $e(i) \leftarrow 1$
    end for
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r - Q_a$
    for All $a \in A(s')$ **do**
      $F_a \leftarrow$ set of features present in s', a
      $Q_a \leftarrow \sum_{i \in F_a} \theta(i)$
    end for
    $a' \leftarrow$ arg $max_a Q_a$
    With probability ϵ : $a' \leftarrow$ a random action $\in A(s)$
    $\delta \leftarrow \delta + \gamma Q_{a'}$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    $a \leftarrow a'$
  end for
  until s' is terminal
end for

4.4 UAV Build

This section details the physical building of the UAV that will be used in autonomous landing.

4.4.1 Technologies Used

The technologies that we currently have are listed below and include the controllers required for autonomous flight.

- Turnigy Talong V1.0 Hexcopter
- 6x DC Motors
- 6x Electronic Speed Controllers
- 3DR Pixhawk
- Odroid-XU4

4.4.2 Phase Overview

We are currently in the first phase of UAV construction. In this phase we have completed the construction of the **Turnigy Talong V1.0 Hexcopter** frame, and mounting all of the DC motors and electronic speed controllers.

4.4.3 Design Details

Current documentation on the build progress made in Sprint 3 is shown below in our BuildProcess.pdf document.

Turnigy Talon Hexcopter v2.0 Build

Dylan Geyer

December 4, 2015

1 Intro

This document details how the Turnigy Talon Hexcopter v2.0 was assembled and the peripheral devices were mounted so that others may replicate this process. It first details assembling the frame, and then mounting each peripheral device (motors, ESC's, Odroid-XU4, etc..).

2 Turnigy Talon Hexcopter v2.0 Frame

This first section will detail how the carbon fiber frame is put together.

2.1 Bolts

The first thing to do is make sure you know which screws correspond to each label in the figures that will follow.

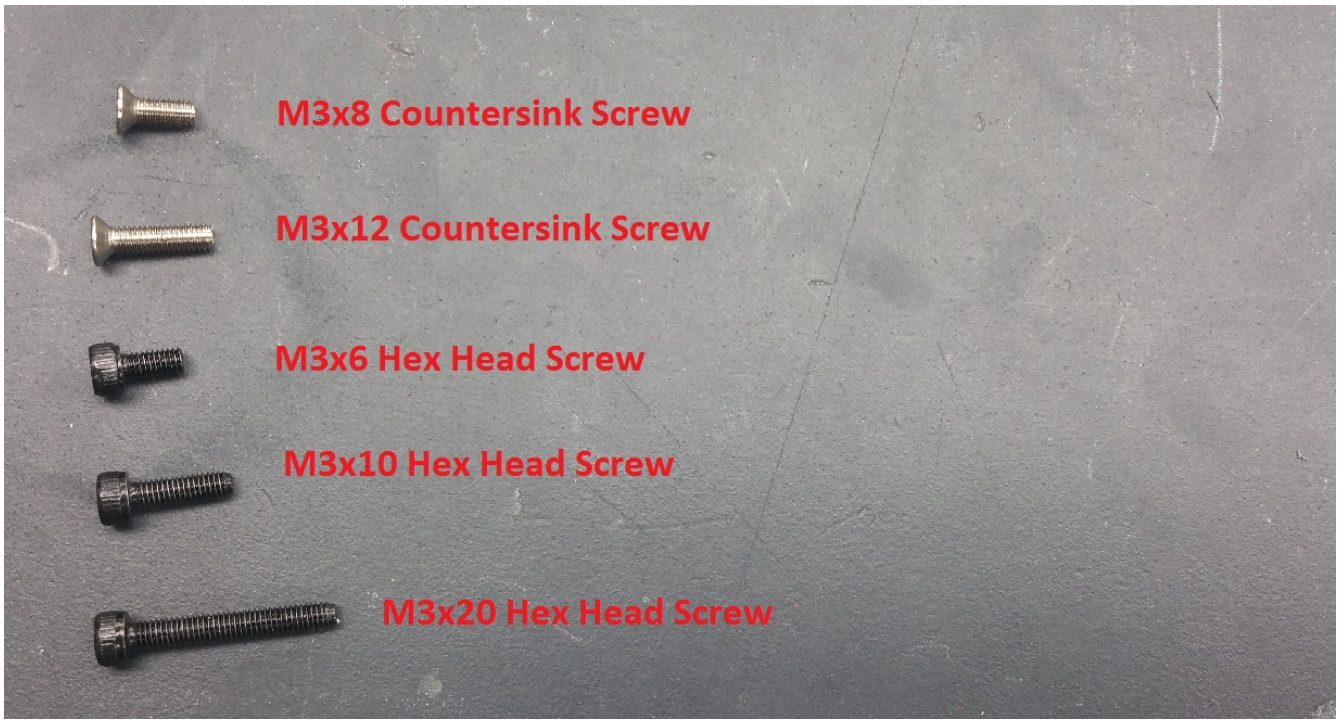


Figure 1: Bolt definitions.

Figure 2: Another view of the bolts.

2.2 Arm

Now that there is a quick reference guide for each of the types of screw, we can begin building the frame. The first part to be constructed is the motor mount and the tip of each arm.

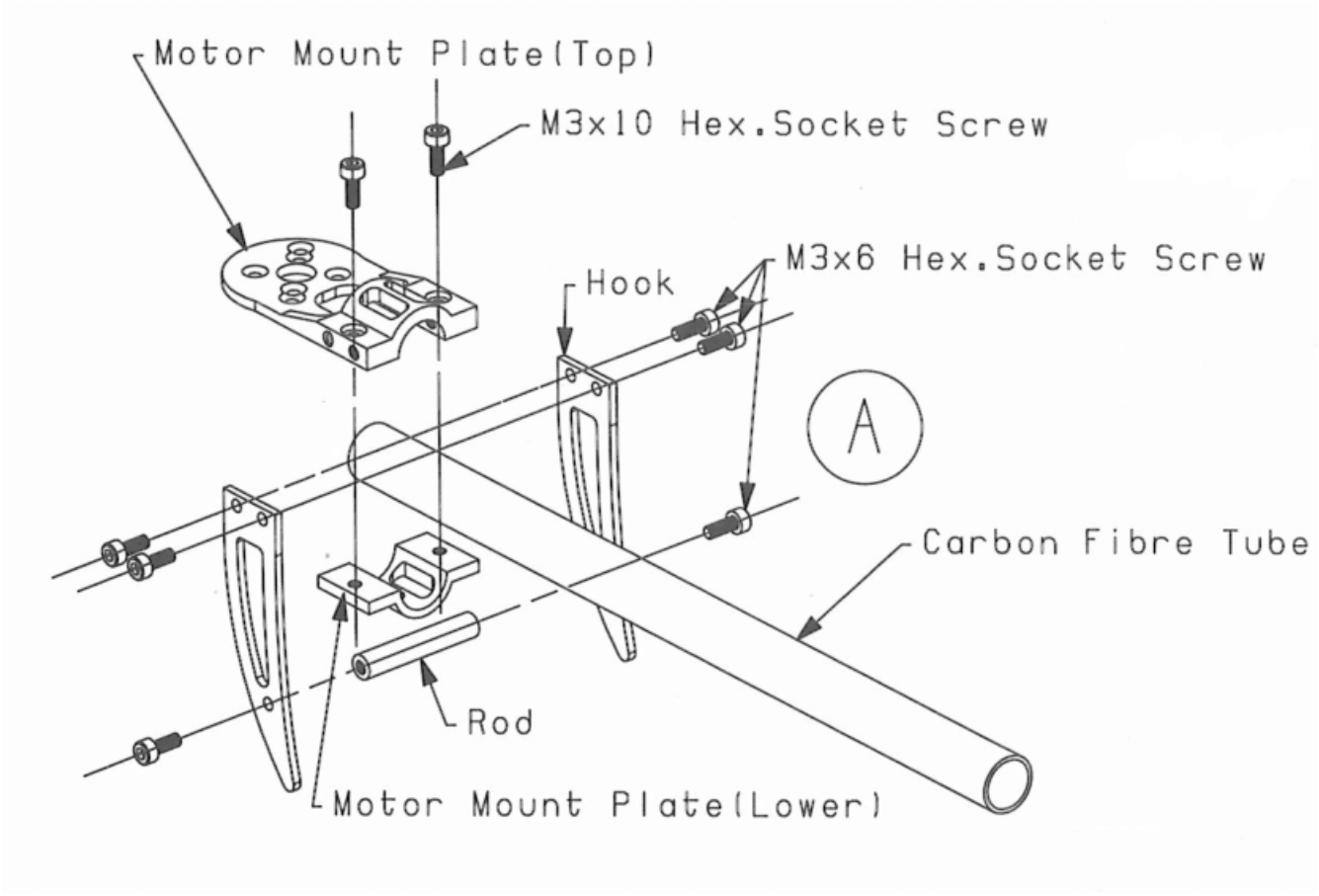


Figure 3: Instructions for assembling motor mount.

After carefully following the directions above we are left with a carbon fiber tube with an assembled motor mount which is shown in the image below.

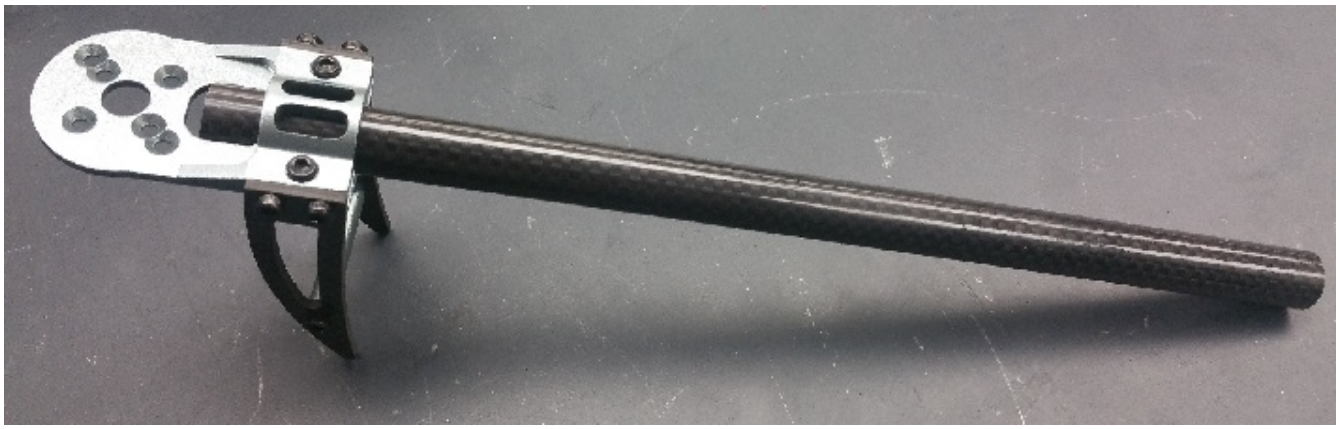
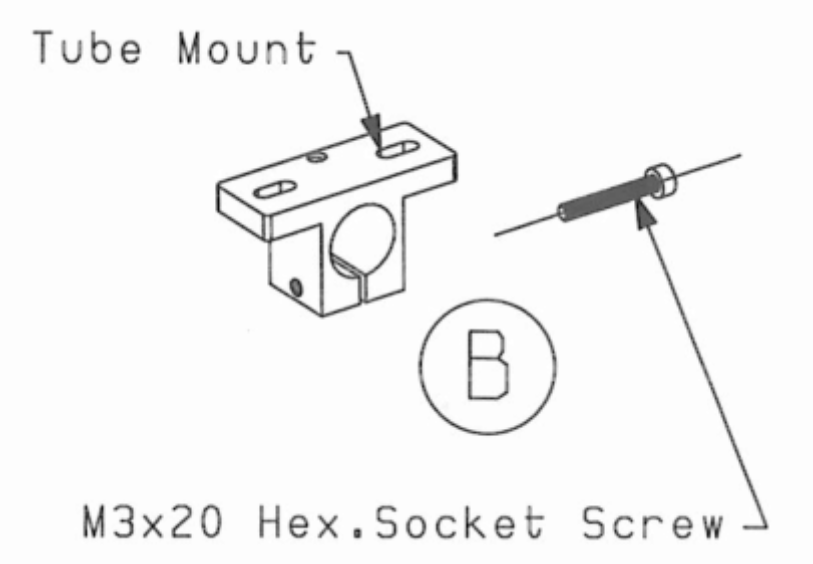


Figure 4: My completed motor mount.

2.2.1 Frame Mount

Now that the motor mount has been attached to one end of the carbon fiber arm, it is time to attach the frame mount to the other end so that the center plates will be able to hold each arm in place.



2.3 Center Support

Once all of the arms have been assembled they must be affixed to the top and bottom plates to keep everything stable. This step is a bit tricky as each arm must be loosely connected to the top and bottom plates before tightening the screws down or it will be impossible to insert the other arms.

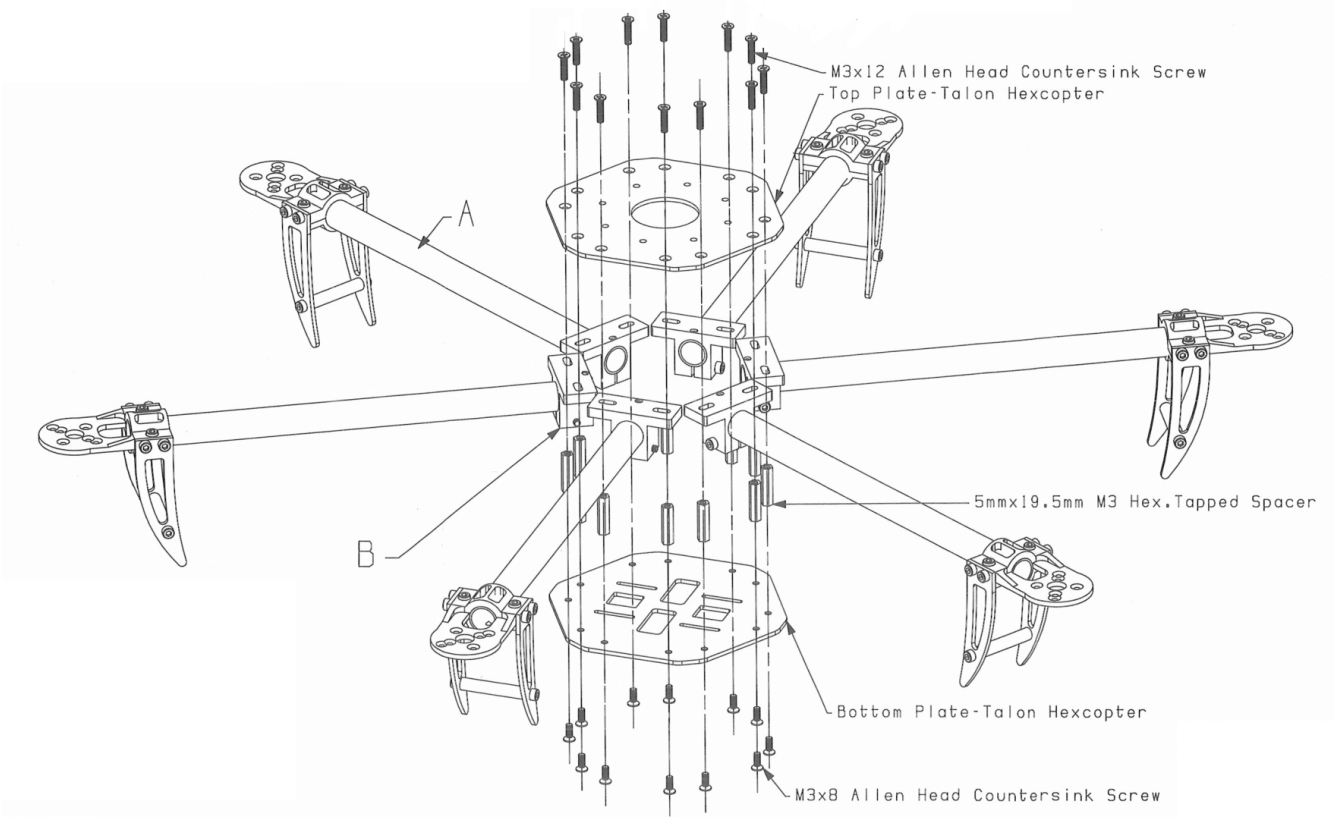


Figure 5: Instructions for Top/Bottom plates.

