UAV/UGV Coordinated Automation

Senior Design 2015 Project Documentation

Eye In the Sky

Colter Assman Julian Brackins Sam Carroll Hafiza Farzami
Charles Parsons Alex Wulff

May 8, 2015

Contents

List of Figures

List of Tables

List of Algorithms

Mission

Our company, Eye In The Sky, is dedicated to providing un-manned ground and air vehicle solutions (UGV/ UAV) for assistance of fire department and search and rescue assistance in the state of South Dakota. It is our purpose to create solutions to assist in the evaluation of fire safety and recovery of people in need of assistance by leveraging a combination technologies to handle information gathering and telemetry in otherwise dangerous environmental conditions.

Document Preparation and Updates

Current Version [1.2.0]

Prepared By:
Alex Wulff
Julian Brackins

Revision History

Date	***Author***	***Version***	***Comments***
11/06/14	*Alex Wulff*	*1.0.0*	*Initial version*
12/05/14	*Alex Wulff*	*1.1.0*	*General documentation updates.*
12/06/14	*Alex Wulff*	*1.1.1*	*Integrated sprint reports.*
03/13/15	*Julian Brackins*	*1.2.0*	*Sprint Timeline and other misc. project requirements*
03/15/15	*Julian Brackins*	*1.2.1*	*Revised Contract, Path Planning, cv_tracker, UAV*
04/07/15	*Julian Brackins*	*1.2.2*	*Additional Chapter 4 Content*
04/18/15	*Julian Brackins*	*1.2.3*	*Chapter 4 Complete, formatted back material*
05/06/15	*Julian Brackins*	*1.2.4*	*Control Panel, Drive System, UAV, UGV material*
05/07/15	*Julian Brackins*	*2.0.0*	*Final Editorial Adjustments*

1

Overview and concept of operations

Commensurate with the User Stories and Architecute Design, we have isolated three key components. These are intrinsic to performing the operations of UAV, UGV, and common ROS-based infrastructure. To better illustrate these components, this section will overview the operational entities as semi-independent sections.

1.1 Scope

This section of the Senior Design documentation covers the three main components that are being constructed to deliver upon the aforementioned User Stories.

1.2 Purpose

This system will assist in the collection of data in otherwise unsafe zones for human occupancy as it relates to:

- Natural Disaster Reconnaissance
- Forest Fires - Spread and Destructive Evaluation
- Missing Persons Tracking

1.2.1 UAV

The Unmanned Arial Vehicle will fulfill the reconnaissance tasks in this project. This will consist of video and sensor data collection, while in flight. Upon completion of data collection this data will be transfered back to a control base station via radio telemetry. It is crucial that the UAV be able to return to the ground vehicle before energy reserves are depleted to recharge and be transported securly to the next location where data collection can begin again.

The current configuration for this component is:

- AMP 2.6 - flight control board
- 3dr ublox GPS with Compass - compass and gps
- R5800X receiver and TXV582 - video transmitting
- Sony HAD 520 line camera - on board camera
- Spektrum AR7000 - 7 channel receiver
- 915 MHz radio - telemetry with ground station

1.2.2 UGV

The Unmanned Ground vehicle is computationally more simplistic than the UAV and will fulfill the tasks of navigation to pre-determined GPS coordinates, elevate the launch pad, wait for return of the UAV and recharge the UAV on arrival. The system is set to include the following components after the frame has been constructed in Sprinter Break:

The current ground vehicle loadout contains the following components:

- GPS – Make and model still under investigation
- SLAM Cameras (Asus Xtioc & Microsoft Kinect)
- Recharging Station with mounts
- Tilting platform with LED indicators for low light environments.
- Power supply
- Odroid computational units. Likely Odroid XU 3 models.

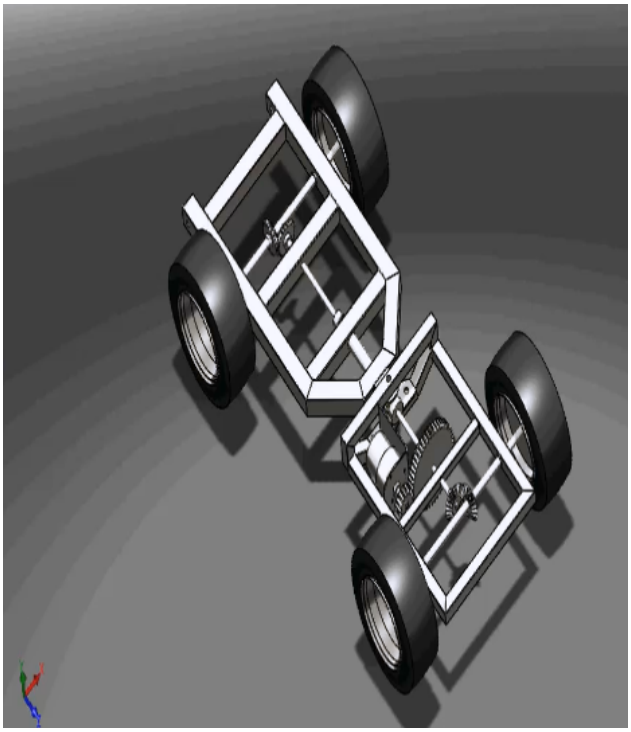

Figure 1.1: Frame proposed for UGV

1.2.3 Common Components

To better distribute work load as well as reduce the testing required for new software developments in ROS, Eye in the Sky has proposed to create common infrastructure programming for input devices. This common protocol will take various input streams (Cameras - Stereo and Monocular, GPS, Motor Control, etc.) and will use common topics or service subscriptions to streamline communication between these devices. This will further assist with common computation such as GPS interpretation.

1.3 Systems Goals

This completed system is intended to:

1. (UGV) Take predetermined GPS waypoint data and navigate to a location to begin reconnaissance.
2. (UGV) Once at the location, level the landing platform to launch the UAV & disconnect any charging and securing mechanisms.
3. (UAV) Start mapping and elevate to a predetermined altitude directly above the UAV and navigate linearly to a perimiter of a flight circle.
4. (UAV) Once on the radius of the flight path, begin traversing the path while recording video and other sensor data.
5. (UAV) After completing the flight path, return to the UAV with both visual and mapping assistance.
6. (UAV) Orient for landing and touch down on the UAV's platform.
7. (UGV) Secure the UAV and begin charging.

This process is scheduled to repeat until the UGV has expired all GPS way-points. At this time the UGV will return to the start location or any ending location for retrieval by the owners.

1.4 System Overview and Diagram

To illustrate the connectivity of these various components, the following system diagram has been created.

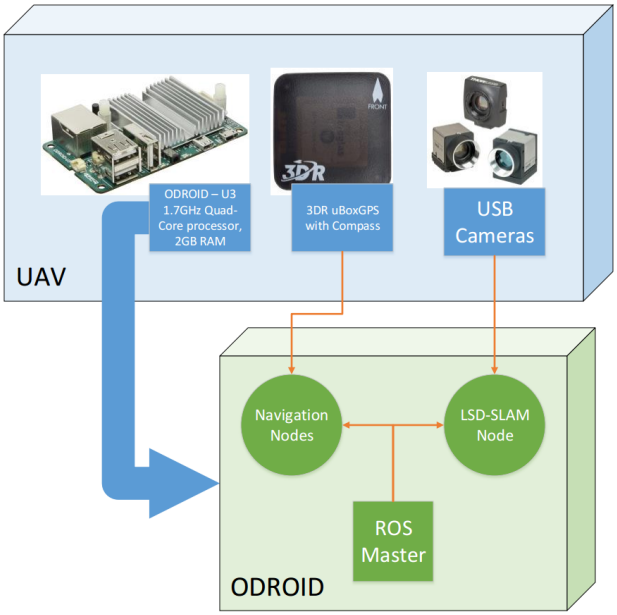

Figure 1.2: UAV System Diagram

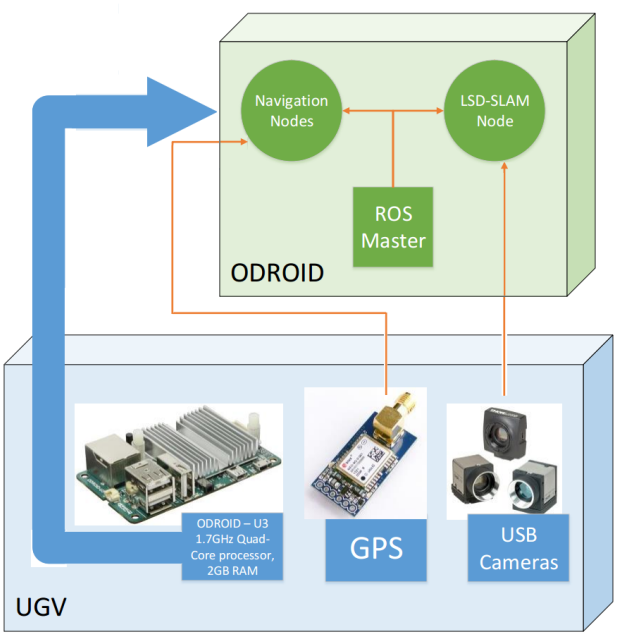

Figure 1.3: UGV System Diagram

1.5 Technologies Overview

A brief list of the technologies included and their applications:

1. OpenCV
   (a) Distance Detection to known design on Landing Pad
   (b) Possible Homography integration
2. SLAM
   (a) Varieties Used
      i. Hector-SLAM - Integrated with UAV mapping structure
      ii. LSD-SLAM - Additional mapping and localization for UAV
      iii. Point Cloud Maps- Obsticle and path detection for UGV
3. GPS - Global coordination and additional verification on UAV / UGV localization.

2

Project Overview

This section contains information regarding the team, project progress, and our testing methodology

2.1 Team Members and Roles

The team consists of the following members.

- Colter Assman - UAV Specialist. Colter is a member of SDSM&T's UAV team and liaison for Search and Rescue, FAA, and ourselves.
- Julian Brackins - Scrum Master. Julian is responsible for documentation, team management, presentation preparation, and craft orientation software for UAV landing.
- Samuel Carroll - Navigations Specialist focused in developing object avoidance subroutines and UGV construction.
- Hafiza Farzami - Control Panel designer. Also in charge of AR Tag implementation for determining craft pose and positioning for UAV landing.
- Charles Parsons - Navigations Specialist. Charles has experience with various SLAM algorithms and their integrations and is investigating the path planning procedures for the UGV.
- Alex Wulff - Technical Lead. Alex resolves technical issues and is responsible for overseeing informal code reviews. Alex has been the architect for the custom ROS API layer, SVN logger for monitoring team collaboration, and the lead designer for the UGV construction.

2.2 Project Management Approach

The structure of Eye in the Sky's development is based on Agile methodology. There are a series of six formal three-week sprints through out the development of this solution with a short sprint during the SDSM&T winter break (Sprinter Break). All backlog structure is housed on Trello, an internet resource for project management.

The development process will focus heavily on code review and simulation testing before builds are pushed to the hardware. This is primarily due to the budget constraints on this project, and we can not afford continual repairs on either the UGV or UAV. All System and Unit Testing is available here.

2.3 Phase Overview

The initial sprints for this project have been primarily research driven. As a result Sprints 4,5, and 6 will focus on development of UAV / UGV in parallel. For development, the following System and Unit Testing structure has been put into place.

2.4 Terminology and Acronyms

- UAV - Unmanned Aerial Vehicle. Commonly referred to as a drone.
- UGV - Unmanned Ground Vehicle
- Point cloud - A set of data points in a given coordinate system.
- QR Code - Quick Response Code. A matrix barcode first used for the automotive industry in Japan. Has recently become popular due to quick readability and increased information capacity over standard UPC barcodes
- AR Tag - A marker used to support augmented reality.

3

Project Agreement

SDSMT SENIOR DESIGN
SOFTWARE DEVELOPMENT AGREEMENT

This Software Development Agreement is made between the SDSMT Computer Science

Senior Design Team Eye In The Sky ,

consisting of team members Colter Assman, Julian Brackins, Samuel Carroll, ,

Hafiza Farzami, Charles Parsons, Alex Wulff ,

and Sponsor Jeff McGough ,

with address: 501 E. Saint Joseph Street Rapid City, SD 57701 .

3.1 RECITALS

1. Sponsor desires Senior Design Team to develop software for use in visual odometry and mapping for UAVs and UGVs.
2. Sponsor desires Senior Design Team to develop hardware for UAVs and UGVs to integrate and facilitate recharging batteries.
3. Senior Design Teams willing to develop such Software.

NOW, THEREFORE, in consideration of the mutual covenants and promises herein contained and within the specifics defined in the documentation section entitled "Requirements", the Team and Sponsor agree as follows:

3.2 EFFECTIVE DATE

This Agreement shall be effective as of September 15, 2014 .

3.3 DEFINITIONS

1. "Software" shall mean the computer programs in machine readable object code form and any subsequent error corrections or updates supplied to Sponsor by Senior Design Team pursuant to this

Agreement.

2. “Acceptance Criteria” means the written technical and operational performance and functional criteria and documentation standards set out in the project plan.
3. “Acceptance Date” means the date for each Milestone when all Deliverables included in that Milestone have been accepted by Sponsor in accordance with the Acceptance Criteria and this Agreement.
4. “Deliverable” means a deliverable specified in the project plan.
5. “Delivery Date” shall mean, with respect to a particular Milestone, the date on which University has delivered to Sponsor all of the Deliverables for that Milestone in accordance with the project plan and this Agreement.
6. “Documentation” means the documents, manuals and written materials (including end-user manuals) referenced, indicated or described in the project plan or otherwise developed pursuant to this Agreement.
7. “Milestone” means the completion and delivery of all of the Deliverables or other events which are included or described in [the project plan] scheduled for delivery and/or completion on a given target date; a Milestone will not be considered completed until the Acceptance Date has occurred with respect to all of the Deliverables for that Milestone.

