UAV Landingpad

Senior Design Final Documentation

Team Expeditus

Steven Huerta Christopher Smith Johnathan Dixon Dylan Geyer

April 28, 2016

Contents

List of Figures

Overview Statements

0.1 Mission Statement

The purpose of this project is to develop a method to enable a small multi-rotor UAV to autonomously take-off, navigate to assigned waypoints, and land accurately on a landing pad to recharge its battery.

0.2 Elevator Pitch

Small multi-rotor UAVs provide the ability to quickly survey a large area over difficult terrain, and remain a small investment relative to other larger UAVs. The downside is the short battery life, which can be overcome by implementing an autonomous landing method that will land accurately and with the correct orientation so as to make recharging possible. Our team is working towards that autonomy so that small UAVs can be deployed again and again from a recharging station.

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).

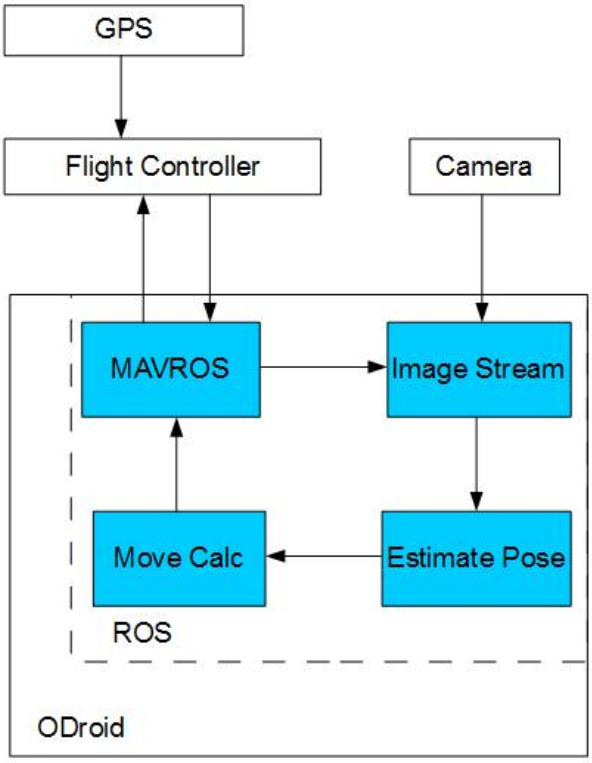


Figure 1.1: Communication between Flight Controller and ODroid

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 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 $\pm$.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

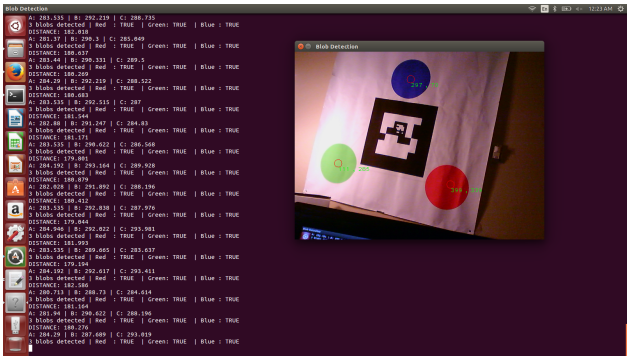
- Ubuntu 14.04 LTS
- Python
- OpenCV

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.

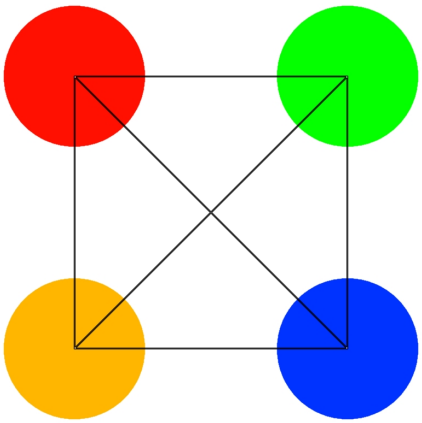
4.3.1.d Design Details

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.

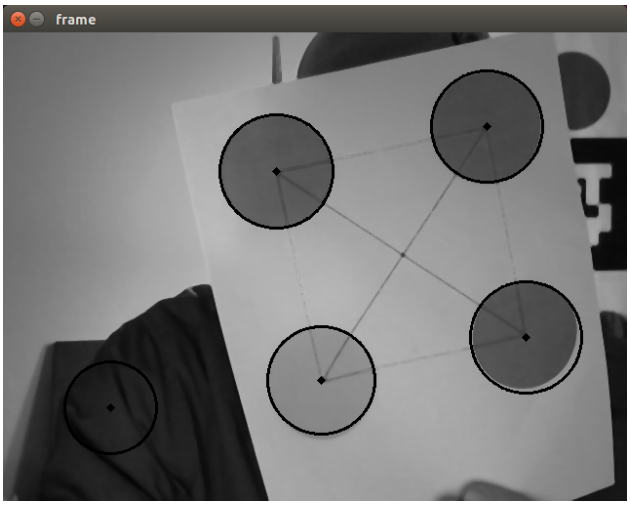


It may be hard to notice, but the text that is currently being displayed in the terminal window shows that there are three blobs detected, exactly one for each color. It is also displaying the range to target, which is correct as far as we can tell.

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 $\epsilon : a \leftarrow$ 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 $\epsilon : a' \leftarrow$ 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

4.3.3 Legacy Code

The purpose of this section is to document the code from previous years that the team attempted to use for the vision portion of the project. While the team did eventually end up deciding not to take this approach, which

involves blob detection, a lot was learned about the process, and if nothing else, we were required to think about the problem of vision much more thoroughly. The code is split up into multiple source files, each file having its own purpose. While this topic is indeed briefly touched upon in the prototypes section of this document, this section will go into much more depth in examining the code, and describing how it functions. The files used are:

- Camera.h/cpp
- Coord.h/cpp
- LED.h/cpp
- Triangle.h/cpp
- tracker.cpp

4.3.3.a tracker.cpp

Overview

The file tracker.cpp contains the main control structure for the blob detection and tracking used by previous teams. Julian Brackins (the original author of the file) provided the following description of the file, which is included in the listing of the contents of the file as well:

> This software should hopefully soon be adapted into a ROS publisher that is capable of sending messages to our UAV to indicate the craft's distance from our landing pad situated on our Unmanned Ground Vechicle (UGV). This is done using three equidistant coloured dots, or "blobs" situated on our landing pad. Utilizing OpenCv's libraries for tracking specific colours, these three blobs are recognized by the software's camera feed, with corresponding coordinates in relation to the camera feed's image plane. The software compares these values to the stored points read in from the configuration file in order to compare the observed size of the objects being tracked to the known size of these objects that has been previously calibrated and written to a config file.

```
/*****************************************************************//**
 * @file tracker.cpp
 *
 * @brief SOURCE - Blob detection software for determining distance
 *
 *
 * @section course_section CSC 465
 *
 * @author Julian Brackins
 *
 * @date December March, 10
 *
 * @par Professor:
 *         Dr. Jeff McGough
 *
 * @par Course:
 *         CSC 465 - M001 - Tues / Thurs - 9:00am
 *
 * @par Location:
 *         McLaury - 304
 *
 * @section program_section Program Information
 *
 * @details
 * This software should hopefully soon be adapted into a ROS publisher that is
```

```
* capable of sending messages to our UAV to indicate the craft's distance
    from
* our landing pad situated on our Unmanned Ground Vechicle (UGV). This is
    done
* using three equidistant coloured dots, or "blobs" situated on our landing
    pad.
* Utilizing OpenCv's libraries for tracking specific colours, these three
    blobs
* are recognized by the software's camera feed, with corresponding
    coordinates
* in relation to the camera feed's image plane. The software compares these
    values
* to the stored points read in from the configuration file in order to
    compare the
* observed size of the objects being tracked to the known size of these
    objects that
* has been previously calibrated and written to a config file.
*
* @section compile_section Compiling and Usage
*
* @par Compiling Instructions:
*      (Linux) - catkin_make
*
* @par Usage:
@verbatim
./tracker -[options] [configfile]
@endverbatim
*
* @section todo_bugs_modification_section Todo, Bugs, and Modifications
*
* @par Modifications and Development Timeline:
@verbatim
Date                 Modification
------------------
    ------------------------------------------------------------------------
* October  21, 2014    Uploading qr reader with distance algorithm
*
* November  4, 2014    Adding led stuff
*
* February  2, 2015    Set up LED tracking code
*
* February  8, 2015    Created Coordinates and Triangle classes, need to
    start
*                      working on distance algorithm.
*
* February 16, 2015    Triangle is set up and printing, but lawOfCosines
*                      calculation isn't working right.
*
* February 21, 2015    Cleaning out led directory's cmake files so that code
*                      builds more easily on other machines
*
* February 22, 2015    Distance approximations now working through blob
*                      detection, Now just have to comment it all.... -.-
*
* February 28, 2015    A few tweaks
```

```
 *
 * March 10, 2015       Retooled tracker system so that it takes in command
    arguments
 *                      properly. run <tracker -h> to see the command line
    options.
 *                      Tracker can now determine distance from just two blobs
    . Need
 *                      to implement config files still
 *
 @endverbatim
 *
 ********************************************************************************
    */

/*
   ********************************************************************************

 *
 * INCLUDE
 *
 ********************************************************************************
    */


#include <sstream>
#include <string>
#include <iostream>
#include <vector>
#include <unistd.h>
#include "opencv2/highgui/highgui.hpp"
#include "opencv/cv.h"
#include "LED.h"
#include "Coord.h"
#include "Triangle.h"
#include "Camera.h"
#include <math.h>

using namespace cv;
using namespace std;

//default capture width and height
const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;

//max number of objects to be detected in frame
const int MAX_OBJS=50;

//minimum and maximum object area
const int MIN_OBJ_AREA = 15*15;
const int MAX_OBJ_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;
//names that will appear at the top of each window
const string winName = "Blob Detection";


int _arg_cam = 0;
```

```
//Global flags for command arguments
bool _arg_calibrate = false;
bool _arg_distance  = false;
bool _arg_points    = false;
bool _arg_booleans  = false;
bool _arg_ros       = false;

string _arg_conf     = "hsv.conf";

string intToString(int number);

Coord drawObject(LED light, Mat &frame);
void drawObject(vector<LED> lights, Mat &frame);
void trackFilteredObject(Mat threshold,Mat HSV, Mat &cameraFeed);
Coord trackFilteredObject(LED lights,Mat threshold,Mat HSV, Mat &cameraFeed);
void morpher(Mat &threshold);
ostream& operator<<(ostream& os, Coord& cd);
void Process_Arguments(int argc, char **argv);


/*
    ********************************************************************************

 * @author Julian Brackins
 *
 * @par Description:
 * Set up infinite loop for program. Sorry for using main() for my whole prog
    ..
 *
 * @param[in] argc - count of args.
 * @param[in] argv - arguments themselves.
 *
 ********************************************************************************
    */
int main(int argc, char* argv[])
{
    //true to set calibration

    Process_Arguments( argc, argv);


   //Matrix to store each frame of the webcam feed
   Mat cameraFeed;
   Mat threshold;
   Mat HSV;


   //video capture object to acquire webcam feed
   VideoCapture capture;
   //open capture object at location one (default location for usb cam)

       capture.open(_arg_cam);


   //set height and width of capture frame
```

```
capture.set(CV_CAP_PROP_FRAME_WIDTH,FRAME_WIDTH);
capture.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);

//infinite loop where all operations occur.
while(1)
{
   //store image to matrix
   capture.read(cameraFeed);

   //convert frame from BGR to HSV colorspace
   cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);

    //Create LEDs
    LED green("green"), blue("blue"), red("red");

    //Create Triangle
    Triangle camTri;

    //Create the coordinate sets for all three blobs
     Coord p1, p2, p3;

     //Set up Camera class that calculates distance based on focal length &
     //observed object distance.
     Camera distCam;

    //Set up Red Tracker, send coordinates to p1
    cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
    inRange(HSV,red.getHSVmin(),red.getHSVmax(),threshold);
    morpher(threshold);
    p1 = trackFilteredObject(red,threshold,HSV,cameraFeed);

    //Set up Green Tracker, send coordinates to p2
    cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
    inRange(HSV,green.getHSVmin(),green.getHSVmax(),threshold);
    morpher(threshold);
    p2 = trackFilteredObject(green,threshold,HSV,cameraFeed);

    //Set up Blue Tracker, send coordinates to p3
    cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
    inRange(HSV,blue.getHSVmin(),blue.getHSVmax(),threshold);
    morpher(threshold);
    p3 = trackFilteredObject(blue,threshold,HSV,cameraFeed);

     //Display Image Feed in window
   imshow(winName,cameraFeed);

     //Set up the three sides of the Triangle
     if(p2.isTracking() && p3.isTracking())
         camTri.setSide("A", p2,p3);
     if(p3.isTracking() && p1.isTracking())
         camTri.setSide("B", p3,p1);
     if(p1.isTracking() && p2.isTracking())
         camTri.setSide("C", p1,p2);

     //Set the A,B,C sides of the triangle in the Camera Class
```

```
        distCam.setA( camTri.getSide("A") );
        distCam.setB( camTri.getSide("B") );
        distCam.setC( camTri.getSide("C") );

        distCam.resetBlobs();
        if(p1.isTracking())
            distCam.incBlobs();
        if(p2.isTracking())
            distCam.incBlobs();
        if(p3.isTracking())
            distCam.incBlobs();
        //Print out distance to stdout
        if(_arg_calibrate)
        {

            //cout << "A: " << dist(p2,p3) << " | ";
            ///cout << "B: " << dist(p3,p1) << " | ";
            //cout << "C: " << dist(p1,p2) << endl;
        }
        if(_arg_distance)
        {
            cout << "DISTANCE: " << distCam.getDist() << endl;
        }
        if(_arg_points)
        {
            cout <<   "A: " << camTri.getSide("A") << " | B: " << camTri.
               getSide("B") << " | C: " << camTri.getSide("C") << endl;
        }
        if(_arg_booleans)
        {
            cout << distCam.blobTotal() << " blobs detected | ";
            cout << "Red  : " << p1.printTracking() << " | ";
            cout << "Green: " << p2.printTracking() << " | ";
            cout << "Blue : " << p3.printTracking() << endl;
        }


      //delay 30ms so that screen can refresh.
      //image will not appear without this waitKey() command
        if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc'
           key is pressed, break loop
        {
            break;
        }
    }
    return 0;
}

/*
    ********************************************************************************

 * @author Julian Brackins
 *
 * @par Description:
 * Convert integers to strings lmaoooo.
```

```
 *
 * @param[in] number - an integer lmfaoo.
 *
 * @returns that number as a STRING yo
 *
 ********************************************************************************
    */
string intToString(int number)
{
   std::stringstream ss;
   ss << number;
   return ss.str();
}

/*
   ********************************************************************************

 * @author Julian Brackins
 *
 * @par Description:
 * Convert integers to strings lmaoooo.
 *
 * @param[in] light - an led light being tracked
 * @param[in] frame - image matrix
 *
 * @returns Set of coordinates in the Coord class structure, giving you (x,y)
 *
 ********************************************************************************
    */
Coord drawObject(LED light, Mat &frame)
{
    //Draw a circle around the object being tracked

    cv::circle( frame, cv::Point( light.getX(),light.getY() ), 10, cv::Scalar
       (0,0,255));
    cv::putText(frame, intToString( light.getX() ) + " , " + intToString(
       light.getY() ),
                cv::Point(light.getX(), light.getY() + 20 ), 1, 1, cv::Scalar
                   (0,255,0));
    Coord point;
    point.setX(light.getX());
    point.setY(light.getY());
    return point;
}

/*
   ********************************************************************************

 * @author Julian Brackins
 *
 * @par Description:
 * After the images have been filtered properly, this function
 * handles determining if an image contains a blob that fits in the HSV range
 * of the colour being tracked.
 *
```

```
 * @param[in] light - an led light being tracked
 * @param[in] threshold - thresholded image matrix
 * @param[in] HSV - Matrix containing HSV values
 * @param[in] cameraFeed - Matrix containing camera feed
 *
 * @returns Set of coordinates in the Coord class structure, giving you (x,y)
 *
 ************************************************************************
    */
Coord trackFilteredObject(LED lights,Mat threshold,Mat HSV, Mat &cameraFeed)
{
    //Maybe change all this so that It doesn't refer to everything as lights
        ...
   LED blob;

   Coord coordinates;

   Mat tempMatrix;
   threshold.copyTo(tempMatrix);

   //findContours params
   vector<vector<Point>> contours;
   vector<Vec4i> hierarchy;

    //find the contours of the image using openCV
   findContours( tempMatrix, contours, hierarchy,CV_RETR_CCOMP,
      CV_CHAIN_APPROX_SIMPLE );

   //use moments method to find our filtered object
   double refArea = 0;
   bool objectFound = false;

   //Zero out the coordinates
   coordinates.setX(0);
   coordinates.setY(0);

   if (hierarchy.size() > 0)
   {
      int numObjects = hierarchy.size();
      //The filter becomes too noisy if the numObjects is too great...
      if(numObjects < MAX_OBJS)
      {
         for (int i = 0; i >= 0; i = hierarchy[i][0])
         {

            Moments moment = moments((cv::Mat)contours[i]);
            double area = moment.m00;

            //if the area is less than 20 px by 20px then it is probably just
               noise
            //if the area is the same as the 3/2 of the image size, probably
               just a bad filter
            //we only want the object with the largest area so we safe a
               reference area each
            //iteration and compare it to the area in the next iteration.
```

```
            if(area>MIN_OBJ_AREA)
            {
               blob.setX(moment.m10/area);
               blob.setY(moment.m01/area);
               blob.setColour(lights.getColour());
               blob.setText(lights.getText());

               //Push onto vector if allowing multiples of one colour
               //blobvec.push_back(blob);
               objectFound = true;
            }
            else objectFound = false;
         }
         //let user know you found an object
         if(objectFound ==true)
         {
            //draw object location on screen
            //use vector version of drawObject if allowing multiples
            coordinates = drawObject(blob,cameraFeed);
             coordinates.setTracking(true);
         }
         else
         {
             coordinates.setTracking(false);
          }
      }
      else putText(cameraFeed,"Adjust Filter, too much noise.",Point(0,50)
          ,1,2,Scalar(0,0,255),2);
   }
   return coordinates;
}

/*
    ********************************************************************************

 * @author Julian Brackins
 *
 * @par Description:
 * Morphing function to properly erod and dilate the dense array to enhance
 * image visibility.
 *
 * @param[in] threshold - thresholded image matrix
 *
 ********************************************************************************
    */
void morpher(Mat &threshold)
{
    //Erode and dilate the dense array to make the object clearly visible

    //Construct the erode
   Mat erodeElement  = getStructuringElement( MORPH_RECT, Size(3,3));
   Mat dilateElement = getStructuringElement( MORPH_RECT, Size(8,8));

   erode(threshold, threshold, erodeElement);
   erode(threshold, threshold, erodeElement);
```

```
   dilate(threshold, threshold, dilateElement);
   dilate(threshold, threshold, dilateElement);
}

/*
    ********************************************************************************

 * @author Julian Brackins
 *
 * @par Description:
 * Overloaded output stream for the Coord class. Might as well toss this
 * eventually because I'll most likely be switching to printf anyways... hehe.
 *
 * @param[in] os - outstream.
 * @param[in] cd - representatoin of Coord class indicating how output works
 *
 * @returns os - output stream
 *
 ********************************************************************************
    */
ostream& operator<<(ostream& os,  Coord& cd)
{
    //Overloaded Coord << operator
    if (cd.getX() == 0 || cd.getY() == 0)
        os << "( " << "N/A" << "," << "N/A" << " )";
    else
        os << "( " << cd.getX() << "," << cd.getY() << " )";
    return os;
}

/*
    ********************************************************************************

 * @author Julian Brackins
 *
 * @par Description:
 * Super sophisticated arguments processor that I made back at NASA woo.
 * Handles a bunch of different option flags brought in through command line
 * arguments. The first command argument is the program name (./tracker), the
 * second argument is the list of option flags, and the third option is the
 * configuration file being read in. Each flag that is read in will modify a
 * global flag that will determine how the program is executed.
 *
 * @param[in] argc - count of args.
 * @param[in] argv - arguments themselves.
 *
 ********************************************************************************
    */
void Process_Arguments(int argc, char **argv)
{
    int c;
    printf("CVTracker: argc=%d, argv=%s\n", argc-1, argv[1]);
    //tokenize each param and check it individually
```

```
    while ((c = getopt(argc, argv, "01234cdpbrh")) != -1)
    {
        switch (c)
        {
            case '0':
                _arg_cam = 0;
                printf("CVTracker: using camera 0 (-0)\n");
                break;
            case '1':
                _arg_cam = 1;
                printf("CVTracker: using camera 1 (-1)\n");
                break;
            case '2':
                _arg_cam = 2;
                printf("CVTracker: using camera 2 (-2)\n");
                break;
            case '3':
                _arg_cam = 3;
                printf("CVTracker: using camera 3 (-3)\n");
                break;
            case '4':
                _arg_cam = 4;
                printf("CVTracker: using camera 4 (-4)\n");
                break;
            case 'c':
                _arg_calibrate = true;
                printf("CVTracker: running calibration mode (-c)\n");
                break;
            case 'd':
                _arg_distance = true;
                printf("CVTracker: printing Object Distance to stdout (-d)\n")
                    ;
                break;
            case 'p':
                _arg_points = true;
                printf("CVTracker: printing RGB point coordinates to stdout (-
                    p)\n");
                break;
            case 'b':
                _arg_booleans = true;
                printf("CVTracker: printing blob tracking booleans (-b)\n");
                break;
            case 'r':
                _arg_ros = true;
                printf("CVTracker: publishing to ROS (-r)\n");
                break;
            case 'h':
                printf("CVTracker: help\n");
                printf("$ %s [-options] [logfile] \n", argv[0]);
                printf("    Please place a - then however many of the
                    following [option] values:\n");
                printf("    -#  select which camera to use ( 0 - 4, 0 is
                    default).\n");
                printf("    -c  run calibration mode.\n");
                printf("    -d  ouput object distance to stdout\n");
```

```
                printf("    -p  print RGB point coordinates to stdout\n");
                printf("    -b  print blob tracking booleans (true=tracking /
                    false=not tracking)\n");
                printf("    -r  publish info to ROS\n");
                printf("    -h  display help\n");
                printf("    If no [logfile] specified then the default\n");
                printf("    \"hsv.conf\" will be used.\n");
                exit(0);
                break;
            default:
                //gStandalone = 0;
                printf("CVTracker: ignoring option -%c\n", c);
                break;
        }
    }
    /**
     * Handle the configuration file here.
     **/
        if( argc > 2 )
        {
            _arg_conf = argv[2];
            printf("CVTracker: reading HSV & focal length settings from
                configuration file: \"%s\"\n", argv[2]);
        }
        else
        {
            printf("CVTracker: reading HSV & focal length settings from
                default file: \"hsv.conf\"\n");
        }
}
```

