UAV Landingpad

Senior Design Final Documentation

Team Name

Steven Huerta Christopher Smith Dylan Geyer Jonathan Dixon

December 11, 2015

Contents

Overview Statements

0.1 Mission Statement

Our project goal is to produce software to enable a UAV to take-off, navigate through a series of waypoints, and return to the landing pad. The autonomous landing of the craft will result in the UAV successfully land within +/- .1 meters and with the correct orientation of the craft, so as to allow for the recharging of the UAV by the landing pad station.

0.2 Elevator Pitch

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. Our goal is to provide or enable this autonomy for the UAV.

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 $\pm .1$m 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 $\pm .1$m of the center of the landing pad with $\pm 15°$ 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 ± 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 $\pm 15°$.

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).

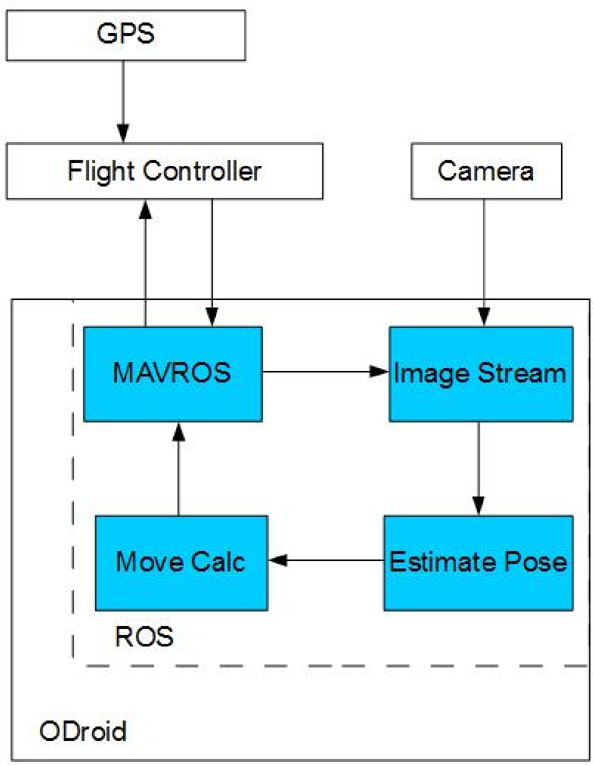


Figure 1.1: Communication between Flight Controller and ODroid

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

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 addition 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 $\pm .1$m 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 orientation

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 $\pm 15°$ 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 autonomouse takeoff and waypoint navigation system is seperate 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 recieve 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 Recieve 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
            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
            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

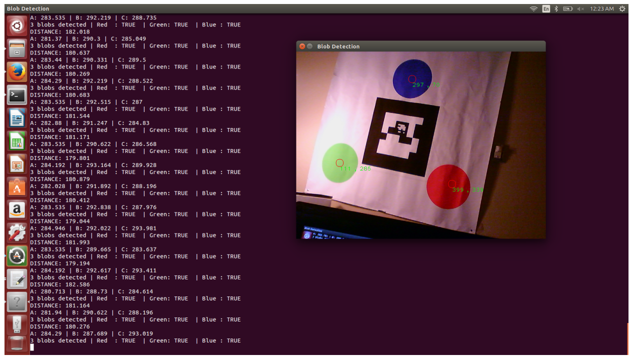
- Ubuntu 14.04 LTS
- Python
- OpenCV

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.

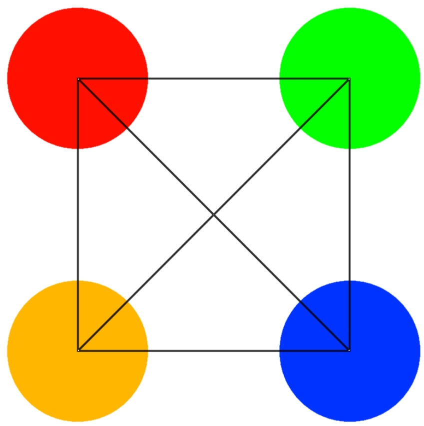
4.3.1.d Design Details

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.