3.4 DEVELOPMENT OF SOFTWARE

1. Eye In the Sky 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 and will be conducted by the Team Lead. The Team will deliver the Software to the satisfaction of the course instructor that reasonable effort has been made to address the needs of the client. The Team understands that failure to deliver the Software is grounds for failing the course.
2. Sponsor understands that the Senior Design course’s mission is education and advancement of knowledge, and, consequently, the development of Software must further that mission. The Senior Design Course does not guarantee specific results or any results, and the Software will be developed only on a best efforts basis. The Software is considered PROOF OF CONCEPT only and is NOT intended for commercial, medical, mission critical or industrial applications.
3. The Senior Design instructor will act as mediator between Sponsor and Team; and resolve any conflicts that may arise.

3.5 COMPENSATION

No compensation will occur in this project.

3.6 CONSULTATION AND REPORTS

1. Sponsor’s designated representative for consultation and communications with the Team Lead shall be Jeff McGough.
2. During the Term of the Agreement, Sponsor’s representatives may consult informally with course instructor regarding the project, both personally and by telephone. Access to work carried on in University facilities, if any, in the course of this Agreement shall be entirely under the control of University personnel but shall be made available on a reasonable basis.
3. The Team Lead will submit written progress reports. At the conclusion of this Agreement, the Team Lead shall submit a comprehensive final report in the form of the formal course documentation at the conclusion of the Senior Design II course.

3.7 CONFIDENTIAL INFORMATION

1. The parties may wish, from time to time, in connection with work contemplated under this Agreement, to disclose confidential information to each other. Each party will use reasonable efforts to prevent the disclosure of any of the other party's Confidential Information to third parties for a period of three (3) years after the termination of this Agreement, provided that the recipient party's obligation shall not apply to information that:
   (a) is not disclosed in writing or reduced to writing and so marked with an appropriate confidentiality legend within thirty (30) days of disclosure;
   (b) is already in the recipient party's possession at the time of disclosure thereof;
   (c) is or later becomes part of the public domain through no fault of the recipient party;
   (d) is received from a third party having no obligations of confidentiality to the disclosing party;
   (e) is independently developed by the recipient party; or
   (f) is required by law or regulation to be disclosed.
2. In the event that information is required to be disclosed pursuant to subsection (6), the party required to make disclosure shall notify the other to allow that party to assert whatever exclusions or exemptions may be available to it under such law or regulation.

3.8 INTELLECTUAL PROPERTY RIGHTS

The claim to the IP is shared between the Sponsor Jeff McGough and the members of the Eye In the Sky Senior Design Team: Colter Assman, Julian Brackins, Charles Parsons, and Alex Wulff.

3.9 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).

3.10 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 TO THIS AGREEMENT NOR THEIR AFFILIATED COMPANIES, NOR THE OFFICERS, AGENTS, STUDENTS AND EMPLOYEES OF ANY OF THE FOREGOING, SHALL BE LIABLE TO ANY OTHER PARTY HERETO IN ANY ACTION OR CLAIM FOR CONSEQUENTIAL OR SPECIAL DAMAGES, LOSS OF PROFITS, LOSS OF OPPORTUNITY, LOSS OF PRODUCT OR LOSS OF USE, WHETHER THE ACTION IN WHICH RECOVERY OF DAMAGES IS SOUGHT IS BASED ON CONTRACT TORT (INCLUDING SOLE, CONCURRENT OR OTHER NEGLIGENCE AND STRICT LIABILITY), STATUTE OR OTHERWISE. TO THE EXTENT PERMITTED BY LAW, ANY STATUTORY REMEDIES WHICH ARE INCONSISTENT WITH THE PROVISIONS OF THESE TERMS ARE WAIVED.

3.11 INDEPENDENT CONTRACTOR

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

3.12 TERM AND TERMINATION

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

3.13 GENERAL

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

3.14 SIGNATURES

Colter Assman	Date
Julian Brackins	Date
Samuel Carroll	Date
Hafiza Farzami	Date
Charles Parsons	Date
Alex Wulff	Date
Jeff McGough	Date

4

User Stories, Backlog and Requirements

4.1 Overview

The following chapter is the overview of the Landing Pad Senior Design project. The intention is to give a generalized top-level report of the project without delving too deeply into any of the technical components. The project's overall goal, stakeholder information, development requirements, and development timeline will all be covered in the following sections. Please refer to Chapter 5 for a more detailed look at the project's design and implementation.

4.1.1 Scope

This section covers project user stories, stakeholder information, and general project requirements.

4.1.2 Purpose of the System

The purpose of this project is to design a system where an Unmanned Ground Vehicle transports an Unmanned Aerial Vehicle to designated flight locations. The UAV will initiate flight off of a landing pad situated on the ground vehicle, which it will autonomously return to after flight is completed.

An Unmanned Aerial Vehicle is capable of collecting sensor data from above a target site and communicate that data back to a home location. The battery life on a standard quad copter does not support extended flight times, so having a UAV fly over a vast expanse of an area is not feasible. The intended purpose of the ground vehicle is to transport the UAV to different flight locations while charging the air vehicle during transportation.

4.2 Stakeholder Information

The following section details the stakeholders in this project, including the Product Owner, the Scrum Master, potential Investors, and the various roles of the developers involved with the project.

4.2.1 Customer or End User (Product Owner)

The product owner of this project is Dr. Jeff McGough. Our team has a weekly meetings with Dr. McGough every Tuesday at 8AM during the first semester of Senior Design and every Thursday at 10AM during the second semester of Senior Design. All project features and product backlog prioritization is conducted by Dr. McGough during these meetings.

4.2.2 Management or Instructor (Scrum Master)

The Scrum Master for the Landing Pad project is Julian Anthony Brackins. The tasks laid out for Julian besides being a standard developer on the team is to facilitate and drive Sprint Meetings, and coordinate meeting times for the team to collaborate and work on the project outside of class. Julian is also in charge

of coordinating the design document for senior design, as well as the sprint recap presentations and poster for the Senior Design Fair.

4.2.3 Investors

As of right now, there currently are no outside investors. The intended goal of this project is to find investors in the search and rescue field that would be interested in using our integrated UAV and UGV system to assist with finding missing persons in situations deemed unsafe for human involvement, including forest fires and other natural disasters.

4.2.4 Developers

With the size of our project, each of our team members has a specified role or multiple roles in the project:

Colter Assman

- **UAV Specialist**. In charge of designing software routine for manual override on the UAV, facilitating fully autonomous flight. Also tasked with general software and hardware component integration on the UAV.

Julian Brackins

- **Scrum Master**. Coordinates team meetings and code reviews, lead author for design document, presentations, and poster.
- **Craft Orientation Specialist**. Using OpenCV to design an object tracking system for the UAV to recognize and navigate towards the landing pad.

Samuel Carroll

- **Navigation Specialist**. Designing an obstacle avoidance routine for the ground vehicle using point cloud data.
- **UGV Specialist**. Assisting with the construction of the new unmanned ground vehicle.

Hafiza Farzami

- **Control Panel Specialist**. Designer for a front-end display panel used to handle manual ground vehicle control and view incoming sensor data from the ground vehicle and aerial vehicle.
- **Craft Orientation Specialist**. Using ROS to design an AR tag tracking routine to determine UAV pose, distance, and orientation in relation to the landing pad.

Charles Parsons

- **Navigation Specialist**. In charge of designing software routine for autonomous path-planning for the ground vehicle.

Alex Wulff

- **Technical Lead**. Oversees various software components of the build environment, including ground vehicle navigation, object detection, and craft orientation homography.
- **Architect**. In charge of building the project's custom API and overall software build environment on both the unmanned ground vehicle and unmanned aerial vehicle, and in charge of ensuring the environments are fully functional on the hardware components.
- **UGV Specialist**. In charge of the construction of the new unmanned ground vehicle.

4.3 Requirements and Design Constraints

A lot of the design constraints for this project stem from our utilization of ROS for our software architecture, which is further detailed in the Development Environment Requirements subsection located on page 15 of this design document. While ROS has been an invaluable tool for the development of the software components of our project, our utilization of the ROS library has restricted this project to devices running Ubuntu 14.04.

Another design constraint our project has to be aware of is the ability to operate all necessary components without internet access. The autonomous UAV is expected to perform all of its sensor data collection and automated landing in locations where internet access is not guaranteed, including the lesser populated Black Hills area

Use this section to discuss what requirements exist that deal with meeting the business need. These requirements might equate to design constraints which can take the form of system, network, and/or user constraints. Examples: Windows Server only, iOS only, slow network constraints, or no offline, local storage capabilities.

4.3.1 System Requirements

The build environment for this project requires an Ubuntu 14.04 operating system.

4.3.2 Network Requirements

The UGV and UAV do not explicitly need to be connected to the internet to operate. On board the UGV is a wireless router which creates a wireless connection between the ground vehicle, the UAV, and potentially another nearby computer for handling the control panel.

4.3.3 Development Environment Requirements

This project utilizes the Robot Operating System, or ROS, which provides libraries and tools for robotic applications using hardware abstraction, device drivers, visualizers, and various other component management. As of the most recent release, ROS Indigo [1], The only stable supported platforms for ROS are for Ubuntu 13.10 (Saucy), Ubuntu 14.04 (Trusty), and Ubuntu Trusty armhf [2]. Experimental ROS versions also exist for Homebrew versions of OS X, Android, Arch Linux, and Debian Wheezy. Because stability and dependency are of the utmost importance for our project, our software environment has been focused specifically on Ubuntu 14.04. All of our software components have been developed on this platform, and as such, all of the configuration instructions enclosed in this document reflect an Ubuntu 14.04 development environment. There are currently no plans to make this project cross-platform.

4.3.4 Project Management Methodology

Because our team is much larger than the standard senior design team in the Computer Science Program at the School of Mines and Technology, team synchronization is a very high priority in order to ensure our project is executed successfully. With the sheer number of hardware and software components expected of this project, a diligently focused project management philosophy must be implemented. The following sub-sections of this document detail the project management process for our team.

4.3.4.a Task Management

For our Senior Design project, our team is utilizing Trello, a free web-based project management application [4]. Our team has utilized Trello for project synchronization as well as a medium for discussing various topics relating to the project. The trello board has cards representing the different topics of our project, organized by category. The following list is an overview of the topic categories found on our Trello board:

[1] ROS jade Turtle, the 8th installment of ROS, is expected to release in May of 2015, right after the completion of this senior design project.

- Code Reviews / Meeting Notes / Misc. Discussion

  A location for meeting notes and general discussion made available to the entire team. Notes collected at our weekly Thursday meetings in conjuction with our established Product Backlog form the basis for our Sprint Backlog.

- Sprint Backlog

  A collection of generalized components not yet being worked on that are required for our project as per the specifications given from our project advisor.

- In Progress

  Tasks from our Sprint Backlog currently being worked on, as well as progress lists and discussions relating to the particular feature or task.

- Ready For Testing

  After completion of a task from the "In Progress" card, the module will be shifted into the testing phase to validate the components before marking the task as complete.

- Complete

  A collection of completed tasks for our project. Each component is added to this category only after successfully making it through the testing phase for component validation.

- Product Backlog

  The product backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product [3]. Our Sprint Backlog is based off of these requirements.

- Deliverables

  After each sprint, a collection of deliverables is sent to our Senior Design professors and Product Owner for progress validation and approval. This includes project prototypes, client presentation slides, the most recent revision of our design document, and a Sprint Report detailing the efforts put in by each member of the Landing Pad team.

- Feedback

  This category contains feedback from the Senior Design instructor, Dr. Larry Pyeatt. The feedback received is based on the content of the project's client presentations given throughout the Fall 2014 / Spring 2015 school year.

- References

  A collection of useful links and documentation used for software configuration or other miscellaneous assistance for the project.

4.3.4.b Sprint Timeline

Please refer to Table 4.1, which details the entire sprint timeline for the UAV Landing Pad senior design project.

4.3.4.c Source Control

The source code for our project is located on a private SVN repository provided by the Mathematics and Computer Science department at the South Dakota School of Mines and Technology. The repository is located at `http://dev.mcs.sdsmt.edu/repos/srdesign1415/robots/trunk/landingpad/`.

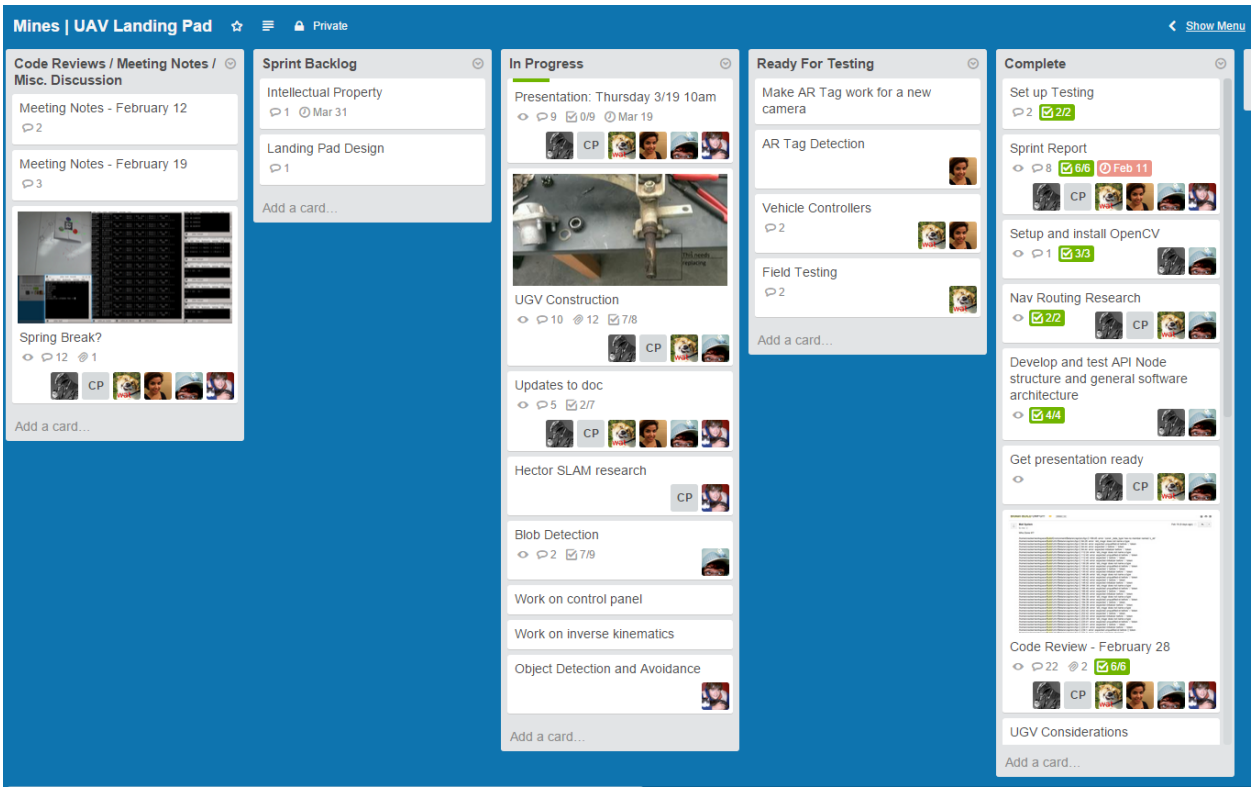

Figure 4.1: UAV Landing Pad Team Trello Board.