3 Motors

Once the whole frame has been constructed it is time to attach the DC motors. These DC motors will attach to the very tip of the each arm with their power cables routing through the hollow carbon fiber arms.

3.1 Mounting

One note for this particular build is that the leads that came attached to the motor were much too short to connect to the controllers in the center console. This was fixed by simply soldering some 16 gauge extension wires onto the motor leads and then connecting these to the ESC's. Once the DC motor leads have been extended route them through the carbon fiber tube arm and affix the motor to the arm using 4 - M3x6 Hex Head screws.

3.2 Electronic Speed Controllers

ESC's are attached to the DC motor leads **after** they come out of the arm holes and are in the center console. The ESC's are fixed to the frame of the hex-copter and the +/- leads from each ESC are attached to the power distribution board, and the signal wires from the ESC' are attached to the Pixhawk.

4 Controllers

Now it is time to attached the brains of the hex-copter to the frame and motors. To do this we will simply stack the Pixhawk, ODROID, and power distrubution boards in the center of the frame.

4.1 Power Distribution Board

The first level in the stack will hold the power distribution board which is simply just a way of connecting six different +/- motor leads to the single +/- terminal on the battery. This board will be the base of the controller tower in the center of the hex-copter.

4.2 Pixhawk

Once the power distribution board has been placed and the ESC's signal lines are connected to the Pixhawk, the Pixhawk can be mounted on top of the power distribution board.

4.3 ODROID-XU4

Finally mount the ODROID on top of the Pixhawk. The ODROID doesn't handle any controls at this point so its only connection right now is power and the camera data lines.

5 Camera

The final thing to mount on the hex-copter is the camera that will be used for viewing the landing pad. This is mounted in a position so that it is facing straight down so that it can see the landing pad from directly above it.

4.5 Software: ROS Vision

This section serves as an overview of the packages used to determine the landing pad, as well as estimate pose, by means of monocular vision. This description will include the packages, launch files, as well as other files needed to implement the AR tag tracking and pose estimation activities. This section is divided into sections covering the different portions of the ROS Vision build, as follows:

- Camera Operation
- Camera Calibration
- Image Processing
- AR Tag Tracking
- Pose Estimation

4.5.1 Camera

usb_cam: this ros package provides a method for bringing usb cameras into the ros environment. The node will publish an image topic.

- Parameters
  - image width:
  - image height:
  - pixel_format:
  - io_method:
  - camera_frame_id:
  - framerate:
  - brightness:
  - saturation:
  - sharpness:
  - autofocus:
  - camera_info_url:
  - camera_name:
- Published Topics
  - /image - containing the sensor_msgs/Image message
- installation: sudo apt-get-install ros-jade-usb-cam
- Sample Launch File

```
<launch>
   <node pkg="usb_cam" type="usb_cam_node" name="usb_cam">
      <param name="video_device" type="string" value="/dev/video0"/>
      <param name="image_width" type="int" value="640" />
      <param name="image_height" type="int" value="480" />
      <param name="pixel_format" type="string" value="yuyv"/>
      <param name="camera_frame_id" value="camera" />
   </node>
</launch>
```

pointgrey_camera_driver: this ros package provides a method for bringing point grey cameras into the ros environment.

- Parameters
  - video_mode:
  - frame_rate: the camera speed in frames per second
  - auto_exposure: Allow the camera to automatically change exposure (Combined Gain, Iris & Shutter control).
  - exposure:
  - auto_shutter:
  - auto_gain:
  - gain:
  - pan:
  - tilt:
  - brightness:
  - gamma:
  - auto_white_balance:
  - white_balance_blue:
  - white_balance_red:
- Published Topics
  - /image - containing the sensor_msgs/Image message
- installation: sudo apt-get install ros-jade-pointgrey-camera-driver
- Usage:

```
<launch>
   <!-- Determine this using rosrun pointgrey_camera_driver list_cameras.
        If not specified, defaults to first camera found. -->
  <arg name="camera_serial" default="0" />
  <arg name="calibrated" default="0" />

  <group ns="camera">
    <node pkg="nodelet" type="nodelet" name="camera_nodelet_manager"
     args="manager" />

    <node pkg="nodelet" type="nodelet" name="camera_nodelet"
          args="load pointgrey_camera_driver/PointGreyCameraNodelet
                camera_nodelet_manager" >
      <param name="frame_id" value="camera" />
      <param name="serial" value="$(arg camera_serial)" />

      <!-- When unspecified, the driver will use the default framerate
           as given by the camera itself. Use this parameter to override
           that value for cameras capable of other framerates. -->
      <!-- <param name="frame_rate" value="15" /> -->

      <!-- Use the camera_calibration package to create this file -->
      <param name="camera_info_url" if="$(arg calibrated)"
             value="file://$(env HOME)/.ros/camera_info/
```

```
                    $(arg camera_serial).yaml" />
    </node>

    <node pkg="nodelet" type="nodelet" name="image_proc_debayer"
          args="load image_proc/debayer camera_nodelet_manager">
    </node>
  </group>
</launch>
```

NOTE: This launch file is THE launch file provided by the repository, and will be available when you install this package. A few notes about what is occuring in this file before we go any further. The call for a camera calibration file will become clear in the next subsection. Additionally, this launch file is somewhat complicated by the use of nodelets which is a thing in ROS designed to reduce the complexity of the node networks in terms of communication. In the words of the ROS deities at Clear Path Robotics

> The primary advantage is the automagic zero-copy transport between nodelets (in one nodelet manager). This means that the pointcloud created by a hardware driver doesn't need to get copied or serialized before it hits your code, assuming you inject the nodelet into the camera's manager, saving you time and trouble.
>
> You get all the modularity of nodes, and all the efficiency of having one monolithic process. This makes nodelets more flexible than bare plugins (via pluginlib) - you can implicitly tap into any of the intra-process communication that occurs.

We leave it to the reader to learn more about nodelets should that be a path that appears to offer substantial benefit. The team made modifications to this file that will be discussed in a later subsection.

4.5.2 Camera Calibration

camera_calibration: This package is used to calibrate both monocular and stereo cameras. The package has two functions, camera calibration and camera check. Camera check is to check the calibration of the camera. This may be a good idea to use after calibration is performed as an extra measure to ensure a good calibration was made by the calibration step.
Before starting, it is necessary to get a calibration board. For this calibration, a chess board image is required. The interior corner count and size of chess board square in meters is needed.

- Parameters
  - video_mode:
  - frame_rate: the camera speed in frames per second
  - auto_exposure: Allow the camera to automatically change exposure (Combined Gain, Iris & Shutter control).
  - exposure:
  - auto_shutter:
  - auto_gain:
  - gain:
  - pan:
  - tilt:
  - brightness:
  - gamma:
  - auto_white_balance:
  - white_balance_blue:
  - white_balance_red:

- installation: ssudo apt-get install ros-jade-camera-calibration
- Usage:
  1. start camera in ROS environment
  2. begin camera calibration by using the following sample (found at the package documentation site):

```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square
    0.108 image:=/my_camera/image camera:=/my_camera
```

4.5.3 Image Processing

pointgrey_camera_driver: this ros package provides a method for bringing point grey cameras into the ros environment.

- Parameters
  - video_mode:
  - frame_rate: the camera speed in frames per second
  - auto_exposure: Allow the camera to automatically change exposure (Combined Gain, Iris & Shutter control).
  - exposure:
  - auto_shutter:
  - auto_gain:
  - gain:
  - pan:
  - tilt:
  - brightness:
  - gamma:
  - auto_white_balance:
  - white_balance_blue:
  - white_balance_red:
- installation: sudo apt-get install ros-jade-pointgrey-camera-driver

4.5.4 Tag Tracking

pointgrey_camera_driver: this ros package provides a method for bringing point grey cameras into the ros environment.

- Parameters
  - video_mode:
  - frame_rate: the camera speed in frames per second
  - auto_exposure: Allow the camera to automatically change exposure (Combined Gain, Iris & Shutter control).
  - exposure:
  - auto_shutter:
  - auto_gain:
  - gain:
  - pan:

  - tilt:
  - brightness:
  - gamma:
  - auto_white_balance:
  - white_balance_blue:
  - white_balance_red:

- installation: sudo apt-get install ros-jade-pointgrey-camera-driver

4.5.5 Localization

pointgrey_camera_driver: this ros package provides a method for bringing point grey cameras into the ros environment.

- Parameters
  - video_mode:
  - frame_rate: the camera speed in frames per second
  - auto_exposure: Allow the camera to automatically change exposure (Combined Gain, Iris & Shutter control).
  - exposure:
  - auto_shutter:
  - auto_gain:
  - gain:
  - pan:
  - tilt:
  - brightness:
  - gamma:
  - auto_white_balance:
  - white_balance_blue:
  - white_balance_red:

- installation: sudo apt-get install ros-jade-pointgrey-camera-driver

5

System and Unit Testing

This section describes the approach taken with regard to system and unit testing.

5.1 Overview

The testing approach for this project was a bit different from traditional software testing. We did not have an automated testing framework to run unit tests or regression tests, instead we verified visually when a piece of the project was behaving as expected. This project used many off the shelf hardware and software products where testing the functionality of these pieces has already been done by the manufacturer of that technology. The tests described below allowed us to evaluate whether or not the different subsystems of the UAV were functioning correctly.

5.2 Dependencies

There were no testing frameworks used for this project. All testing was verified manually.

5.3 Test Setup and Execution

5.3.1 Testing: U-1

As a user, I want to communicate the waypoints to the UAV.

Task No.	Task	Test	Completed
3	Modify mission communication implementation as necessary	User is able to create mission file with ground control station.	Yes
3	Modify mission communication implementation as necessary	User is able to load mission file on offboard(ODroid).	Yes
3	Modify mission communication implementation as necessary	Landing Algorithm on offboard starts after completing last mission.	Yes

Creating mission files from the ground control station was completed using the QGroundControl software. Once GPS way-points were set up a mission file could be generated. A figure showing the use of QGroundControl is below.

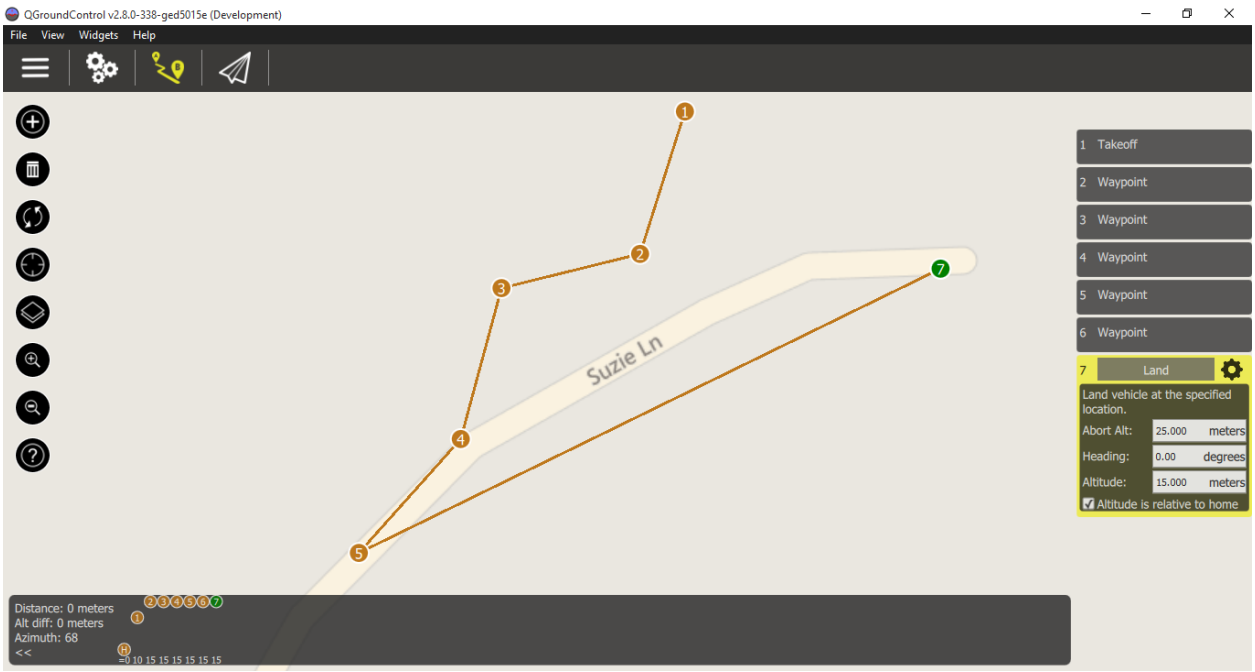


Figure 5.1: QGroundControl mission creation.

5.3.2 Testing: O-1

As an owner, I want the UAV to autonomously take-off from the landing pad.

Task No.	Task	Testing	Completed
3	Modify/Rewrite take-off implementation as necessary	FCU recieves take-off mission from mission from off-board control.	Yes
3	Modify/Rewrite take-off implementation as necessary	FCU executes take-off mission.	Yes

Similar to the previous user story, autonomous takeoff was handled by the flight controller in conjunction with the mission planner. In the previous figure you can see a mission in the QGroundControl software. What is not as obvious as the way-points on the map is the list of events to be executed on the right side of the screen. As you can see in the figure below, autonomous take-off is part of the mission file generated and uploaded to the ODroid.