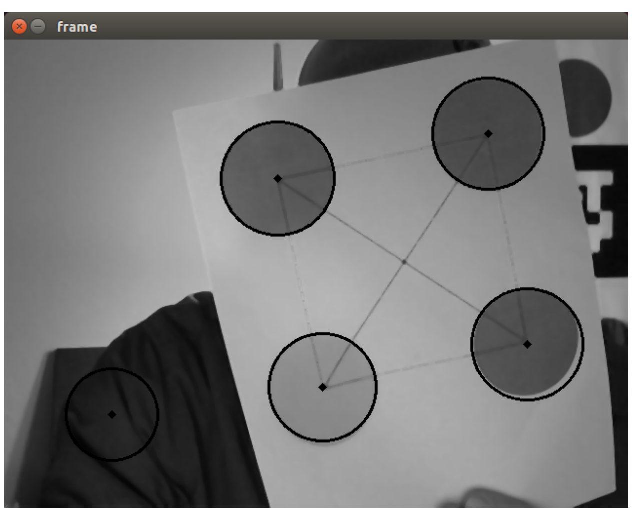


It may be hard to notice, but the text that is currently being displayed in the terminal window shows that there are three blobs detected, exactly one for each color. It is also displaying the range to target, which is correct as far as we can tell.

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

4.3.2.b Component Overview

4.3.2.c Phase Overview

4.3.2.d Design Details

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.