4.4 User Stories

The user stories section of this document describes the applications of this project in a real-life scenario point of view. This will allow stakeholders and anyone else interested or invested in this project to get a more explicit understanding of what we strive to accomplish with our fully autonomous system. Our user stores have been derived from design requirement meetings with our product owner, Dr. Jeff McGough. As such, all functional requirements of our system are described in the following sections.

4.4.1 User Story #1

As a member of a search and rescue team, a volunteer will dedicate their time to saving lives, providing emergency assistance to those in need. Although members of search and rescue teams are certainly highly trained to operate in dangerous conditions, some scenarios, such as during massive forest fires in the Black Hills area, have an incredibly high risk of operational failure. Locations such as a large forest fire would be incredibly difficult for a human to navigate through; a lot of initial time would be wasted wandering through the forested area finding missing persons. Prolonged interactions within the forest fire will quickly put the members of the search and rescue team at risk.

4.4.1.a User Story #1 Breakdown

A solution to the aforementioned user story would be to have a supplementary device capable of providing a bird's eye view of the dangerous location. This device would assist with finding missing person in areas of low radio communication and high human risk, therefore lowering the possibilities of casualties in the rescue mission, both missing persons and members of the rescue team.

Our project proposes to use a combination of a UAV and UGV to solve the problem stated in the user story. The UGV will autonomously drive to specified locations near the danger zone. The UAV will deploy

from a landing pad situated on the UGV and collect sensor data of the danger zone from above in an attempt to determine if any missing persons are in the area. After the UAV has completed the flight, it

Task.....	**Start Date**	**End Date**
Sprint 1	**09/15/14**	**10/03/14**
Project Configuration	09/15/14	09/20/14
Ubuntu 14.04 Installation	09/15/14	09/16/14
ROS Installation, Configuration, & Orientation	09/16/14	10/03/14
Preliminary Technology Research	09/15/14	10/08/14
S1 Summary	10/09/14	10/09/14
Sprint 2	**10/13/14**	**10/31/14**
OpenCV Installation & Configuration	10/21/14	10/22/14
Hardware stress test	10/23/14	10/29/14
LED Recognition/Detection	10/22/14	10/24/14
SVO Decision	10/27/14	11/04/14
Repair Drone	10/27/14	11/04/14
Client Presentation	10/21/14	10/23/14
S2 Summary	11/06/14	11/06/14
Sprint 3	**11/11/14**	**11/29/14**
UGV Design Begins	11/13/14	12/05/14
GPS Research	11/11/14	11/11/14
Complete Architecture Design	11/11/14	11/12/14
Hector SLAM R&D	11/14/14	12/01/14
Client Presentation	12/02/14	12/02/14
S3 Summary	12/05/14	12/05/14
Sprinter Break	**12/22/14**	**1/16/15**
UGV Construction	12/22/14	12/29/14
OpenCV Homography	12/22/14	12/23/14
ROS Node Architecture Setup	12/29/14	01/2/15
Testing Environment Setup	12/29/14	01/16/15
Sprint 4	**01/19/15**	**02/06/15**
Platform Design & Development	01/30/15	02/11/15
UAV best case scenario landing	01/30/15	02/11/15
S4 Summary	02/13/15	02/13/15
Sprint 5	**02/16/15**	**03/06/15**
Finish UGV	02/13/15	03/16/15
Finish GPS & NAV	02/13/15	03/09/15
Design Control Panel	02/13/15	03/16/15
Design UAV Flight Systems	03/02/15	03/16/15
Object Tracking (OpenCV + AR)	02/13/15	3/13/15
Client Presentation	03/19/15	03/19/15
S5 Summary	03/20/15	03/20/15
Sprint 6	**03/23/15**	**04/10/15**
Component Integration	03/20/15	03/27/15
Integrate Control Panel	03/20/15	03/27/15
Handover Documents	03/23/15	05/07/15
Platform Implementation	03/30/15	04/10/15
Intensive Field Testings	03/30/15	04/29/15
S6 Summary	04/10/15	04/10/15
Design Fair	04/14/15	04/14/15
All Materials Due for Senior Design	05/08/15	05/08/15

Table 4.1: Sprint Timeline consisting of 6 Sprints and "Sprinter Break"

will autonomously land back onto the UGV in order to recharge as the ground vehicle moves on to the next destination.

While this user story mainly focuses on the system handling a forest fire situation, our system could be applied to various other rescue missions in locations where human involvement from a rescue team should be kept at a minimum to avoid risking the lives of the team. Additional scenarios such as these include, but are not limited to:

- Radioactive Locations
- Trench Rescues
- Mass Casualty Locations
- Water Rescue
- Other Natural Disaster Locations

4.4.2 User Story #2

User Story #2 further elaborates on User Story #1 but is done from the perspective of a member of the search and rescue team now using our UAV and UGV out on the field. As a search and rescue team member being assisted by our UAV and UGV, a minimal amount of user interaction must occur between the team member and UAV/UGV. The search and rescue team's first and foremost priority is to save lives, and any time spent fussing with hardware will be a severe detriment to their mission. Therefore, the UAV and UGV should be able to operate and navigate the area on their own accord.

4.4.2.a User Story #2 Breakdown

This user story describes the necessity of an autonomous system. The ground vehicle will handle path planning through GPS and point cloud data. GPS will determine the various waypoints for the UGV's route. Point cloud data will be utilized to determine how close any large objects are to the ground vehicle. This will allow the UGV to have an understanding of the environment without having a human controlling the vehicle's steering. Event-handling routines will allow the ground vehicle to modify its navigation path to accommodate any unexpected obstacles in its original path.

The user story also provides the specification that the UAV and UGV must be able to communicate with one another. This requirement is accomplished with the ROS environment, which is build with a very extensive yet simple message passing communication system. This publish/subscribe mechanism works well not only for intercommunication between various components on board each vehicle, but also allows synchronization between the UAV and UGV. Because of the asynchronous behavior of the publisher/subscriber system, the UGV can send commands to the UAV to initiate flight once the ground vehicle has reached a destination waypoint.

4.5 Research or Proof of Concept Results

Sprints 1, 2, and 3 involved an extensive amount of research into what different components we would explore in order to accomplish autonomy of a hardware system. For the UGV, obstacle avoidance would be our greatest challenge in our navigation algorithms. For the UAV, autonomously landing the UAV onto the small landing pad situated on the back of our ground vehicle would prove to be a challenge.

4.5.1 UGV - Point Cloud Research

Point clouds are a set of data points in a given (x,y,z) coordinate system, often created by a 3D scanning system. The team's initial research into point cloud data was in order to establish a mapping system by registering a dense point cloud map of the environment and planning navigation and object avoidance based off of the mapped data.

Using an Asus Xtion Live Pro, which communicates to ROS through openni2, point cloud data can be read in and stored as a 2D vector. This vector can be searched to find if any objects were detected within

Figure 4.2: Point cloud data of McLaury Room 108.

a certain distance in front of the vehicle, approximately 4 meters or however far the ground vehicle's object avoidance methods need to be calibrated. Each frame of the point cloud data is analyzed to determine any obstacles coming towards the ground vehicle.

4.5.2 UAV - OpenCV Research

One of the main challenges in designing the autonomous UAV was determining how the aircraft would handle landing onto the ground vehicle after completing a flight. The solution to this challenge was to use OpenCV, an open-source computer vision library. OpenCV would allow us to use object recognition routines to recognize the landing pad, allowing the autonomous UAV to verify the landing pad is within the aircraft's "vision".

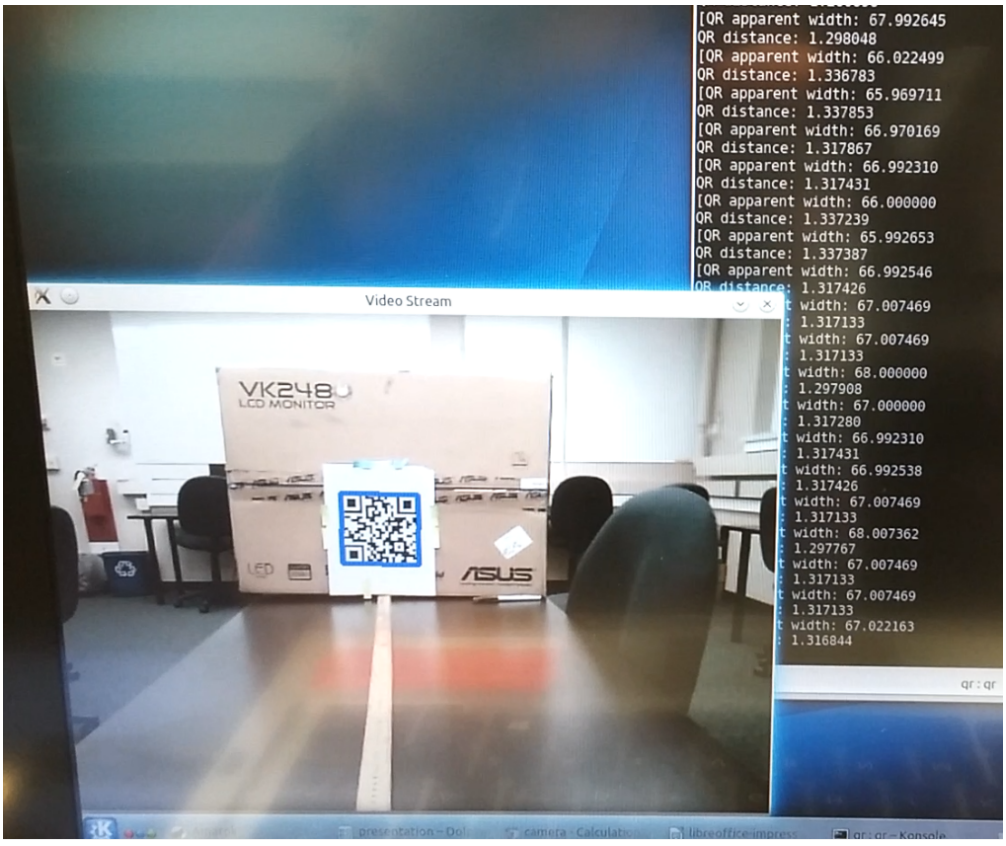

Figure 4.3: QR Code tracking in openCV.

The initial proof of concept test for recognition was to use the Zbar open source barcode library to see if the team could track a qr code with a camera and determine the code's distance from the camera. The qr code was printed and mounted approximately 1.3 meters from the camera, which successfully tracked the code and output the calculated distance based on the camera's focal length and the known size of the printed qr code. The formulas utilized to determine object distance based on camera focal length and the observed size of a known object are detailed in Subsection 5.2.3.a.

The results of this initial research were that the team could certainly utilize openCV as a tracking routine that also returned distance from the tracked object. However, landing the aircraft is still an incredibly complicated and important component of this system, so a redundant system would be desired for returning the aircraft's distance from the landing pad. Therefore, the openCV tracking was shifted away from tracking qr codes and refocused on tracking multiple colored objects in a specified triangular pattern on the landing pad.

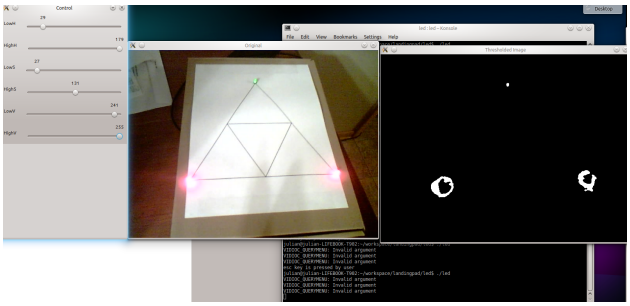

Figure 4.4: LED tracking in openCV.

At one point during Sprint #2, a tracking system using three colored LEDs was investigated. However, complications arose due to the fact that the emissive quality of the LEDs caused the lights to produce a non-uniform color. An artifact of this time of development is evident in the naming conventions of some of the classes in the cv_tracker ROS package. The final design of the landing pad is three colored circles printed onto the landing pad design. Tracking multiple objects introduced multiple distance readings, rather than relying on one potentially unstable distance reading. ROS's AR tag tracking capabilities were also incorporated in conjunction with the openCV tracking system to create a sophisticated and thorough craft orientation and landing routine. Further actualization of this initial research can be detailed in Section 5.2.

5

Design and Implementation

This section is used to describe the design details for each of the major components in the system. As the construction of an automated UAV/ UGV can be broken into the common underlying software components and the separable physical and software differences at each level, this section will be thus divided. In the first section we overview the software, starting with the overall architecture of the system in Section 5.1, and then delving into the various focused components of the project in Sections 5.2 through 5.4. Section 5.6 details the design and construction of the Unmanned Aerial Vehicle, and Section 5.7 details the design and construction of the Unmanned Ground Vehicle.

5.1 Software: Custom API

5.1.1 Technologies Used

The Custom API itself is written in C with the ROS libraries integrated for passing ROS messages. However, the API is designed to handle and interface with the various other technologies used in the landing pad project. The following components that work with the API are listed in the Component Overview.