4.3.3.b Camera.h/cpp

Overview

The Camera class is used to take an image feed that has three colored blobs detected on it, and calculate the distance to the target, based on the known size of the target. Within Camera.h, you can see that there are a couple of options for the paper size, and these will need to be changed at the time of compilation so that the system will be able to accurately calculate the distance to the target. The method used to calculate the distance to the target is to simply take some known distance to the target that has a pixel width associated with it, and calculate the focal length for the camera. Then, we just divide the focal length by the pixel distance on our measurement and multiply that result by the known width of the target in order to get the new distance.

$$focal\_length = pixel\_width * known\_dist\_to\_target / known\_width\_of\_target$$

$$measured\_dist\_to\_target = known\_width\_of\_target * focal\_length / pixel\_width$$

Camera.h

```
#ifndef _CAMERA_H_
#define _CAMERA_H_

#include <string>
#include <iostream>
```

```
class Camera
{

public:
   Camera();
   ~Camera();


    void setFocaLength( );
    double getFocalLength() { return focal_length; }
    double getDist( );
    void setA( double val ) { curr_a = val ; }
    void setB( double val ) { curr_b = val ; }
    void setC( double val ) { curr_c = val ; }
    double getA() { return curr_a; }
    double getB() { return curr_b; }
    double getC() { return curr_c; }
    void resetBlobs();
    void incBlobs();
    double blobTotal();
private:
    //double that keeps track of how many
    //blobs that are being detected
    double tracked_blobs;
    double focal_length;

    //SMALL PAPER
    //const double known_obj_width = 5.5;
    //const double known_obj_dist  = 27.0;

    //LARGE PAPER
    const double known_obj_width = 11.5;
    const double known_obj_dist  = 30.5;
    double dist;

    //Original Pixel Widths

    //SMALL PAPER
    //const double orig_a = 141.00;
    //const double orig_b = 139.87;
    //const double orig_c = 142.68;

    //LARGE PAPER
    const double orig_a = 273.827;
    const double orig_b = 276.123;
    const double orig_c = 270.017;

    //Observed Pixel Widths
    double curr_a;
    double curr_b;
    double curr_c;


};
```

```
#endif
```

Camera.cpp

```
#include "Camera.h"


Camera::Camera()
{
    setFocaLength( );
    Camera::resetBlobs();
}

Camera::~Camera()
{
}

void Camera::setFocaLength( )
{
    double avg_pix = ( orig_a + orig_b + orig_c ) / 3.0;
    focal_length = avg_pix * ( known_obj_dist / known_obj_width );
}

double Camera::getDist( )
{
    double avg_val = 1.0;
    if( blobTotal() >= 3 )
        avg_val = 3.0;
    else
        avg_val = 1.0;
    double avg_pix = ( curr_a + curr_b + curr_c ) / avg_val;
    dist = known_obj_width * ( focal_length / avg_pix );
    return dist;
}


void Camera::resetBlobs( )
{
    tracked_blobs = 0;
}

void Camera::incBlobs( )
{
    tracked_blobs += 1;
}

double Camera::blobTotal()
{
    return tracked_blobs;
}
```

4.3.3.c Coord.h/cpp

Overview

The Coord class is used to provide coordinate functionality, which is to say that it will be used by later classes to

allow them to store coordinates.

Coord.h

```
#ifndef _COORD_H_
#define _COORD_H_

#include <string>

class Coord
{
public:
   Coord();
   ~Coord();



   int getX();
   void setX(int val);

   int getY();
   void setY(int val);

   bool isTracking();
   void setTracking(bool val);
   std::string printTracking();

private:
    bool tracking;
   int  xPos, yPos;

};

#endif
```

Coord.cpp

```
#include "Coord.h"


Coord::Coord()
{
    setX(0);
    setY(0);
    setTracking(false);
}


Coord::~Coord()
{
}


int Coord::getX()
{
   return Coord::xPos;
}
```

```
void Coord::setX(int val)
{
   Coord::xPos = val;
}

int Coord::getY()
{
   return Coord::yPos;
}

void Coord::setY(int val)
{
   Coord::yPos = val;
}


bool Coord::isTracking()
{
    return Coord::tracking;
}

std::string Coord::printTracking()
{
    if(Coord::tracking)
        return "true";
    else
        return "false";
}

void Coord::setTracking(bool val)
{
    Coord::tracking = val;
}
```

4.3.3.d Triangle.h/cpp

Overview

The Triangle class is used to allow the program to store a set of three coordinates that make up the triangle of the blobs on the target. The class stores the coordinates of each vertex of the triangle, along with the lengths of each triangle edge. Additionally, it will store the angle between the edges, that can be used to see how "straight on" the image has been taken from.

Triangle.h

```
#ifndef _TRIANGLE_H_
#define _TRIANGLE_H_

#include <string>
#include <iostream>
#include "Coord.h"
#include "math.h"
#define PI 3.14159265

class Triangle
```

```
{
public:
   Triangle();
   Triangle(Coord c1, Coord c2, Coord c3);
   ~Triangle();


   void setPoint(std::string pointName, Coord point);
   void setSide(std::string sideName, Coord firstPoint, Coord secondPoint);
   double calcSide(Coord P1, Coord P2);
   Coord getPoint(std::string pointName);
   double getSide(std::string sideName);

   //find all angles
    void lawOfCosines();
    double radToDeg(double radVal);

    void printTriangle();

private:

    //Reminder:
    //sideA = dist(pointB, pointC);
    //sideB = dist(pointA, pointC);
    //sideC = dist(pointA, pointB);

    //Angle A = angle at pointA,
    //Angle B = angle at pointB,
    //Angle C = angle at pointC,
    double sideA, sideB, sideC;
   Coord pointA, pointB, pointC;
    double angleA, angleB, angleC;


};

#endif
```

Triangle.cpp

```
#include "Triangle.h"


Triangle::Triangle()
{
    sideA = 0.0;
    sideB = 0.0;
    sideC = 0.0;
}


Triangle::Triangle(Coord c1, Coord c2, Coord c3)
{
    pointA = c1;
    pointB = c2;
    pointC = c3;
```

```
}

Triangle::~Triangle()
{
}

void Triangle::setPoint(std::string pointName, Coord point)
{
    if(pointName == "A")
        pointA = point;
    else if(pointName == "B")
        pointB = point;
    else if(pointName == "C")
        pointC = point;
}

void Triangle::setSide(std::string sideName, Coord firstPoint, Coord
   secondPoint)
{
    double tempSide = 0.0;

    tempSide = calcSide(firstPoint, secondPoint);

    if(sideName == "A")
        sideA = tempSide;
    else if(sideName == "B")
        sideB = tempSide;
    else if(sideName == "C")
        sideC = tempSide;
}

double Triangle::calcSide(Coord P1, Coord P2)
{
    double xSquared = (P2.getX() - P1.getX()) * (P2.getX() - P1.getX());
    double ySquared = (P2.getY() - P1.getY()) * (P2.getY() - P1.getY());
    return sqrt( xSquared + ySquared );
}

Coord Triangle::getPoint(std::string pointName)
{
    if(pointName == "A")
        return pointA;
    else if(pointName == "B")
        return pointB;
    else if(pointName == "C")
        return pointC;

}

double Triangle::getSide(std::string sideName)
{
    if(sideName == "A")
        return sideA;
    else if(sideName == "B")
        return sideB;
```

```
    else if(sideName == "C")
        return sideC;
    else
        return -200.0;
}

void Triangle::lawOfCosines()
{
    //Calculate angleA
    angleA = ( (sideB*sideB) + (sideC*sideC) - (sideA*sideA) ) / (2 * sideB *
       sideC);
    angleA = acos(angleA);
    angleA = radToDeg(angleA);

    //Calculate angleB
    angleB = ( (sideC*sideC) + (sideA*sideA) - (sideB*sideB) ) / (2 * sideC *
       sideA);
    angleB = acos(angleB);
    angleB = radToDeg(angleB);

    angleC = 180.0 - angleA - angleB;
}

double Triangle::radToDeg(double radVal)
{
    return radVal * 180.0 / PI;
}

void Triangle::printTriangle()
{
    lawOfCosines();
    std::cout << "A: " << angleA << " degrees | ";
    std::cout << "B: " << angleB << " degrees | ";
    std::cout << "C: " << angleC << " degrees | ";
    std::cout << "Total: " << angleA+angleB+angleC << " degrees" << std::endl;
}
```

4.3.3.e LED.h/cpp

Overview

The LED class essentially is there to allow definitions of the different colors that are in the blobs being searched for. While the class is named LED, it would be more accurate to rename it to blobs, since detecting colored LEDs was replaced with detecting colored blobs.

LED.h

```
/*********************************************************************//**
 * @file LED.h
 *
 * @author Julian Brackins
 *
 * @brief HEADER - class for LED groupings. HSV values stored in this class.
 *
 ************************************************************************
    */
```

```
#ifndef _LED_H_
#define _LED_H_

/*
    ********************************************************************************

 *
 * INCLUDE
 *
 ********************************************************************************
    */

#include <string>
#include "opencv/cv.h"
#include "opencv/highgui.h"

/*********************************************************************************//**
* @class LED
*
* @author Julian Brackins
*
* @brief LEDs! Not really anymore but who wants to mess with names.
*
*********************************************************************************/
class LED
{
public:
   LED();
    LED(std::string name);
   ~LED();

   int getX()                  { return xPos; }
   void setX(int val)          { xPos = val; }

   int getY()                  { return LED::yPos; }
   void setY(int val)          { LED::yPos = val; }

    cv::Scalar getHSVmin()      { return LED::HSVmin; }
    cv::Scalar getHSVmax()      { return LED::HSVmax; }

    void setHSVmax(cv::Scalar val)  { LED::HSVmax = val; }
    void setHSVmin(cv::Scalar val)  { LED::HSVmin = val; }

    std::string getColour()         { return colour; }
    void setColour(std::string val) { colour = val;  }

    cv::Scalar getText()         { return txtColour; }
    void setText(cv::Scalar val) { txtColour = val;  }

private:
   int xPos, yPos;
   std::string colour;
   cv::Scalar HSVmin, HSVmax;
   cv::Scalar txtColour;
```

```
};

#endif //LED_H
```

LED.cpp

```
/*****************************************************************************//**
 * @file LED.cpp
 *
 * @author Julian Brackins
 *
 * @brief SOURCE - class for LED groupings. HSV values stored in this class.
 *
 ********************************************************************************
    */

#include "LED.h"

/*****************************************************************************//*
    *
 * @author Julian Brackins
 *
 * @par Description:
 * LED Constructor. The empty constructor should never be used...
 *
 ********************************************************************************
    */
LED::LED()
{

}

/*****************************************************************************//*
    *
 * @author Julian Brackins
 *
 * @par Description:
 * LED Constructor for when a name is passed in as a parameter. This decides
 * what colour the LED is.
 *
 ********************************************************************************
    */
LED::LED(std::string name)
{

   setColour(name);

    //TODO: Adjust HSV vals to something more appropriate...
   if(name=="green")
   {
      setHSVmin(cv::Scalar(32, 55, 0));
      setHSVmax(cv::Scalar(105, 256, 256));

      //BGR value for Green:
      setText(cv::Scalar(0,255,0));
   }
```

```
    if(name=="yellow")
    {
        setHSVmin(cv::Scalar(17, 34, 201));
        setHSVmax(cv::Scalar(45, 256, 256));

        //BGR value for Yellow:
        setText(cv::Scalar(0,255,255));
    }
    if(name=="red")
    {
          setHSVmin(cv::Scalar(0,181,0));
          setHSVmax(cv::Scalar(217,256,224));

        //BGR value for Red:
        setText(cv::Scalar(0,0,255));
    }
    if(name=="blue")
    {
        setHSVmin(cv::Scalar(87, 110, 0));
        setHSVmax(cv::Scalar(200, 256, 139));

        //BGR value for Red:
        setText(cv::Scalar(0,0,255));
    }

}

/*************************************************************************//*
     *
 * @author Julian Brackins
 *
 * @par Description:
 * LED Destructor.
 *
 ************************************************************************
     */
LED::~LED()
{

}
```

4.3.3.f Final Conclusions About Legacy Code

Functionality

Fortunately, the legacy code provided by the 2014-2015 UAV/UGV team is definitely functional. However, due to time constraints, that team was unable to "ROSify" the code by creating a ROS publisher for the data being generated within tracker.cpp. While three blobs being detected should be plenty to determine orientation of and distance to the target, after having members complete the Computer Vision course at school, it was decided that it may be worthwhile to pursue a four-blob approach to allow use of some homography techniques that were used in that course. However, this never ended up turning into anything. Eventually, it was decided that it would be best to use the ROS library AR_Track_Alvar, since it is already implemented in ROS, and provides all functionality that would be needed by the team.