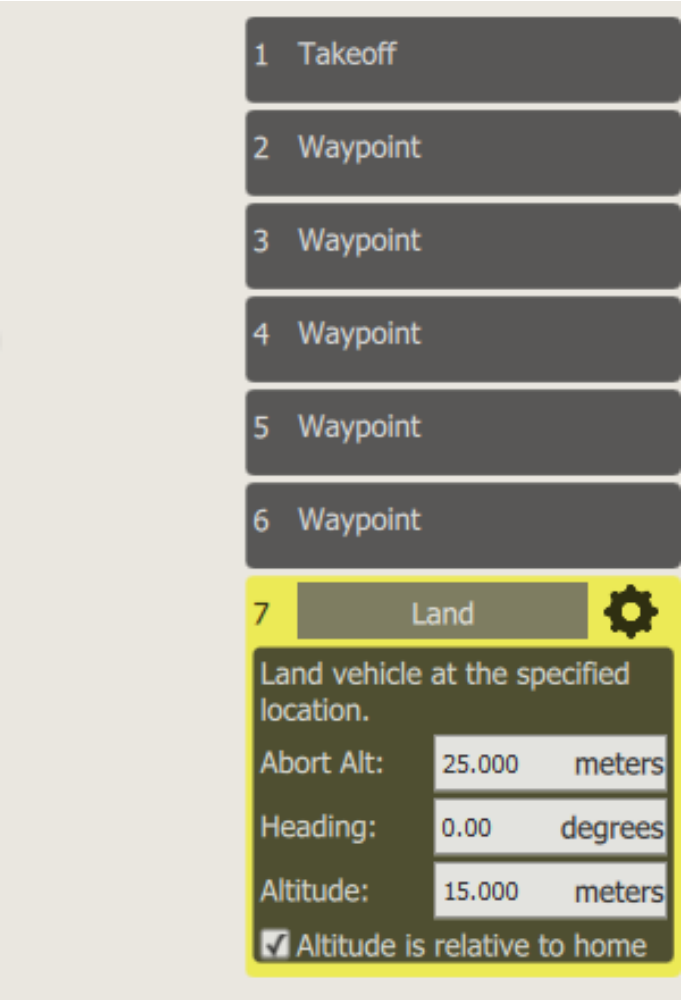


Figure 5.2: QGroundControl showing autonomous takeoff event.

5.3.3 Testing: O-2

As an owner, I want the UAV to autonomously navigate through a set of waypoints.

Task No.	Task	Test	Completed
3	Modify/Rewrite waypoint navigation implementation as necessary	FCU receives navigation missions from offboard control.	Yes
3	Modify/Rewrite waypoint navigation implementation as necessary	FCU executes navigation in sequence.	Yes
3	Modify/Rewrite waypoint navigation implementation as necessary	FCU follows, reasonably, the planned navigation.	Yes

As can been seen in the previous two figures QGroundControl was able to handle the autonomous takeoff and way-point navigation of the UAV. What remained to be tested was our UAV's ability to execute these missions. This was tested by uploading missions to off-board control (ODROID) and letting them be run. There are videos showing the UAV perform the autonomous takeoff and way-point navigation which verify that the UAV was able to perform those operations. Finally we needed to ensure the UAV followed the GPS way-points 'reasonably' well so we measured the error in the UAV's target landing location. As you can see from the picture below the UAV was able to land within 18 inches of its target using only GPS.

Figure 5.3: UAV landing error.

5.3.4 Testing: O-3

As an owner, I want the UAV to autonomously return to the location of the landing pad.

Task No.	Task	Test	Completed
3	Modify navigate to landing waypoint implementation as necessary	FCU receives last navigation mission from offboard control.	Yes
3	Modify navigate to landing waypoint implementation as necessary	FCU executes navigation mission.	Yes
3	Modify navigate to landing waypoint implementation as necessary	FCU navigates within a reasonable distance($<$10m) to waypoint.	Yes

In order to return to the location of the landing pad, the mission must specify the final GPS way-point as the starting way-point. This was done and verified by actual flight that can be seen in this YouTube video.

5.3.5 Testing: O-4

As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft

Task No.	Task	Test	Completed
4	Modify landing pose implementation as necessary	UAV accurately estimates pose of AR tag	Yes
4	Modify landing pose implementation as necessary	UAV centers over AR tag within (<0.1m).	No
4	Modify landing pose implementation as necessary	The UAV remains centered during descent within (<0.1m).	No
4	Modify landing pose implementation as necessary	The UAV lands without damaging craft or legs.	No

Although we were not able to test the physical UAVs ability to react to the AR Tag we were able to get AR Tag tracking up and running. The AR Tag tracking was able to accurately estimate the pose of an AR Tag within 15 ft. A demonstration of this tracking is shown below.

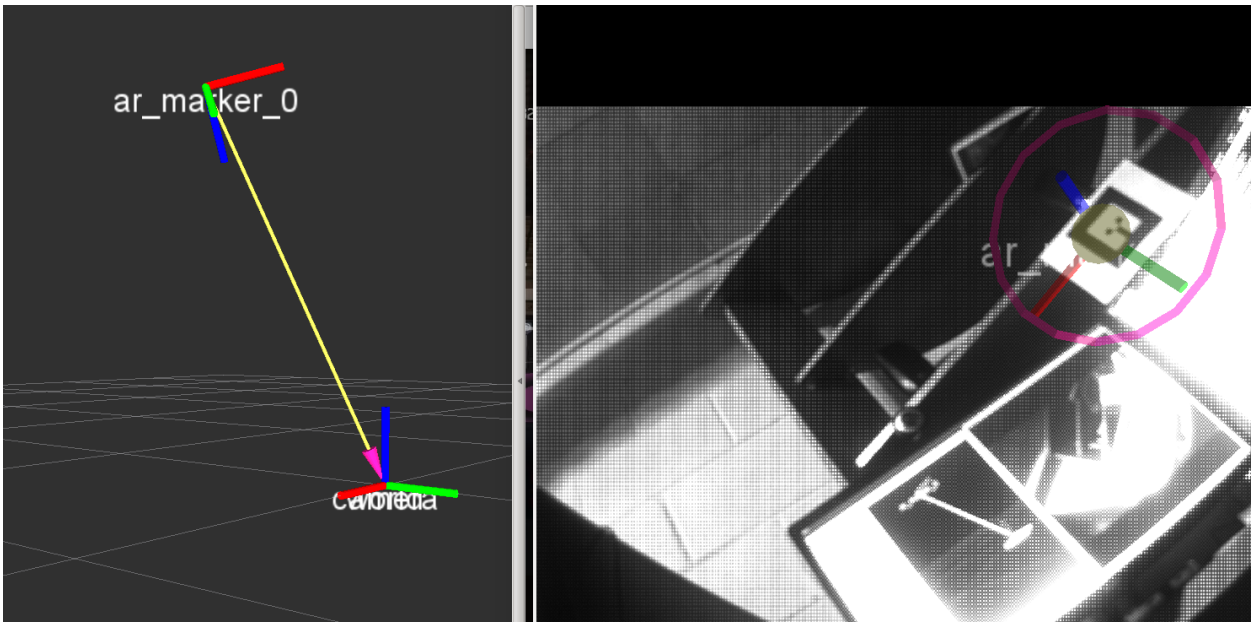


Figure 5.4: AR Tag tracking with AR Track ALVAR

5.3.6 Testing: O-5

As an owner, I want the UAV to autonomously land on the landing pad with the correct orientation.

Task No.	Task	Test	Completed
4	Modify landing orientation implementation as necessary	UAV accurately estimates the orientation of AR tag	Yes
4	Modify landing orientation implementation as necessary	UAV changes orientation to align with AR tag, within $15\mathrm{deg}$	No
4	Modify landing orientation implementation as necessary	UAV maintains correct orientation during descent within $15\mathrm{deg}$	No

On this final user story we were only able to complete the first task which is a common task between this user story and the previous one (O-4). Which to reiterate is the correct pose estimation of an AR Tag. If you would like to see the results of the AR Tracking look at the previous page.

6

Prototypes

Prototype Sprint #1

Project Overview

The capability of UAVs to rapidly search a large area, especially an area that is difficult to traverse by foot or vehicle, would be invaluable to operations such as search & rescue. However, small UAVs have a very limited flight time.

A system incorporating a UVG equipped with a landing pad that also serves as a charging station would allow the UAV to be delivered to areas of limited access. The UAV could then, being provided with way-points by the user, autonomously take-off, and navigate through the waypoints. After moving through the waypoints, or when the UAV requires recharging, the UAV will return to the UVG and safely land in such a way that the charging unit can connect to the UAV.

Team Expeditus aims to specifically create and deliver software that allows a UAV to autonomously take-off, navigate to way-points designated by the user, and return to the landing pad in manner that allows for charging and redeployment.

Goals

- Autonomous Take-off
  - Reaching a specific height before navigation
  - Provided no obstacles in reaching operating height
- Autonomous Navigation of Waypoints
  - Provided an obstacle free operation space
- Autonomous Landing of UAV that is:
  - Accurate within +/- .1 meters
  - Oriented correctly

Approach

Phase I
This first phase requires the testing of hardware and ensuring the quadrotor is properly configured and is capable of stable and controllable flight. This will mean many iterations of manual flight. Take-off and way-point navigation is solved by the use of MavLink. Testing of these capabilities will first be tested through the use of Mission Planner or APM Planner, which provides a handy interface for setting up take-off and

way point navigation. The last goal of this phase will be a simulation environment to simulate landing algorithms.

Objectives

1. Test of Manual Flight
2. Test of Mavlink Autonomous Control
3. Setting up Simulation Evironment for Quadrotor

Phase II

Computer vision has been determined to hold the most promise to solve the landing goal. Autonomous landing provided by the GPS (detailed in the hardware section below) provides an accuracy of nav-point navigation, given optimal conditions, to within $+/-10$ meters. This is obviously unsatisfactory to use for purposes of landing on a small platform, but it should bring us within a distance of our goal to detect the landing platform from our downward facing camera.

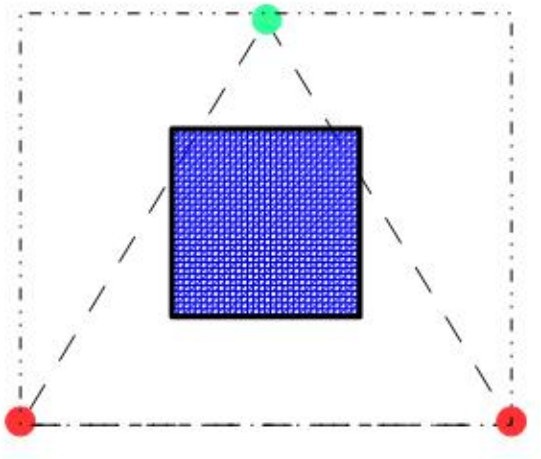


Figure 1: Landing Pad Light & QR Code Layout

Lights: Colored lights arranged in either a rectangle or triangle with different colors on selected lights(figure 1) to provide orientation information for the UAV. The warping and scale of the shape would provide the orientation by the operation needed to unwarp the shape, and the scale would provide distance information. The UAV will use the lights to provide information to the flight controller to manuever above the landing pad at a certain height, and with a specific orientation before switching state to use the QR code.

QR Code: Similar to the lights, warping and scale will provide the UAV with information regarding the orientation and distance between the UAV and the QR code. The QR code will be centered on the landing pad(figure 1).The UAV will be generally centered above the landing pad and at a much closer height, and so this state will be the last fine maneuvers of the UAV to land accurately and with the correct orientation.

ROS will be used as the glue to allow the communications(figure 2) needed between camera, flight controller, and Odroid. Although ROS is not a real-time operating system, it has near real-time capability provided through use of ROS communication tools such as actions, which can provide needed interrupts to activities, and switch state. As the UAV nears the platform, the state will change with the initial image acquisition of the landing platform. This state change will initiate controls to be fed from the software on the Odroid to the flight controller to bring the UAV onto the landing pad with accuracy and correct orientation.

1. Test Autonomous Landing Capability in Simulation
2. Implement Autonomous Landing Capability
3. Test Autonomous Landing Capability

4. Tie Capabilities together
5. Final Testing of UAV with Landing Pad

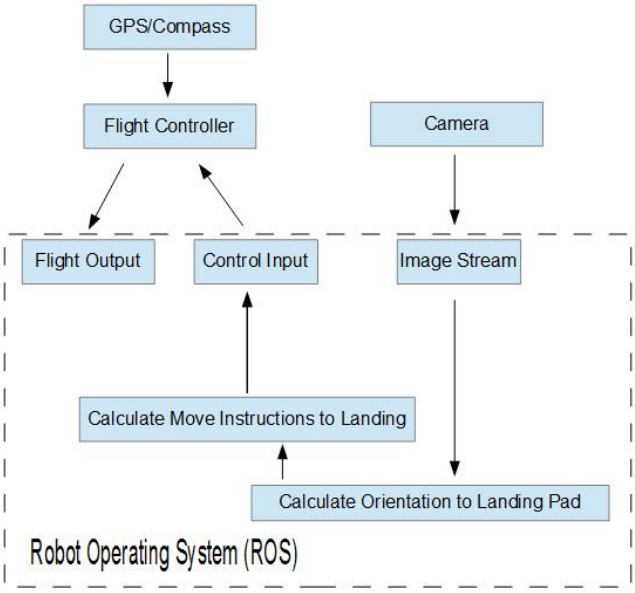


Figure 2: General ROS & Flight Control Setup

Components

Figure 3: Current UAV Build

UAV Hardware
The hardware platform is a quadrotor that had been assembled from a previous iteration of the project(figure 3). To achieve our goals, replacement parts will need to be ordered, as well as some additional equipment to allow the Odroid to coordinate specific landing commands given evaluation of images it receives(figure 4). The platform consists of:

- *APM 2.6+ Assembled*: This serves as the flight controller for the quadrotor. It includes a 3-axis gyro, accelometer, and barometer. It is protected within an enclosure.
- *3DR uBlox GPS with Compass Kit*: GPS and Compass unit with enclosure, external to flight controller to be located on the quadrotor where magnetic interference is least.
- *DIY Quad Kit*: Kit consisting of body and other components such as:
  - Landing Struts(4)
  - 850kV motors(4)
  - Power Distribution Board
  - Electronic Speed Controllers(4)
  - Propellers(4)
  - Power Module & Adapter
  - Radio module
  - Transmitter

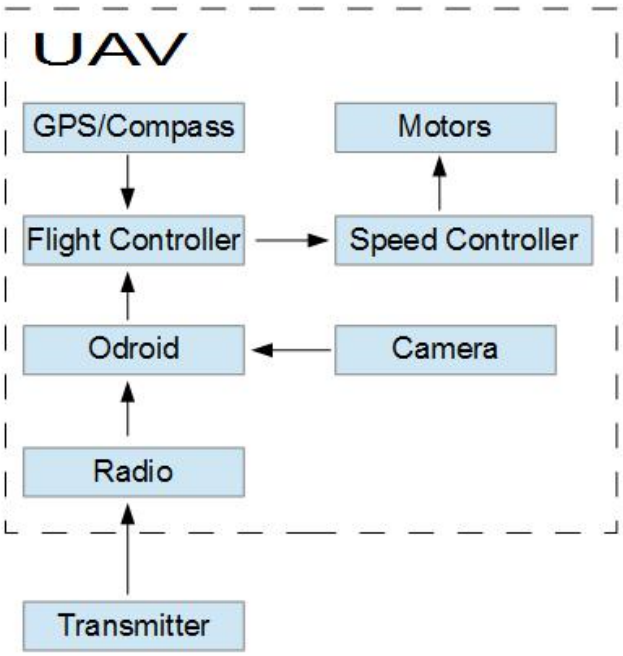


Figure 4: General Hardware Configurations

Additional Hardware

- *USB Camera(2)*: To be used in conjunction with Odroid to provide image stream.
- *Odroid XU4*: To be used to process images and provide instructions to the flight controller during the autonomous landing phase.