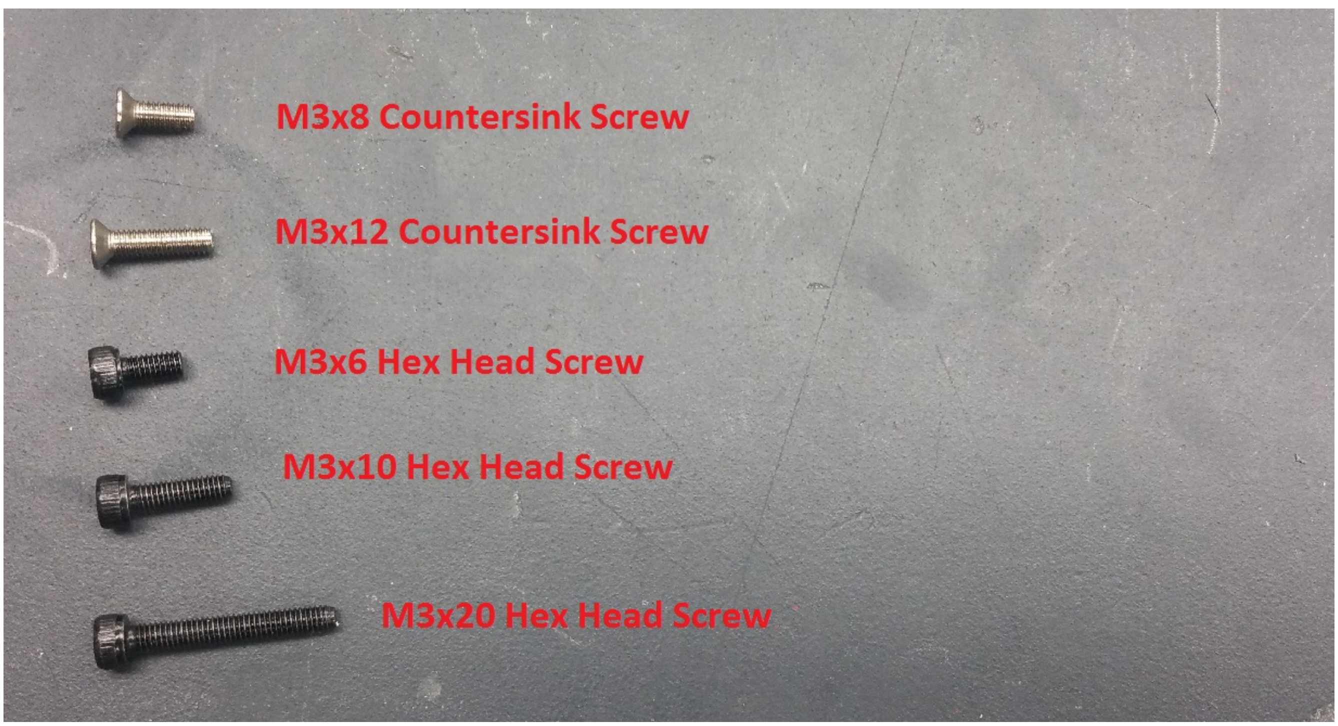


Figure 1: Bolt definitions.

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.