Using the Legacy Code

It is important to understand how to run this code to see the progress made by previous teams. This provided a huge amount of help to the current team, as we were able to have a jumping off point to begin work from, and understand some approaches that worked well, and some things that maybe didn't work as well. To find instructions on running this code, look to the prototypes section of this document, and find the prototypes for Sprint 2, as that will provide information on building and running this code.

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.

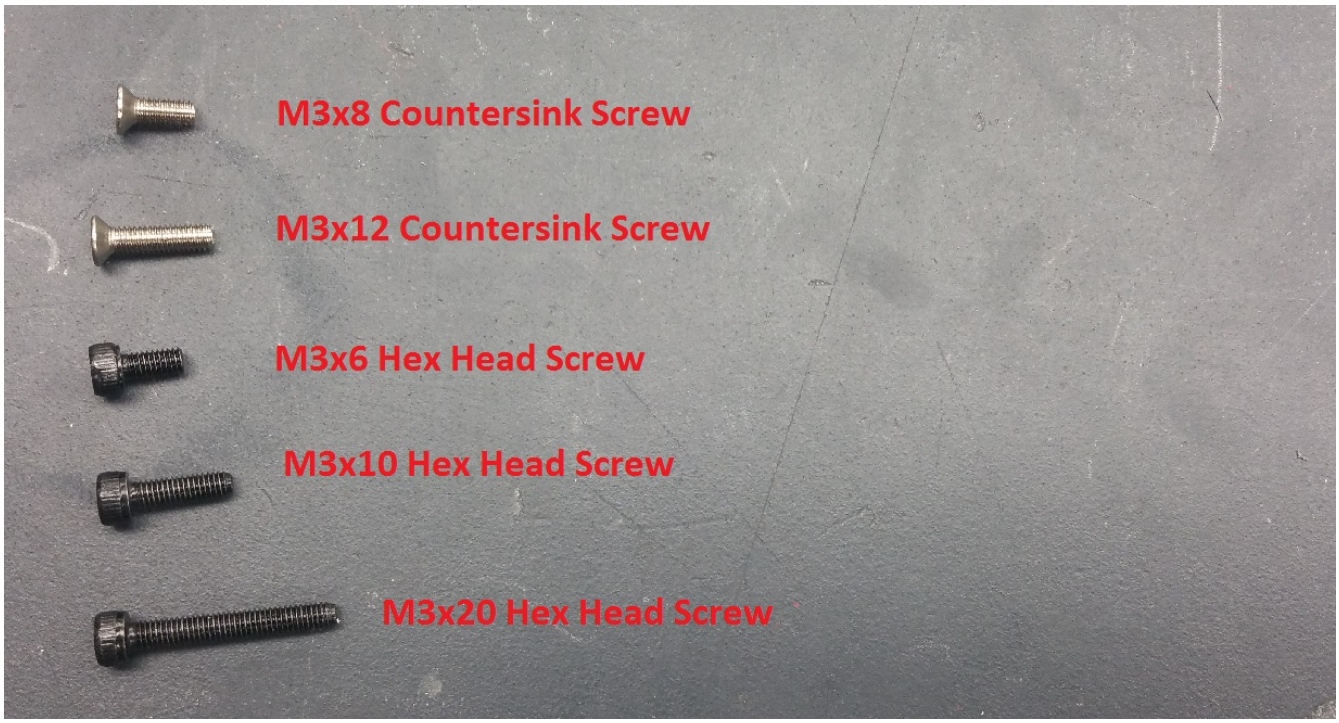


Figure 1: Bolt definitions.

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.

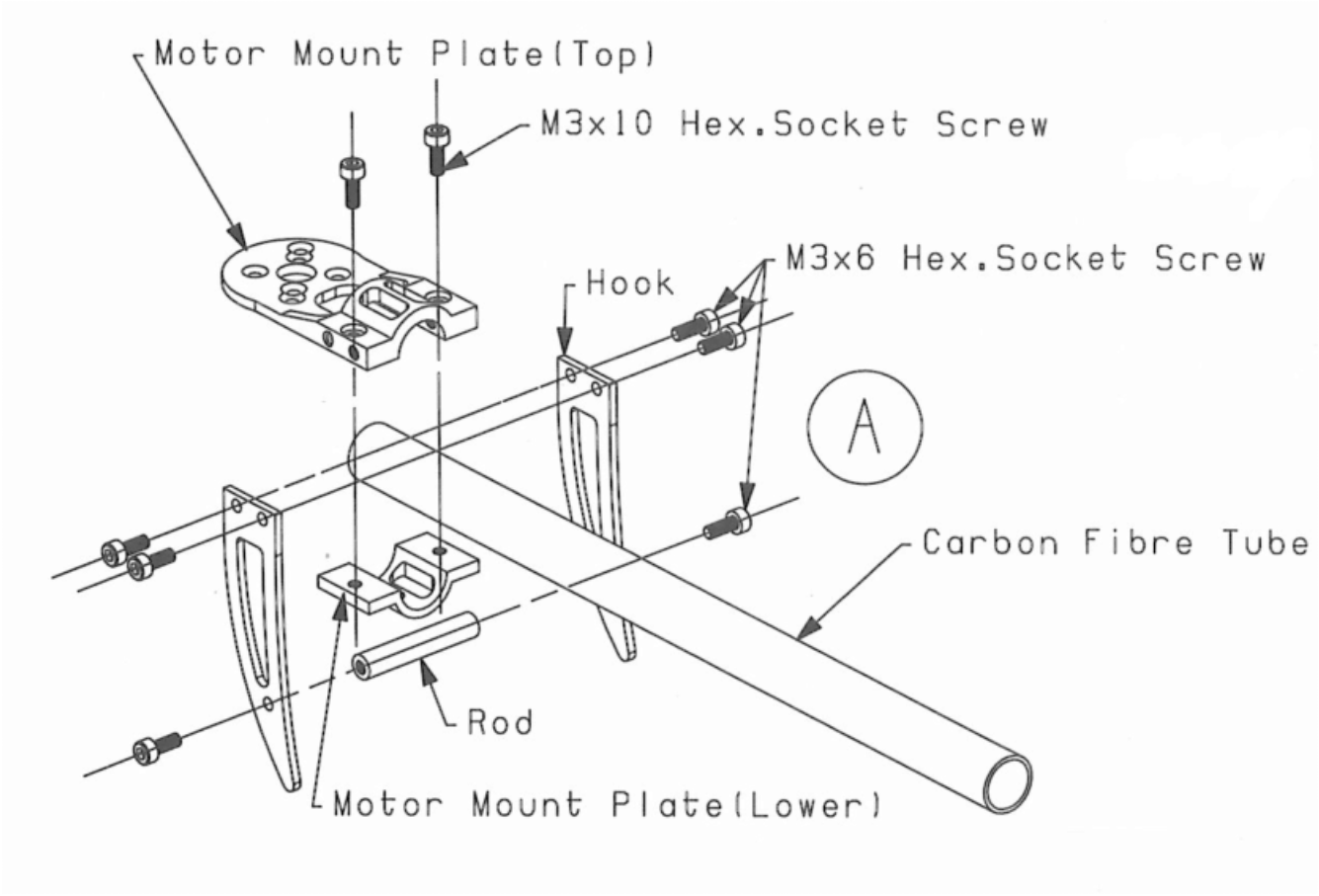


Figure 3: Instructions for assembling motor mount.

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.

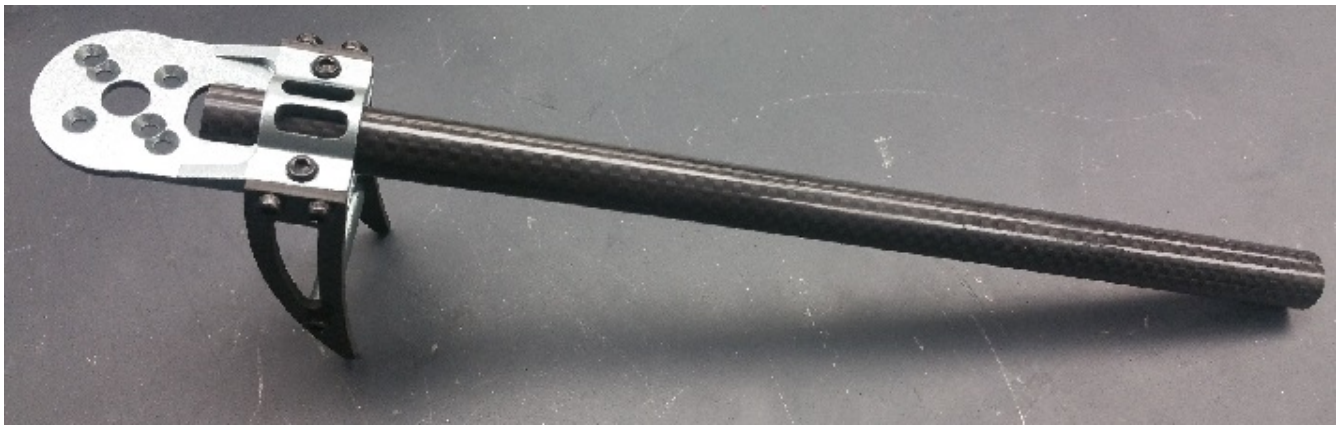
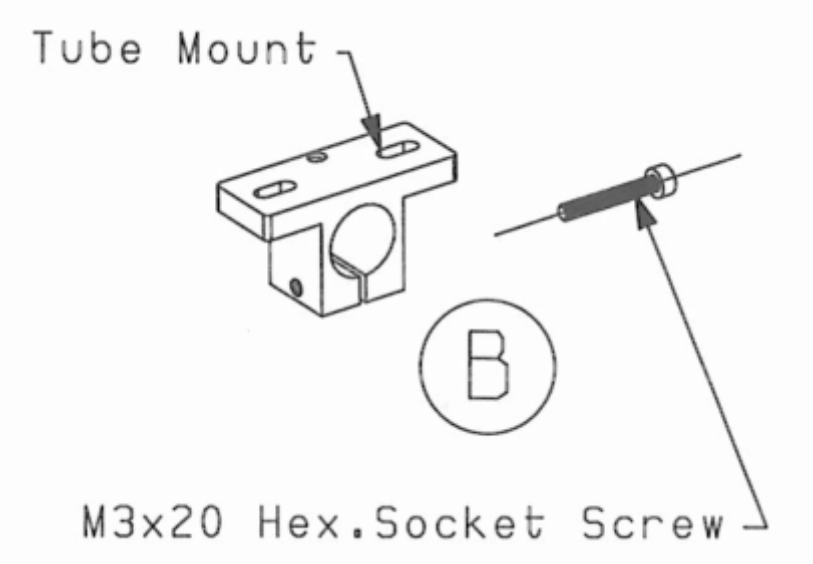


Figure 4: My completed motor mount.

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.

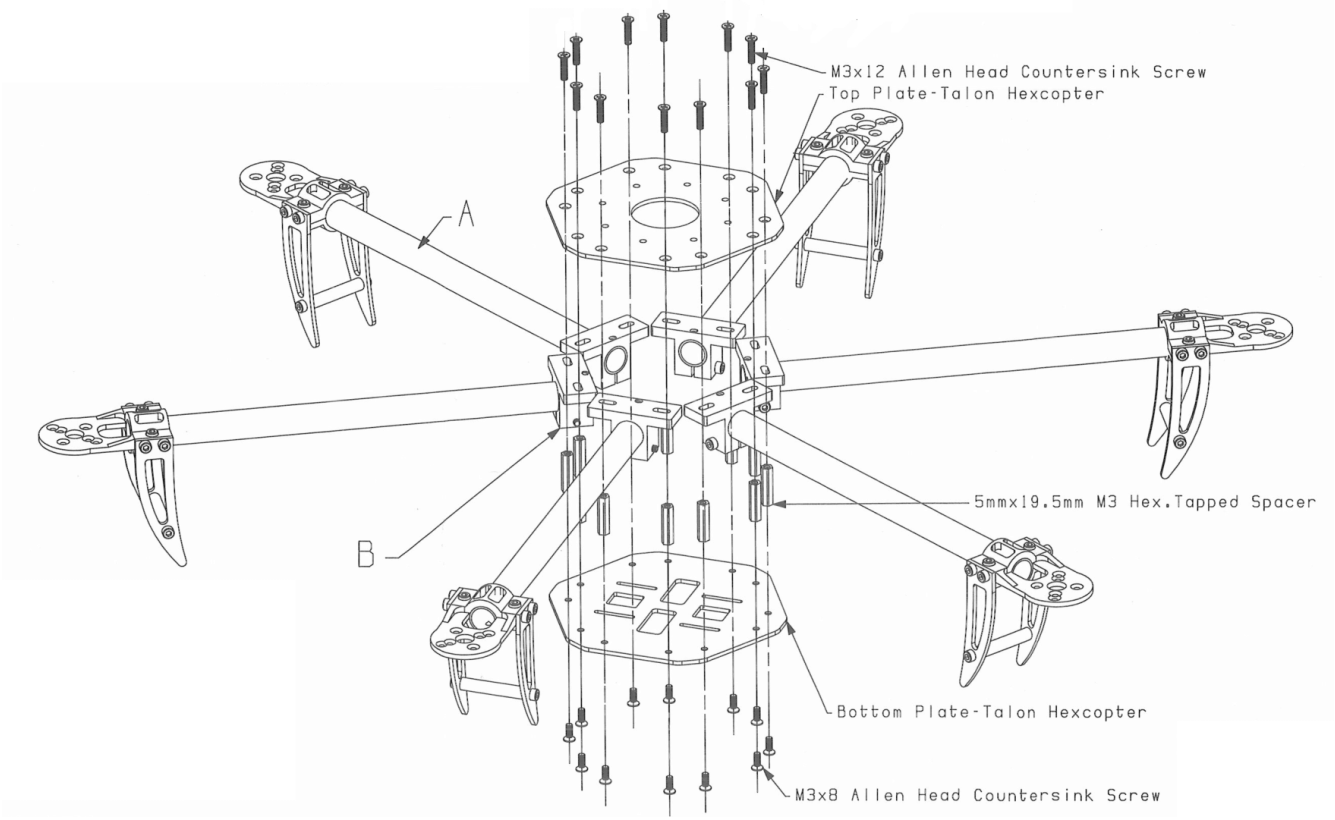


Figure 5: Instructions for Top/Bottom plates.

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.

- License: BSD
- Documentation: http://wiki.ros.org/usb_cam
- Source: https://github.com/bosch-ros-pkg/usb_cam
- Parameters
  - image width: pixel width. Default: 640
    param data type: integer
  - image height: pixel height. Default: 480
    param data type: integer
  - pixel_format: mjpeg, yuyv, or uyvy. Default: mjpeg
    param data type: string
  - io_method: mmap, read, or userptr. Default: mmap
    param data type: string
  - camera_frame_id: name of the camera tf frame. Default: head_camera
    param data type: string
  - framerate: framerate of the camera. Default: 30
    param data type: integer
  - contrast: contrast of the image (integer value [0-255]). Default: 32
    param data type: integer
  - brightness: brightness of the image (integer value [0-255]). Default: 32
    param data type: integer
  - saturation: saturation of the image (integer value [0-255]). Default: 32
    param data type: integer
  - sharpness: sharpness of the image (integer value [0-255]). Default: 22
    param data type: integer
  - autofocus: enable/disable autofocus. Default: False.
    param data type: boolean
  - camera_info_url: string value of url of calibration file.
    param data type: string

  - camera_name: the camera name, which must match the name in the calibration file.
    param data type: string

- 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.