UAV Software

Team Expeditus will take advantage of developed software that interfaces and controls the quadrotor. As stated previously, this will provide the autonomous take-off and waypoint navigation. To achieve autonomous landing, the flight controller API will be used as a node within a ROS system.

- *Mission Planner (Windows)*: To be used during testing autonomous take-off flight and way-point navigation.
- *APM Planner(Windows/Linux)*: Alternative to Mission Planner, has Linux support.
- *MavLink*: API to receive output from the flight controller, as well as providing instructions to the flight controller.

Additional Software

- *OpenCV*: Provides a open-source library with image processing and computer vision capabilities.
- *ROS (Robot Operating System)*: An open-source pseudo-operating system employing network communications to link together component functionality to allow fast prototyping and implementation of real robots.
- *MavROS*: A ROS-ified version of MavLink. It functions as a node within the ROS framework.
- *Gazebo*: A simulation environment that can be linked with ROS.

Development Environment

Hardware:

- *Assembly & Repair*: The team has access to the Robotics Lab within the McLaury Bldg on the SDSMT campus. This lab is equipped with all necessary tools for removing, replacing, or otherwise altering the hardware configuration.
- *Flight Training*: The SDSMT UAV Team has provided orientation and access to a UAV flight simulator, so that team members are able to train on manual flight control.

Software:

- *Language*: C++ will be used for the project. No other language is currently forecasted for use.
- *OS*: Development will mostly be conducted in Ubuntu 14.04 which supports the current version of ROS & Gazebo. Current versions of Windows may be used briefly to test hardware through the use of Mission Planner.

Immediate Needs

Hardware:

- *APM 2.6+ Assembled*: Unit was unresponsive during testing.
- *3DR uBlox GPS with Compass Kit*: Unit was unresponsive during testing. Visual inspection revealed a crack across entire enclosure.
- *Odroid XU4*: Missing.
- *Landing Struts(8)*: Missing.
- *USB Cameras(2)*: Required for image capture.

Software:

- None

Sprint #2 Prototype

Team Expeditus

11/04/15

The purpose of this document is to review the demonstrable work products from Sprint 2. These work products are:

- Visual Homography for Autonomous Landing
- Simulation
- Purchasing UAV Parts

Visual Homography for Autonomous Landing

Introduction

The purpose of this report is to give a brief overview of the current status of the existing code from the 2014-2015 landing pad team's repository. Fortunately, it appears that much of the code is reusable. If not in full, parts of the code can be pulled from the repository to get the ball rolling. In the report, we'll go through where to find the existing code in the repository, and what it's currently capable of.

Location in Repository and Building Source

The code being discussed throughout is in the GitHub repository here. To run this code on your local machine (assuming you have opencv installed as well as the latest compilers) navigate to the above directory, and type 'cmake .'. Then, run make. Once you've done this, you can run the code with './tracker'. Assuming you're on your school laptop, it'll automatically access the front-facing camera, and you'll see a happy little picture of your smiling face! I'm assuming that you're happy, since you've just succeeded in getting the software to run. As this runs, you may notice that it draws some circles and coordinates on the video feed. This overlay is meant to point out the positions of the RGB blobs in the frame. We've taken the large printout of the AR tag and RGB blobs that the team used last semester, and tested the code with it. Fortunately, it works! You can see the results of this in the image below.

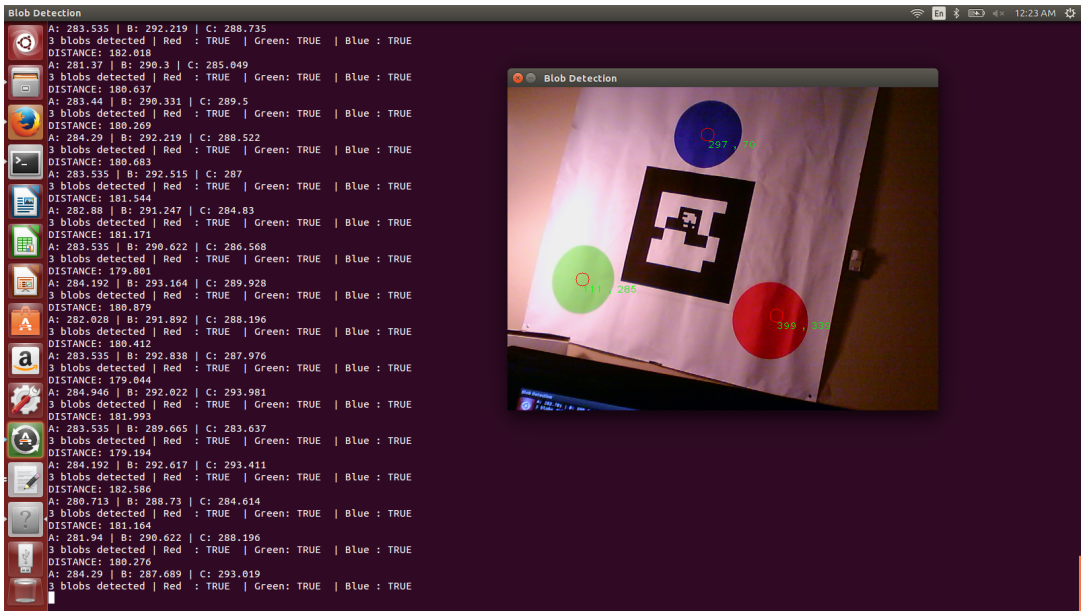


Figure 1: Blobs Detected

Next Steps
Since the existing code outputs a correct distance in centimeters, the next step will to be able to detect the angle of the image. This will allow us to determine orientation. Since we have working code, we should just be able to clean it up, and copy it over into a new branch of our current repository. From there, we will begin to develop orientation detection!

Simulation

Introduction
To test our landing algorithms in simulation, it would be very useful to have something that approximates the Pixhawk flight controller to communicate with for the purpose of supplying instructions to the controller, as well as receiving flight data. The PX4 development team have provided both Software-In-The-Loop and Hardware-In-The-Loop simulation meta-packages for use in ROS.

Build Instructions
This will require Linux 14.04, ROS Distro Indigo or Jade, and the installation of a few repositories. These are detailed well in the setup document provided by the group here. More roughly, after installation of the Ubuntu 14.04 and ROS Jade, other dependencies will need to be installed. After which, a number of repositories from GitHub will need to be cloned into the ROS workspace src directory. These repos include PX4/Firmware, PX4/rotors_ simulator, PX4/mav_ comm, ethz-asl/glog_ catkin, and catkin/catkin_ simple.
However, there were some issues found in following the instructions. For a successful build, it is easier to run each line of code from the provided bash scripts. Often, updates would fail, and the remainder of the script would fail to execute. Additionally, the **catkin_ make** command did not build the meta-package correctly, as detailed in the instructions. It would repeatedly fail in finding the correct path to necessary files for the build. Rather, **catkin build** will build the meta-package properly and the simulation will run with the assistance of an XBox controller (PS controllers will not work). Here is a demonstration of the SITL meta-package created by the PX4 Autopilot Project.

Next Steps
We will next begin building our commands to the simulated Pixhawk through the use of the MavLink API, implemented as MavROS for use in the ROS environment. We will also test the HITL loop to ensure that our commands are being received by the actual hardware.

Purchasing UAV Parts

Introduction
Over the course of Sprint 2, the team met with our advisor and faculty for purpose of receiving funding to create a new UAV platform for this project. The team also met with members of the UAV team to receive assistance and guidance in purchasing hardware that would be compatible with UAV team hardware. In the event of a component not functioning, our team would be able to utilize a component from the UAV team. After building a parts list for a hexrotor, we received approval from faculty for our purchase. Following is a list of the parts list with links to the pages containing information regarding the component.

Parts List

Item	#	Unit Cost	Unit Total
Frame	1	$79.99	$79.99
Motors	8	$23.99	$191.92
ESCs	8	$17.78	$142.24
Pixhawk	1	$199.99	$199.99
Power Distribution	1	$19.99	$19.99
GPS Mast	2	$10.00	$20.00
GPS	2	$89.99	$179.98
Power Module	1	$24.99	$24.99
ODroid XU4	1	$75.95	$75.95
Props	3	$7.55	$22.65
TOTAL			$957.70

Next Steps

After receiving the parts, the UAV will be assembled. After assembly, the team will be able to test flight control, which will provide feedback concerning a correctly assembled and functioning UAV. Afterwards, we will be able to test the autonomy in the flight controller, which will provide information regarding accuracy in waypoint navigation.

Computer Vision Sprint 3 Status

Jon Dixon

12/3/15

1 Introduction

The purpose of this report is to give a brief overview of the current status of the computer vision approach to landing. I've been playing around with some different ideas for how to do this, and have read a few papers, which will have links in this document.

2 Blob Detection Approach

The prototype 2 document located in the repository in the directory: landingpad/Documents/Prototypes/Sprint_2 gives details on the old code, so this will focus on progress made in sprint 3. For information on how to compile the old C code, see the document in that directory.

After the work done in the computer vision class, we decided to move away from the three-blob approach, and use four colored circles. The hope is that this will allow us to detect four points, and compute an accurate homography matrix for orientation, range, etc... Below is an image of the current layout being used in testing and development.

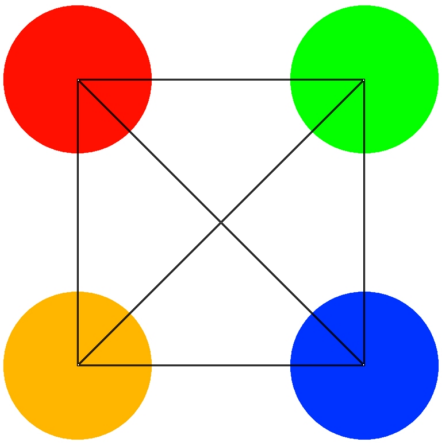


Figure 1: Four-blob landing guide

The approach we've been messing with is to compute a homography matrix based on the four detected points, which would then be used to warp the image so that the blobs are essentially in a "square." The next step will be to use the homography matrix to calculate the angle of rotation, and translation. This will allow us to see how far the target is offset, and calculate the angle that we will need to yaw the quad.

3 Other possible approaches

While doing research for graduate seminar, I read a number of papers on the topic of autonomous landing that all had various approaches to pad target design, and actual landing algorithms. These approaches all ended up using similar methods of image preprocessing, using binarization and thresholding. The next step will be to attempt to apply these to our own target design to see if we are able to use a similar approach.

At the same time, I have been working with learning OpenCV within python. One of the interesting things I discovered is the hough transform for circle detection. some preliminary testing seems to indicate that this will work relatively well for detecting circles, so I may play around with that some to see if there is anything useful I can come up with. Below is an image of the target above and some circles detected on it. One issue that I have had with this is there are a lot of false positives being detected, so if I play around with image preprocessing, I may be able to reduce the amount of these being detected.

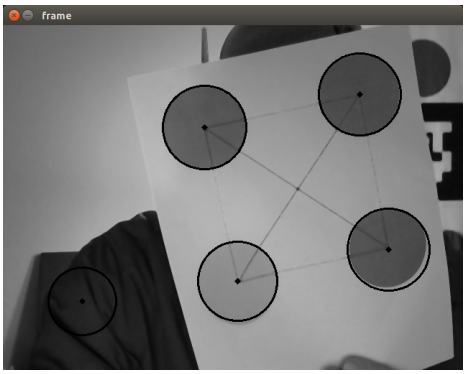


Figure 2: Hough detection on the target

Additionally, I am still interested in the possibility of using AR tags for this, so I will be looking into that in the future.

4 Next Steps

My plan now is to work to do more actual image processing to improve the reliability of any feature detection we will be doing. At the same time, I will be working to learn OpenCV, and find any useful functions that we can implement. In the meantime, I will be working in the github repository under the cv_python branch, in the cv_tracker directory.

5 Useful Reading

Here are links to the possibly useful papers on autonomous vehicle landing algorithms:
Paper with thresholding and corner detection
Paper focusing on image thresholding
Interesting paper using concentric circles on landing pad

Sprint 3.5-4 Prototypes

4/9/2016

1 Introduction

This is a prototype report to give the status of the UAV Lander project after sprints 3.5 and 4. The team was able to make progress and achieve the goals of manual flight and autonymous GPS waypoint navigation.

2 Manual Flight

After the UAV was successfully assembled, the team needed to do more than just spin the motors in the robotics lab. The UAV was taken up to the school gym, where the team was able to sucessfully fly it under manual control, sending commands directly over radio to the flight controller.

Figure 1: UAV Flying in the Gym

The manual flight in the gym was a good measure of first success for the team, as it showed that the UAV was assembled properly, and that the flight controller functioned well with our hexacopter setup. One thing that was mentioned to us here was that the handheld radio that the team was using was designed for fixed-wing aircraft such as model airplanes. This posed a problem when flying the UAV under manual control, because the throttle inputs would click into place, rather than being continuous. This was a problem because in order to maintain altitude, the pilot would have to constantly be clicking the throttle stick between various positions.

3 Autonymous GPS Waypoint Navigation

Shortly after the indoor manual flight, the team was able to get the UAV set up for autonymous GPS waypoint navigation. This is important to the project because one of the major tasks is to build a UAV that is able to navigate through a set of GPS waypoints. The results of the team's testing of the waypoint navigation point to the possibility that GPS will be sufficient to allow the UAV to position itself over the landing pad. This means that little to no work should need to be put into developing a search algorithm to get the UAV to locate the pad after flying to the waypoint, in the event that GPS was inaccurate.

Figure 2: UAV Navigating Through Waypoints in the Wind

One exciting result during testing was that the team flew the UAV on a windy day. In the above image, you can see that a flag is blowing in the wind. The UAV did not show any signs of being blown around, even despite this.

Figure 3: Location of UAV Before Takeoff

Figure 4: Location of UAV After Landing

The above two images show just how accurate the GPS waypoint navigation was. The UAV started at the location in the first image, took off, flew through a series of waypoints, and landed at the location in the second image. This test was done the same day, so this result was observed even in the wind. Another success that makes the team believe that GPS is reliable is that during these tests, one of the motor arms on the UAV had come loose and rotated. The flight controller was still able to keep the UAV under control, even despite these problems

Installing Alvar

Jon Dixon, Dylan Geyer

3/17/2016

1 Introduction

This document will give a brief description on how to install Alvar. It is important to note that this is not AR Track Alvar, which is the ROS library for AR tracking, but just a standalone AR track software.