5.1.2 Component Overview

- Linux OS
- gcc
- python
- R.O.S.
- OpenCV
- SLAM (HectorSLAM, LSDSLAM, PTAM)

5.1.3 Phase Overview

The modular design of ROS made the software environment a sensible choice for our project which incorporates the use of

While it was the focus of this project to use mostly existing architectures to bring about the functionality of UAV and UGV, variability was discovered within the connectivity of these components. As such a custom API was implemented to facilitate the building of custom new functionality and simplify the communication between custom built software and existing code.

5.1.4 Architecture Diagram

Figure 5.1 details the architecture of the Custom API and demonstrates how it is used to facilitate communication amongst the various components in the software architecture.

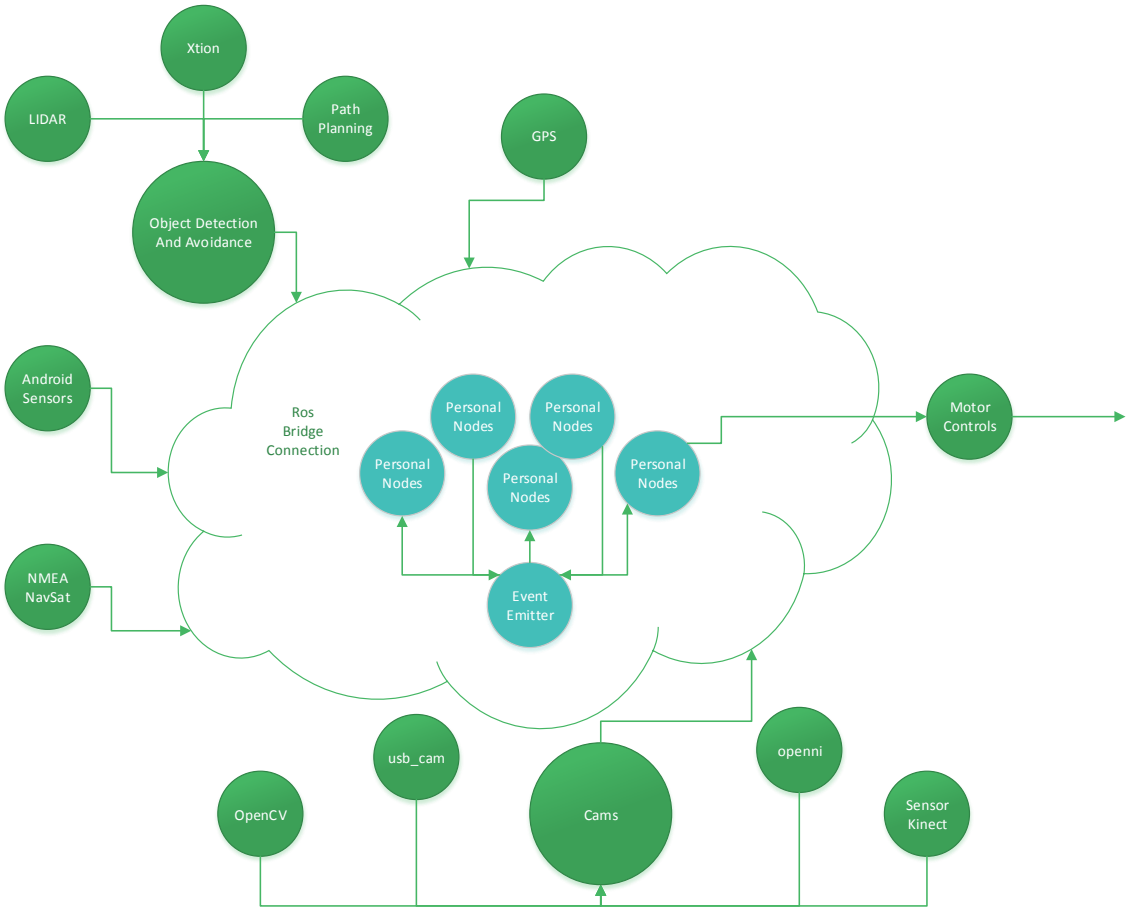

Figure 5.1: Software Architecture for the project build environment

5.1.5 Design Details

The following code segment is a skeleton of the Custom API, which demonstrates how various callbacks are used to communicate messages being sent from various ROS topics at a given time.

```
/*****************************************************************************
 *
 * INCLUDE
 *
 *****************************************************************************/
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <pthread.h>
/*****************************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Callback for general chatter line.
 *
 * @param[in] msg - the message being sent down the topic.
 *
```

```
 ***************************************************************************/

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("Heard: [%s] on Chatter", msg->data.c_str());
}

/***************************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Callback for general gps line.
 *
 * @param[in] msg - the message being sent down the topic.
 *
 ***************************************************************************/

void GPSCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("Heard: [%s] on GPS", msg->data.c_str());
}

/***************************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Callback for errors line.
 *
 * @param[in] msg - the message being sent down the topic.
 *
 ***************************************************************************/

void ErrorCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("Heard: [%s] on Error", msg->data.c_str());
}

/***************************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Callback for other lines... Mostly for debugging.
 *
 * @param[in] msg - the message being sent down the topic.
 *
 ***************************************************************************/

void OtherCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s] on Other Callback", msg->data.c_str());
}

/***************************************************************************
 * @author Alex Wulff
 *
```

```
 * @par Description:
 * Callback for general camera line.
 *
 * @param[in] msg - the message being sent down the topic.
 *
 ********************************************************************************/

void CamCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("Heard: [%s] on Cam", msg->data.c_str());
}

/********************************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Callback for general CV datas.
 *
 * @param[in] msg - the message being sent down the topic.
 *
 ********************************************************************************/

void CVCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("Heard: [%s] on CV", msg->data.c_str());
}

/********************************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Callback for general motor control line.
 *
 * @param[in] msg - the message being sent down the topic.
 *
 ********************************************************************************/

void MotorCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("Heard: [%s] on Motor", msg->data.c_str());
}

/********************************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Callback for SLAM data feeds.
 *
 * @param[in] msg - the message being sent down the topic.
 *
 ********************************************************************************/

void SlamCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("Heard: [%s] on Slams", msg->data.c_str());
```

```
}

/**********************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Callback for simulation data I/O.
 *
 * @param[in] msg - the message being sent down the topic.
 *
 **********************************************************************/

void SimCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("Heard: [%s] on Sim", msg->data.c_str());
}

/**********************************************************************
 * @author Alex Wulff
 *
 * @par Description:
 * Main thread to set up call backs and link topic names as static strings.
 *
 * @param[in] argc - count of args.
 * @param[in] argv - arguments themselves.
 *
 **********************************************************************/
int main(int argc, char **argv)
{
  ros::init(argc, argv, "APINode");

  ros::NodeHandle n,n1,n2;

  ros::Subscriber sub0 = n.subscribe("chatter", 1000, chatterCallback);
  ros::Subscriber sub1 = n.subscribe("Error", 1000, ErrorCallback);
  ros::Subscriber sub2 = n.subscribe("GPSStream", 1000, GPSCallback);
  ros::Subscriber sub3 = n.subscribe("Other", 1000, OtherCallback);
  ros::Subscriber sub4 = n.subscribe("CamStream", 1000, CamCallback);
  ros::Subscriber sub5 = n.subscribe("SlamStream", 1000, SlamCallback);
  ros::Subscriber sub6 = n.subscribe("SimStream", 1000, SimCallback);
  ros::Subscriber sub7 = n.subscribe("CVStream", 1000, CVCallback);
  ros::Subscriber sub8 = n.subscribe("MotorStream", 1000, MotorCallback);

  ros::spin();
  return 0;
}
```

5.2 Software: Craft Orientation

UAV orientation is an important aspect of the Landing Pad project. In order to successfully land the UAV on the ground vehicle's landing pad, the aerial craft must ensure stable landing conditions. As such, our craft orientation software involves two different components for an overly redundant, yet thorough landing procedure.

Due to there being multiple components for craft orientation, this section will detail both components individually, as well as discuss the overlap in the two systems in the Design Details.

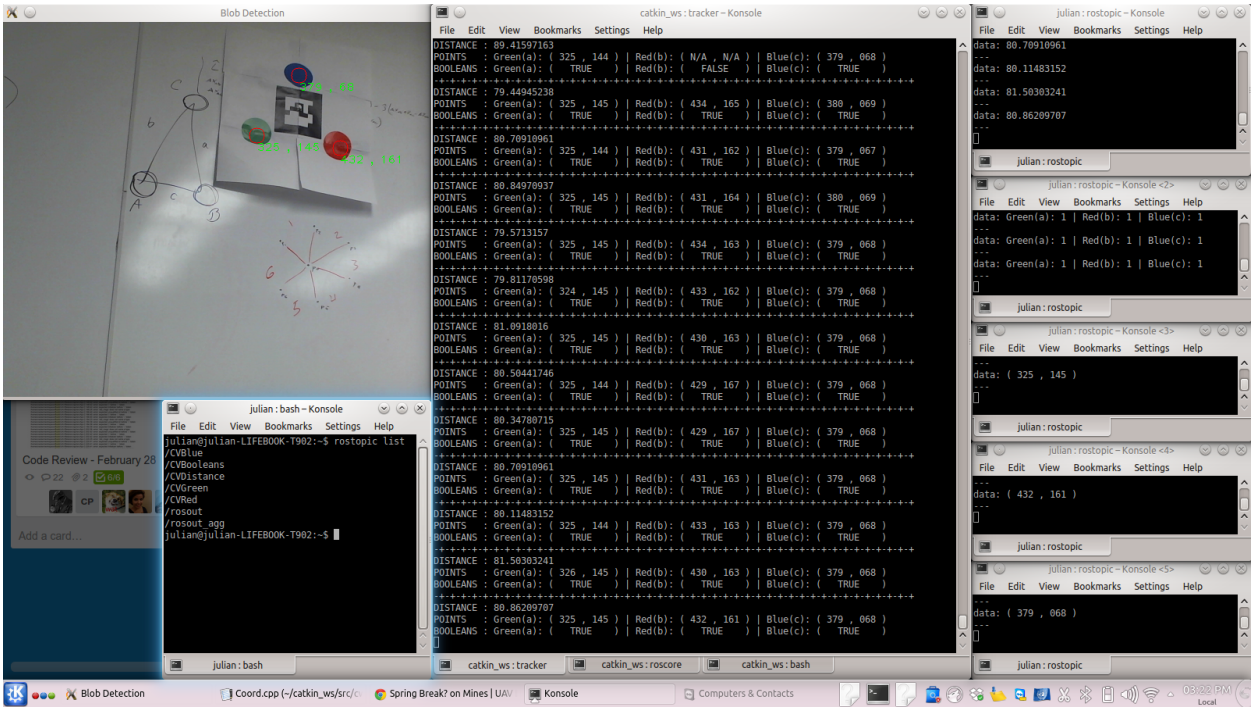

Figure 5.2: cv_tracker tracking three coloured circles in the Mobile Computing Lab.

5.2.1 Technologies Used

The object detection and recognition functions for cv_tracker were written using OpenCV, and the AR tracking is done through the ROS package ar_track_alvar.

5.2.2 Component Overview

- cv_tracker

  custom OpenCV ROS node designed by Julian Brackins and Alex Wulff. Handles craft landing by determining UAV's distance from landing pad based on specified object detection.

- ar_track_alvar

  Existing ROS package integrated into our project by Hafiza Farzami. Handles craft landing and orientation by identifying and tracking the pose of AR tags.

5.2.3 Phase Overview

5.2.3.a cv_tracker

The cv_tracker ros package started out in earlier phases of the Landing Pad project as a proof of concept for identifying object distance using QR Codes. In a camera's video feed, a given subject will appear to be smaller as the camera moves away from that object. Conversely, the item will appear larger the closer the camera is. While this concept is a trivial observation, this change in object size can be utilized to determine the object's current distance from the camera as long as there is prior knowledge of the object's intended size.

A given object's current distance from a camera is the camera's focal length divided by the object's current pixel size from the camera feed, multiplied by the object's original distance. This is expressed in the following equation:

$$d'_k = w_k \times \frac{f}{w'_a} \tag{5.1}$$

where:

- w_k = The object's Known Width[1] in meters.
- d_k = The object's Known Distance in meters.
- d'_k = The change in the object's Distance in meters.
- w_a = The object's Apparent Width in pixels. This is simply the object's current width within the video feed coordinate plane.
- f = The camera's focal length in pixels. The focal length is a measure of how strongly the camera converges or diverges light.

In order to calculate the change in distance, or d_k, the camera's focal length must first be determined using the following formula:

$$f = w_a \times (\frac{d_k}{w_k}) \tag{5.2}$$

The original design in Sprint 2 focused on determining the distance from a QR code by measuring the width of the tag and determining the distance based on the change in size of the code. By Sprint 3, this concept was revised to track coloured circles, arranged in a triangular pattern. This pattern is intended to be implemented as the design for the landing pad. The three coloured circles will be utilized to determine the distance from the landing pad in a similar fashion to the QR code.

With the three coloured circles, an additional layer of redundancy is incorporated into the tracking routine to ensure accurate distance readings. The triangular pattern allows the tracking software to be later improved to allow the UAV to approach the landing pad from a specific configuration. This functionality would come in to play when integrating a charger to the landing pad, which would facilitate autonomous UAV battery recharge upon landing.

5.2.3.b ar_track_alvar

As a way for the UAV to determine the distance away from the UGV and land with the appropriate roll, pitch, and yaw orientation. After Hafiza Farzami joined the team during Sprint 4, the ROS AR tag tracking technology offered by ar_track_alvar was incorporated into the craft orientation and landing procedure to be used in conjunction with cv_tracker in order to provide two separate distance determination methods. The main functionalities of this package used by this project are as follows:

- Generating AR tags of different size, resolution, and data encoding
- Identifying and tracking the pose of individual AR tags

Various colors were experimented with to determine the highest accuracy and the best combination of colors that could be detected from a distance. Ultimately, it was concluded that the black and white AR tags yielded the best results.

[1] The equation only deals with one dimension of an object's size. One could use height rather than width as long as consistency is maintained in the rest of the calculations.