- License: BSD
- Documentation: http://wiki.ros.org/pointgrey_camera_driver
- Source: https://github.com/ros-drivers/pointgrey_camera_driver
- Parameters
  - video_mode: There are several types supported by the driver. It is important the correct one is used.
    * use "Format0_Mode5" for video mode 640x480_mono8
    * use "Format0_Mode6" for video mode 640x480_mono16
    * use "Format2_Mode1" for video mode 1280x960_bayer8
    * use "Format2_Mode2" for video mode 1280x960_mono8
    * use "Format2_Mode6" for video mode 1280x960_mono16
    * use "Format7_Mode0" for video mode format7_mode0
    * use "Format7_Mode1" for video mode format7_mode1
    * use "Format7_Mode2" for video mode format7_mode2
    * use "Format7_Mode3" for video mode format7_mode3

    param data type: string
  - frame_rate: the camera speed in frames per second
    param data type: double
  - auto_exposure: allow the camera to automatically change exposure (Combined Gain, Iris & Shutter control).
    param data type: boolean
  - exposure: similar to contrast (use double, instead of integer).
    param data type: double
  - auto_shutter: enable/disable auto shutter
    param data type: boolean

  - shutter_speed: the time (in seconds) aperture remains open
    param data type: double
  - auto_gain: emable/disable auto gain
    param data type: boolean
  - gain: set the relative circuit gain
    param data type: double
  - pan: control camera pan
    param data type: integer
  - tilt: control camera tilt
    param data type: integer
  - brightness: set the black level offset
    param data type: double
  - gamma: set the gamma expansion exponent
    param data type: double
  - auto_white_balance: enable/disable automatic white balancing. Default: True
    param data type: boolean
  - white_balance_blue: set the blue component of white balance
    param data type: integer
  - white_balance_red: set the red component of white balance
    param data type: integer
  - format7_roi_width: width of Format7 Region of Interest in unbinned pixels, full width if zero
    param data type: integer
  - format7_roi_height: height of Format7 Region of Interest in unbinned pixels, full height if zero
    param data type: integer
  - format7_x_offset: horizontal offset for left side of Format7 ROI in unbinned pixels
    param data type: integer
  - format7_y_offset: Vertical offset for top of Format7 ROI in unbinned pixels
    param data type: integer
  - format7_color_coding: Only use if using Format7 modes
    * use "Mono8" for color coding format mono8
    * use "Mono16" for color coding format mono16
    * use "Raw8" for color coding format raw8
    * use "Raw16" for color coding format raw16

    param data type: string

- Published Topics
  - <camera>/image_raw - the image file
  - <camera>/camera_info - information from the calibration file

- Installation:

```
$ sudo apt-get install ros-jade-pointgrey-camera-driver
```

- Sample Launch File

```
<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. A sample board is shown in the figure HERE below. This board can be found here. This board is a 6×8 board (count the interior corners of the squares along the width and height, you should get 6 and 8). Be sure to measure a square after printing in

meters and use that value in the call to run this package.

- License: BSD
- Documentation: http://wiki.ros.org/camera_calibration
- Source: https://github.com/ros-perception/image_pipeline
- Subscribed Topics
  - image: this is the image being published by the respective camera
  - camera: this is the camera topic being published by the respective camera
- Installation:

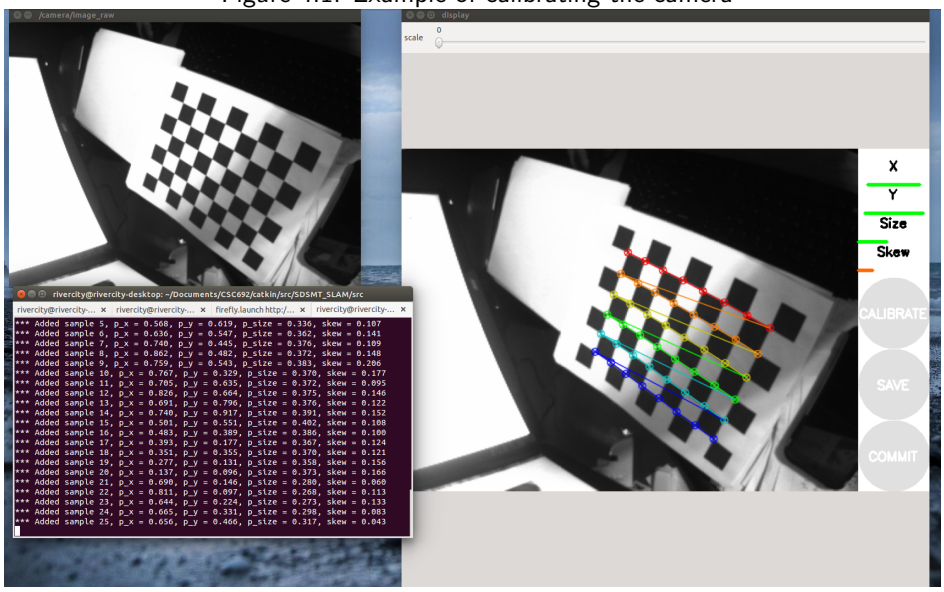
```
$ sudo apt-get install ros-jade-camera-calibration
```

- Usage:
  1. start camera in ROS environment
  2. begin camera calibration by using the following sample command, which has been modified for a board printed on standard A4 paper. (the original sample is found at the package documentation site):

```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square
    0.0254 image:=/camera/raw_image camera:=/camera
```

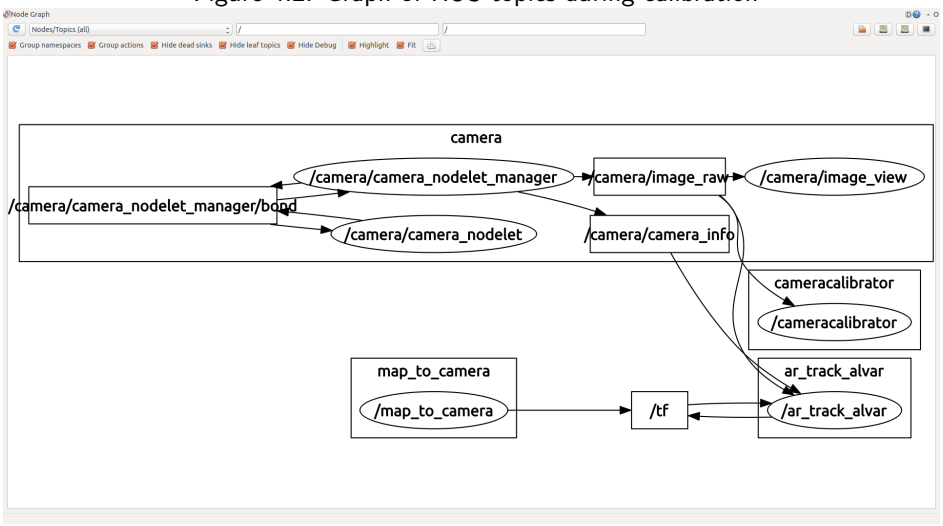
  3. A window will open containing an image like that in figure 4.1. It is important to move the chess board about traversing the frame near and distant, and about the extremes of the image frame. It is VERY important to be slow and deliberate with your movements while calibrating. Quick jerky movements will still result in a valid calibration, but the calibration may not be accurate enough when it comes time to track the AR tag or use this information for visual odometry, discussed later in the localization subsection.
     As a note, figure 4.2 is a useful example of the calibration with the firefly camera. Both firefly and

Figure 4.1: Example of calibrating the camera



and webcam calibrations will necessitate the subscription of the raw image by the calibration package.

Figure 4.2: Graph of ROS topics during calibration



The camera_info topic is useless here, because we are trying to discover the calibration, and so we only want the raw image, and the output will be the calibration file.

4. After the calibration is complete, the file will be saved to in the ros camera calibration file found by navigating to:

```
$ cd ~/.ros/camera_info
```

A quick peek into this directory will reveal, yaml files for the calibrated camera.

```
$ ls
0.yaml   head_camera.yaml
```

In this example, the firefly and webcam have both been calibrated, represented by the 0.yaml and head_camera.yaml files respectively.

This calibration will be used by the tag tracking directly, and there will some changes that will need to be made to make it usable. That will be covered in the localization section. However, it is very important to understand where this file is located. You can change this location, but the parameter in the launch file will need to be altered as well to ensure that the package can find the calibration file for reference. Note in the image file we can see that camera name, image dimensions, camera matrix, and distortion coefficients. These become very, very important later on.

```
image_width: 640
image_height: 480
camera_name: head_camera
camera_matrix:
  rows: 3
  cols: 3
  data: [974.876804668264, 0, 310.0364238884084, 0, 977.9747558767541,
      183.9139863795955, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.09363555823455676, -0.1141346935352986,
      -0.008866235140581833, -0.0048644735641969, 0]
```

```
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [962.1740112304688, 0, 307.6517383159608, 0, 0,
      965.6265258789062, 181.2353823390822, 0, 0, 0, 1, 0]
```

4.5.3 Image Processing

image_proc: This package is necessary for color cameras, as the svo step requires the use of grayscale images. This package provides a number of convenient tools in addition to translating color to grayscale. The images can also be rectified through this tool. However, for the purpose of this project, it is important that the launch file is NOT configured to do this. Doing so will result in calibration being applied twice, which will result in a distorted image.

- License: BSD
- Documentation: http://wiki.ros.org/image_proc
- Source: https://github.com/ros-perception/image_pipeline
- Parameters
  - queue_size: message queue for synchronizing image and camera_info topics. May need adjustment if camera_info is coming much faster than image_raw messages. Default: 5
    param data type: integer
  - decimation_x: the number of pixels to decimate to one horizontally. Range: [1-16]. Default: 1
    param data type: integer
  - decimation_y: the number of pixels to decimate to one vertically. Range: [1, 16]. Default: 1
    param data type: integer
  - x_offset: X offset of the region of interest. Range: [0-2447]. Default: 0
    param data type: integer
  - y_offset: Y offset of the region of interest. Range: [0-2049]. Default: 0
    param data type: integer
  - width: width of the region of interest. Range: [0-2448]. Default: 0
    param data type: integer
  - height: height offset of the region of interest. Range: [0-2050]. Default: 0
    param data type: integer
  - interpolation:
    * Nearest-neighbor sampling. Value = 0
    * Bilinear interpolation. Value = 1
    * Bicubic interpolation over 4x4 neighborhood. Value = 2
    * Resampling using pixel area relation. Value = 3
    * Lanczos interpolation over 8x8 neighborhood. Value = 4

    param data type: integer
- Subscribed Topics
  - <camera_topic_node>/image_raw: image frames from camera

  - <camera_topic_node>/camera_info: camera metadata from respective calibration file

- Published Topics
  - <camera_topic_node>/image_raw: adjusted output image
  - <camera_topic_node>/camera_info: adjusted camera metadata

- Installation:

```
$ sudo apt-get install ros-jade-image-proc
```

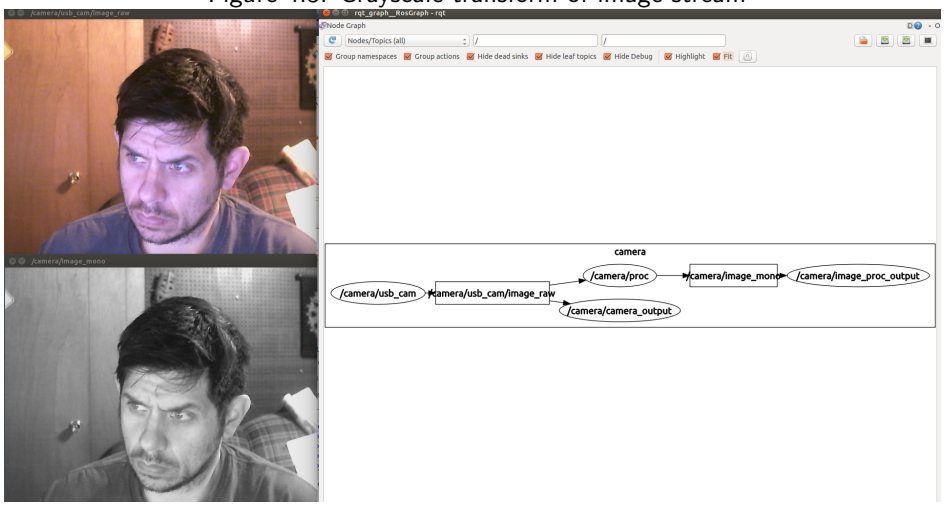
- Sample Launch File:

```
<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>

    <node name="proc" pkg="image_proc" type="image_proc" args="">
        <remap from="image_raw" to="usb_cam/image_raw"/>
        <remap from="camera_info" to="usb_cam/camera_info"/>
    </node>
</launch>
```

It may be useful here to explain the remap function. Looking at figure 4.3, we see the grayscale tranformed image from the color image. When the camera package is advertising the topic, the topic advertised will be coming from its parent node. The processing node, seen in the middle of figure 4.3 won't find image_raw without a little help. Thus, we remap the topic the processing node is looking for from image_raw to usb_cam/image_raw. So, remap does nothing more than tell the node to subscribe to the topic using a different name.

Figure 4.3: Grayscale transform of image stream



4.5.4 Tag Tracking

ar_track_alvar: this package tracks ar tags in the image frame and performs the visual homography necessary to calculate the distance and orientation of the tag in reference to the camera. It is capable of tracking multiple tags, however, we only need it to track one. Another nice feature of this package is that it will identify the tag being observed. This may be advantageous in later iterations if there are multiple UAVs requiring multiple landing pads.

- License: LGPL-2.1
- Documentation: http://wiki.ros.org/ar_track_alvar
- Source: https://github.com/sniekum/ar_track_alvar
- Parameters
  - enabled: enable/disable tracking, unsubscribing from camera topic to stop openni processing. Default: True
    param data type: boolean
  - max_frequency: maximum processing rate; frames coming at a higher rate are discarded. Range: [1.0-30.0]. Default: 10.0
    param data type: double
  - marker_size: the width in centimeters of one side of the black square marker border. Range: [1.0-100.0]. Default: 10.0
    param data type: double
  - max_new_marker_error: threshold determining when new markers can be detected under uncertainty. Range: [0.0-2.0]. Default: 0.08
    param data type: double
  - max_track_error: threshold determining how much tracking error can be observed before an tag is considered to have disappeared. Range: [0.0-4.0]. Default: 0.2
    param data type: double
- Subscribed Topics
  - <camera_topic_node>/image_raw: image frames from camera
  - <camera_topic_node>/camera_info: camera metadata from respective calibration file
- Published Topics
  - /ar_pose_marker: used in rviz for visualization. A colored square will be displayed at the location of the tag.
  - /visualization_marker: pose information of the AR tag with respect to the output frame.
- Installation:

```
$ sudo apt-get install ros-indigo-ar-track-alvar
```

- Sample Launch File:

```
<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" />

  <node pkg="tf" type="static_transform_publisher" name="map_to_camera"
     output="screen" args="0 0 0 0 0 0 world camera 10" />
```

```
<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)" />
      <param name="video_mode" type="string" value="Format0_Mode5"/>
      <param name="frame_rate" type="double" value="70" />
      <param name="shutter_speed" type="double" value="0.00005"/>

      <!-- 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 name="image_view" pkg="image_view" type="image_view" respawn="
        false" output="screen">
      <remap from="image" to="image_raw" />
      <param name="autosize" value="true" />
    </node>
</group>

<arg name="marker_size" default="12.7" />
<arg name="max_freq" default="10.0" />
<arg name="max_new_marker_error" default="0.08" />
<arg name="max_track_error" default="0.2" />
<arg name="cam_image_topic" default="/camera/image_raw" />
<arg name="cam_info_topic" default="/camera/camera_info" />
<arg name="output_frame" default="/camera" />

<node name="ar_track_alvar" pkg="ar_track_alvar" type="
    individualMarkersNoKinect" respawn="false" output="screen" args="$(
    arg marker_size) $(arg max_new_marker_error) $(arg max_track_error)
    $(arg cam_image_topic) $(arg cam_info_topic) $(arg output_frame) 30"
     />


</launch>
```

Above we see many of the previous packages discussed wrapped up into a single launch file along with ar_track_alvar. Looking solely at what is needed to call this package we can see in the snippet below from the launch file. Without an image topic and camera_info topic being actively published, the package will be unable to calculate any type of detect any tag.