2 Installing Alvar

Installing Alvar is easy! All you do is just run the installalvar.sh script. This will download and build all dependencies for Alvar, including OpenCV and ROS. The most important thing to note is that the shell script must be located in build directory of the Alvar directory that can be downloaded from the Alvar website.

```
#!/bin/bash

#Install ROS

#Setup your sources.list
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
    $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros
    -latest.list'

#Set up your keys
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.
    net:80 --recv-key 0xB01FA116

#Installation
sudo apt-get update
sudo apt-get install ros-jade-desktop-full

#Initialize rosdep
sudo rosdep init
rosdep update

#ROS Environment setup
echo "source /opt/ros/jade/setup.bash" >> ~/.bashrc
source ~/.bashrc

#Getting rosinstall
sudo apt-get install python-rosinstall

#Install opencv
version="$(wget -q -O - http://sourceforge.net/projects/
    opencvlibrary/files/opencv-unix | egrep -m1 -o
    '\"[0-9](\.[0-9]+)+' | cut -c2-)"
```

```
echo "Installing OpenCV" $version
mkdir OpenCV
cd OpenCV
echo "Removing any pre-installed ffmpeg and x264"
sudo apt-get -qq remove ffmpeg x264 libx264-dev
echo "Installing Dependenices"
sudo apt-get -qq install libopencv-dev build-essential
    checkinstall cmake pkg-config yasm libjpeg-dev
    libjasper-dev libavcodec-dev libavformat-dev
    libswscale-dev libdc1394-22-dev libxine-dev
    libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev
     libv4l-dev python-dev python-numpy libtbb-dev libqt4-
    dev libgtk2.0-dev libfaac-dev libmp3lame-dev
    libopencore-amrnb-dev libopencore-amrwb-dev libtheora-
    dev libvorbis-dev libxvidcore-dev x264 v4l-utils
    ffmpeg cmake qt5-default checkinstall
echo "Downloading OpenCV" $version
sudo wget -O OpenCV-$version.zip http://sourceforge.net/
    projects/opencvlibrary/files/opencv-unix/$version/
    opencv-"$version".zip/download
echo "Installing OpenCV" $version
sudo unzip OpenCV-$version.zip
cd opencv-$version
mkdir build
cd build
sudo cmake -D CMAKE_BUILD_TYPE=RELEASE -D
    CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D
    BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D
    INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
     BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON ..
sudo make -j2
sudo checkinstall
sudo sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/
    opencv.conf'
sudo ldconfig
echo "OpenCV" $version "ready to be used"

# Download and extract alvar-2.0.0-sdk-linux64-gcc44.tar.
    gz
# Go into the /alvar-2.0.0-sdk-linux64-gcc44/build
sudo apt-get install build-essential cmake
sudo apt-get install freeglut3-dev
sudo apt-get install libopenscenegraph-dev
sudo apt-get install gcc-4.4
sudo apt-get install g++-4.4
sudo apt-get install cmake-qt-gui
```

```
chmod +x generate*.sh
sudo ./generate_gcc44.sh
cd build_gcc44_release
sudo make install
```

7

User Documentation

This section should contain the basis for any end user documentation for the system. End user documentation would cover the basic steps for setup and use of the system. It is likely that the majority of this section would be present in its own document to be delivered to the end user. However, it is recommended the original is contained and maintained in this document.

7.1 User Guide

The source for the user guide can go here. You have some options for how to handle the user docs. If you have some `newpage` commands around the guide then you can just print out those pages. If a different formatting is required, then have the source in a separate file `userguide.tex` and include that file here. That file can also be included into a driver (like the senior design template) which has the client specified formatting. Again, this is a single source approach.

7.2 Installation Guide

The following two sections will go through the set up of a laptop or desktop as a workstation and the odroid that will be attached to the UAV.

7.2.1 Setting Up a Workstation

To set up the workstation the first thing that is required is a computer with Ubuntu 14.04 Long Term Support(LTS) installed as the operating system.

7.2.1.a Dependencies

Once a Ubuntu machine is acquired make sure git and cmake are installed by running the follwoing commands:

```
$ sudo apt-get install git
$ sudo apt-get install cmake
```

Then create the following two directories:

```
$ mkdir -p catkin_ws/src cmake_ws
```

These directories will be used to manage the libraries and software required by the UAV. The cmake_ws will contain the source of the libraries required by a semi-direct monocular visual odometry pipeline package in ROS also known as SVO. The catkin_ws directory will contain the source for SVO and other packages that will be written and downloaded for the UAV. Before any of the ROS packages can be installed the dependencies for SVO will need to be installed.

The following commands will install Sophus which is required by SVO:

```
$ cd cmake_ws
$ git clone https://github.com/strasdat/Sophus.git
$ cd Sophus
$ git checkout a621ff
$ mkdir build
$ cd build
$ cmake ..
$ make
```

The following commands will install Fast detector which is used by SVO to detect corners:

```
$ cd cmake_ws
$ git clone https://github.com/uzh-rpg/fast.git
$ cd fast
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Once Sophus and Fast are installed the ROS packages can start to be installed to the system.

7.2.1.b ROS

To install ROS refer to http://wiki.ros.org/indigo/Installation/Ubuntu. It will always have the most up to date installation instructions for installing ROS indigo onto Ubuntu. Everything has successfully worked under both indigo and jade versions of ROS. If ROS jade has reached EOL then indigo should be used. The commands and links are given with ros-indigo instead of ros-jade for this purpose.

To install mavros, ar-track-alvar, and the camera driver the following commands needs to be executed:

```
$ sudo apt-get install ros-indigo-mavros*
$ sudo apt-get install ros-indigo-ar-track alvar
$ sudo apt-get install ros-indigo-pointgrey-camera-driver
```

In order to install ROS packages the catkin workspace will have to be initialized so that ROS libraries can be found for ROS packages that are written by a user or to compile source code.

To initialize the catkin workspace execute the following:

```
$ cd catkin_ws/src
$ catkin_init_workspace
$ cd ..
$ catkin_make
$ source devel/setup.bash
```

The source command above can be added to the .bashrc file in the home directory so the workspace doesn't have to be explicitly sourced. This will allow usage of packages compiled in this directory without having to remember to source it.

Before SVO can be installed a ROS dependency vikit will need to be downloaded into the workspace alongside SVO.

```
$ cd catkin_ws/src
$ git clone https://github.com/uzh-rpg/rpg_vikit.git
$ git clone https://github.com/uzh-rpg/rpg_svo.git
$ cd ..
$ catkin_make
```

7.2.2 Setting Up the Odroid

To set up the odroid that will be attached to the UAV the instructions for burning an image can be found here http://odroid.com/dokuwiki/doku.php?id=en:odroid_flashing_tools. The image that is required is the same as the odroid xu3 found at http://odroid.com/dokuwiki/doku.php?id=en:xu3_release_linux_ubuntu. It will be less demanding on the odroid if the sever version of the xu3 is used instead a desktop variant. Also space will be saved since a user interface is not necessary to run the commands on the odroid.

Once the odroid is booting properly connect it to a monitor or ssh into it through the network to complete the rest of the installation. Before installing anything run the following command in the console:

```
$ export ARM_ARCHITECTURE=True
```

After the above command is run the instructions for creating the directories and compiling the cmake libraries will work just like the workstation. However, the link above for the ROS directions is for a laptop or desktop computer that will be running ROS. The directions change since the odroid is a single board computer that uses the ARM architecture. The instructions for this can be found on the ROS website at http://wiki.ros.org/indigo/Installation/UbuntuARM. The rest of the instructions in the workstation section can be followed once ROS is installed on the odroid.

7.3 Programmer Manual

8

Class Index

8.1 Class List

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

9

Class Documentation

9.1 Poly Class Reference

Public Member Functions

- Poly ()
- ∼Poly ()
- int myfunction (int)

9.1.1 Constructor & Destructor Documentation

9.1.1.a Poly::Poly ( )

My constructor

9.1.1.b Poly::∼Poly ( )

My destructor

9.1.2 Member Function Documentation

9.1.2.a int Poly::myfunction (int *a*)

my own example function fancy new function
new variable
The documentation for this class was generated from the following file:

- hello.cpp

10

Experimental Logs

For research projects one needs to keep a log of all research/lab activities.

10.1 Takeoff Log

10/15/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

10.2 Navigation Log

10/03/15 Tested ardupilot with windows. Could not get landingpad ardupilot to connect, but was able to get Robotics Team ardupilot to work with windows. Also was able to get this ardupilot connected in Ubuntu and working with MavRos. Chris Smith

10.3 Landing Log

10/15/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

11/14/15 AI Landing. Created vague sketch of AI approach to landing problem. Utilizing resource provided by Dr. Larry Pyeatt (Advisor/Client) to learn more on the subject of Reinforcement Learning.
Initial thoughts are:
Visualize an inverted cone above the landing pad, so that the point is located in the center of the landing pad. The cone is divided horizontally into slices which represent the height above the landing pad. Each slice is further segmented into wedges, much like a pie. Each one of these pie wedges will represent the state of the UAV.
The goal is to land the UAV on the center of the landing pad with correct orientation. The AI will be penalized for choices that result in greater distance from landing pad, greater distance from correct orientation, greater distance from the vertical center of the cone. Getting closer will be rewarded.
The inputs will be the image, motor speeds, IMU(roll, pitch, yaw). The outputs will motor speeds.

Christopher Smith, Steven Huerta

11/15/15 AI Landing. Briefly discussed inputs, outputs, and rewards. Uploaded Dr. Pyeatt's Artificial Neural Net to repo. We will review this code and likely use some or most of it as a foundation for the Landing AI.

Christopher Smith, Steven Huerta

11/22/15 AI Landing. Discussed Sarsa algorithm to solve our landing problem. Outlined rewards as -1 penalty for every time step landing has not been found, and $+1$ reward for minimizing distance from landing pad. We discussed how to use Dr. Pyeatt's work to inform our own. Decided that using a 2-agent solution would

make use of the gradient field for radial distance and height.

Christopher Smith, Steven Huerta

10.4 Mechanical Log

10.4.1 UAV

10/15/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

10.4.2 UGV

10/15/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

10.5 Simulation Log

9/26/15 Installed Ubuntu LTS, ROS Jade, and Gazebo5 Chris Smith

10/3/15 Went through tutorials on using Gazebo and Ros Jade. Chris Smith

10/9/15 Installed Rotors-Simulator for ROS and Gazebo. Chris Smith

10/15/15 Worked on Gazebo Simulation with Rotors-Simulator. Chris Smith

11/14/15 Worked on integrating waypoint passing with Pix4 SITL simulation.
First attempt: Attempted to create a waypoint publisher to pass waypoints via ROS. Abandoned this method when waypoint file loader was found in Mavlink documentation.
Second attempt: Attempted to utilize waypoint file loader to pass a file of waypoints and instructions to simulated UAV. Abandoned this method when we were unable to connect to MavROS implementation. Calling waypoint loader method would result in communication error. Third attempt: Attempted to use a different simulation of PixHawk and UAV in order to implement file loading. Abandoned after many unsuccessful attempts to load waypoint files.

Chris Smith, Steven Huerta

11/15/15 Continued working on simulation waypoint passing. Returning to original conceived method of creating a waypoint publisher node to pass waypoints from a file to the simulated PixHawk. Created fork in Git to work on waypoint publisher.

Chris Smith, Steven Huerta

11/21/15 Continued working on simulation environment setup. Started from stratch with SITL Pix4 simulation. Can now see pose messages generated from joystick operation of simulated UAV being published via mavros_sitl node. Still unable to successfully control UAV through mavros commands.

Steven Huerta

11/23/15 Worked on setting up ground control station, QGroundControl. Possible to setup message passing between ground control station and pixhawk simulation.

Steven Huerta

11/29/15 Working on ground control station setup with pixhawk.

Steven Huerta

11

Research Results

This chapter describes the results and conclusions of your research. This would be the final report for a research project.

11.1 Result 1

11.2 Result 2

11.3 Conclusions

11.4 Further work

Bibliography

[1] R. Arkin. *Governing Lethal Behavior in Autonomous Robots*. Taylor & Francis, 2009.

[2] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[3] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[4] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automation moving amidst unknown obstacles of arbirary shape. *Algorithmica*, pages 403–430, 1987.

[5] S.A. NOLFI and D.A. FLOREANO. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. A Bradford book. A BRADFORD BOOK/THE MIT PRESS, 2000.

[6] Wikipedia. Asimo — Wikipedia, the free encyclopedia. http://upload.wikimedia.org/wikipedia/commons/thumb/0/05/HONDA_ASIMO.jpg/450px-HONDA_ASIMO.jpg, 2013. [Online; accessed June 23, 2013].

SDSMT SENIOR DESIGN
SOFTWARE DEVELOPMENT AGREEMENT

This Software Development Agreement (the "Agreement") is made between the SDSMT Computer Science

Senior Design Team _____,
("Student Group")

consisting of team members _____,
("Student Names")

and Sponsor _____,
("Company Name")

with address: _____.

[Note: Bracketed material is included to suggest content that will vary with each agreement. I STRONGLY SUGGEST THAT THE INSTRUCTOR LOOK AT THE COMPLETED AGREEMENT BEFORE YOU SIGN IT!!]

1 RECITALS

1. Sponsor desires Senior Design Team to develop software [for use in Sponsor's simulation platform for optical fiber transmissions of digitized video signals] (the "Field").
2. Senior Design Teams willing to develop such Software.

NOW, THEREFORE, in consideration of the mutual covenants and promises herein contained, the Team and Sponsor agree as follows:

2 EFFECTIVE DATE

This Agreement shall be effective as of _____ (the "Effective Date").

3 DEFINITIONS

1. "Software" shall mean [the computer programs in machine readable object code form and any subsequent error corrections or updates supplied to Sponsor by Senior Design Team pursuant to this Agreement.] [Depending on the particulars of each agreement, any or all of the following may need to be specified. If they are relevant, they should be used throughout, modifying the standard form as appropriate.]
2. "Acceptance Criteria" means the written technical and operational performance and functional criteria and documentation standards set out in the [project plan.]
3. "Acceptance Date" means [the date for each Milestone when all Deliverables included in that Milestone have been accepted by Sponsor in accordance with the Acceptance Criteria and this Agreement.]
4. "Deliverable" means a deliverable specified in the [project plan.]
5. "Delivery Date" shall mean, [with respect to a particular Milestone,] the date on which University has delivered to Sponsor all of the Deliverables [for that Milestone] in accordance with [the project plan and] this Agreement.

6. "Documentation" means the documents, manuals and written materials (including end-user manuals) referenced, indicated or described in [the project plan] or otherwise developed pursuant to this Agreement.
7. "Milestone" means the completion and delivery of all of the Deliverables or other events which are included or described in [the project plan] scheduled for delivery and/or completion on a given target date; a Milestone will not be considered completed until the Acceptance Date has occurred with respect to all of the Deliverables for that Milestone.

4 DEVELOPMENT OF SOFTWARE

1. Senior Design Team will use its best efforts to develop the Software described in [the project plan.] The Software development will be under the direction of or his/her successors as mutually agreed to by the parties ("Team Lead") and will be conducted by the Team Lead. The Team will deliver the Software to the satisfaction of the course instructor that reasonable effort has been made to address the needs of the client. The Team understands that failure to deliver the Software is grounds for failing the course.
2. Sponsor understands that the Senior Design course's mission is education and advancement of knowledge, and, consequently, the development of Software must further that mission. The Senior Design Course does not guarantee specific results or any results, and the Software will be developed only on a best efforts basis. The Software is considered PROOF OF CONCEPT only and is NOT intended for commercial, medical, mission critical or industrial applications.
3. The Senior Design instructor will act as mediator between Sponsor and Team; and resolve any conflicts that may arise.

5 COMPENSATION

[This is entirely subject to negotiation. Normally NO COMPENSATION occurs in a Senior Design Project. On occasion an intern status and wage is appropriate.]

6 CONSULTATION AND REPORTS

1. Sponsor's designated representative for consultation and communications with the Team Lead shall be

   _____ or such other person as Sponsor may from time to time designate to the Team Lead ("Designated Representative").
2. During the Term of the Agreement, Sponsor's representatives may consult informally with course instructor regarding the project, both personally and by telephone. Access to work carried on in University facilities, if any, in the course of this Agreement shall be entirely under the control of University personnel but shall be made available on a reasonable basis.
3. The Team Lead will submit written progress reports. At the conclusion of this Agreement, the Team Lead shall submit a comprehensive final report in the form of the formal course documentation at the conclusion of the Senior Design II course.

7 CONFIDENTIAL INFORMATION

1. The parties may wish, from time to time, in connection with work contemplated under this Agreement, to disclose confidential information to each other ("Confidential Information"). Each party will use reasonable efforts to prevent the disclosure of any of the other party's Confidential Information to third parties for

a period of three (3) years after the termination of this Agreement, provided that the recipient party's obligation shall not apply to information that:

(a) is not disclosed in writing or reduced to writing and so marked with an appropriate confidentiality legend within thirty (30) days of disclosure;

(b) is already in the recipient party's possession at the time of disclosure thereof;

(c) is or later becomes part of the public domain through no fault of the recipient party;

(d) is received from a third party having no obligations of confidentiality to the disclosing party;

(e) is independently developed by the recipient party; or

(f) is required by law or regulation to be disclosed.

2. In the event that information is required to be disclosed pursuant to subsection (6), the party required to make disclosure shall notify the other to allow that party to assert whatever exclusions or exemptions may be available to it under such law or regulation.

8 INTELLECTUAL PROPERTY RIGHTS

[Negotiated on a case-by-case basis. This must address who owns the algorithms and who owns the source code. For example: All deliverables become property of the Sponsor. Roughly: If the idea originates with the sponsor, or if a sponsor pays you to develop an idea, then they have legitimate claim to the IP. If the idea originates from the University (through faculty or staff) then the University has legitimate claim. If the idea is yours (student) and you develop it without external compensation then you have legitimate claim.]

9 WARRANTIES

The Senior Design Team represents and warrants to Sponsor that:

1. the Software is the original work of the Senior Design Team in each and all aspects;
2. the Software and its use do not infringe any copyright or trade secret rights of any third party.

No agreements will be made beyond items (1) and (2).

10 INDEMNITY

1. Sponsor is responsible for claims and damages, losses or expenses held against the Sponsor. [Sponsor may have something to add here.]
2. Sponsor shall indemnify and hold harmless the Senior Design Team, its affiliated companies and the officers, agents, directors and employees of the same from any and all claims and damages, losses or expenses, including attorney's fees, caused by any negligent act of Sponsor or any of Sponsor's agents, employees, subcontractors, or suppliers.
3. NEITHER PARTY TO THIS AGREEMENT NOR THEIR AFFILIATED COMPANIES, NOR THE OFFICERS, AGENTS, STUDENTS AND EMPLOYEES OF ANY OF THE FOREGOING, SHALL BE LIABLE TO ANY OTHER PARTY HERETO IN ANY ACTION OR CLAIM FOR CONSEQUENTIAL OR SPECIAL DAMAGES, LOSS OF PROFITS, LOSS OF OPPORTUNITY, LOSS OF PRODUCT OR LOSS OF USE, WHETHER THE ACTION IN WHICH RECOVERY OF DAMAGES IS SOUGHT IS BASED ON CONTRACT TORT (INCLUDING SOLE, CONCURRENT OR OTHER NEGLIGENCE AND STRICT

LIABILITY), STATUTE OR OTHERWISE. TO THE EXTENT PERMITTED BY LAW, ANY STATUTORY REMEDIES WHICH ARE INCONSISTENT WITH THE PROVISIONS OF THESE TERMS ARE WAIVED.

11 INDEPENDENT CONTRACTOR

For the purposes of this Agreement and all services to be provided hereunder, the parties shall be, and shall be deemed to be, independent contractors and not agents or employees of the other party. Neither party shall have authority to make any statements, representations or commitments of any kind, or to take any action which shall be binding on the other party, except as may be expressly provided for herein or authorized in writing.

12 TERM AND TERMINATION

1. This Agreement shall commence on the Effective Date and extend until the end of classes of the second semester of Senior Design (CSC 467), unless sooner terminated in accordance with the provisions of this Section ("Term").
2. This Agreement may be terminated by the written agreement of both parties.
3. In the event that either party shall be in default of its materials obligations under this Agreement and shall fail to remedy such default within thirty (30) days after receipt of written notice thereof, this Agreement shall terminate upon expiration of the thirty (30) day period.
4. Any provisions of this Agreement which by their nature extend beyond termination shall survive such termination.

13 ATTACHMENTS

Attachments A and B are incorporated and made a part of this Agreement for all purposes.

14 GENERAL

1. This Agreement constitutes the entire and only agreement between the parties relating to the Senior Design Course, and all prior negotiations, representations, agreements and understandings are superseded hereby. No agreements altering or supplementing the terms hereof may be made except by means of a written document signed by the duly authorized representatives of the parties.
2. This Agreement shall be governed by, construed, and enforced in accordance with the internal laws of the State of South Dakota.

15 SIGNATURES

Senior Design Team

Jonathan Dixon	9-29-15 Date
Dylan Geyer	9-29-15 Date
Steven Huerta	9-29-15 Date
Christopher Smith	29 sept 15 Date

Sponsor

Larry Pyeatt	10-1-15 Date

A

Sprint Reports

1 Sprint Report #1

Team Overview

Name
Expeditus

Members
Jonathan Dixon, Dylan Geyer, Steven Huerta, Christopher Smith

Project Title
UAV Landing Pad

Sponsor
Dr. Larry Pyeatt, SDSMT MCS

Sponsor Overview

Sponsor Description
The Math and Computer Science Department of South Dakota School of Mines and Technology, in addition to providing ABET certified education to students, conducts software-side robotics research including autonomy, navigation, and computer vision.

Sponsor Problem
The capability of UAVs to rapidly search a large area, especially one that is difficult to traverse by foot or vehicle, would be invaluable to operations such as search & rescue. However, small UAVs have a very limited flight time. A system incorporating a UVG equipped with a landing pad that also serves as a charging station would allow the UAV to be delivered to areas of limited access. The UAV could then, being provided with waypoints by the user, autonomously take-off, and navigate through the waypoints. After moving through the waypoints, or when the UAV requires recharging, the UAV will return to the UVG and safely land in such a way that the charging unit can connect to the UAV.

Sponsor Needs

- Ability to communicate waypoints to UAV.
- UAV can autonomously take-off.
- UAV can autonomously navigate through waypoints.
- UAV can autonomously navigate back to landing pad.
- UAV can autonomously land safely and with the correct orientation.

Project Overview

Phase 1 First phase will focus on finalizing the autonomous take-off and waypoint navigation by the UAV. Previous development will be reviewed, implemented, and tested. Simulation environment will be created for the purpose of testing landing algorithms.

Phase 2 Second phase will focus on finalizing autonomous landing

Project Environment

Project Boundaries

- Project is constrained to the UAV autonomy problems of take-off, navigation, and landing.
- Autonomous landing is constrained by fixed position landing platform with ideal operating conditions.
- Autonomous take-off is constrained by taking flight from a fixed position platform, with ideal operating conditions.
- Autonomous waypoint navigation is constrained by absence of obstacles, and operating with ideal operating conditions.

Project Context

- Project will utilize stable ROS distribution
- Project simulations will utilize Gazebo 6.+ & ROS package Rviz
- Project will be developed in Linux environment compliant with ROS & Gazebo

Deliverables

Phase 1

- Requirements documentation
- Overview documentation

Phase 2

- Project software
- Log
- Refence manual (software documentation)
- User documentation
- System design documentation
- Testing documentation
- Deployment documentation

Product Backlog

Phase 1

- **O-1**: As an owner, I want the UAV to autonomously take-off from the landing pad
- **O-2**: As an owner, I want the UAV to autonomously navigate through a series of waypoints

Phase 2

- **U-1**: As a user, I want to communicate the waypoints to the UAV
- **O-3**: As an owner, I want the UAV to autonomously return to the location of the landing pad
- **O-4**: As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft
- **O-5**: As an owner, I want the UAV to autonomously land on the landing pad with the correct orientation

Sprint Report

Completed Tasks

- Install Ubuntu 14.04 or some other ROS Indigo/Jade distro compliant OS.
- Setup Gazebo 6.+
- Download Rviz package
- Review previous iteration of project documentation
- Inspect current quadrotor configuration
- Identify parts needed for quadrotor

Tasks Carried to Next Sprint

- Acquire parts needed for quadrotor

2 Sprint Report #2

Summary