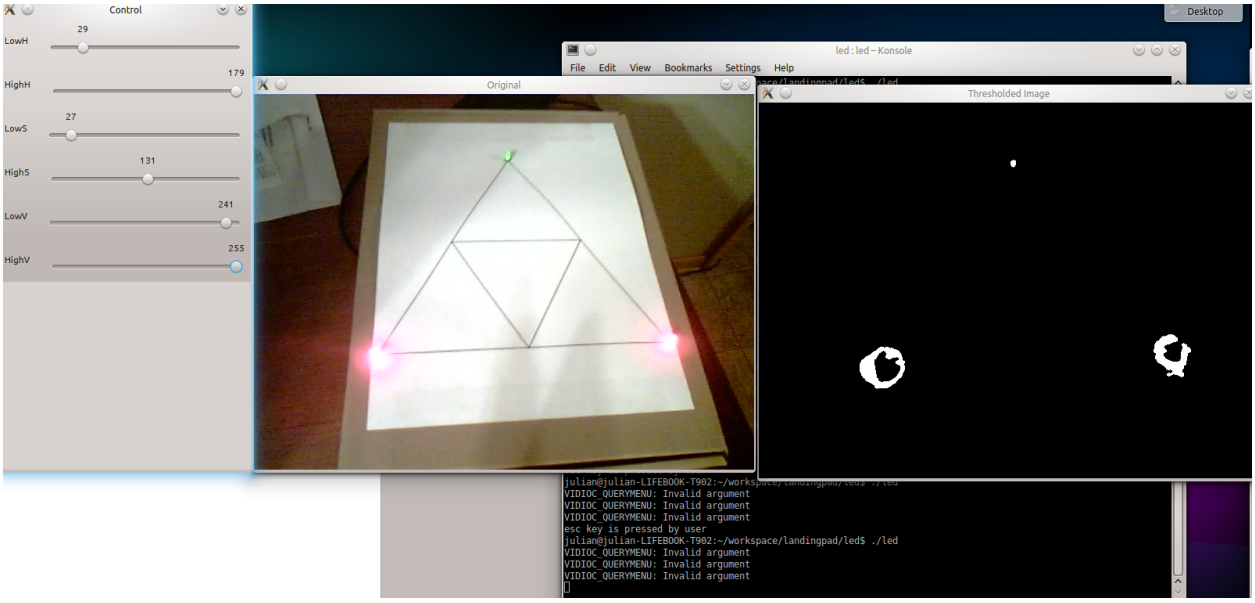

Figure 5.3: Early cv_tracker proof-of-concept using LEDs showing inconsistent color tracking.

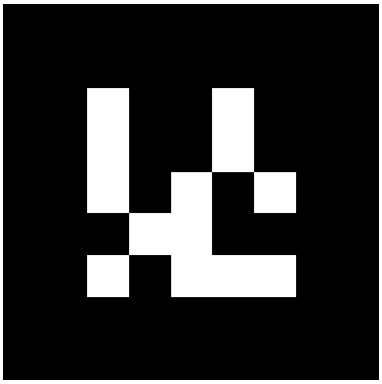

Figure 5.4: An AR Tag.

5.2.4 Design Details

5.2.4.a cv_tracker

The cv_tracker ROS package currently publishes the following topics:

- /CVDistance - this displays the distance from the camera to the landing pad in centimeters.
- /CVBooleans - displays the tracking state of the circles. true = colour is being tracked. false = colour is not being tracked.
- /CVCam - video feed from the camera with the drawn circles to indicate an object is being tracked.
- /CVPoints - displays the (x,y) coordinates of the colored objects being detected. The (x,y) are from the camera feed plane.

These topics should provide adequate data to the control panel and API in order to issue specified commands for reorienting the UAV for successful landing.

5.2.4.b ar_track_alvar

After initializing ROS using roscore and launching the camera, ar_track_alvar publishes the ar_pose_marker topic, which contains information about the tag it is detecting as well as a pose message regarding the tag. The pose message contains a point (x,y,z) and a quaternion (x,y,z,w). The quaternion values are converted into roll, pitch and yaw for the control panel display. Figure 5.5 displays the ar_pose_marker topic output.

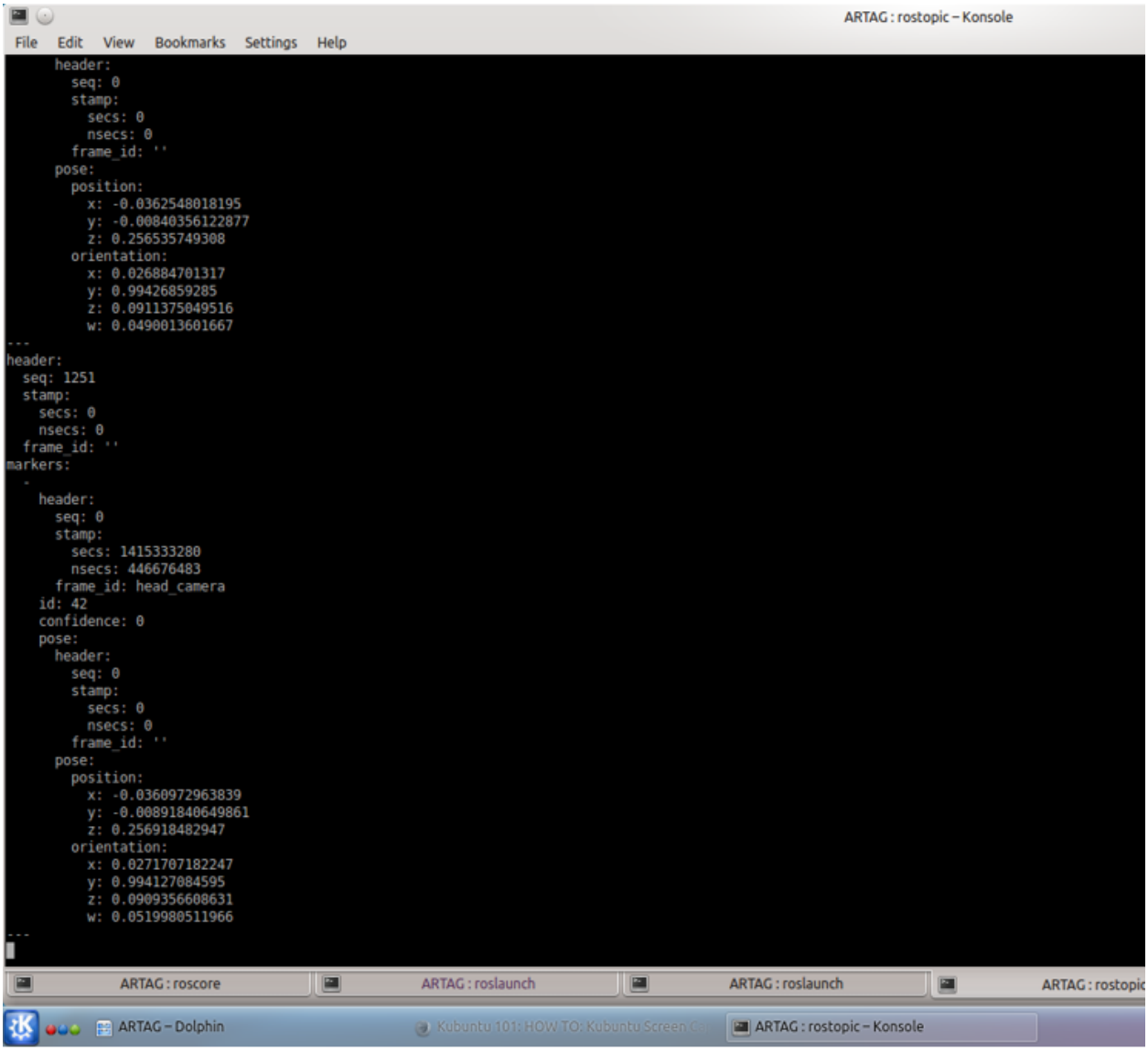

Figure 5.5: Snapshot of the ar_pose_marker topic

5.3 Software: Path Planning

5.3.1 Technologies Used

The path planning is performed by using the Wavefront algorithm, and the obstacle detection is done by reading in point cloud data.

5.3.2 Component Overview

The Wavefront algorithm utilizes GPS information processed by ROS. The GPS is located on the UAV, so the waypoints are sent to the UGV through message passing. Due to budget constraints, only one GPS is present, which is simply shared by the ground vehicle and air vehicle since the ground vehicle will only require GPS waypoints while it is moving. At this point in time, the UAV should be situated on the landing pad attached to the UGV.

Point cloud data is generated using the Asus Xtion.

5.3.3 Phase Overview

During Sprints 4 and 5, Samuell Carroll has been utilizing the Asus Xtion, a professional motion sensor for generating point cloud. Using the program PCL-ROS, the Xtion can take in a point cloud image from a ROS topic and output the information as human readable data in a 640x480 lines of 3 float values and an 11 line header. Skipping the header and first 153,600 lines, the numbers are read in, with each set of three being a specific (x,y,z) point in the cloud. The coordinate plane is measured straight ahead from the Xtion in meters.

The object avoidance routine looks at a minimum distance of 4 meters and anything within that range of the is flagged as a trouble zone for the next image sweep to investigate. Unless the trouble zone was an anomaly, the second sweep will verify the approaching obstacle. If this occurs, the avoidance routine will compare the plane of the object to the drive plane. If the angle between the planes is greater than a given angle, the vehicle will attempt to maneuver around the obstacle. Because the data is being read from a file, the detection code is currently running at a slower rate than originally anticipated. The object avoidance group has been investigating PCL libraries to take the point cloud data and stream the x, y, z coordinates from a ROS topic to streamline the runtime for the object detection routine.

The current object avoidance component can read in 6MB of point cloud data, parse and start looking for objects in approximately 0.5 seconds. Without the files, the team anticipates processing time to be reduced to approximately 0.1 seconds.

5.3.4 Design Details

5.3.4.a Navigation Design

The following code segment details how the wavefront algorithm is used by the ground vehicle, written by Charles Parsons. The UGV performs the wavefront algorithm to fill in values that represent steps from the goal. It starts out by marking the row and column that the goal is represented in, then iterates through the remaining rows using the value from the goal's column as the "seed" value. The algorithm stops evaluating the values once it reaches the starting location. At this point, the path from the starting position and the goal has been established.

```
void wavefrontFill(int row, int col, int** &map)
{
    int i, j;
    bool goal_found = false;
    int goal_index[2] = {0}; //index 0 = i, index 1 = j
    bool start_found = false;
    int start_location[2] = {0};

    //find the indexes of the goal
    for(i = 0; i < row && !goal_found; ++i)
    {
        for(j = 0; j < col && !goal_found; ++j)
        {
            if(map[i][j] == -1)
            {
                goal_found = true;
```

```
                goal_index[0] = i;
                goal_index[1] = j;
            }
        }
    }

    //mark the row and column of the goal with their values
    for(j = goal_index[1] - 1; j >= 0; --j)
    {
        if(map[goal_index[0]][j + 1] == -1)
        {
            map[goal_index[0]][j] = 1;
        }
        else if(map[goal_index[0]][j] == 0)
        {
            map[goal_index[0]][j] = map[goal_index[0]][j + 1] + 1;
        }
    }
    for(j = goal_index[1] + 1; j < col; ++j)
    {
        if(map[goal_index[0]][j - 1] == -1)
        {
            map[goal_index[0]][j] = 1;
        }
        else if(map[goal_index[0]][j] == 0)
        {
            map[goal_index[0]][j] = map[goal_index[0]][j - 1] + 1;
        }
    }
    for(i = goal_index[0] - 1; i >= 0; --i)
    {
        if(map[i + 1][goal_index[1]] == -1)
        {
            map[i][goal_index[1]] = 1;
        }
        else if(map[i][goal_index[1]] == 0)
        {
            map[i][goal_index[1]] = map[i + 1][goal_index[1]] + 1;
        }
    }
    for(i = goal_index[0] + 1; i < row; ++i)
    {
        if(map[i - 1][goal_index[1]] == -1)
        {
            map[i][goal_index[1]] = 1;
        }
        else if(map[i][goal_index[1]] == 0)
        {
            map[i][goal_index[1]] = map[i - 1][goal_index[1]] + 1;
        }
    }
    //mark the rest of the rows and columns using the already marked ones as
    //seed values

    for(i = 0; i < row; ++i)
```

```
    {
        start_found = false;
        //skip the row that the goal is on
        if(i != goal_index[0])
        {
            for(j = goal_index[1] - 1; j >= 0 && !start_found; --j)
            {
                if(map[i][j] == 0)
                {
                    map[i][j] = map[i][j + 1] + 1;
                }
                else if(map[i][j] == -2)
                {
                    start_found = true;
                    start_location[0] = i;
                    start_location[1] = j;
                }
            }
            start_found = false;
            for(j = goal_index[1] + 1; j < col && !start_found; ++j)
            {
                if(map[i][j] == 0)
                {
                    map[i][j] = map[i][j - 1] + 1;
                }
                else if(map[i][j] == -2)
                {
                    start_found = true;
                    start_location[0] = i;
                    start_location[1] = j;
                }
            }
        }
    }
    for(j = start_location[1] + 1; j < col; ++j)
    {
        if(map[start_location[0]][j] == 0)
        {
            if(map[start_location[0] - 1][j] < map[start_location[0] + 1][j])
            {
                map[start_location[0]][j] = map[start_location[0] - 1][j] + 1;
            }
            else
            {
                map[start_location[0]][j] = map[start_location[0] + 1][j] + 1;
            }
        }
    }
    for(j = start_location[1] - 1; j >= 0; --j)
    {
        if(map[start_location[0]][j] == 0)
        {
            if(map[start_location[0] - 1][j] < map[start_location[0] + 1][j])
            {
                map[start_location[0]][j] = map[start_location[0] - 1][j] + 1;
```

```
            }
            else
            {
                map[start_location[0]][j] = map[start_location[0] + 1][j] + 1;
            }
        }
    }
}
```