```
<arg name="marker_size" default="12.7" />
<arg name="max_freq" default="10.0" />
<arg name="max_new_marker_error" default="0.08" />
<arg name="max_track_error" default="0.2" />
<arg name="cam_image_topic" default="/camera/image_raw" />
<arg name="cam_info_topic" default="/camera/camera_info" />
<arg name="output_frame" default="/camera" />

<node name="ar_track_alvar" pkg="ar_track_alvar" type="
    individualMarkersNoKinect" respawn="false" output="screen" args="$(
    arg marker_size) $(arg max_new_marker_error) $(arg max_track_error)
    $(arg cam_image_topic) $(arg cam_info_topic) $(arg output_frame) 30"
     />
```

It is crucial that these are set correctly, such as marker_size, which will determine the distance of the tag from the camera based on that size.

This other bit below is a way to instantiate a window to view our camera output image. This is very helpful when attempting to gain pose information as feedback, as image view will let us view our target frame and include the AR tag.

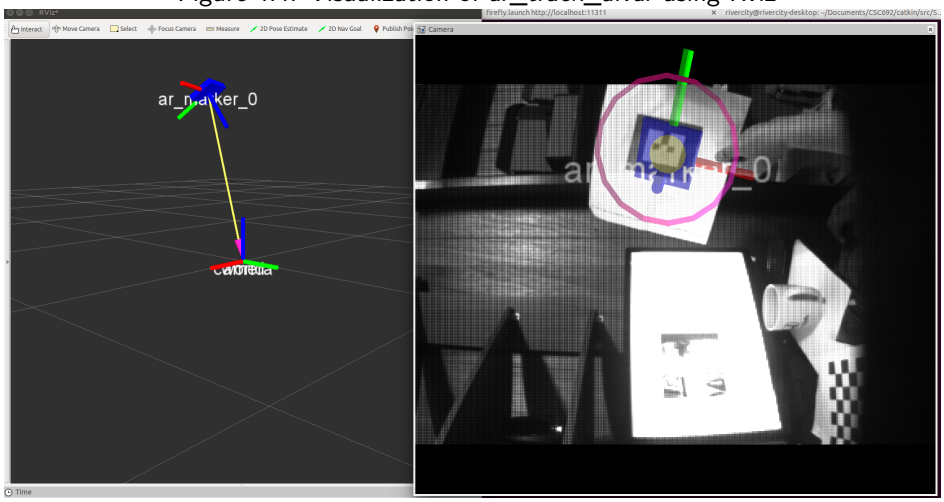
```
    <node name="image_view" pkg="image_view" type="image_view" respawn="
        false" output="screen">
      <remap from="image" to="image_raw" />
      <param name="autosize" value="true" />
    </node>
```

To gain a fantastic visualization of our pose information within the image frame we can call rviz.

```
$ rosrun rviz rviz
```

After calling rviz, we need to change our frame to that of our camera. Then we can add the camera, TF, and marker by using the add buttons in rviz and selecting those topics to visualize. The result should be similar to what we see in in figure 4.4.

Figure 4.4: Visualization of ar_track_alvar using Rviz



Before leaving discussion of this package, we will review the graph of topic communication seen in figure 4.5. We see that ar_track_alvar subscribe to both image_raw and camera_info topics. The node then passes that information to tf to calculate the pose of the tag relative to the camera frame. In the launch file you may have noticed the tf call.

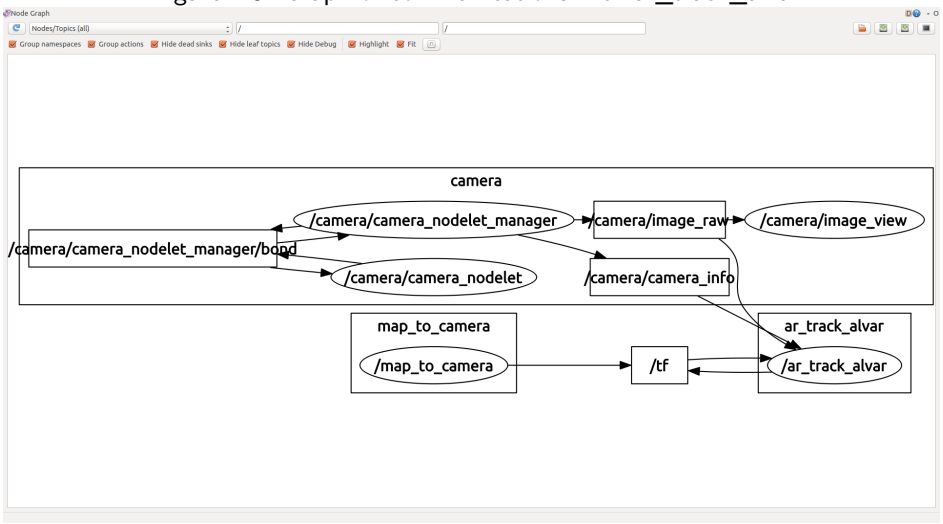
```
<node pkg="tf" type="static_transform_publisher" name="map_to_camera"
    output="screen" args="0 0 0 0 0 0 world camera 10" />
```

This is providing us the ability to set the transform, as well as adjust the alignment of the pose in the argument list. These arguments are, in order:

- x offset in meters
- y offset in meters
- z offset in meters
- yaw in radians
- pitch in radians
- roll in radians
- frame_id
- child_frame_id
- frequency of sending the tf message in milliseconds

Also present, but not appearing on the graph, are the published topics for the tag pose and the marker visualization. These topics will remain 'silent' until there is another node subscribes to those topics.

Figure 4.5: Graph of communications with ar_track_alvar



4.5.5 Localization

svo:Localization is the backbone of pose estimation. The downside to relying upon a GPS localization is that in an area with weak or no GPS signal (indoors, near buildings, in canyons or ravines), the pose estimate becomes poor or useless. Without reliable pose information, the UAV is unable to solely rely upon its IMU data to judge its movements.

- License: GPLv3
- Documentation: https://github.com/uzh-rpg/rpg_svo/wiki

- Source: https://github.com/uzh-rpg/rpg_svo
- Parameters
  - trace_name: Base-name of the tracefiles.
    param data type: string
  - trace_dir: Directory where the tracefiles are saved.
    param data type: string
  - n_pyr_levels: Number of pyramid levels used for features.
    param data type: integer
  - use_imu: Use the IMU to get relative rotations.
    param data type: boolean
  - core_n_kfs: Number of keyframes in the core. The core-kfs are optimized through bundle adjustment.
    param data type: integer
  - map_scale: Initial scale of the map. Depends on the distance the camera is moved for the initialization.
    param data type: double
  - grid_size: Feature grid size of a cell in [px].
    param data type: integer
  - init_min_disparity: Initialization: Minimum required disparity between the first two frames.
    param data type: double
  - init_min_tracked: Minimum number of tracked features.
    param data type: integer
  - init_min_inliers: Minimum number of inliers after RANSAC.
    param data type: integer
  - klt_max_level: Maximum level of the Lucas Kanade tracker.
    param data type: integer
  - klt_min_level: Minimum level of the Lucas Kanade tracker.
    param data type: integer
  - reproj_thresh: Reprojection threshold [px].
    param data type: double
  - poseoptim_thresh: Reprojection threshold after pose optimization.
    param data type: double
  - poseoptim_num_iter: Number of iterations in local bundle adjustment.
    param data type: integer
  - structureoptim_max_pts: Maximum number of points to optimize at every iteration.
    param data type: integer
  - structureoptim_num_iter: Number of iterations in structure optimization.
    param data type: integer
  - loba_thresh: Reprojection threshold after bundle adjustment.
    param data type: double
  - loba_robust_huber_width: Threshold for the robust Huber kernel of the local bundle adjustment.
    param data type: double
  - loba_num_iter: Number of iterations in the local bundle adjustment.
    param data type: integer
  - kfselect_mindist: Minimum distance between two keyframes. Relative to the average height in the map.
    param data type: double
  - triang_min_corner_score: Select only features with a minimum Harris corner score for triangulation.
    param data type: double

  - triang_half_patch_size: Subpixel refinement of reprojection and triangulation. Set to 0 if no subpix refinement required!
    param data type: integer
  - max_n_kfs: Limit the number of keyframes in the map. This makes nslam essentially a Visual Odometry. Set to 0 if unlimited number of keyframes are allowed. Minimum number of keyframes is 3.
    param data type: integer
  - img_imu_delay: How much (in milliseconds) is the camera delayed with respect to the imu.
    param data type: double
  - max_fts: Maximum number of features that should be tracked.
    param data type: integer
  - quality_min_fts: If the number of tracked features drops below this threshold. Tracking quality is bad.
    param data type: integer
  - quality_max_drop_fts: If within one frame, this amount of features are dropped. Tracking quality is bad.
    param data type: integer

- Subscribed Topics
  - <camera_topic_node>/image_raw: image frames from camera
- Published Topics
  - /svo/dense_input
  - /svo/image
  - /svo/info
  - /svo/keyframes
  - /svo/points
  - /svo/pose: this is the pose estimate with x,y,z coordinates, along with orientation as a quaternion.
  - /svo/remote_key
- Installation:

  Installation of this package is a bit different from the previous package installation instructions, and a bit more involved.

  Step by Step:

  1. create a workspace for two CMake projects

```
$ mkdir vis_workspace
```

  2. install our first project, Sophus

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

Sophus (by Hauke Strasdat) implements Lie groups which are needed to describe rigid body transforms.

3. install the second project, Fast

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

This is the Fast corner detector by Edward Rosten, which is needed to quickly find features in the image.

4. make a catkin workspace if one is not available

```
$ mkdir vis_catkin
$ cd vis_catkin
$ mkdir src
$ src
$ catkin_init_workspace
```

5. let's install vikit

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

vikit contains lots of tools which are required for SVO such as camera models and interpolation functions.

6. now for SVO!

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

7. before we go any further, let's make sure we have our cmake modules

```
$ sudo apt-get install ros-jade-cmake-modules
```

8. now let's build

```
$ cd vis_catkin
$ catkin_make
```

9. Before running the launch file
   (a) have the correct camera calibration file located in the folder located at <catkin_space>/src/rpg_svo/param
   (b) have the vo_px4.yaml file created in that same folder location as the camera calibration file. This is most easily achieved by making a copy of the vo_fast.yaml file located there and renaming it.

- Launch File & Usage:

Before launching SVO, be sure to start the camera, so that the appropriate topics are published and advertised, so that SVO will be able to subscribe to the image topic.

```
<launch>

    <node pkg="svo_ros" type="vo" name="svo" clear_params="true" output="
        screen">

        <!-- Camera topic to subscribe to -->
        <param name="cam_topic" value="/camera/image_raw" type="str" />
        <param name="max_fts" value="200" type="int" />
        <param name="min_fts" value="20" type="int" />
        <param name="init_min_disparity" value="10.0" type="double" />

        <!-- Camera calibration file -->
        <rosparam file="$(find svo_ros)/param/firefly.yaml" />

        <!-- Default parameter settings: choose between vo_fast and
            vo_accurate -->
        <rosparam file="$(find svo_ros)/param/vo_px4.yaml" />

    </node>

</launch>
```

To use this launch file, there are at least two files that must be modified or created to use this package.

– The first file that will need to be created is the <camera>.yaml file. For our implementation, this is the firefly.yaml file. To make this file, we can use the calibration file that was created earlier (the firefly calibration is used here).

```
image_width: 640
image_height: 480
camera_name: 0
camera_matrix:
  rows: 3
  cols: 3
  data: [502.0479811571255, 0, 328.3379912823597, 0,
      502.5040566931246, 249.276443286676, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.3385831965124953, 0.1221738505255268,
      -0.000681702092234282, 0.0005802657058398539, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [424.6744384765625, 0, 331.5108446465238, 0, 0,
      459.6941223144531, 250.1992172933205, 0, 0, 0, 1, 0]
```

We first need the image height and width, which should be fairly obvious. Next we need the camera model where the camera_matrix statistics in the above example is more easily discernible as:

$$\begin{bmatrix} cam\_fx & 0 & cam\_cx \\ 0 & cam\_fy & cam\_cy \\ 0 & 0 & 1 \end{bmatrix}$$

Next, the distortion_cofficients data array will compose the cam_d0 to cam_d4. With these values we can create a calibration file that meets the format requirements of svo.
We just need to create a file named <camera>.yaml and fill it with the following information:

```
cam_model: Pinhole
cam_width: 640
cam_height: 480
cam_fx: 502.047981
cam_fy: 502.504057
cam_cx: 328.337991
cam_cy: 249.276443
cam_d0: -0.338583
cam_d1: 0.122174
cam_d2: -0.000682
cam_d3: 0.000580
```

We make this a Pinhole camera, as the pinhole camera type is the most similar to the model of the camera we are using. The supported types are:

* ATAN - Uses FOV distortion model, and is the preferred model by the authors of this package. The authors state that this model computes projections much faster than other camera models. Requires the ethzasl_ptam calibration tool.
* Pinhole - Standard model for OpenCV and ROS. Requires the camera_calibration tool to determine calibration parameters.
* Ocam - Used to model cameras with high field of view and omnidirectional cameras. Requires the OCamCalib to determine calibration parameters.

– The second file that is required is the vo_px4.yaml.

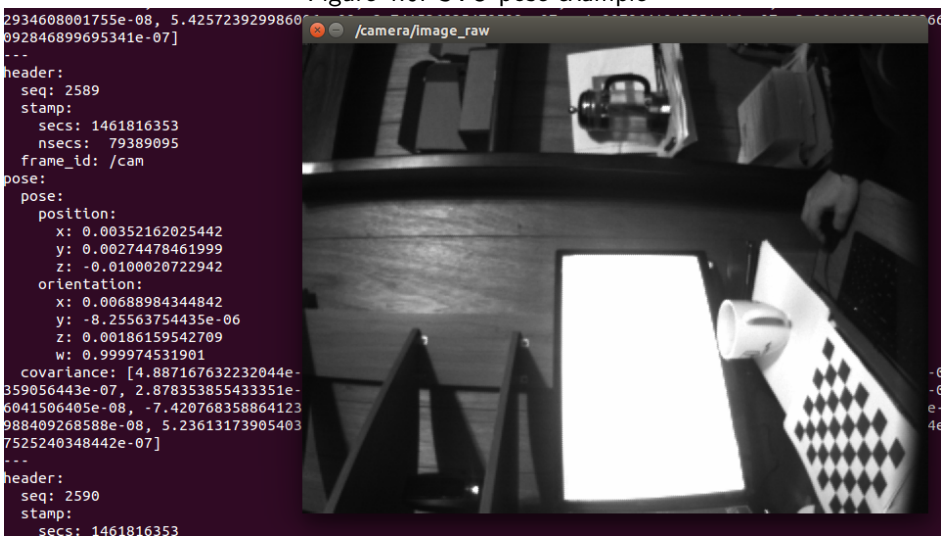
```
grid_size: 30
max_n_kfs: 20
loba_num_iter: 5
max_fts: 180
```

This file contains parameter information that is read and set at instantiation of the svo node. This example is the file we are using and does the following:

* sets the grid size of a single cell to be 30 pixels square.
* sets the maximum number of keyframes to 20.
* caps the number of iterations in the bundle adjustment to 5.
* caps the number of features tracked to 180.

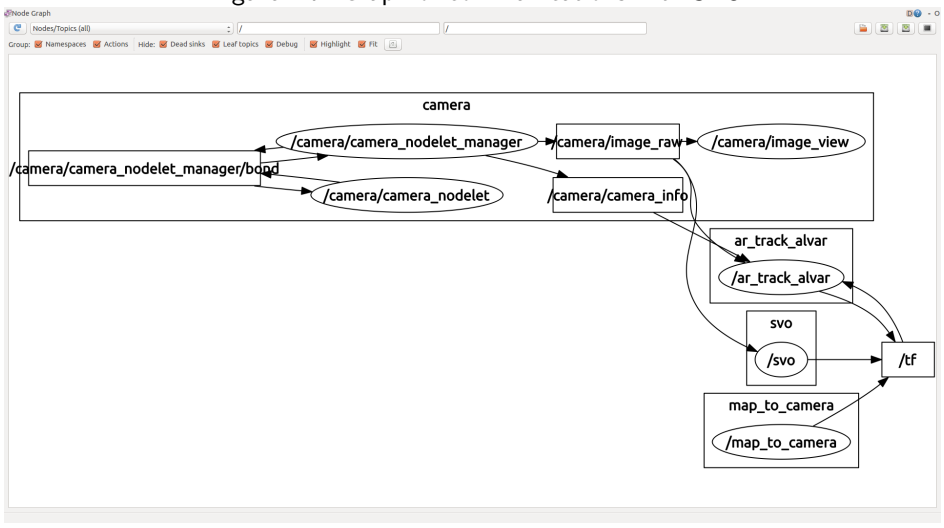
In running this program, the SVO package will gather up a number of features and initialize this point as the origin for its pose. Outputting the pose information will look much like the pose displayed on the terminal in figure 4.6. This example is taken immediately after localization is achieved, and so our coordinates are very close to the origin.

Figure 4.6: SVO pose example



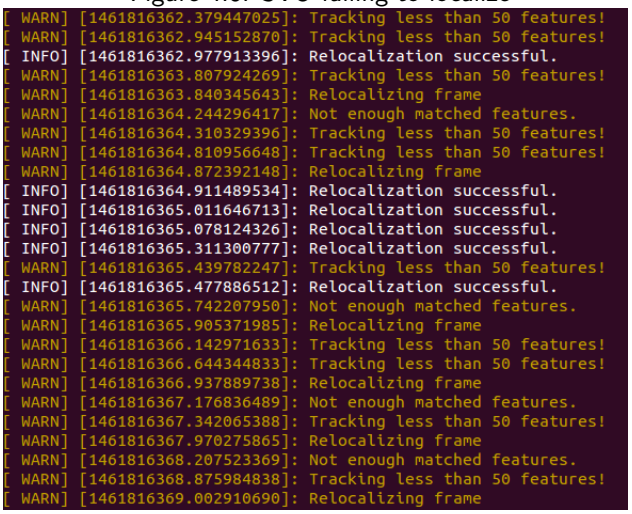
Looking at our communication graph in figure 4.7, we see the camera namespace. This namespace is publishing the image topics, which are being subscribed to by both ar_track_alvar and svo nodes. Both nodes send this to the tf node to estimate the pose of the UAV and the pose of the AR tag.

Figure 4.7: Graph of communications with SVO



Unfortunately, we found that SVO is extremely fragile in our implementation. Robust performance and reliable results are not guaranteed by the authors of the package, and we found that the camera could only be moved slightly before losing localization due to a loss of tracked features. This is demonstrated in figure 4.8.

Figure 4.8: SVO failing to localize



The team feels that the promise of localization is worth the investment with this package. Likely culprits may be slow processing of pose relative to the number of frames being sent. Another issue may be the camera is not achieving crisp (clear, not blurry, images), which would make feature matching that much more difficult. The answer may lay in playing with the various parameters respective of current image processing ability.

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

1. **TEST: U-1-3-1**
   Relating to Task: 3
   Task Description: Modify mission communication implementation as necessary
   Task Test: User is able to create mission file with ground control station.
   Results: The team found that QGroundControl worked more reliably, for the purpose of creating navigation files, within a Windows environment. Specifically, the files were created on an install on Windows 10.

   The team was able to create a series of missions including takeoff, navigate to waypoints, and landing. The waypoint mission file is exported.

2. **TEST: U-1-3-2**
   Relating to Task: 3
   Task Description: Modify mission communication implementation as necessary
   Task Test: User is able to load mission file on offboard(ODroid).
   Results: The team is able to load a mission on the ODroid by connecting directly to the ODroid via network cable. The files are then transferred from a laptop to the ODroid.

3. **TEST: U-1-3-3**
   Relating to Task: 3
   Task Description: Modify mission communication implementation as necessary
   Task Test: Landing Algorithm on offboard starts after completing last mission.
   Results: The team has tested that missions will feed the pixhawk via the Mavlink protocol when the pixhawk has indicated that the previous mission parameter has been met.

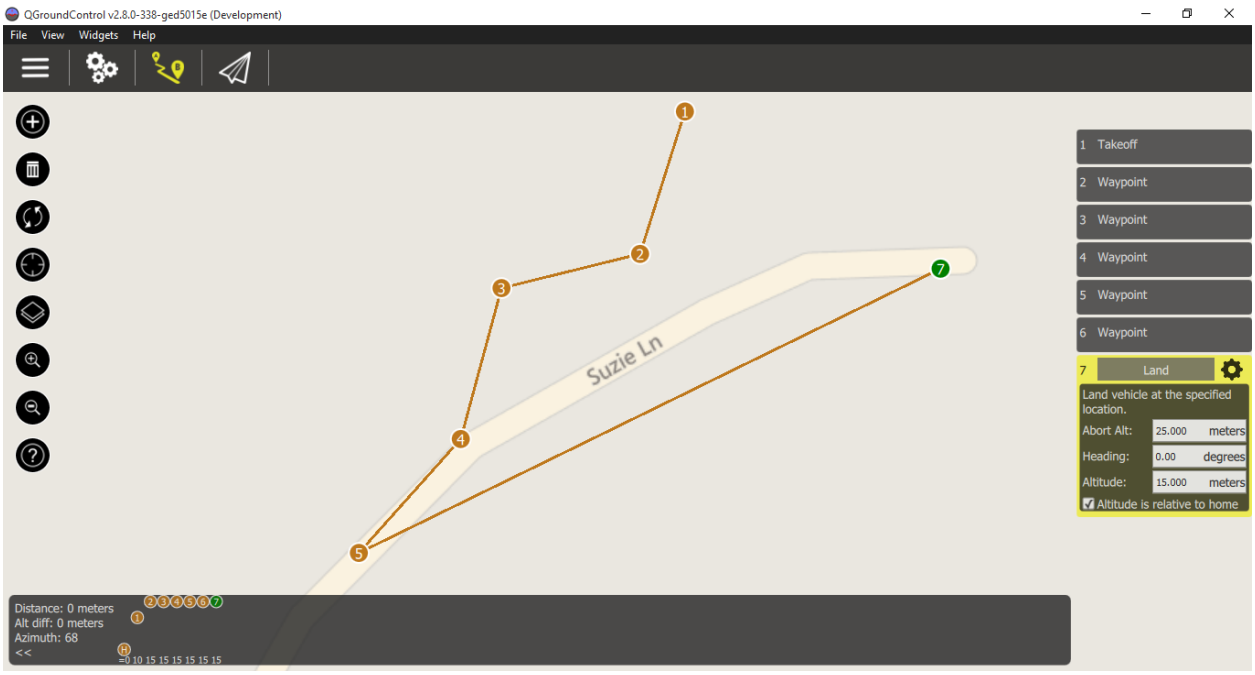


Figure 5.1: QGroundControl mission creation.

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 offboard control.	Yes
3	Modify/Rewrite take-off implementation as necessary	FCU executes take-off mission.	Yes

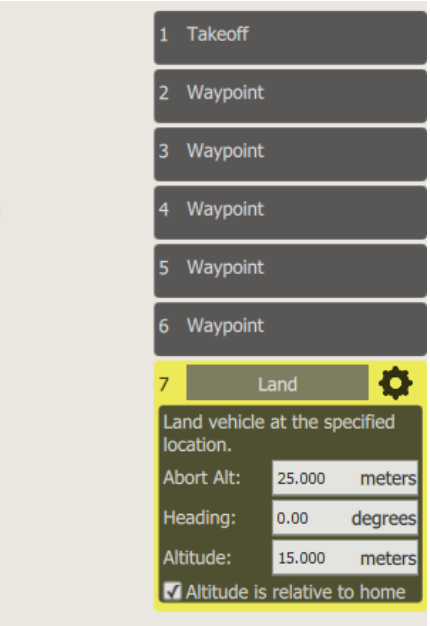


Figure 5.2: QGroundControl showing autonomous takeoff event.

1. **TEST: O-1-3-1**
   Relating to Task: 3
   Task Description: Modify/Rewrite take-off implementation as necessary
   Task Test: FCU recieves take-off mission from offboard control..
   Results: The team successfully tested this by uploading a mission file that included a takeoff event. A sample mission with takeoff event is shown in Figure 5.2.

2. **TEST: O-1-3-2**
   Relating to Task: 3
   Task Description: Modify/Rewrite take-off implementation as necessary
   Task Test: FCU executes take-off mission
   Results: The team successfully tested the execution of a takeoff mission by uploading a takeoff mission to the FCU and then allowing the flight controller to run the current mission.

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

1. **TEST: O-2-3-1**
   Relating to Task: 3
   Task Description: Modify/Rewrite waypoint navigation implementation as necessary
   Task Test: FCU receives navigation missions from offboard control.
   Results: This was successfully tested by simply observing a flight to a single way-point at the end of a driveway.

2. **TEST: O-2-3-2**
   Relating to Task: 3
   Task Description: Modify/Rewrite waypoint navigation implementation as necessary
   Task Test: FCU executes navigation in sequence.
   Results: This was verified by creating a multi-way-point mission and ensuring that each way-point was reached in the correct order. For a mission that traced a semicircle in GPS coordinates the UAV traveled along the desired path and made a semicircle before arriving at the final way-point.

3. **TEST: O-2-3-3**
   Relating to Task: 3
   Task Description: Modify/Rewrite waypoint navigation implementation as necessary
   Task Test: FCU follows, reasonably, the planned navigation.
   Results: Now that we are sure the way-points are being traversed in order, we must ensure they are being reached accurately. This was verified by having the UAV take off and land on the same exact way-point and marking its takeoff position. The UAV returned and landed with 2 feet of its original takeoff position.

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

1. **TEST: O-3-3-1**
   Relating to Task: 3
   Task Description: Modify navigate to landing waypoint implementation as necessary
   Task Test: FCU receives last navigation mission from offboard control.
   Results: This was successfully tested by simply observing a flight to a single way-point at the end of a driveway.

2. **TEST: O-3-3-2**
   Relating to Task: 3
   Task Description: Modify navigate to landing waypoint implementation as necessary
   Task Test: FCU executes navigation mission.
   Results: This was verified by creating a multi-way-point mission and ensuring that each way-point was reached in the correct order. For a mission that traced a semicircle in GPS coordinates the UAV traveled along the desired path and made a semicircle before arriving at the final way-point.

3. **TEST: O-3-3-3**
   Relating to Task: 3
   Task Description: Modify navigate to landing waypoint implementation as necessary
   Task Test: FCU navigates within a reasonable distance($<$10m) to waypoint.
   Results: In order to test accuracy of a way-point flyby mid mission, we set way-points to landmarks in the environment and had the UAV fly from landmark to landmark. The UAV came well within 10m of each landmark and ends up setting down at the landing zone within 0.5m of the target.

A video showing the successful takeoff, navigation, return and landing is provided here.

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

1. **TEST: O-4-4-1**
   Relating to Task: 4
   Task Description: Modify landing pose implementation as necessary
   Task Test: UAV accurately estimates pose of AR tag.
   Results: Using AR Track ALVAR, an AR Tag can be seen from about 15 feet away and its pose estimated very accurately.

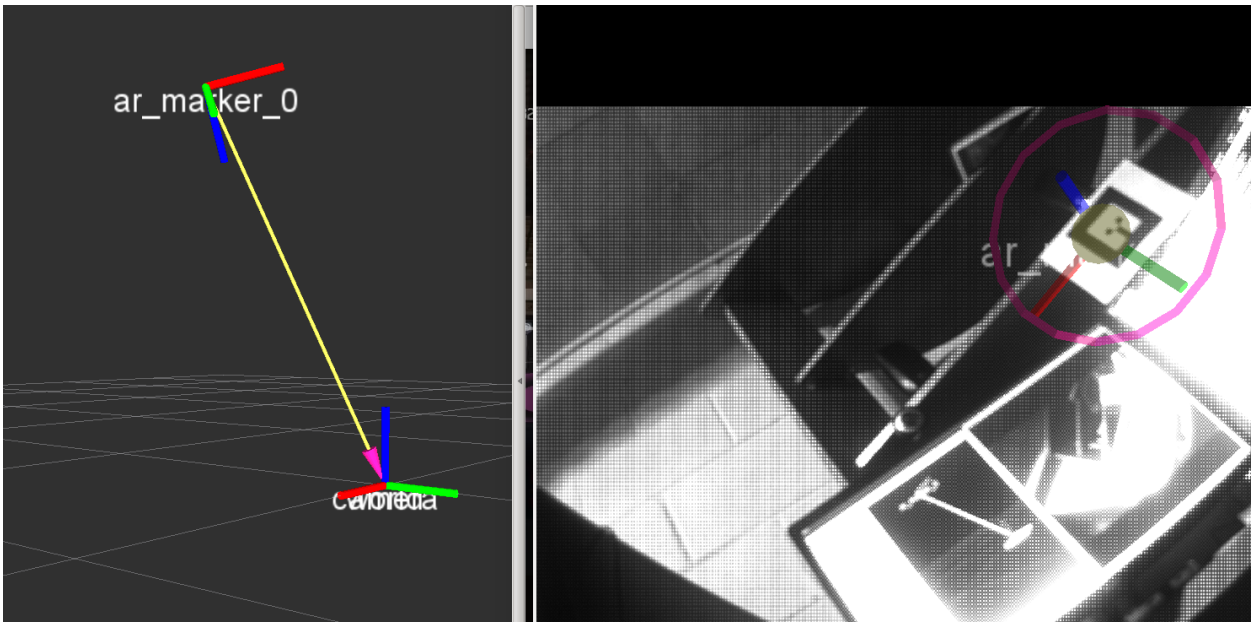


Figure 5.4: AR Tag tracking with AR Track ALVAR

2. **TEST: O-4-4-2**
   Relating to Task: 4
   Task Description: Modify landing pose implementation as necessary
   Task Test: UAV centers over AR tag within (<0.1m).
   Results: Not tested.

3. **TEST: O-4-4-3**
   Relating to Task: 4

Task Description: Modify landing pose implementation as necessary
Task Test: The UAV remains centered during descent within ($<$0.1m).
Results: Not tested.

4. **TEST: O-4-4-4**
Relating to Task: 4
Task Description: Modify landing pose implementation as necessary
Task Test: The UAV lands without damaging craft or legs.
Results: The UAV was able to land without damaging the craft under control of the FCU. Offboard control of the landing was not tested.

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

1. **TEST: O-5-4-1**
   Relating to Task: 3
   Task Description: Modify landing orientation implementation as necessary
   Task Test: UAV accurately estimates the orientation of AR tag.
   Results: See Figure 5.4 for results of AR Tag tracking.

2. **TEST: O-5-4-2**
   Relating to Task: 3
   Task Description: Modify landing orientation implementation as necessary
   Task Test: UAV changes orientation to align with AR tag, within $15\mathrm{deg}$.
   Results: Not tested.

3. **TEST: O-5-4-3**
   Relating to Task: 3
   Task Description: Modify landing orientation implementation as necessary
   Task Test: UAV maintains correct orientation during descent within $15\mathrm{deg}$.
   Results: Not tested.

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.

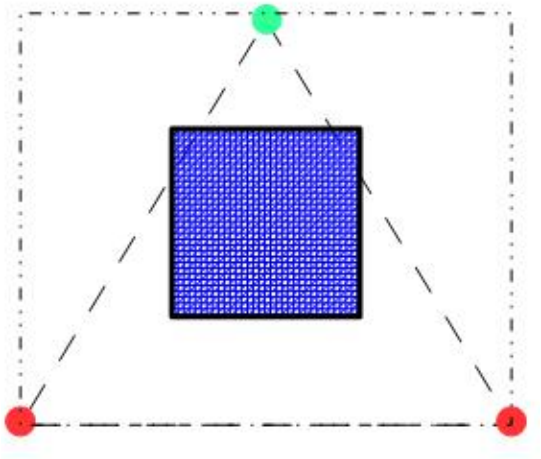


Figure 1: Landing Pad Light & QR Code Layout

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

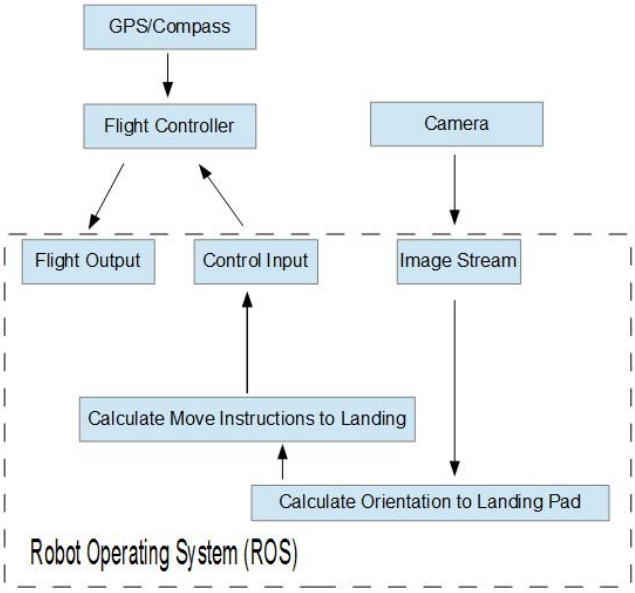


Figure 2: General ROS & Flight Control Setup

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

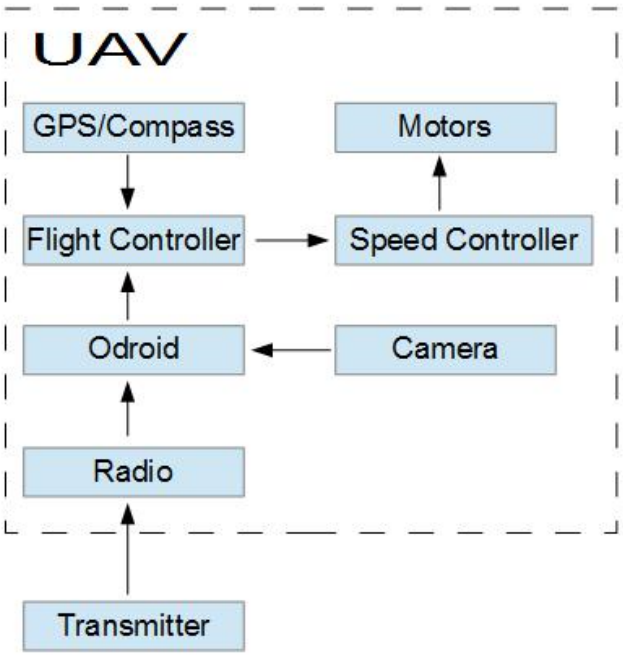


Figure 4: General Hardware Configurations

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.

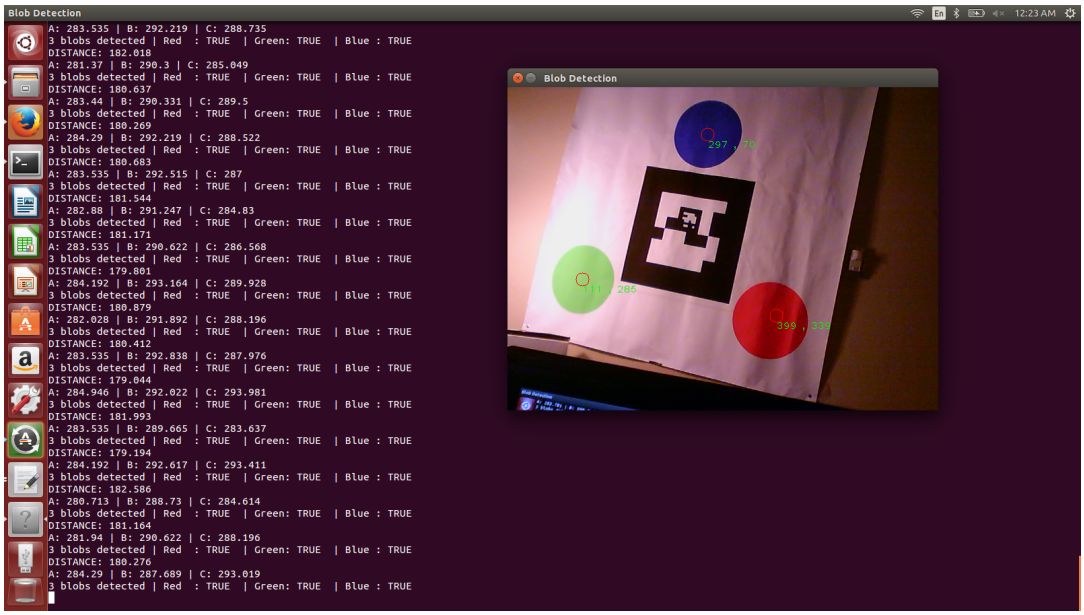


Figure 1: Blobs Detected

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.

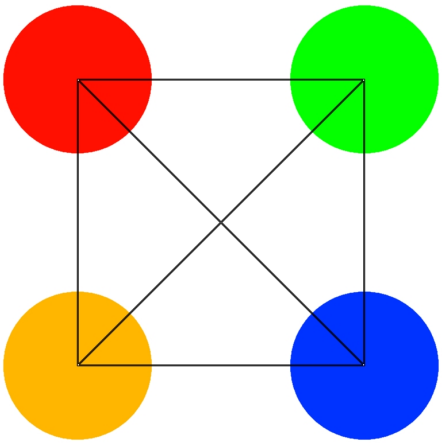


Figure 1: Four-blob landing guide

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.

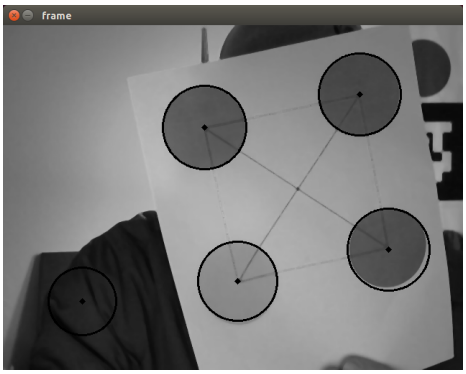


Figure 2: Hough detection on the target

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 chapter will cover the usage instructions and how to install the software required to get the UAV up an running with a Pixhawk flight controller and a odroid xu4 companion computer. The User Guide section will cover how to run all the software installed during the Installation Guide. The Programmer Manual section will cover where to get up to speed with programming in ROS as well as details regarding the flight controller code that was used to demonstrate offboard control.

7.1 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.

7.1.1 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.

7.1.1.a Dependencies

Once a Ubuntu machine is acquired make sure git and cmake are installed by running the following 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.

7.1.1.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
```