Team Expeditus was able to adapt to setbacks, and make progress on other tasks and goals of the project. Specifically, the team was able to make progress on the landing software, as well as find and implement a visual simulation that models the Pixhawk flight controller. The team was also able to coordinate with our advisor and school faculty for funding and ordering of our UAV platform and components. The restructuring of tasks for Sprint 2, does have a knock-on effect for Sprint 3. Sprint 3 will focus heavily on the building and testing of the UAV and its off-the-shelf components. Additionally, our client/advisor has suggested a different AI approach than the Artificial Neural Network(ANN). We will pursue this development during Sprint 3. Lastly, after meeting with the CENG/EE team several times, it was determined that while there is an opportunity for collaboration, time frames for both teams will not support collaboration. Our team will continue, however, to work with the ME UGV team on the development of a landing pad functional for both teams.

Team Work

- **Julian Brackins:** Worked on tasks relating to Autonomous Landing.
- **Jonathan Dixon:** Worked on tasks relating to Autonomous Landing.
- **Dylan Geyer:** Worked on tasks relating to ordering parts for the UAV,
- **Christopher Smith:** Worked on tasks relating to ordering parts for the UAV, as well as tasks relating to setting up a Simulation Environment.
- **Steven Huerta:** Worked on tasks relating to ordering parts for the UAV, as well as tasks relating to setting up a Simulation Environment.

Completed Backlog

Common Development Tasks

- **Setup Simulation Environment.**
  The team now has a working software simulation of the Pixhawk 4, the flight controller for this build, that utilizes both ROS and Gazebo.
- **Identify Parts Needed for UAV.**
  The team was supplied with funding source. The team needed to additionally coordinate with the SDSMT UAV Team to order correct parts, as well as parts that would be useful to both groups to ensure redundancy in the event of component failure.
- **Acquire parts needed for quadrotor**
  Received approval for the ordering of the parts. Parts ordered. Expected delivery date of 11/9/15.

As a user, I want to communicate the waypoints to the UAV

- **Review code that communicates with quadrotor.**
  Software is available to access the flight controller through a GUI called APM Planner, available for Linux/Windows. Additionally, there is Mission Planner, available for Windows. Both will provide the ability of a user to communicate with the UAV.
- **Review code that allows a user to input waypoints.**
  Both APM Planner and Mission Planner allow the user to input waypoints through the GUI.

As an owner, I want the UAV to autonomously take-off from the landing pad.

- **Review code that enables the quadrotor to autonomously take-off from landing pad.**
  This will be handled by Mission Planner/APM Planner.

As an owner, I want the UAV to autonomously navigate through a set of waypoints.

- **Review previous implementation for navigating waypoints.**
  This will be handled by Mission Planner or APM Planner

As an owner, I want the UAV to autonomously return to the location of the landing pad.

- **Review code that allows the autonomous return of the UAV to the landing pad.**
  This will be handled by Mission Planner or APM Planner. The built in autonomy will bring the UAV to a position near the landing pad, where either Visual Homography, Artificial Intelligence, or combination of the two will be responsible for landing the craft. It is estimated that the craft will be within 10 meters of the designated area. Discussions with UAV team members provide an estimate of 5 meters from their observations.

As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft

- **Review previous implementation for autonomous landing.**
  The code was reviewed and is running. The software is correctly identifying the RGB lights and accurately reporting distance.

As an owner, I want the UAV to autonomously land on the landing pad with the correct orientation.

- **Review previous implementation for autonomous landing.**
  As reported above, the software is running and is able to detect the lights. This detection will allow the UAV to orient itself to align correctly with the landing pad.

Uncompleted Tasks

Common Development Tasks

- **Build UAV**
  This will be completed during Sprint 3. Waiting for UAV parts to arrive.

- **Test flight under manual control**
  Testing will be completed by Sprint 3. Waiting for UAV to be built.

Prototype

There is a prototype document for Sprint 2 (found here in the repository), where this same material will be covered in much greater detail. This is only a brief description.

- **Visual Homography Code**
  The Visual Homography Code that was developed last year for the UAV Landing Project has been reviewed. The program successfully builds and much of it is likely to be reused towards providing the landing algorithm. The code can be found here in the repository. The code requires that OpenCV has been installed. A cmake file is contained within the directory, so that after running cmake and make the program can be run by ./tracker. The tracker is looking for RGB blobs, so it may be better for testing to have some primary colors about to test. This program is currently providing correct distance in centimeters.

- **Simulation**
  To test our landing algorithms in simulation, it would be very useful to have something that approximates the Pixhawk flight controller to communicate with for the purpose of supplying instructions to the controller, as well as receiving flight data. The PX4 development team have provided both Software-In-The-Loop and Hardware-In-The-Loop simulation environments. This will require Linux 14.04, ROS Distro Indigo or Jade, and the installation of a few repositories. These are detailed well in the setup document provided by the group here. However, **catkin_ make** command did not build the meta-package correctly, as detailed in the instructions. Rather, **catkin build** will build the meta-package properly and the simulation will run with the assistance of an XBox controller (PS controllers will not work).

- **Ordering Parts**
  Over the course of Sprint 2, the team met with our advisor and faculty for purpose of receiving funding to create a new UAV platform for this project. The team also met with members of the UAV team to receive assistance and guidance in purchasing hardware that would be compatible with UAV team hardware. In the event of a component not functioning, our team would be able to utilize a component from the UAV team. After building a parts list for a hexrotor, we received approval from faculty for our purchase. A complete order list will be detailed in the Prototype document.

3 Sprint Report #3

Summary

Team Expeditus was unable to meet the expectations that it set for the conclusion of the first phase, ending with the conclusion of the third sprint. The team began this Sprint adapting to delays in receiving and testing UAV. The final contruction of the UAV was delayed beyond the end of this sprint due to ordering issues. Although the team had made progress on UAV construction, Simulation, and Landing approaches, the team acknowledges that we are currently behind schedule for Phase II. Workdays will be scheduled for team members to make up lost ground before the beginning of Sprint 4.

Team Work

- **Julian Brackins:** Worked on tasks relating to Autonomous Landing.
- **Jonathan Dixon:** Worked on tasks relating to Autonomous Landing.
- **Dylan Geyer:** Worked on tasks relating to assembly of UAV
- **Christopher Smith:** Worked on tasks relating to setting up simulation environment and artificial intelligence landing approach.
- **Steven Huerta:** Worked on tasks relating to setting up simulation environment and artificial intelligence landing approach.

Completed Backlog

Common Development Tasks

- **Setup Simulation Environment.**
  Software simulation of Pixhawk and Ardupilot (both make use of Mavlink which will be the handle used by our team to send commands and receive information from the flight controller). Simulations are successful in setup, however issues with communicating with Software-in-the-loop simulations.
  - Pixhawk simulation (Pix4 ROS SITL): is working. The simulation will successfully emulate the flight controller and set up a simulation in Gazebo using ROS framework. Unable to send instructions to flight controller. Can successfully control UAV with Joystick to provide manual commands. Attempted to setup ground control station software (QGroundControl) to connect to Pixhawk SITL simulation. Unsuccessful in attempts to relay waypoint missions or arm the simulated UAV.
  - Arducopter (Ardupilot SITL with simulated Quadrotor): This simulation is working and the UAV can be controlled with a controller. Unable to communicate missions or controls via Mavlink API. A ROS waypoint publisher was developed to supply the simulation with commands to navigate to designated waypoints via MavROS(ROS wrapper for Mavlink). The publisher is successful in reaching the simulated ardu, however, the simulated UAV is unresponsive to ARM attempts, as well as commands to take-off or to navigate to waypoints.

  Team members are hopeful that using the actual flight controller integration will be much smoother, as networking issues specific to simulated hardware are no longer a possible culprit, limiting the scope of troubleshooting.

- **Acquire parts needed for quadrotor**
  Frame, motors, and ESCs were received by Nov 13th. Waiting for remainder of order, including Flight Controller, GPS, power distribution board, and battery connector.
- **Build UAV**
  Construction began on the UAV. UAV frame is assembled. Motors and ESCs are attached.

As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft

- **Modify/Rewrite implementation as necessary**
  The current setup of three points was found to insufficient to calculate the necessary transformation to unwarp the image. Team members have implemented a method to use a square, rather than a triangle.

As an owner, I want the UAV to autonomously land on the landing pad with the correct orientation.

- **Modify/Rewrite implementation as necessary**
  Changing the number of points from three to four causes some additional problem-solving. With three points, each can easily be assigned a color for clear distinction between points (Red, Green, Blue). The addition of a fourth causes some solving, as before only each of the RGB was used. Testing on simulated images, team members used Black. However, it is acknowledged that this will not be an available color. Once the color problem is solved, orientation will be regained.

Uncompleted Tasks

As a user, I want to communicate the waypoints to the UAV

- **Modify/Rewrite imlementation as necessary**
  - Waypoint Publisher: This is a test implementation to take a file of waypoints generated by a ground control station(GCS) and sequentially read them to supply the simulated flight controller with commands. This publisher is implemented with ROS, which will be the ultimate form of our landing command framework. The publisher is successful in sending messages, however, the UAV is not acting on the messages being passed. More troubleshooting will be needed.
  - Ground Control Station(GCS): There are some great GUI GCSs available. We have attempted to use QGroundControl to connect to the simulated flight controlled. This GCS is suggested by the developers of the flight controller simulations.

As an owner, I want the UAV to autonomously take-off from the landing pad.

- **Modify/Rewrite implementation as necessary**
  This is dependent on successful communication using Mavlink/MavROS. Once the communication problem has been solved, the team will be able to use the GCS to provide the command to take-off. Additional support will not be needed (ie, no need to code custom interface).

As an owner, I want the UAV to autonomously navigate through a set of waypoints.

- **Modify/Rewrite implementation as necessary**
  This is dependent on successful communication using Mavlink/MavROS. Once the communication problem has been solved, the team will be able to use the GCS to provide the command to navigation. Additional support will not be needed.

As an owner, I want the UAV to autonomously return to the location of the landing pad.

- **Modify/Rewrite implementation as necessary**
  This is dependent on successful communication using Mavlink/MavROS. Once the communication problem has been solved, the team will be able to use the GCS to provide the command to navigate back to the landing pad. Additional support will not be needed.

As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft

- **Propose AI Landing Algorithm**
  Team members have discussed approaches, but were unable to dedicate a sufficient amount of time to create or begin an initial prototype.

Prototype

There is a prototype document for Sprint 3 (found here in the repository), where this same material will be covered in much greater detail. This is only a brief description.

- **SIMULATION: Ardu SITL**
  Software in the loop (SITL) was attempted again with many different simulators that work with ROS. One of them implemented there on version of a Mavlink and ROS interface and was not well documented on how to send commands besides direct motor. Another used Mavros and Mavlink for direct communication with the Pixhawk. However, it gave errors of invalid flight control unit and files did not hash properly so believed it was recieving invalid waypoint files that were generated by the mission planner. Hardware in the loop (HITL) was also attempted using the ardupilot with many different simulators. The same errors were also received with the file checksum integrety (MD5) and bot the SITL and HITL gave the same errors with all commands sent. Pixhawk was not attempted since it was received the week after the sprint, but will be attempted over break.

- **SIMULATION: Pix ROS SITL Setup**
  These are the instructions for setting up the ROS SITL simulation. This simulation has the advantage of completely modelling the Pixhawk flight controller, the FCU that will be used by the team. The simulation will start an instance of Gazebo where a representation of a UAV controlled by the FCU is implemented. The UAV may be manually flown by the use of a joystick (XBox controller needed).

- **SIMULATION: QGroundControl Setup**
  These are the instructions for setting up the QGroundControl ground control station software for Ubuntu 14.04. QGroundControl provides a GUI that can link to the UAV's flight controller unit and supply missions such as autonomous take-off, waypoint navigation, and landing.

- **LANDING: Visual Homography**
  This document discusses the current direction of Visual Landing approach.

- **UAV BUILD: Initial UAV Build**
  This document details the initial build of the UAV with current parts available.

4 Sprint Report #4

Summary

Team Expeditus was able to accomplish most of the objectives for sprint 3.5 and 4. Sprint 3.5 focused mostly on designing a new base plate for the hexrotor and calibrating the flight controller. During this sprint it was agreed on by the team members to set aside work on the simulation as too much time was being consumed by issues encountered. The UAV was able to flown under manual control successfully by the end of Sprint 3.5. Sprint 4 focused on autonomous flight testing, flight control specific message passing in the ROS framework, and reapproach to vision. By the end of the sprint the autonomous flight was successfully tested and a software package for AR tag tracking was successfully compiled and run.

Team Work

- **Jonathan Dixon:** Worked on UAV build, AR tracking, and UAV flight.
- **Dylan Geyer:** Worked on UAV build, AR tracking, and UAV flight.
- **Christopher Smith:** Worked on UAV build, message passing, AR tracking, and UAV flight.
- **Steven Huerta:** Worked on UAV build, AR tracking, and UAV flight.

Completed Backlog

Common Development Tasks

- **Build UAV.**
  The UAV was assembled before winter break, however, it was apparent that the power distribution board did not fit appropriately, and in addition, the team realized that more room would be needed on the center plate to accommodate the control hardware and sensors. Team members modeled a new plate that was larger and printed the plates. These printed plates will be temporary, as carbon fiber will be used for the final build. The new plates provided the necessary room from the distro board, as well as being able to easily accommodate the flight controller, Odroid, and other sensors.

- **Test flight under manual control**
  The UAV was first tested under manual control within the King Center at SDSMT. The manual flight was successful and validated the build. The UAV was responsive to pilot controls.

- **Autonomous Flight**
  The testing of the autonomous flight validated the capability of the flight controller to provide the hexrotor speed controllers the necessary and correct signals, as well as validating the build of the UAV.

  This objective is also relevant to the following user stories:

  - **U-1 As a user, I want to communicate the waypoints to the UAV:**
    The team successfully sent a series of "missions" to the flight controller including take-off, waypoint navigation goals, and landing. The team used QGroundControl GUI to easily mark points and define the associated action for the UAV.
  - **O-1 As an owner, I want the UAV to autonomously take-off from the landing pad:**
    The mission provided to the flight controller instructed the UAV to take-off and achieve a certain height before executing the next mission. The UAV repeatedly executed these instructions according to parameters provided through the GUI mission setup.
  - **O-2 As an owner, I want the UAV to autonomously navigate through a set of waypoints:**
    The UAV navigated to the waypoints assigned. This process was repeated for a set of waypoints to ensure that navigation of these points was being successfully handled by the flight controller. The team

also observed that despite a flexible frame, caused by the material used in our temporary assembly plates, and some unsecured rotation in one of the rotor arms, the flight controller still successfully navigated through the waypoints.

– **O-3 As an owner, I want the UAV to autonomously return to the location of the landing pad:** The UAV navigated back to the location of the landing area designated. Though this was achieved simply by defining the return as the last waypoint mission that the flight controller will execute. It was also observed that if the user uses the landing mission, the flight controller will determine the direct line to land at that point, which is definitely problematic if the landing area is higher than the altitude of the UAV. It causes the UAV to "skip" along the surface. However, this will not be an issue for our team, as we will not be using the landing mission.
The navigation is very accurate relative to expectations. The same navigation points used for takeoff were used for the final point of navigation and landing. The flight controller consistently set UAV within two feet of the takeoff point. This has caused the team to re-evaluate the needs for a landing pad. The team feels confident that with the GPS, the flight controller will allow the UAV to navigate to a point where an AR tag alone can be used to provide visual guidance to the UAV.

Uncompleted Tasks

Common Development Tasks

- **Setup Hexrotor Simulation in Gazebo** The team has decided to indefinitely postpone development of the specific hexrotor simulation. The time invested on resolving software and hardware conflicts was monopolizing project time.

As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft

- **Landing Algorithm Simulation**