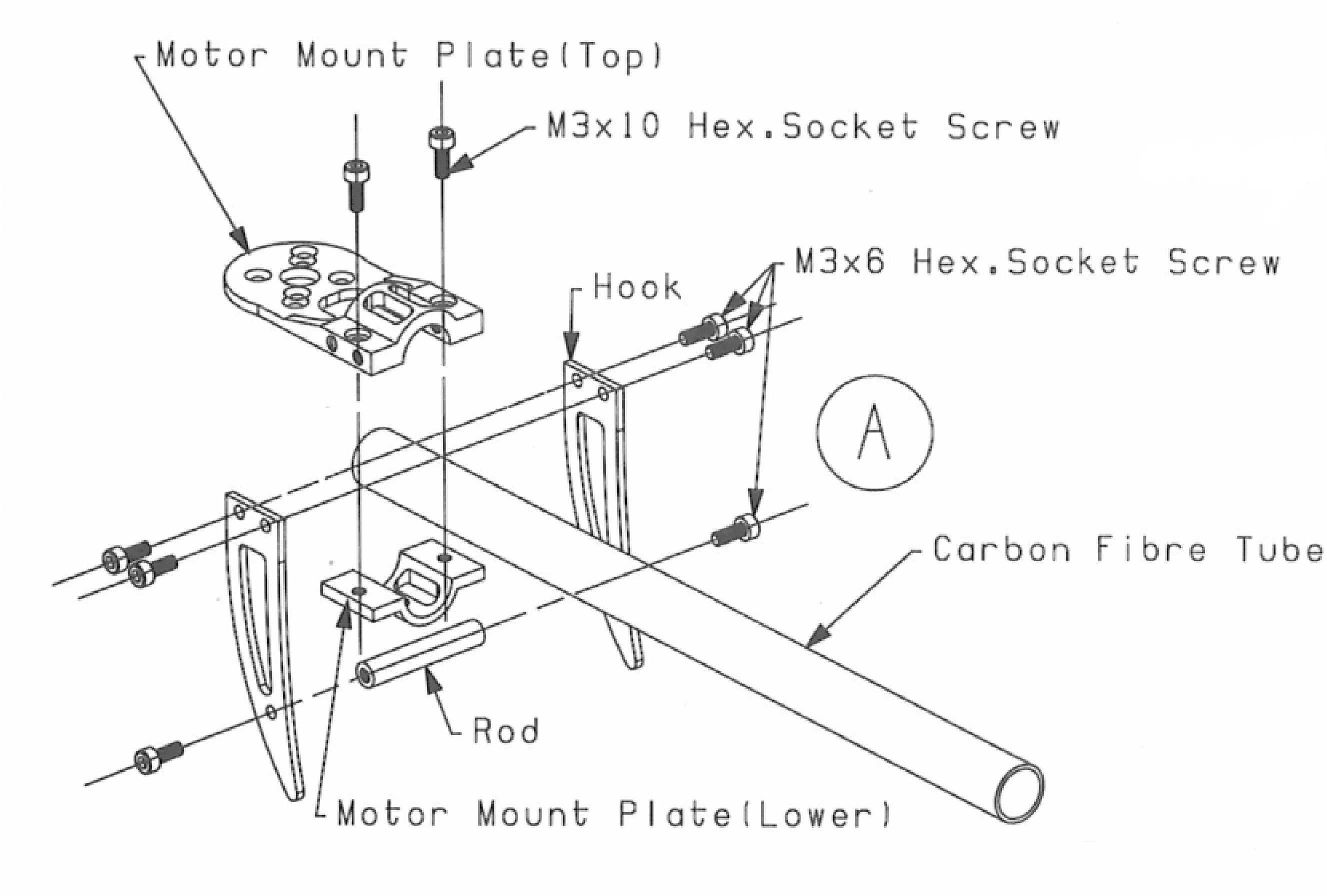


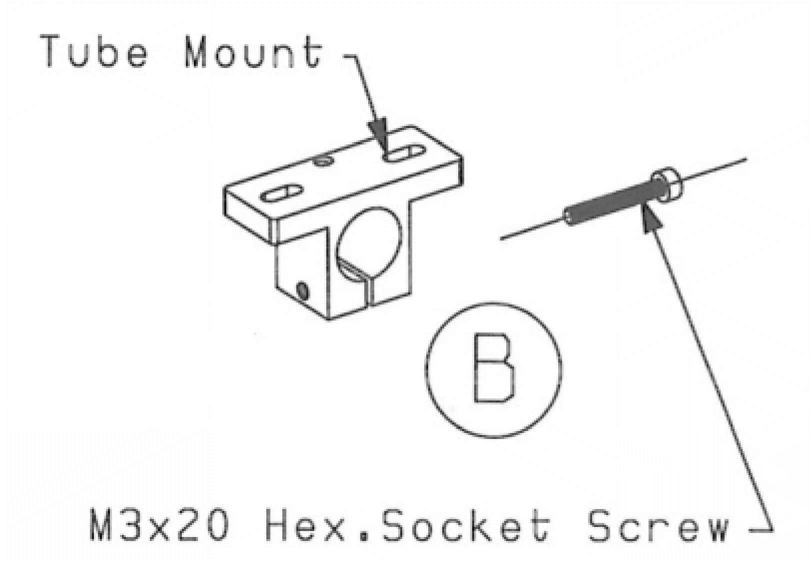
Figure 3: Instructions for assembling motor mount.

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.