Once all the previous software is set up, a couple of items need to be moved to specific directories on the system. The flight controller directory in the landingpad repository needs to be moved into the catkin/src directory that was used for SVO. Once that is done catkin_make the directory again like above. Also the udev rules in the udev directory in the repository need to be moved to /etc/udev/rules.d/. The following commands will do the above, just replace directories with how it is set up on the system. This all assumes everything is in the home directory of the user.

```
$ cd
$ git clone https://github.com/SDSMT-CSC464-F15/landingpad
$ cd landingpad
$ cp -r flight_controller ../catkin_ws/src
$ cp udev/* /etc/udev/rules.d/
$ cd ../catkin_ws
$ catkin_make
$ source /devel/setup.bash
```

After all the steps are completed, the system is now ready for use and the user can run through the usage documentation on how to run everything that has just been installed. The next section will cover the instructions for the odroid.

7.1.2 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. Only the flight controller directory and udev rules from the repository should go on the odroid so space is not taken up by documents such as the design document and other miscellaneous items.

7.2 User Guide

Once all the software is installed successfully everything can start to be tested and ran. Make sure the ROS install is sourced correctly as well as the catkin workspace that was created during the installation section.

This Section will cover how to run the flight controller code that was used to demonstrate offboard control, Ar-Track-Alvar for the tracking of ar tags, and the visual odometry package SVO for position estimates.

A couple of ROS tools will be used to run the nodes that were downloaded and installed. The tools in ROS that will be used to run these commands are detailed in the following section in the programmers manual.

7.2.1 Flight Controller

The flight controller node is a simple python node that can control the UAV by sending vectors of direction to go where the origin is where the UAV was turned on. Make sure the Pixhawk is plugged into either the workstation or odroid and if everything is being run in the odroid multiple ssh sessions or screen is needed. To run flgight controller node execute the following commands:

```
$ roslaunch flight_controller px4.launch
$ rosrun flight_controller flight_conroller
```

Once the above commands are executed the topics can be viewed by using rostopic list or rostopic echo to view the data on a specific topic. As of this writing the flight controller node will wait until a position is reached based on the setpoint local topics and what the code is telling it to go to. The end goal is published in the console for the first three points so the user can verify that it reaches the positions appropriately.

7.2.2 Ar-Track-Alvar

Ar-track-alvar is the node that calculates the pose of a camera relative to a ar tag that is in its line of sight. This node can be ran by running the following commands depending on a camera that is used. If the user wants to use a different camera besides the ones provided they will need to calibrate the camera with the ROS camera calibration node. The commands are:

```
$ roslaunch flight_controller webcam.launch
$ rosrun rviz rviz
```

Once rviz open there will be a section on the right side where a camera topic, marker topic, and the transform (TF) topic can be added to visualize the camera and ar tags.

7.2.3 SVO

SVO is the visual odometry node that is used to give pose estimates to the Pixhawk based on the camera. To run the SVO node run the following commands:

```
$ roslaunch flight_controller webcam_svo.launch
$ rostopic echo <pose_topic>
```

7.3 Programmer Manual

To write more software to control the UAV a meta-operating system known as the Robot Operating System (ROS) is needed. ROS is the framework of everything used in this project. Other frameworks such as Mavlink could be used, however that requires writing much of the software without the basis of the communication that ROS can provide. The following subsection will describe ROS and some of the tools used in running the packages in the user guide.

7.3.1 ROS

ROS was the main the tool used in getting all the software to work together as well as many of the libraries in ROS are required to get different nodes to communicate with each other. The framework of ROS is complicated, but at its core it is an easy way to get different modules of code to communicate to each other. It has software built in that will allow a user to visualize the data that is getting streamed across the ROS network known as ROS topics.

ROS provides wrappers for interfacing with low level libraries such as Mavlink. This allows data to get streamed across ROS topics for the user to see what GPS location the flight controller and other useful information. The visualization tools such as Rviz allow a user to see pose information between a camera and ar tag for example as well as the map.

7.3.1.a Core ROS Tools

This section will cover the tools used specifically in the user guide. More information can be found at the ROS wiki regarding other tools or more in depth explanations.

Roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate. It is rand by using the roscore command.

```
$ roscore
```

Rosrun allows you to run an executable in an arbitrary package from anywhere without having to give its full path.

```
$ rosrun package executable
```

Roslaunch is a tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the Parameter Server. roslaunch can be run by executing the following commands.

```
$ roslaunch package_name file.launch
```

Rostopic contains the rostopic command-line tool for displaying debug information about ROS Topics, including publishers, subscribers, publishing rate, and ROS Messages.

```
$ rostopic list
$ rostopic echo topic_name
```

Rviz is a 3D visualization tool for ROS. This node allows for the

```
$ rosrun rviz rviz
```

7.3.1.b Programming with ROS

Information regarding programming in ROS can be found on the wiki page in the tutorial page for writing a simple publisher and subscriber in C++ and python. The flight controller node is example of python node that subscribes to information from the Pixhawk topics and also for publishing to topics for the Pixhawk to be controlled. Examples of more ROS code and be found at https://github.com/smd-ros-devel. The following listing is the flight controller code used to demonstrate offboard control. Instead of using global functions for callbacks as in the simple examples on the ROS wiki it uses a class structure to encapsulate the callbacks so data isn't global.

```
#!/usr/bin/python
import rospy
import thread
import threading
import time
import mavros

from math import *
from mavros.utils import *
from mavros import setpoint as SP
from tf.transformations import quaternion_from_euler


class SetPointPosition:
    """
    This class sends set point data to the fcu.
    """

    def __init__(self):
        self.x = 0.0
        self.y = 0.0
        self.z = 0.0

        self.pub = SP.get_pub_position_local(queue_size=10)
        self.sub = rospy.Subscriber( mavros.get_topic('local_position', 'pose'),
            SP.PoseStamped, self.reached)
```

```
        try:
            thread.start_new_thread( self.navigate, () )
        except:
            fault("Error: Unable to start the thread")

        self.done = False
        self.done_evt = threading.Event()

    def navigate(self):
        rate = rospy.Rate(10)

        msg = SP.PoseStamped( header = SP.Header( frame_id = "base_footprint",
            stamp = rospy.Time.now() ), )

        while not rospy.is_shutdown():
            msg.pose.position.x = self.x
            msg.pose.position.y = self.y
            msg.pose.position.z = self.z

            yaw_degrees = 0
            yaw = radians(yaw_degrees)

            quaternion = quaternion_from_euler(0,0,yaw)

            msg.pose.orientation = SP.Quaternion(*quaternion)

            self.pub.publish(msg)

            rate.sleep()

    def set(self, x, y, z, delay = 0, wait = True ):
        self.done = False
        self.x = x
        self.y = y
        self.z = z

        if wait:
            rate = rospy.Rate(5)
            while not self.done and not rospy.is_shutdown():
                rate.sleep()
        time.sleep(delay)

    def reached( self, topic ):
        def is_near( msg, x, y ):
            rospy.logdebug("Position %s: local: %d, target: %d, abs diff: %d",
                msg, x, y, abs(x-y) )
            return abs( x - y ) < 0.5

        if is_near('X', topic.pose.position.x, self.x ) and is_near('Y', topic.
            pose.position.y, self.y ) and is_near('Z', topic.pose.position.z,
            self.z):
            self.done = True
            self.done_evt.set()
```

```
def setpoint_demo():
    rospy.init_node('setpoint_position_demo')
    mavros.set_namespace()
    rate = rospy.Rate(10)

    setpoint = SetPointPosition()

    rospy.loginfo("Climb")
    setpoint.set(0.0, 0.0, 0.5, 0)
    setpoint.set(0.0, 0.0, 2.0, 5)

    rospy.loginfo("Sink")
    setpoint.set(0.0, 0.0, 1.0, 5)


    rospy.loginfo("Climb")
    setpoint.set(0.0, 0.0, 2.0, 5)

    rospy.loginfo("Fly to the right")
    setpoint.set( 10.0, 4.0, 8.0, 5)

    rospy.loginfo("Fly to the left")
    setpoint.set( 0.0, 0.0, 8.0, 5)

    rospy.loginfo("Landing")
    setpoint.set(0.0, 0.0,  8.0, 5)
    setpoint.set(0.0, 0.0,  3.0, 5)
    setpoint.set(0.0, 0.0,  2.0, 2)
    setpoint.set(0.0, 0.0,  1.0, 2)
    setpoint.set(0.0, 0.0,  0.0, 2)
    setpoint.set(0.0, 0.0, -0.2, 2)

if __name__ == '__main__':
    try:
        setpoint_demo()
    except rospy.ROSInterruptException:
        pass
```

8

UAV Landing Project Build Logs

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

8.1 Takeoff Log

SPRINT 1

9/14/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

9/21/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

9/28/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

SPRINT 2

10/12/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

10/19/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

10/26/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

SPRINT 3

11/9/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

11/16/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

11/23/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

SPRINT 3.5

12/21/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

12/28/15 Ran modified filter on data sets 1 - 6. Results were ... First Last

1/4/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

SPRINT 4

1/18/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

1/25/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

2/1/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

SPRINT 5

2/15/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

2/22/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

2/29/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

SPRINT 6

3/21/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

3/28/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

4/4/16 Ran modified filter on data sets 1 - 6. Results were ... First Last

8.2 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

SPRINT 2

SPRINT 3

11/16/15 QGroundControl installed and working...maybe. The linux side install seems to be a bit more problematic (some crashing and buttons not loading properly). Windows side install went smoothly, seems to work just fine. Able to generate mission files using either side. Steven Huerta

SPRINT 3.5

12/21/15 QGroundControl is incredibly frustrating to use to calibrate the pix on the linux side. Unable to complete the calibration for some reason, some information is not getting updated and preventing the final calibration from being completed. This has required walking through the calibration from the beginning each time. Steven Huerta

1/4/16 Pixhawk is calibrated and working with QGroundControl. We have some motor tests to complete and then we are ready to fly. Steven Huerta

SPRINT 4

1/18/16 UAV flying under manual control, ready to test autonomous flight soon. Steven Huerta

1/25/16 UAV performed beautifully. Recent outdoor tests confirm that the pixhawk is able to execute waypoint navigation missions. We discovered after a few flights that one of the rotor arms was twisting, and the flight controller was able to compensate for this and still fly accurately. We had consistent results of less than a few feet when the flight controller was given the same point to take off from and land on. Very surprising and we may be able to modify our landing approach. Steven Huerta

SPRINT 5

2/15/16 Offboard control for the UAV established for switching to the mode from mission mode. Noticed the local position of the UAV was not where it should be. The Pixhawk should be at the origin or close to since it has an imu. The position of the UAV in ROS needs to be investigated further. Chris Smith

2/22/16 The position that UAV wants is in vector format of x, y, z, theta. Since the UAV does not start at the origin there is no telling what could happen if offboard control is used in the current state. Chris Smith

SPRINT 6

3/21/16 Over spring break visual odometry was tried in order to get a better position estimate since no other method of controlling the UAV was satisfactory compared to the vector. Chris Smith

3/28/16 Offboard mode was tested without props to demonstrate offboard control. THe motors did increase in speed. The UAV also gave feedback of takeoff detected and was trying reach a vector that was straight above the origin in its local frame. However that location according to the Pixhawk was below it and meters to the left. The UAV would have flipped into the ground if there had been Props. If visual odometry was working we would be able to integrate another sensor in the pose estimate and gain better accuracy for the UAV on its local position. Chris Smith

8.3 Landing Log

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

SPRINT 3.5

SPRINT 4

SPRINT 5

2/22/16 AR tag tracking tool cannot be used to localize. When using the localization methods provided by mavlink protocol, we are getting garbage. Changing different localization methods to receive information from pixhawk does not seem to help. Steven Huerta

SPRINT 6

3/28/16 SVO not working, IMU data from pixhawk is overwhelming the sensor data fusion after SVO breaks and is producing very poor pose estimates. If I can get the SVO package to be a bit more robust, sensor fusion should fall in line and we should have some good pose estimations, then the landing. Steven Huerta

4/4/16 SVO still not working. Unable to get reliable pose estimates. Steven Huerta

8.4 Mechanical Log

8.4.1 UAV

SPRINT 1

9/14/15 Started to look at compenents for UAV. Chris, Dylan, Jon, and Steven

9/21/15 Compenents for UAV were not in a usable state so new components needed to be looked at for ordering. Chris, Dylan, Jon, and Steven

9/28/15 Components decided on for ordering for new hex rotor UAV. Chris, Dylan, Jon, and Steven

SPRINT 2

10/12/15 Components were ordered and waiting on arrival Steven Huerta

SPRINT 3

11/9/15 Received the Turnigy Talon Hexcopter and got the frame put together. Dylan Geyer

11/11/15 Attached the 6 DC Motors and Electronic speed controllers to each arm of the frame. Noted that there may not be enough room on the base plate of the hexcopter to mount two GPS modules a flight controller and an ODroid. Possible looking at 3D Printing a larger center plate to attach all of the arms to. I burned my fingers a lot trying to solder the ESC's to the motors.