5.3.4.b Obstacle Detection Design

The following code segment details the findObjects() function written by Saumel Carroll. This function will traverse the image looking for points in a given threshold value. If a point within the threshold, the function will look around the point and try to determine if the point is an object or just an anomaly by flagging trouble spots. If the function decides the point is part of an object, the program will try to find its plane and decide if it is a troublesome object. Troublesome objects are classified as objects within 30 degrees of the drive plane.

```
void findObjects (vector < vector < point > > &image)
{
  int goLeft, goRight = 0;
 // if the object is within the minimum threshold try to find
 //the object plane
 // do this by taking points around it in the i and j positions and if they
 //are about the same distance away assume they are the same object
 // start at multiple rows and see if we have any trouble areas save
 //all trouble
 // i and j zones that we found on the previous sweep and see if the
 //object is still there, if so

  int listSize = _lastSweep.size();
  list<rowcol>::iterator lastSweepIter;
  if (listSize > 0)
  {
    for (lastSweepIter = _lastSweep.begin();
           lastSweepIter != _lastSweep.end(); lastSweepIter++)
    {
      if (!scanArea(image, *lastSweepIter, lastSweepIter))
      {
            // remove from our _lastSweep list
             list<rowcol>::iterator tempIter = lastSweepIter;
             if (!_lastSweep.size() != 1)
     {
                 lastSweepIter--;
                 // decrement by one so we don't skip a possible object.
      }
          _lastSweep.erase(tempIter);
      }
    }
  }

// put in a for loop that scans 8 random rows that aren't equal to each other
// then save the trouble spots (i and j) so we can determine what to do
// If an object is still there avoid if not it moved and keep going
  for (int scannedRows = 0; scannedRows < 30; scannedRows++)
```

```
  {
    // scan random rows looking for troubled spots
    int scanRow = rand() % 240; // randomly choose a row from 0-239
    scanRows (image, scanRow, goLeft, goRight); // scan that row
  }
// add all elements from _currSweep to _lastSweep
  _lastSweep.splice(_lastSweep.end(), _currSweep);
  _currSweep.clear(); // ensure _currSweep is empty
}
```

5.4 Software: Control Panel

5.4.1 Technologies Used

The control panel is a top level graphical user interface written in C++ using Qt. Underneath the graphical layer, the control panel utilizes ROS to read and send data to ROS topics.

5.4.2 Component Overview

The following is a list of features the Control Panel is capable of performing:

- Displays the (x, y) coordinates of the Red, Green, and Blue blobs located on the landing pad.
- Subscribes to the ar_pose_marker topic form AR tags and displays (x, y, z) from the UAV to the landing pad on the UGV in the "Position" field. Quarternion values are also calculated from the AR tags, giving the roll, pitch and yaw data from the aircraft.
- Subscribes to the GPS node in the UAV and displays the latitude, longitude, and altitude.
- Displays whether or not the control panel is connected and running, confirms whether or not a joystick is connected to the computer running the control panel, and displays the height of the UAV in relation to the landing pad.

5.4.3 Phase Overview

The control panel is a centralized tool where it displays all the sensor data and is able to control the UGV via the computer keyboard or a USB Playstation 3 controller. The control panel was initially built by Matt Richard and Scott Logan. This semester, Hafiza Farzami added additional functionality for our project. For instance, the blob detection data, the AR tag data, and distance display have been integrated into the control panel to display the relevant data for the ground vehicle and air vehicle. The GUI and the new ROS nodes for subscribing to the topics published from the new nodes and displaying them on control panel were written and integrated into the existing control panel system.

5.4.4 Design Details

Figure 5.6 displays the control panel after the modifications had been made to it this semester.

5.5 Drive System

5.5.1 Technologies Used

The drive system uses the Ackerman Steering system, and is based off the skid_steer drive, a differential drive node written by former SDSM&T student Andrew Pierson. Further details on how the drive system functions are in Section 5.5.3.

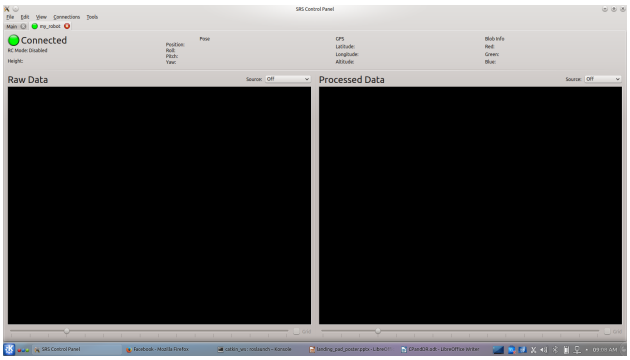

Figure 5.6: Customized Control Panel.

5.5.2 Drive System Diagram

Figure 5.7 is an excerpt from Dr. McGough's Introduction to Robotics book detailing how the Ackerman Steering system functions.

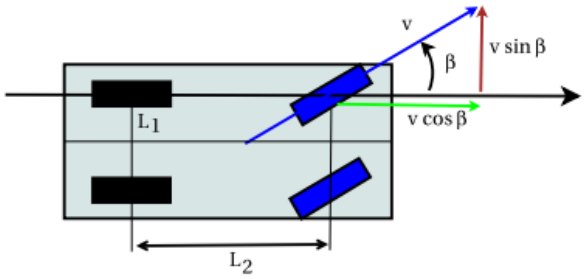

Figure 5.7: Ackerman Steering System

5.5.3 Design Details

The drive system takes angular and linear velocity inputs from a USB controller and uses the linear equations in Figure 5.8 to send the velocity values to the back wheels through the motor controller attached to the back wheels. The turning angle values is sent to the front wheels through the front motor controller.

$$\begin{bmatrix} v \\ \dot{\theta} \end{bmatrix} = r\dot{\phi} \begin{bmatrix} 1 \\ \frac{\sin\beta}{L_2} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \dot{\phi} \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{v}{r} \\ \sin^{-1}\frac{L_2\dot{\theta}}{v} \end{bmatrix}$$

Figure 5.8: Ackerman Steering Formula

5.6 UAV

5.6.1 Component Overview

The following subsection details the components onboard the current UAV:

- AMP 2.6 - flight control board
- 3dr ublox GPS with Compass - compass and gps
- R5800X receiver and TXV582 - video transmitting
- Sony HAD 520 line camera - on board camera
- Spektrum AR7000 - 7 channel receiver
- 915 MHz radio - telemetry with ground station

5.6.2 Phase Overview

The Phase Overview for the UAV will be broken up into the different progress reports returned throughout the project timeline.

- Sprint 2

  The quadcopter used for research and development during the landing pad project was a device inherited by our team from the UAV team at the South Dakota School of Mines and Technology. As a result, there were a number of repairs required to perform before the UAV was in working condition. During Sprint 2, 2 new batteries and battery adapters as well as a new Electronic Speed Controller were purchased in order to get the quad rotor working for testing in later sprints.

- Sprint 4

  By Sprint 4, the first person view camera was set up, allowing video to be streamed from the aerial vehicle to a ground computer. Autonomous flight was also implemented at this time by changing settings on the UAV controller. The vehicle is now able to switch from transmitter control to autonomous control.

- Sprint 5

  Using the ROS package Mavros, a user is now able to send commands to the UAV using ROS. by connecting the UAV to a flight control board, a user is able to issue commands to the UAV by sending the vehicle a list of waypoints to navigate. Using the onboard telemetry unit, the UAV is able to wirelessly link to a ground computer within a mile range.

5.6.3 Design Details

Roscopter is a ROS interface designed for Arducopter utilizing the Mavlink 1.0 interface [1].

The package publishes various sensor topics that our custom API and control panel can monitor:

- /attitude - imu information
- /gps - gps
- /rc - value of current raw rc input
- /state - displays whether the uav is armed or not
- /vfr_hud - airspeed, ground speed, heading, throttle, alt, climb

UAV control instructions can be sent to a vehicle running Roscopter by publishing to /send_rc topic. This topic reads in raw RC channel values between 1000-2000 and uses these values to override the values given from an RC controller.

5.7 UGV

5.7.1 Component Overview

The following components interface with the UGV. Please refer to each specified component's design section for additional details

- Custom API - The ground vehicle, just like the UAV, uses the custom API designed by Alex Wulff detailed in Section 5.1
- Drive System - an Ackerman-based rear drive system detailed in Section 5.5
- Path Planning and Object using Stereo vision - Section 5.3

5.7.2 Phase Overview

The Phase Overview for the UGV will be broken up into the different progress reports returned throughout the project timeline.

- Sprint 3

  By the end of Sprint 3, the frame design for the ground vehicle had been completed, with the intention to start building the vehicle before Sprint 4. At this time, a front-end steering system from a golf cart was purchased for use as the UGV drive system.

- Sprint 4

  Under the supervision of Alex Wulff, the UGV designs reached completion, incorporating the front differential acquired from the golf steering system. The external frame is nearly complete; the design requires additional support gussets and a connection to the power supply and landing pad.

- Sprint 5

  During Sprint 5, Alex Wulff mounted the steering system to the frame and attached the motors, motor controls and tires to the frame. The drive wheels still need to be secured to ensure slip does not occur while propelling the vehicle and to create a better gear interface with the Ackerman differential.

- Sprint 6

  During Sprint 6, the worm gear used for controlling the steering system via a motor was installed using a bracket which secured the motor drive in place. Wiring the motor controls was done by Samuel Carroll, and Hafiza Farzami worked on the motor controls for the drive system. On the way to the design fair, the bracket securing the motor for the drive system was shifted out of alignment. A new, more rigid bracket will need to be installed in order to facilitate steering for the ground vehicle in the future.

6

System and Unit Testing

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

6.1 Overview

Our testing methodology is based around the following seven areas.

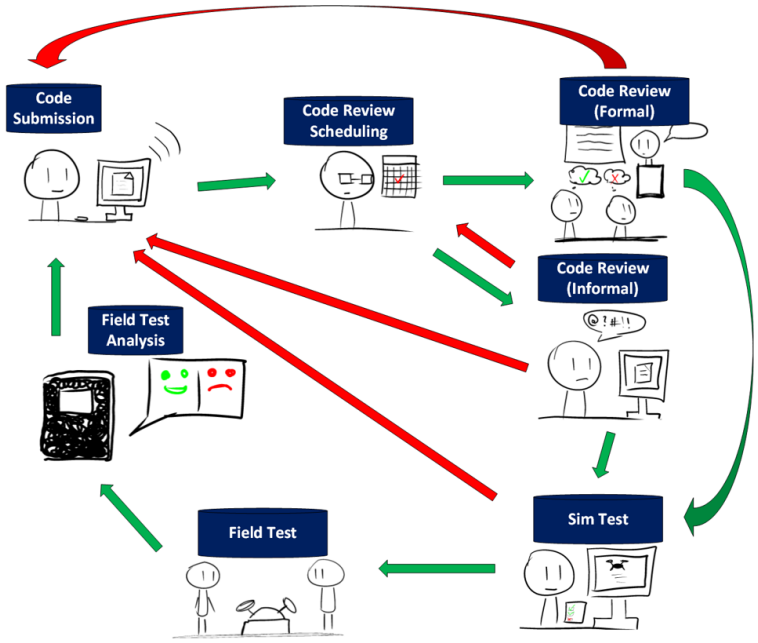

Figure 6.1: Testing Structure

6.1.1 Code Submission

As the development team submits code to the repo, the scrum master will receive an email about the changes. These changes will be based on either the sprint backlog or field test analysis and the changes needed to be made from the results there of. This change will then trigger code review scheduling.

6.1.2 Code Review & Scheduling

The scrum master will look at the commit logs, filter out documentation and determine what code needs to be scheduled for review. Either formal (entire team participation) or informal (reviewed soley by a team

Figure 6.2: Code Submission

member or technical lead) reviews will then be scheduled on team member's availibility. At least two formal reviews will take place during a sprint to provide assurance of quality. If a code review should fail, the code will be returned and kept from the newest build.

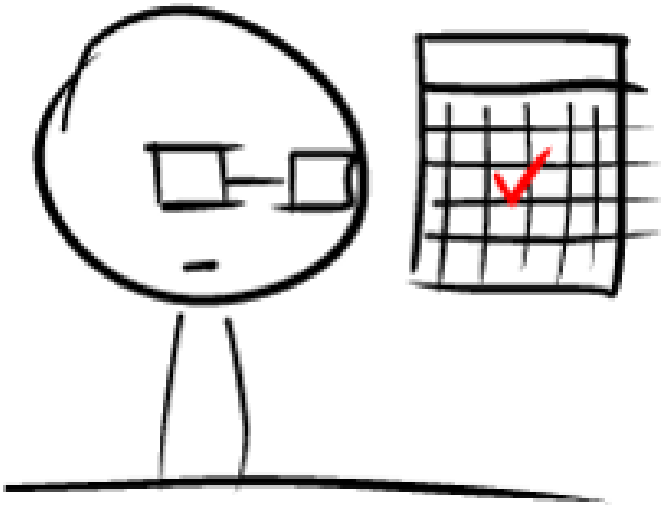

Figure 6.3: Code Review Scheduling

6.1.3 Simulation and Field testing

Upon completion of code review, a Team Member will test the new build on the ROS Simulator, Gazebo. These tests will determine behavior in current software environment and the newest build will not be added to the physical hardware without passing simulation tests. This will happen for all code reviews and can be performed as part of a formal code review. Field testing may not occur immediately after passing simulation testing, however due to scheduling.

During field tests, two team members must be present for documentation purposes. Reports on the progress of the test will be tracked in lab notebooks or other media and will be uploaded to source control upon completion of the tests. Problems in software are to be added to the sprint backlog immdediately if not corrected, and resubmitted for code review at this time. The field testing team members will collaborate with other developers to determine changes for next build.