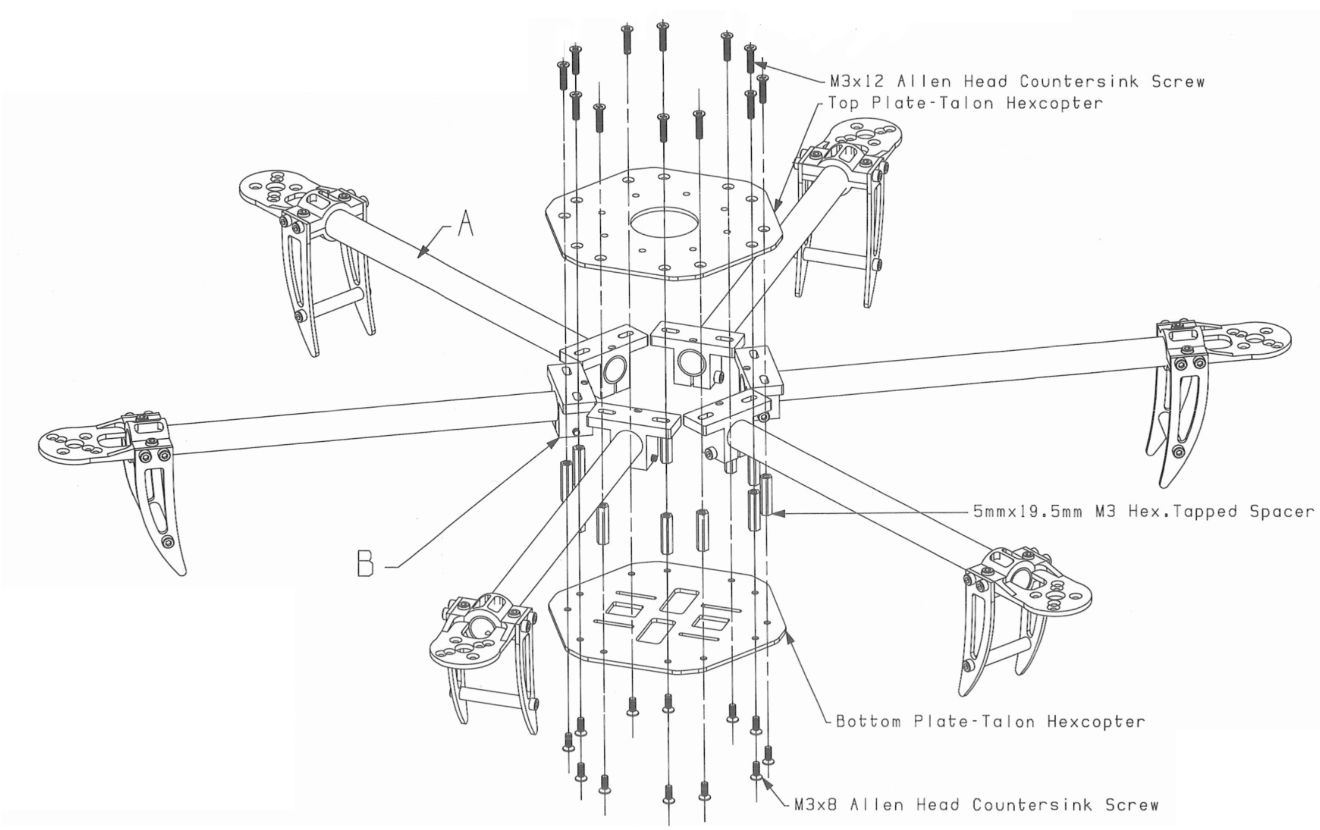


Figure 5: Instructions for Top/Bottom plates.

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.

SDSMT SENIOR DESIGN
SOFTWARE DEVELOPMENT AGREEMENT

This Software Development Agreement (hereafter referred to as **Agreement**) is made between the SDSMT Computer Science Senior Design Team, **Expeditus**, consisting of team members: **Jonathan Dixon, Dylan Geyer, Steven Huerta, Christopher Smith** (hereafter referred to as **Senior Design Team**), and Sponsor: **Larry Pyeatt** (hereafter referred to as **Sponsor**), with address: **501 E. Saint Joseph Street Rapid City, SD 57701**
.

1 RECITALS

1. **Sponsor** desires the **Senior Design Team** to develop software to enable the autonomous take-off, navigation, and landing of a UAV.
2. **Sponsor** desires the **Senior Design Team** to develop UAV autonomous landing to include correct orientation and position.
3. **Senior Design Team** is willing to develop such software.

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

2 EFFECTIVE DATE

Agreement shall be effective as of September 17, 2015.

3 DEFINITIONS

1. "Software" shall mean the computer programs in machine readable object code form and any subsequent error corrections or updates supplied to the **Sponsor** by the **Senior Design Team** pursuant to **Agreement**.
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 the **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 the **Sponsor** all of the Deliverables for that Milestone in accordance with the project plan and the **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. The **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 **Senior Design 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 **Senior Design 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 **Senior Design Team**; and resolve any conflicts that may arise.

5 COMPENSATION

No compensation will occur in this project

6 CONSULTATION AND REPORTS

1. **Sponsor**'s designated representative for consultation and communications with the **Team Lead** shall be the **Sponsor** or such other person as **Sponsor** may from time to time designate to the **Team Lead**.
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 the **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 the **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 the **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 the **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

Sponsor holds claim to IP.

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**.
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 IN THE **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 the **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. The **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. The **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 the **Agreement** and shall fail to remedy such default within thirty (30) days after receipt of written notice thereof, the **Agreement** shall terminate upon expiration of the thirty (30) day period.
4. Any provisions of the **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 the **Agreement** for all purposes.

14 GENERAL

1. The **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. The **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

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

Sponsor

10-1-15	
Larry Pyeatt	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 way-points.

- **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 ...

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

2.4 Steven Huerta

–No Industrial Experience–