As the team has ceased simulation development, initial testing for the landing algorithm will not occur using the simulation. The proposed method for testing is to review logs generated by our vision guided landing solution while passing the UAV, without props, over the landing AR tag and reviewing the messages passed to the flight controller for descent. The team will specifically test for a controlled landing that will prevent damage to the UAV or the landing pad from the force of impact.

- **Modify/Rewrite implementation as necessary**
Though not completed, progress was made on modifying our implementation of the landing algorithm to only include the AR tag tracking. The tracking tool (fig. A.2) that we have successfully tested will provide us with distance information necessary to correctly judge the distance from our landing target, as well as calculate a safe rate of descent.

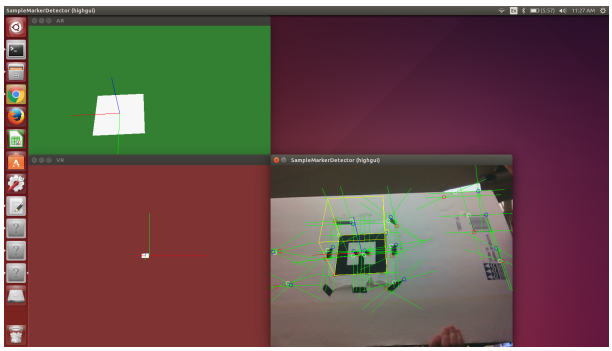


Figure A.1: ALVAR AR Tracker testing with laptop webcam

As an owner, I want the UAV to autonomously land on the landing pad with the correct orientation.

- **Landing Algorithm Simulation**
  As mentioned previously, as our simulation development has ended, our team will utilize the sensors to generate log data to evaluate. The team will specifically test for orientation corrections that will provide the correct alignment on landing.
- **Modify/Rewrite implementation as necessary**
  As mentioned previously, our visual landing approach has changed. We hope to be able to easily modify the existing software to provide information regarding the orientation of the AR tag to use to change the orientation of the UAV as necessary during landing.

Prototype

Our prototype for this sprint is mainly just an assembled hexrotor UAV that is able to follow GPS waypoints. Throughout the assembly process, we did small tests to be sure that we were on the right track. At first, we just kept the UAV in the lab and turned to rotors to verify that they were wired correctly and spinning in the right directions. We then had a manual test flight in the gym, where we discovered that the UAV is quite stable and responsive. Finally, we tested the GPS waypoint navigation with a series of flights. We started small, just having the UAV fly to the end of the driveway at first, eventually working our way up to having the UAV take off, move through a series of waypoints, and return to land on the same location it took off from. Over the course of our testing, we estimate that we will be able to trust the GPS to get us above our landing pad. As stated above, this means that we feel confident that we will be able to trust GPS to get us directly above the landing pad.

We have videos of these GPS tests up on YouTube.

5 Sprint Report #5

Summary

Team Expeditus was unfortunately unable to complete the tasks for sprint 5, but the team was able to make measurable progress towards the goals. After deciding not to pursue simulation, the team members began to focus on message passing and offboard control between the flight controller and the Odroid. After multiple issues with multiple approaches to the AR tag tracking, the team was able to successfully get the AR Track Alvar library up and running.

Team Work

- **Jonathan Dixon:** Worked on AR tracking.
- **Dylan Geyer:** Worked on AR tracking.
- **Christopher Smith:** Worked on message passing and offboard control.
- **Steven Huerta:** Worked on AR tracking.

Uncompleted Tasks

As an owner, I want the UAV to autonomously land on the landing pad without damaging the craft

- **Modify/Rewrite implementation as necessary**
  - **Visual Tracking by use of non-ROS version of AR tracker**
    The team was able to successfully download, install, and run a non-ROS version of ALVAR to track AR Tags. The GUI elements of the AR Tracking program were removed and replaced by printing the quaternian and translation data of the AR Tag to the terminal.

    The team decided not to pursue this any further since we were able to get the AR Track Alvar ROS package working which will save us the time of creating our own ROS wrapper around alvar.
  - **Visual Tracking by use of ROS version of AR tracker**
    AR_TRACK_ALVAR is a package in ROS that provides AR tracking. This, as the name suggests, provides the ability to track AR tags. This is a ROS implementation of the ALVAR tool. Initially, this was the tool that the team wanted to use to estimate the UAV's pose relative to AR tag. This estimation is important, as it would allow the UAV to accurately estimate the location of the landing pad with the correct orientation.

    The team, however, was unable to get this package functioning correctly. The documentation of the package is poor, and after struggling with making this package functional, we abandoned this path. However, after struggling with making the necessary components of ALVAR functional, we again attempted to use the ROS Package which that was available. After spending a full week stumbling through poor documentation, user forums, and tinkering with various launch files, we are now able to use the package.

    This package will now serve as our means to relate the position of the UAV to the landing site, including orientation. As demonstrated in Fig. A.2, this package provides AR tag identification, pose and orientation of the tag. We will use this information to provide our flight controller with necessary move commands to land the UAV.
  - **Offboard Local Control of UAV**
    Offboard control was successfully achieved. After the successful flights using just QgroundControl and the pixhawk we attached the odroid to the uav on stands mounted right above the pixhawk. ROS Jade is installed on the ubuntu lts 14.04 odroid. Mavros is also installed and was able to connect to the pixhawk through USB and not throw any errors of connection.

    Once we achieved this connectivity we wrote an simple offboard flight control node that would send location setpoints of a pose message in ROS composed of a position (x, y, z) and yaw. The first

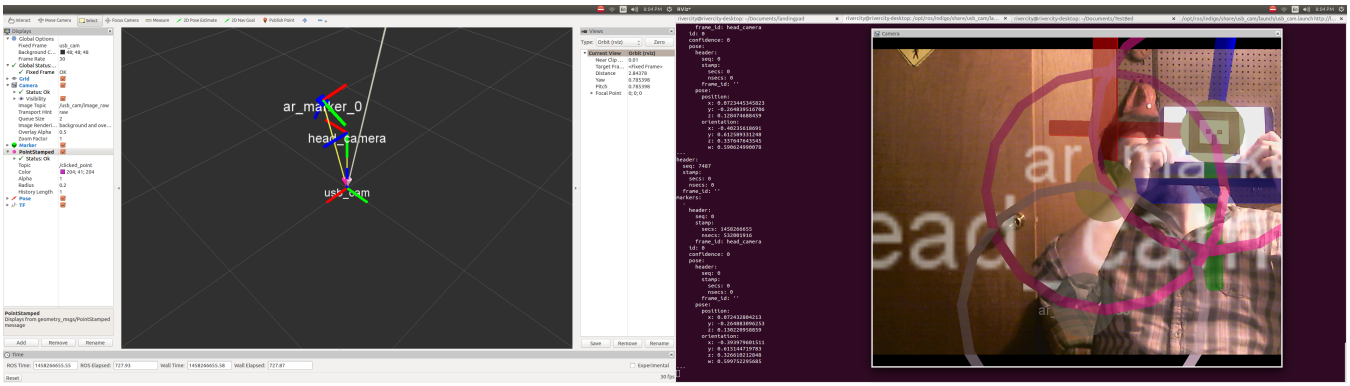


Figure A.2: ALVAR AR Tracker Demo in RViz

message that was tested was the position [0, 0, 0], and 0 yaw. When the drone was switched to offboard mode once setpoint data was being streamed nothing happened as was expected. The motors did not increase in velocity.

The second test was to send a setpoint of [0, 0, 1], and 0 yaw. During this test the motors did increase in velocity significantly as the controller switched the uav from stabilize mode to offboard mode. We switched it back and forth between the two modes making sure all motors were increasing in speed. Once we did that we physically picked it up and moved it up and to the 1 meter mark. Once the setpoint position was either reached or passed the motors did slow down in speed to try and hold that position or lower itself to that position.

All the tests appeared to be working without monitoring the actual positions that were being streamed across the ROS topics. However, once we started looking at the data that was on the topics we noticed that where the uav started the local position topic would drift considerably in location. This issue was caused by being inside and in a basement. The x, y, and z location the pixhawk was reporting was not consistent. On some boot ups it would say it was at roughly [0, 0, -4] but the z value -4 would fluctuate severely by plus or minus 5. The x and y values would hover around the 0 marks usually but occasionally would fluctuate in the plus or minus 20 area. This was when we found that the onboard sensors would not be enough to perform at the fidelity we wanted in an ideal testing environment.

Once a nice day happened we took it outside to make sure that the building was the only severe drift effects on the sensors of the pixhawk. We took it out fifty feet from structures and performed the same tests on the uav as above. There were still errors but this time they were less but was off plus or minus a couple meters in all directions. The yaw appeared to have no issues or if it was off in the decimal places due to picking up the uav ourselves.

Since all the onboard sensors would not provide us the fidelity in control that we wanted on their own we started researching different things to implement localization with the pixhawk sensors and exterior ones like a camera. So far it appears that visual odometry or optitrack are going to be the the best solutions to give us localization when we are above the landing pad. What still needs to be done is sending a pose estimate from the camera to the pixhawk using a kalman filter. We wish to treat the tag on the landing pad as the origin of the uav so that only setpoints that increasingly get it closer to [0, 0, 0] are sent.

As an owner, I want the UAV to autonomously land on the landing pad with the correct orientation.

- **Modify/Rewrite implementation as necessary**
  As mentioned in the previous section, our team has not completed the tasks relating to the landing, which are dependent on:
  - Estimate of Pose

Figure A.3: Pose Estimation by Flight Controller

– Providing the commands to the flight controller to move the UAV to the landing pad.

AR_TRACK_ALVAR will provide estimation of the orientation of the AR tag relative to the camera. This will be used to correct the orientation of the craft. As message passing from and to the flight controller is solved through the use of MavROS, the team will now concentrate its efforts to correct issues offboard movement control. This, as mentioned previously, is the result of insufficient information for the flight controller to estimate its position.

6 Sprint Report ...

B

Industrial Experience and Resumes

1 Resumes

JONATHAN DIXON

3311 Hogan Ct.
Rapid City, SD 57702

jonathan.dixon@mines.sdsmt.edu
605.415.8371

OBJECTIVE

To obtain an internship or full time offer with a high-profile company engaged in Software Development

EDUCATION

Student, South Dakota School of Mines and Technology, 3.143 GPA September 2011-present
Computer Science Major, Expected to Graduate May 2016
Diploma, Rapid City Stevens High School May 2011
Fluent in: C/C++/C#, Python, VB.NET, Java, Assembly Language, QT, Lisp, MySQL, BASH

AWARDS AND RECOGNITION

- Fall Semester Dean's List, SDSM&T 2011
- Phi Eta Sigma Honor Society 2012
- National Honor Society 2010-2011

ACTIVITIES

- Lambda Chi Alpha Fraternity 2013-present
- KTEQ Assistant Station Manager 2012-2014
- SDSM&T Orchestra 2011-2014
- Black Hills Symphony Orchestra 2007-2014

WORK EXPERIENCE

NASA Systems Software Development Intern Fall 2014 - Summer 2015
Kennedy Space Center, Florida

- Development and maintenance of new Launch Control System software
- Test current software, develop new features, address any bugs in previous versions
- Develop graphical user interface test automation suite using Sikuli, Fitnesse, and Jenkins

FAST Enterprises Intern
Oklahoma City, Oklahoma Summer 2014

- Assist with the implementation of the GenTax software for the Oklahoma DMV
- Create automatically generated letters with VB.NET that will be mailed to dealerships

NASA Journey into Space Intern 2013-2014
The Journey Museum, Rapid City

- Assist with youth education programs, including a course on robotics, run and program the planetarium software

Halberstadt's Men's Clothiers 2013-2014

- Salesperson

SDSM&T Foundation Phonathon 2011, 2012

- Call SDSM&T alumni, recorded pledges and donations, kept records

CURRENT PROJECTS

Oculus Rift Quadcopter

- Hobby project to create a quadcopter that can be controlled with the head-tracking from an Oculus Rift
- Currently overcoming hardware issues with the quadcopter itself

Simple C++ Grading Program

- Class Project
- Using a team agile approach, created software to compile and run a directory of simple C++ programs, and compare their output against expected output to give each student a grade.

3203 Ponderosa Place, Rapid City, SD 57702
(605)415-5245 dylan.geyer@mines.sdsmt.edu

Dylan Geyer

Programming Languages: C/C++/C#, Assembly, Java, Visual Basic

Work History:

Software Engineer Intern, Microsoft. (May 2015 – August 2015)

- Created tools to visualize data relationships and gain actionable insights
- Automated time consuming security analyst tasks
- Updated prototype code to meet coding standards

Software Engineer Intern, OEM Solutions. (May 2014 – May 2015)

- Designed Graphical User Interfaces using Visual Basic, Visual Studio 2013
- Created automated testing/calibration systems for seven product lines (C,VB)
- Modified existing firmware to add sensor calibration functionality (Assembly)
- Developed firmware for three new product lines (C)

Academics:

South Dakota School of Mines and Technology, Rapid City, SD

- B.S. in Computer Engineering — expected May 2016
- B.S. in Computer Science — expected May 2016
- **GPA:** 3.74

Activities:

SDSM&T Programming Team — Fall 2013 – Present

Institute for Electronics and Electrical Engineers (IEEE) — (Fall 2011 – present)

- Vice President — Fall 2014 – Spring 2015
- Treasurer — Fall 2013 – Spring 2014
- Freshman Activities Chair — Fall 2012 – Spring 2013

Honors & Awards:

Honors

- Dean's List — Fall 2011 – Present
- Tau Beta Pi – Engineering Honor Society — Fall 2013 – Present
- Eta Kappa Nu – Electrical and Computer Engineering Honor Society — Spring 2014 – Present

Scholarships

- Tech Challenge Scholarship — Fall 2011 – Spring 2013
- John T. Vucurevich Scholarship — Fall 2012 – Spring 2016
- Fawcett Familty CENG/EE Scholarship — Fall 2015 – Spring 2016
- Tim & Laura Pike CSC Scholarship — Fall 2015 – Spring 2016

Christopher Smith

Personal Data

Address: 6850 Suzie Ln, Black Hawk, South Dakota
Phone: (605) 786-6599
Email: chris.sdsmt@gmail.com

Objective: To gain experience in a work environment in a computer science field. Also to be challenged continuosly to improve my critical thinking, analyzing, programming, and math skills.

Education

May 2017 Bachelors of Science in Computer Science
May 2017 Bachelors of Science in Applied and Computational Mathematics
South Dakota School of Mines, Rapid City, SD

Elecvtives: Cryptography, Cybersecurity, Computer Graphics, Natural, and Parallel Computing

Programming Experience

Aug 2015-Present | **Senior Design (Project: UAV Landing Pad)**
- Programming a UAV to autonomously navigate a series of waypoints, takeoff and land autonomusly on a landing platform.
- Landing approches: Visual Homography and Reinfocement Learning

Jan-May 2015 | **Natural Computing (Final Project: Artificial Intelligence)**
- Created an AI to play the board game Puerto Rico using artificial neural networks
- The AI was improved through a genetic algorithm

Nov-Dec 2014 | **Parrallel Computing (Final Project: Image Processing)**
- Created a shared memory image processing library in Qt to benchmark performance
- Fast Fourier Transform, pixel, and mask based image operations were tested

Sept 2011-Present | **SDSMT Robotics Team**
- Working as a team to design and build robots that can autonomously navigate courses for competitions based on input from sensors, GPS and cameras
- Focusing on programming camera interface with Opencv and ROS

Computer Skills

Programming Languages: C++, Python, Java, C#, SQL, Common Lisp
General Knowledge: Linux, BASH, Makefiles, LaTeX, Microsoft Office and VS

Work Experience

July 2012-Present | **Backroom Associate at Target, Rapid City SD**
Responsible for:
- Pulling merchandise down for stocking on the sales floor
- Placing extra merchandise in the appropriate areas in the stockroom

Volunteer Experience

SDSMT Robotics 2012-2015
- Helped boy scouts recieve their Robotics Merit Badge by teaching them about robots (mechanical, sensors, design, and programming)
- Assisted in teaching students programming and design so they could complete their challeges in First Lego League

JROTC 2006-2011
- Cleaned up Liberty Blvd twice a year
- Bought gifts for children with low income families in the school district around Christmass

3624 Reder St. Rapid City, SD 57702 Mobile: 605.728.9043 steven.huerta@gmail.com

Steven Huerta

EDUCATION: Enrolled in **SDSMT Accelerated MS Program**:

- Computational Sciences & Robotics
- Expected Graduation: May 2017

2012-Current **South Dakota School of Mines & Technology** Rapid City, SD

- B.S. in Computer Science
- Expected Graduation: May 2016
- 3.5 GPA

1999-2003 **University of Oregon** Eugene, OR

- B.A. in Sociology, Minor in Japanese
- High Honors from Sociology Department
- 3.5 GPA

COMPUTER LANGUAGES: Proficient with C++, Python. Familiar with C, Java, CLISP, PHP, HTML

OTHER SKILLS: Windows, Linux (Ubuntu, Fedora), MySql, Robot Operating System, GitHub

PROJECTS: Face Detection (C++), Face Detection & Recognition (Python), Planetarium GUI (Java), Bug Tracker (MySql, PHP, Javascript, HTML)

EMPLOYMENT:

Undergraduate Researcher - Robotics **Jun 15 ~ Aug 15** Corvallis, OR
Oregon State University
Conducted research pertaining to increasing adoption and development of Robot Operating System.
Created tools to retrieve data on current ROS users and user activity.
Worked on extending existing tools to observe real-time behavior on ROS implementations.

Response Coordinator
Weston County Public Health **Feb 08 ~ Dec 11** Newcastle, WY
Request, maintain, and exercise equipment, software, and applications.
Develop, provide, coordinate, and track training for organization personnel and partner agencies.

Senior Customer Service Representative
Symantec **Feb 05 ~ Nov 06** Springfield, OR
Provide corporate and small business customers with information regarding software licensing.
Research and troubleshoot software activation issues for customers.

English Instructor
NOVA **Nov 03 ~ Oct 04** Okazaki, Japan
Provide instruction for standardized English proficiency tests.
Evaluate student progress and provide recommendations based on needs and strengths.

Fire Support Specialist
US ARMY **Sep 95 ~ Feb 99** Ft. Bragg, NC
Responsible for coordinating and planning for indirect and support fire weapons.
Responsible for operating and maintaining communication systems.

ACTIVITIES: SDSMT Robotics Club (Aug 2014 ~ Current), currently Outreach Coordinator (Aug 2015)
Foster Parent (Jan 2013 ~ Current)

2 ABET: Industrial Experience Reports

2.1 Jonathan Dixon

NASA Internship (September 2014 - August 2015)

- Year-long internship at the Kennedy Space Center in Cape Canaveral, FL.
- Worked with the Launch Control System team.
- Spent the Fall term developing system tests for the Launch Control System API. Tested over 150 function calls, confirming that the correct status codes were received.
- Spent the spring term cleaning compiler warnings from the Launch Control system. Successfully removed all 1000+ compiler warnings from the C++ code base, and repaired a number of memory leak issues that were being reported by Valgrind.
- Spent the summer term developing a framework for automating the build-deploy-test process of the Launch Control System. Investigated techniques to allow automated GUI tests to be run on Launch Control Center machines on a weekly basis.

FAST Enterprises Internship (May 2014 - September 2014)

- Worked with the FAST team in order to create DMV software for the state of Oklahoma.
- Developed forms that the DMV employees would fill out when registering dealerships.
- Created automatically generated forms that would be mailed to dealerships when it comes time to renew their tags.
- Tested the DMV software during development.

2.2 Dylan Geyer

Microsoft Internship (May 2015 - Aug 2015)

- Created tools to visualize data relationships and gain actionable insights.
- Automated time consuming security analyst tasks.
- Updated prototype code to meet coding standards.

OEM Solutions Internship (May 2014 - Current)

- Used Visual Basic and .NET framework to interface PC with embedded controllers.
- Created automated testing/calibration systems for seven different heat controllers.
- Added features to existing heat controller firmware (Assembly).
- Wrote firmware for three new heater controllers (C).

2.3 Christopher Smith

–No Industrial Experience–

2.4 Steven Huerta

–No Industrial Experience–