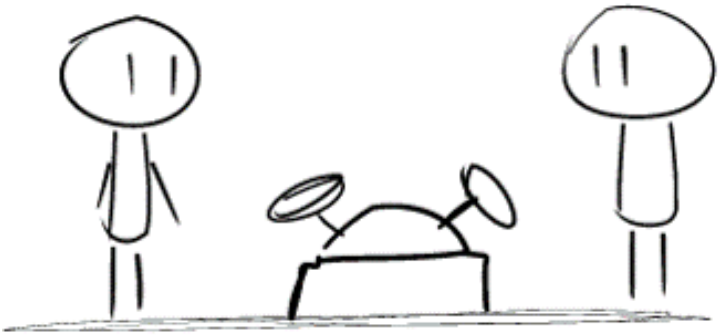

Figure 6.4: Field Testing

7

Development Environment

The basic purpose for this section is to give a developer all of the necessary information to setup their development environment to run, test, and/or develop.

7.1 Source Control

The source code for our project is located on a private SVN repository provided by the Mathematics and Computer Science department at the South Dakota School of Mines and Technology. The repository is located at `http://dev.mcs.sdsmt.edu/repos/srdesign1415/robots/trunk/landingpad/`.

All of the code designed specifically to be put onto the UAV and UGV are located in the Build Directory, which is broken up into Build/UAV and Build/UGV environments. The Build/Environment directory is used for components that are present in both the UAV and the UGV.

7.2 Development Machine Setup

An extensive list of walkthroughs for installing all major components of this project are outlined in the Appendix. Please refer to Section 7.1 for more information on how to build each component for the development setup.

Supporting Materials

The following section contains supporting materials for the design document. This includes industrial experience of the students involved with the project in the form of resumes, as well as the sprint reports which document the phase overviews of each section of the project.

Sprint Reports

7.1 Sprint Report #1

Weeks Sept. 22, 2014 to Oct. 10, 2014:

Teams InSight (Joseph Lillo, Daniel Nix & Elizabeth Woody) and Eye In The Sky (Colter Assman, Julian Brackins, Charles Parsons & Alex Wulff) paired to research common design considerations, goals and architecure dependencies. During this sprint, the following topics were researched:

1. Development Operating System

   Both teams required a foundational operating system for the development of our independent software solutions. For familiarity linux was selected as the developing kernel; the distribution was selected as Ubuntu for its integration with robotics documentation and support for Robotics Operating Systems (ROS). Version 14.04 was selected as it was the most current of the long-term support (LTS) editions available. The attempt to run Ubuntu as a virtual machine (VM) posed too many problems to be viable - webcam access was difficult, general performance was greatly reduced, etc. - so dedicated installations were made on laptops and desktops for development.

2. Common API

   As Eye In The Sky would be utilizing UGV/UAV technologies and InSight would be utilizing a small deployable Odroid-like computers, a common API would greatly benefit the teams. It was found that keeping with Ubuntu and ROS, nearly all separate software dependencies could be installed to this common platform and reduce the need for redundant research.

3. Computer Vision

   Both teams have a need for computer aided recognition of visual queues. To this end, the software package openCV was selected for its integration with ROS based solutions as well as the ability to customize the variations of triggers to be recognized. This software was built into the workspaces used by both team's ROS installations.

4. Verbal & Auditory Interaction

   While not immediately apparent to the needs of Eye In The Sky, verbal interaction was a requirement for the customer experience of InSight. After much deliberation, the package PocketSphynx based on CMU Sphynx was accepted for the light size, functionality, C++ integration and extended customization options that this solution provides over alternatives like SpeechLion or Gillesdemey which required either more computational power or continuous connection for Software As A Service (SAAS) processing.

   For the simplicity and extensive documentation of Text To Speech (TTS), FreeTTS was selected as the method that would be used to route verbal queues on directions, obstacles and other updates to the InSight users.

5. Visual Odometry

   Each team required a computer based tracking system to coordinate visual signals with motion tracking. Both teams have vastly different needs for accurate visual odometry, but a solution was found that will facilitate both, Parallel Tracking And Mapping (PTAM) with optional Semi-direct monocular Visual

Odometry (SVO) enhancement. InSight would use the odometry to precisely integrate navigational maps with verbal queues for the visually impaired while Eye In The Sky will use odometry to validate flight progress on UAV deployments. While there is still some question as to how to integrate PTAM, SVO and ROS accurately. PTAM's setup provides a basis for odometrical readings for each team.

Alternate methods were researched on a per team basis, such as InSight's discovery of San Francisco Airport's assisted navigation system which leverages radio frequency triangulation. After further investigation, however, to keep the number of dependencies minimal, the above list was accepted as the best approach.

After research decisions were made, the remaining time spent through this sprint was split into collaborative development environment setup assistance (led by the InSight team), lab meetings, and weekly progress reporting to Dr. Jeff McGough, project owner.

7.2 Sprint Report #2

Weeks Oct. 13, 2014 to Oct. 31, 2014: Teams Insight and Eye in the Sky have split during this sprint to being design and construction tasks that are unique to each project. This sprint consisted of:

1. Computational Stress Tests

   Alex Wulff has installations of ROS, Ubuntu and all aforementioned nodes completed on various integrated computational units. Cheif among these are the ODroid XU modles (including XU-3 and XU+E). An addtional model will be added to the testing as Dr. McGuough has access to a similiar unit, but that will allow homogeneous proccessing time against the ARM 15 and ARM 7 quad core bays.

   Testing has yet to begin on odometry and tracking as getting display from the XU models requires either an "active" display or a micro-HDMI cable. Neither of these were in the team's possession at the time of this report. Completion and updates to the results will be uploaded within a day or two from the conclusion of this report.

2. LED Construction of Landing Pad.

   Julian Brackins has started construction of a simple LED landing pad using one green and two red LEDs to construct an openCV representation of a plane and allow for orientation. His research indicates that LED's will assist greatly with the detection of the landing pad as they provide a high contrast to the design-on-surface method we'd been considering.

3. Repair Estimates and Timeline Regarding Quadrotor

   We have found the quadrotor in disrepair. Colter Assman has taken the lead to evaluate repairs needed and has made the following decisions to fix it.

   - Buying 2 batteries ($32.50 each).
   - Adapters for the batteries ($10.00).

   We expect that the ESC(electronic speed controller) is defective as well; we have identified spare ESC's so no aditional purchases will be made at this time.

4. Documentation Updates, Extensions - How To's

   Charles Parsons and Alex Wulff have been extending the existing documentation on design. This includes an extension into a "How To" section to describe how to overcome the dificulties encountered in the development process, revisions to all senior design documents and reworking the repository to reflect up-to-date changes. Additions can be found in the appendix section for walkthroughs.

7.3 Sprint Report #3

Weeks Nov. 3, 2014 to Dec. 1, 2014: The focus of sprint three was to finalize research and begin implimentation of our components. This included

1. Computational Stress Tests

   Alex Wulff has been working on finishing stress testing, but was unable to complete the testing due to the software variations between x86 and arm repos and linux distributions.

2. Testing Environments

   Julian Brackins has constructed a basis for testing of ROS structures by combining RViz and Gazebo. This new framework will greatly help in reduced costs regarding repairing UAVs / UGVs after testing in the field.

3. Code Testing

   Julian Brackins has established a testing life-cycle for Eye in the Sky. Here in, formal code reviews and simulation testing will provide the basis for quality assurance.

4. Common Communication and Control Nodes

   Alex Wulff has established a template for an API node structure to reduce the number of nodes that need to be made within each environment (This should help code reuse in UAV and UGV systems).

5. Technologies Solidified The following components for navigation, localization, mapping and imaging have been tested and selected for integration with UAV and/ or UGV: Nav, SVO, Ptam, SLAM & PointClouds.

6. SVN Notification System

   A system to allert the Scrum Master and Technical Lead has been put into place that will allert these team members every time a submission is made to the SVN. This will allow for immediate triggering of code reviews and ensure that quality is maintained.

7. UAV / UGV Designs All design process for UAV and all the frame configuration for the ground vehicle have been completed. Over Sprinter Break, it is our intention to construct the UGV frame and start loading it with the power supplies and motors needed for basic movement.

7.4 Sprint Report #4

Weeks Jan. 19, 2015 to February. 6, 2015: The focus of sprint four was to focus on implementation of our components with existing ROS nodes and developing the autonomous ground vehicle.

1. Computational Stress Tests

   Alex Wulff has reworked stress tests for the new architectures. This was completed and led to the purchases of both two Odroid XU3-Lite modles and the addition of two PcDuino8 units for communication and API support.

2. Blob Detection

   At the beginning of Sprint 4, Julian Brackins started working on an object recognition and tracking program using a USB camera and OpenCv. The tracking system was intended to locate a triangular setup of LED lights, calculate the size of the triangle in relation to the camera feed, and determine the UAV's distance in relation to the known triangle size.

   The program contains an LED.cpp class. This class contains the HSV values needed to detect specific colours. HSV stands for Hue, Saturation, and Value, and is used to track specific colours in OpenCv, rather than the standard RGB model. This LED class allows for multiple lights to be instantiated and tracked, allowing this project to expand to tracking more or less than 3 coloured lights at a given time.

   A complication discovered in this tracking program was that the specific colour of an LED would vary too greatly for reliable tracking due to the nature of how colour is emitted from an LED. Therefore, the design of the landing pad will still use colours for the camera to track, but instead of LEDs, the colours will most likely be painted onto the landing pad. LEDs could still be used on the landing pad to illuminate the coloured circles in low light scenarios.

3. SLAM Research and Implementation

   Charles Parsons has successfully calibrated the camera and compiled both Hector SLAM and LSD SLAM. He is currently working on getting the camera and slam algorithms to work together to build a map. Samuel Carroll was recently added to the Landing Pad team during Sprint 4. In order to get familiarized with the project, he has been setting up his ROS environment and calibrating SLAM components.

4. Common Communication and Control Nodes

   Alex Wulff has created a simplified API node and sub-node structure for the connectivity of new nodes needed for the UAV / UGV, and for all other general communication. This new structure enables a simple message passing interface left in a "Debug" mode for output. All that remains for this to be complete is a launcher file to spin up all nodes in the API and a common logger for the messages to be passed.

5. UGV - Construction & Design

   All design process for UGV and all the frame configuration for the ground vehicle have been completed. Under the supervision of Alex Wulff, the UGV has been designed and sent off to be constructed. Alex also secured a front differential for the ground vehicle. The external frame has been finished aside from additional structural support gussets and the connection to the powersupply and landing pad. Both of the latter should be finished within the next sprint.

6. UAV - Design

   Colter Assman has been tasked with working on the UAV. the FPV (First person view) camera is now working, using the camera currently situated on the quad rotor. The FPV is capable of streaming video to a ground computer. Soldering work was done on the camera wiring and it has been concluded that if the camera stops working in the future, it will need to be completely replaced.

7. UAV - Autonomous Flight

   Over the course of Sprint 4, Colter has been readying the UAV for autonomous flight. By changing settings on the UAV controller, the vehicle can switch from transmitter control to autonomous control.

However, the propeller motors were not firing off in order to fly the vehicle to the intended destination. It was determined that the compass was not facing the proper direction.

7.5 Sprint Report #5

Weeks Feb. 16, 2015 to March 06, 2015: The focus of the penultimate sprint is to finalize the design of each of the main components of our Landing Pad project, with the intent to begin component integration and testing in Sprint 6.

Our Sprint Reports have been partitioned this sprint based on the work and tasks performed by each member of the team:

1. Colter Assman

   Colter has been working on sending commands to the UAV over a wireless connection. He found a ROS package called Mavros, which allows him to send commands to the UAV using ROS. He first connected to the UAV using a wired connection to the FCB (flight control board). When he was first trying to use it, he was getting some error with some commands, such as arming the UAV. After doing some research, he found out that there was a newer version of firmware for the FCB. After getting the newer version on there, he was able to use all commands as expected. The first very useful command he found was sending a GPS way-point mission to the UAV. This allows him to change where the UAV would go when autopilot was activated. After learning how to use the command, he was successful at sending it new way-points.

   The next goal was to get the wireless link to the UAV. The UAV kit was bought last year by the previous group. Lucky for us, they bought it with a telemetry unit. This allows for a wireless link to be made between the UAV and a computer. After some fiddling around and updating of firmware on the units, it was working. So now we are able to send the GPS way-points to the UAV as long as we have the wireless connection. The telemetry unit lists the it can reach a mile, but most people say you will only get 300 - 500m. We still need to test this to see what range we can get. We had a test flight where we used the telemetry to change the way-points.

2. Julian Brackins

   Craft Orientation:

   This Sprint, Julian started the process of reworking the blob-tracking OpenCv system into a ros package. Named cv_tracker, the package is now located in the Build environment. The package requires openCV 2.4.

   The package utilizes a camera to compare the observed size of an object with the known size of that object to determine the distance between the camera and object. Using the three separate sides of a triangle to compare sizes, the package forms a triply redundant calculation of the change in distance observed. The program dynamically adjusts to determining the distance using only one side if one of the blobs stops being tracked at any point during the program execution.

   There are option flags that can be input via command line to determine specific runtime configurations, including whether or not to display data on standard output, or whether or not the ROS topics should be published during runtime.

   The program publishes 5 different ROS topics: /CVDistance, the distance from the blobs to the camera, /CVBooleans, which displays the state of the three blobs, and /CVGreen, /CVRed, /CVBlue, which handle tracking the positioning of each blob.

   The cv_tracker software package is completely ready to begin being integrated with the rest of our software components in Sprint 6.

   Project Management:

   As Scrum Master, Julian has been in charge of drafting the most recent updates to our design document for Senior Design. The Sprint 4 and 5 client presentation was put together by Julian and can be located at the following link: `https://prezi.com/bkwjltymrewr/`.

   Julian has also started preliminary work on the poster that will accompany the team at the design fair in April.

3. Samuel Carroll

Basic object detection: We use a point cloud gathered by an Asus Xtion Live Pro talking with ROS via openni2. We then use pcd files generated by pcl_to_pcd provided by the perception_pcl package. We read in a pcd file skipping the header and the first half of the image (since that should all be above us and irrelevant). The second half of the image is stored in a 2D vector array which we then search 8 random lines of to find any object within a certain distance in front of us (4 meters currently) if we have an object in range we will mark it for the next frame. When the next frame comes in we look around the trouble zone to see if the object is still there if it is we pass it to a plane finding algorithm and compare it's plane to the drive plane. If the two planes are in an acceptable angle of each other we don't mark the object (since we should be able to drive over it) if it's greater than that angle we mark it as an obstacle.