SPRINT 3.5

12/28/15 New base plates for craft was printed and assembled. There is now plenty of room for GPS stands, Flight Controller, and ODroid. Printed base plates are temporary until carbon plates can be made.

SPRINT 4

SPRINT 5

SPRINT 6

8.5 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

SPRINT 3.5

12/21/15 Hardware in the Loop simulation still not working. Simulation requires calibration that I am unable to get working for the simulation. Have researched the issue through forums. I am voting to drop simulation completely and focus on the UAV and landing problem without simulation. Steven Huerta

12/28/15 Calibration done for the Pixhawk and still issues with file check sums. Simulation will not be possible if a product is to be finished in time so it is being abandoned. Chris Smith

SPRINT 4

SPRINT 5

SPRINT 6

8.6 Vision Log

SPRINT 1

9/14/15 Began looking into original code. Jon Dixon

9/21/15 Ran original code with landing pad target from the robotics lab. Code was able to detect the three blobs, and give what appeared to be an accurate distance reading in centimeters. Jon Dixon

9/28/15 Understood legacy vision code and was able to develop an idea of a plan for future sprints. Jon Dixon

SPRINT 2

10/19/15 Continued working with legacy vision code. Determined a plan of attack - find the angle of the target so that we can detect orientation. Jon Dixon

10/26/15 Continued with attempt to add orientation sensitivity, but not making any real progress. Jon Dixon

SPRINT 3

11/9/15 Developed a plan of attack to change from a three blob approach to blob detection to a four blob approach. This idea was prompted by the computer vision class, as we were taught that for full-blown homography it helps to have four known features to use for the math. Jon Dixon

11/16/15 Worked with Julian to develop a new landing target that would have four colored blobs instead of three. Jon Dixon

11/23/15 After playing with legacy code and attempting to make it work for the new four blob system, Julian and I determined that we had no idea how to actually go from a homography matrix to an orientation angle. Jon Dixon

11/23/15 Began work and research with other vision tracking systems. One potential option is using hough transforms to search for circles in an image, since circles are going to be relatively uncommon in nature. Began research on QR/AR tag tracking libraries. Jon Dixon

12/28/15 Attempted to implement ar_track_alvar. Unable to get ros package working. Camera is working. Tag is being detected, but pose information is not being outputted. Changed sizes of AR tag, no change. Changed parameters in launch file, no change. Steven Huerta

SPRINT 4

1/18/16 Got an ALVAR environment set up in Ubuntu 14.04 using an image stream from the webcam on the school laptop. ALVAR sample programs build and run successfully, but note that there is no ROS implementation in the raw ALVAR source. Dylan Geyer

1/25/16 Attempting to get ar_track_alvar working. Using a webcam instead of firefly to see if results are different. They are not. Results are the same. Tag is being detected, including correct identification of tag type, but no pose information is being produced. Steven Huerta

2/1/16 Non-ROS version of ALVAR is able to see AR Tags and estimate their pose. This information is stored as 3 translation variables(x,y,z) and 4 quaternian terms. Future implementations could bundle this data into a ROS Message but for now it is just printed to the terminal. Dylan Geyer

SPRINT 5

2/15/16 ar_track_alvar now working consistently and reliably. ROS package requires presence of camera calibration file. Absence of file will not produce errors, however no pose data will be outputted. Calibration MUST occur before ar_track_alvar package is implemented. Steven Huerta

2/22/16 Reproduced calibration files for both firefly and web cameras. Pose estimation is much more accurate. AR tag metrics must be adjusted if switching to a different sized tag. This must be done to get reliable and accurate pose estimates of the AR tag. Steven Huerta

2/29/16 Starting work on installing SVO project. SVO project is implementation of Semi-Direct FAST feature detection. SVO project package is designed to incorporate into the ROS environment. Steven Huerta

SPRINT 6

3/21/16 Needed to adapt camera calibration files to create necessary parameter files for SVO. SVO ATAN calibration method proved to be somewhat problematic. ATAN is recommended, but calibration application continues to fail during calibration. Files are now generated for both firefly and webcam. Both are using PINHOLE camera types. Hoping that this is not a deal breaker. Steven Huerta

3/28/16 SVO continues to break regardless of camera being used. SVO can initially localize, but any transition or rotation of the camera results in a loss of features so that localization can no longer be calculated. Not sure where this error is coming from. If camera moves incredibly slowly and not too far, SVO will continue to report pose. We are very close. Steven Huerta

4/4/16 SVO continues to fail. Attempted to change some camera parameters (shutter speed, frame rate, and autofocus), and did not succeed with either firefly or web camera. Also attempted to change some of the SVO parameters in the launch file such as minimum features. Performance increased slightly, but not enough to overcome likely movement of UAV. Steven Huerta

9

Research Results

This chapter describes the results and conclusions from our work on developing the UAV lander. This report should serve as the first stop for future teams interested in extending this project or conducting similar research.

9.1 User Interface

QGroundControl is likely the best means of generating mission files. There are definitely some problems with the application. However, it is important to remember that QGroundControl, like many of the tools we are using in this project, are currently ongoing and in development. There is an active community surrounding this project that addresses bugs, updates, and fielding questions.

It may be worth the time to develop a small message passing tool so that the file containing the files is not as difficult to transfer. The current process of connecting to the ODroid via network cable, logging into the ODroid, and pulling the file from the laptop from the laptop to the ODroid is clunky and slows down testing. As the team was not unable to afford the time to invest in developing this tool, this may be a good place to invest effort initially for future teams.

9.2 Autonomous Take-off

The take-off is solved through the functionality on the pixhawk. The pixhawk flight controller solves many of the problems of this project, so that the team was left with solving for the landing.

The take-off works very well, given an environment with GPS signal. There was a small issue that was found that the UAV needed to be at or very near the take-off waypoint specified in the take-off mission parameters. It is therefore much easier to drag around a laptop to connect to the UAV at the take-off point to ensure that when activated the UAV will perform the take-off mission consistently.

9.3 Autonomous Navigation

As mentioned in the previous section, the UAV had autonomous navigation solved with the incorporation of the pixhawk flight controller. Given an area with GPS signal, the UAV will navigate to the waypoint, and upon reaching that waypoint, load the next waypoint into as the current mission. This built-in functionality allowed the team to focus on the landing problem.

Future teams should be aware that when using the pixhawk, that the navigation will solve for shortest distance. Terrain will be a large issue if not considered in creating a navigation mission list. For example, if the UAV is navigating to a point that is at a higher elevation, the UAV may bounce along the ground as it navigates to the

next point. This is relatively easy to address when setting up the waypoint missions through QGroundControl. The user will need to simply ensure that the height set in the missions is above the height for the entire navigation.

When testing the pixhawk navigation with a strong GPS signal, our tests indicated that the pixhawk was capable of consistently bringing the UAV within 1 meter of the defined navigation point. This resulted in simplifying the landing pad. However, it also merits mentioning that these results depend on accurate mapping by companies that provide the satellite maps, such as Google. If those coordinates are not consistent between the satellite map and the GPS, then, of course, the resulting navigation will be poor.

It may be useful in future iterations to utilize the vision approaches to localization as an end towards estimating its height. Stereo vision may be the best, but a much more costly approach to achieve this end.

9.4 Autonomous Landing

This was the central problem of our project. The pixhawk provided us with fairly accurate results when navigating to the landing area. This allowed us to constrain the problem from needing to locate the landing pad from a predicted distance of 10 meters to locating the landing pad from a predicted distance of 1 meter, assuming that we had a good GPS signal available.

As mentioned in the previous section, we need to assume that the mapping between our satellite is consistent. With that assumption, we were able to remove the landing lights from the landing pad, as we should now be at a small distance directly or nearly directly over the landing pad. We can switch over from missions being executed directly by the pixhawk to the offboard control on the Odroid, which will pass the move commands to the pixhawk via the Mavlink protocol. We can perform this action after navigating to a point much closer than previously estimated. This allowed us to contrain the landing pad problem to that of the AR tag, rather than the AR tag and landing lights.

The AR tag tracking, after initial difficulties, proved to be very robust and accurate, tracking distances to less than a few centimeters. However, that is with the understanding that the size of the tag is made known to the tracking node, otherwise it is not able to estimate the distance accurately. The tracking tool also provided accurate orientation information of the tag relative to the camera. The distance and orientation of the tag from the camera serves as the basis for "dragging" the UAV to the landing pad by reducing the difference between both orientation and distance. Resolving the difference means that we have landed (or close enough to, so the rotors can gently reduce power until off).

However, after this redesign to our approach, we encountered the obstacle which stopped us, localization. Although we were getting very accurate and consistent results with the GPS peripheral for guiding the UAV through waypoint navigation, for some reason that we could not pinpoint, our pose estimation message sent from the pixhawk to the ODroid was incredibly poor, and the move estimate was poor as well. The result of this was we would localize to an unknown and unpredictable pose and moving the UAV did not show a consistent transition across the space, making move commands poor at best, dangerous to researchers at worst.

9.5 Conclusion

The problem of autonomous landing has been solved many times, however, these solutions typically rely upon external observation systems in a known space (such as motion capture). It is a much more difficult problem to solve for the landing by estimating the pose of the UAV by other means, such as using vision. Although the authors of the SVO package which we were relying upon for localization do not guarantee results, we felt that we should have at least had some limited success in worst case, if we had set up the package properly. That in mind we feel that the possible issues to the failure of our project lay in:

1. Camera Type. We are using an inexpensive camera, but a camera without a wide lens and a low frame

rate. These two issues may be causing the lack of feature acquisition to match across subsequent image frames, causing us to lose our relative position from our initial localization. Use of a different camera may solve this issue.

2. Number of Features. There are a few different parameters that can be adjusted to limit the number of features needed and to cap the number of features tracked. Setting the needed number of features low may initially seem like a good idea because we get a rapid localization, those features may be missing in the next frame after the camera has moved. Setting the cap to the number of features being tracked to a very large number has the drawback of contributing to a large increase in processing time to perform RANSAC, as well as compare features between frames to find matches. Unfortunately, this issue will require a good deal of time to tweak the values to find a good balance that delivers reliable tracking with a minimum of processing time.

3. Shutter Type. This is another potential issue. There are different types of shutters that have different effects, or can have, on the image that results. A rolling shutter is very common for inexpensive web cameras. A rolling shutter will update cells line by line. More expensive cameras may feature a global shutter, where the cells are updated all at once and so avoid the potential blurring effect that may result from motion. The team feels that cheaper solutions are better, but perhaps this may be an issue where there is no current compromise. This would be the last area of exploration for post review, as it is not as accessible for testing as the previous two.

The pieces to solving this project are contained here. The UAV is built and flying autonomously with the pixhawk. The camera is able to detect the position of the tag accurately so that we can measure our position from the landing pad and make the necessary corrections to land the craft on the landing pad with the correct orientation so that it may recharge before its next flight.

9.6 Further work

There are many different ways that this work can be extended from where our team has ended its research. The team has gathered these few possiblities:

1. Solve how to localize in a GPS denied environment. This was our final obstacle, and certainly another group may utilize the work we have done and solve for the localization with vision, or perhaps another approach.

2. Add another camera and add the task of obstacle avoidance by means of computer vision. This project has assumed that in take-off, navigation, and landing, the UAV has an unobstructed path. In actual deployment, it may not be the case that the craft will be unobstructed along its path. Certain terrain features, such as rocks, trees, buildings may not be available on the map, and the craft will be unable to proceed as is.

3. Solve for navigation in GPS denied environments. The UAV may be deployed and navigate to areas where GPS signal is not available. The UAV should be able to continue to navigate to the location where it believes those coordinates lay. This would be of great assistance to application where navigation across steep terrain or dense canopy may prevent access to a reliable GPS signal.

These extensions are difficult problems that are compounded by the amount of processor available, as well as the draw that a processing has on power consumption, reducing the overall flight time of the UAV. However, this project should provide a firm base for launching in any of these directions.

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

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.

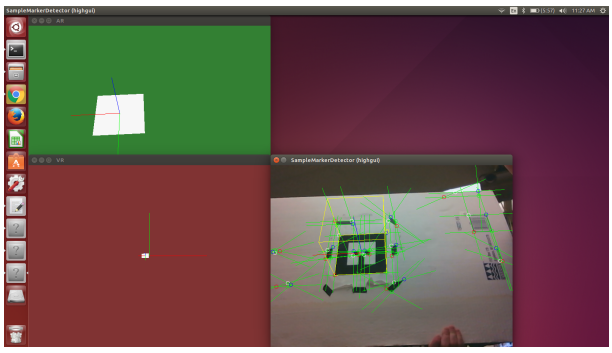


Figure A.1: ALVAR AR Tracker testing with laptop webcam

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

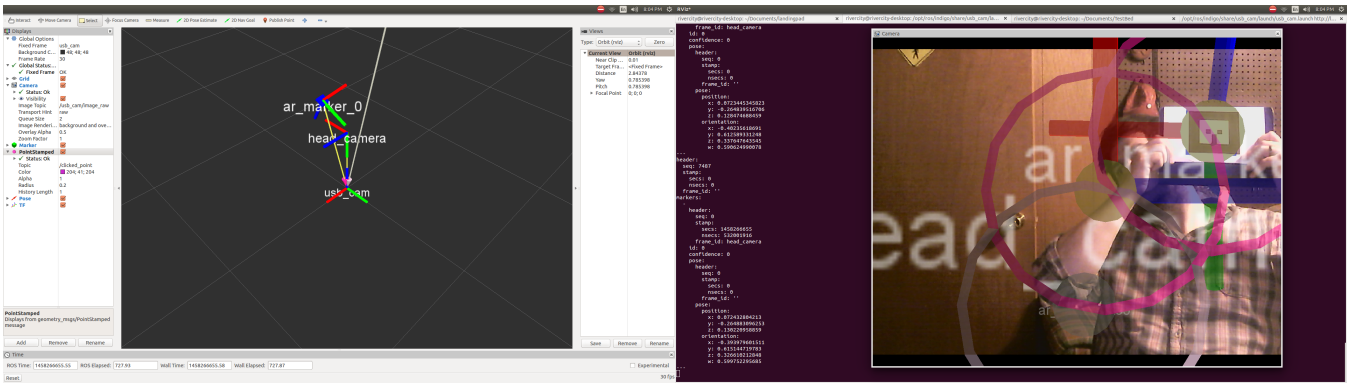


Figure A.2: ALVAR AR Tracker Demo in RViz

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.

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: gitcsmithsd@gmail.com

Education

Bachelors of Science in Computer Science — Dec 2016
Minor in Applied and Computational Mathematics — Dec 2016
South Dakota School of Mines, Rapid City, SD

Electives: Probabilistic Robotics, Cryptography, Cybersecurity, Computer Graphics, Natural Computing, Parallel Computing

Programming Experience

Probabilistic Robotics (Project: SLAM Algorithm) — Jan 2016-Present
- Programming a robot to determine its location
- Implementing a particle filter based on a Bayes filter

Senior Design (Project: Unmanned Aerial Vehicle (UAV) Landing Pad) — Aug 2015-Present
- Programming a UAV to autonomously navigate and land on a platform using computer vision

SDSMT Robotics Team — Sept 2011-Present
- Working as a team to design and build robots for competitions
- Previous vice president and president in 2014

Natural Computing (Final Project: Artificial Intelligence) — Jan-May 2015
- Created an AI to play the board game Puerto Rico
- A genetic algorithm was used to improve the AI

Computer Skills

Programming Languages: C++, Python, ARM Assembly, Java, C#, Common Lisp, SQL
General Knowledge: Linux, ROS, Makefiles, LaTeX, Microsoft VS, Slicer, Repetier-host

Work Experience

Backroom Associate at Target, Rapid City SD — July 2012-Present
- Pulling merchandise down for stocking on the sales floor
- Placing extra merchandise in the appropriate areas in the stockroom
- Organizing and putting items in the coolers and freezers that come off of a food truck

Volunteer Experience

Association of Computing Machinery and Linux Users Group — Jan 2016-Present
- Helped other students with debugging code written in ARM assembly, c, c++, lisp, and python

SDSMT Robotics — 2012-Present
- Helped boy scouts earn a merit badge by teaching them about the designs and programming
- Assisted in teaching students programming and design so they could complete their challenges in First Lego League

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–