I have also started research (with Alex) into getting the pcd data streaming across ROS so we don't have to use costly file reads.

4. Hafiza Farzami

   Hafiza mainly focused on the control panel that is based on Matt Richard and Scott Logan's code. There were initial complications getting the inherited code figured out. Hafiza has been able to add GUI files to the Widgets directory and their corresponding ROS nodes in the Nodes directory. So far, the control panel is able to get raw data, processed data, GPS information, blob detection information, and ar tag distance. The micro-controller related components are still in progress.

5. Charles Parsons

   Research on Ackermann steering kinematics.

   Looked into using the navigation stack, but cannot because that requires a holonomic robot and our robot is non-holonomic.

   Build map using robot position and goal point. Will make a new map for each new waypoint using the previous waypoint as the starting point.

   Path planning for UGV using wavefront algorithm. The wavefront algorithm takes a grid map, it starts from the goal point and, using the 4 neighbors method, starts counting out from the start point. The nodes directly adjacent to the goal are marked as 1 away. The nodes adjacent to the nodes marked with 1's are marked with 2's. this pattern repeats until the entire map has values that represent how many steps until goal.

   Spent time working with odroid-c1. Can ssh into the odroid to interface with it. Got ROS installed and running and will use that for testing hardware performance for path planning.

   Came to the decision to not use hector slam. Will instead use a point cloud or laser scan for obstacle detection.

6. Alex Wulff

   Simulation & Build Environments:

   Alex Wulff created some additional node topics to assist with simulation. After evaluation of both Gazebo and Vrep, these were abandoned. This however, became the foundation for what was to be a new testing framework. This framework centered around an SVN listener to inspect the Landing pad group repository (every 50 minutes). This listener would notify Alex and Julian Brackins of the changes as Julian (Scrum Master) would have to schedule code reviews) and Alex wanted verification of the system working. As new builds were detected a virtual machine running Ubuntu 14.04 with the minimal ROS install (to mirror live computation on ODroids and PcDuinos) would begin a sequence of actions to make for a Continuous Integration (CI) framework. This consists of updating the VM's local repository, executing a build test against the code set to the build environments underneath the packages directory. Should any builds fail, this test would notify the SVN listener of the break and would report a much cleaned version of the CMake error outputs to describe succinctly the path and all errors within the code as CMake and Catkin_Make found them. Any errors found would be emailed to the whole team with all commit data that is associated with the broken code in the hopes that the offending commit will be fixed by the first available team member if not the person who committed the broken code. This was further enhanced by creating mock-able data sets that could be used to

facilitate unit testing given image, PCD (point cloud data) or GPS signals. Some of this is available for inspection under the repository, but due to the size of these tests, a vast majority have been kept local.

Assistance:

Alex also assisted with research about path planning, SLAM, camera use (Kinect, Xtion, Sterio-Disparity Images) and obstacle avoidance for both Sam and Charles. The research of which resulted in a driver custom tailored to match stereo Logitech cameras and bind them to left and right topic namespaces for further disparity checking, a depth cloud as read out as X,Y,Z coordinates from the Kinect, Xtion or LIDAR for obstacle detection as well as a mathematical model for road (planar) detection and a method by which GPS points can be both used to avoid trickier elements of path planning such as forks and false paths as well as assistance with the method of inserting autonomous waypoints into the UAV. Alex is also working on creating a method of attaining the precise global XYZ offset and twist angle to the landing pad given the design as proposed and implemented by Julian.

UGV:

Alex worked on the UGV throughout this sprint and despite the difficulties encountered with mounting the existing Ackerman steering taken from a golfcart and the frame construction, he has managed to secure a method for attaching motors, motor controls and bike wheels (and tires) to the frame. All that remains for completion of construction is securing the drive wheels to ensure no slip while propelling and create a better gear interface with the Ackerman differential.

Other:

Alex has created a plan to duplicate component usage for ground and air vehicles using an ad-hoc connection between both PcDuino's that will be connected to each device. This process is in test, but it will significantly reduce the cost of separate IMU / GPS components.

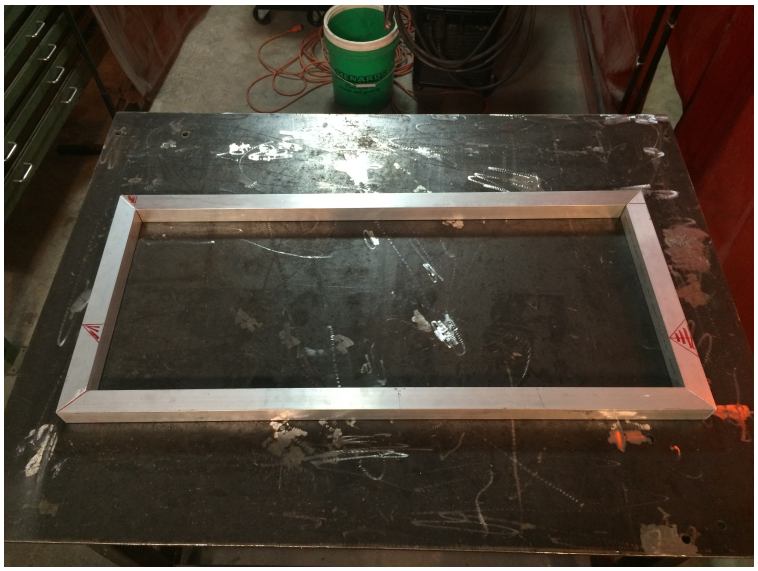

Figure 7.1: UGV Frame in Construction

7.6 Sprint Report #6

Weeks March 23, 2015 to April 10, 2015: The focus of the final sprint is to integrate all of the components of the project.

1. UGV

   During Sprint 6, the worm gear used for controlling the steering system via a motor was installed using a bracket which secured the motor drive in place. Wiring the motor controls was done by Samuel Carroll, and Hafiza Farzami worked on the motor controls for the drive system. On the way to the design fair, the bracket securing the motor for the drive system was shifted out of alignment. A new, more rigid bracket will need to be installed in order to facilitate steering for the ground vehicle in the future.

2. UAV

   Autonomy has been acheived with the aerial vehicle. The vehicle is capable of Auto-takeoff when set to autonomous mode by a remotely connected computer.

3. Control Panel

   The control panel has been integrated into the software architecture and handles the display of the UAV data. The camera feed from the UAV is displayed, as well as the relevant tracking information sent from cv_tracker and ar_track_alvar.

4. Craft Orientation

   The cv_tracker package has been re-structured to read the same raw video feed that ar_track_alvar subscribes to, instead of initializing its own camera feed. This cuts down processor load by sharing an already existing camera feed instead of running two separate feeds simultaneously.

5. Navigation

   The wavefront algorithm for the path planning for the ground vehicle has been completed. the design details are located in Section 5.3 of the design document.

Industrial Experience

7.7 Resumes

Colter Assman

1229 Spearfish Mtn. Ln.
Spearfish, SD 57783
Phone: 605-222-8358
Email: colter.assman@mines.sdsmt.edu

Objective

Seeking for a full-time position at “The Company” as a Software Developer to apply my skills in computer science.

Education

South Dakota School of Mines and Technology
BS, Computer Science Expected June or July 2015
Overall GPA – 2.81

Professional Experience

Software Developer Intern
Golden West, Rapid City, SD
May 2014 – current

- Did web development in JavaScript, using ExtJS Framework, that interacted with backend via Ajax
- Wrote backend code to interact with database (PostgreSQL)
- Worked on both Linux and Windows operating systems
- Made a small unit testing suite for an API
- Worked on a team of two
- Languages used: Python and JavaScript
- Code repository used: Subversion(SVN)

Languages

C++, Python, JavaScript, Java

Activities

Unmanned Arial Vehicle: Programmer / Pilot (2012 – current)

Ultimate Frisbee Club (2012 – 2013)

JULIAN ANTHONY BRACKINS

Address
2717 Lawndale Drive
Rapid City, SD 57702

Contact
Phone: (605) 415-1443
Email: julian.brackins@mines.sdsmt.edu

Education
South Dakota School of Mines & Technology, Rapid City, SD
B.S. Computer Science, 3.20 GPA, Expected May 2015
SDSM&T Orchestra 2nd Violin Section Leader
KTEQ Campus Radio Assistant Station Engineer
NASA Student Ambassador
National Science Foundation Funded Undergraduate Researcher

Employment Experience
Undergraduate Student Researcher June 2014 - Present
Security Printing & Anti-Counterfeiting Technology, Rapid City, SD
NSF-funded Research Experience for Undergraduates (Grant#EEC-1263343)

Research Field 1: Smartphone Application Development for Reading Nanoparticle-Based Inks
Developed an Android-based smartphone application for reading covert barcodes as a method of product authentication. Software features include QR code capture, QR image manipulation, and QR code data storage in a database. Application controls a near-infrared laser remotely in order to upconvert invisible QR codes, making the codes visible for scanning. Will be presenting a prototype reader demo with the rest of the development team at the South Dakota Legislature Poster Session in March of 2015.

Research Field 2: Mass Spectrometry Analysis for the detection of counterfeit Pharmaceuticals
Ongoing research into developing a simulation environment for isotope behavior and abundances.

Developing a Python script and user interface to determine the authenticity of pharmaceutical drugs by comparing natural isotope abundances found in authentic products with the abundances found in counterfeit versions.

Undergraduate Student Researcher January 2013 - April 2013
NASA Johnson Space Center, Houston, TX
Undergraduate Student Research Program (USRP)
Involved with two projects in the Spacecraft Software Engineering Branch:

Member of the Cabin Flight Software Team responsible for designing and developing the Core Flight Software (CFS) system that controls the 2B version of the Multi-Mission Space Exploration Vehicle (MMSEV) cabin. Developed a ground display for monitoring MMSEV Thruster firing activity. Developed a CFS application for interfacing with the MMSEV Potable Water System (PWS) through a Web Relay connection. Assembled and tested the initial On-Board Display Hardware configuration that supports the software test environment.

Member of the Active Debris Removal (ADR) Software Team designing and developing the software system that controls and integrates the components of a prototype vehicle designed to identify and capture orbital debris. Modified the existing software load (C code) to increase efficiency of IMU and pressure transducer processing. Wrote software to cyclically log time-tagged test data for post-test analysis.

Programming Language Experience
C, C++, Python, Java, Visual Basic, Lisp, Assembly Language, PHP

Computer Science Experience
Experience in coding both in Linux & Windows, Computer Graphics (OpenGL), Graphical User Interfaces (C++ with Qt, Java), Robotic Operating System (ROS), Android Mobile Development (Android Studio, Eclipse), Parallel Computing (Pthreads, OMP, OpenMPI), performance analysis (gprof), debugging (gdb), Software Version Control (SVN, Git), NASA Integrated Test and Operations Systems (ITOS), NASA Core Flight Software system (CFS), Technical Writing Skills (Technical Communications I & II, REU Research Paper).

Other Experience
Violinist September 2008 - Present
Black Hills Symphony Orchestra, Rapid City, SD
2nd Violinist. Attend weekly rehearsals for five symphony concerts each year.

References available on request.

Samuel Carroll

samuel.carroll@mines.sdsmt.edu | (605) 391-0602 | 131 Cleveland St. Rapid City SD

Employment History

May 2014 – January 2015: Software Development Intern at CHR Solution Rapid City, SD.

- C# learned and used to build web enabling software
- Parameterized SQL and Dapper with C# for Database access/manipulation
- Windows Installer XML (WiX) self-taught to build installers multiple products.
- Taught others how to use WiX

July 2012 – May 2014: Construction worker at Carroll Construction Rapid City, SD.

- General laborer
- Various tasks involved

Education

South Dakota School of Mines and Technology
Computer Science BS Math Minor
Graduate December 2015
GPA 3.131

Rapid City Central High School
Class of 2011
GPA ~ 4.102

Software Development Tools

Languages (C/C++ ARM Assembly Python Java C#)
Profiler (gprof)
Debuggers (Windows, GDB, DDD)

Source Code Control (SVN, Git, TFS)
IDE's (gedit, VIM, Qt, Visual Studio Android Studio)

Honors and Activities

- Hatterscheidt Foundation scholarship Fall 2011 – Spring 2012
- SDSM&T Orchestra Spring 2012 - Current
- Dean's List Fall 2012
- Public Relations officer for the Unmanned Aerial Vehicle team 2013-2014
- Secretary for the SDSM&T ACM chapter 2014
- President of Unmanned Aerial Vehicle team Fall 2014 – Current
- Secretary for Newman Club Fall 2014 – Current

Hafiza Farzami
(605) 389-2305
hafiza.farzami@mines.sdsmt.edu
151 Kansas City St. Apt. 209
Rapid City, SD 57701

Education

South Dakota School of Mines and Technology (SDSM&T), Rapid City, SD

- B.S. Computer Science and Minor in Mathematics
- Expected Graduation Date: December 2015

Experience

Microsoft Corporation – Redmond, WA

- Engineering Intern (Summer 2014)
  - Worked for the OWA(Outlook Web Access) team

EchoStar Corporation – Englewood, CO

- Engineering Intern (Summer 2013)
  - Wrote code for VOD (Video On Demand) system

Humane Society of the Black Hills – Rapid City, SD

- Volunteer – care for and walk rescue dogs (2015 – Present)

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

- SDSM&T Cultural Presenter (2009-Present)
  - Develop and present items on cultural diversity for various groups in Rapid City, SD
- SDSM&T Summer Youth Programs
  - Counselor (2011-2012)
- Tech Learning Center
  - Academic Tutor (Fall 2011)
- SDSM&T Math Department
  - Academic/Secretary Assistant (Spring- 2012)

Skills

Languages: C/C++, C#, Scala, Python, MySQL
IDEs: Visual Studio, Vim, gedit, Eclipse, QT Creator
Others: Linux, Windows, LaTeX, Multilingual (English, Persian, Hindi/Urdu, Pashto)

Activities

Cultural Expo Committee president (2014 – Present)
Association for Computing Machinery (ACM) member (2015 – Present)
Women in Science and Engineering (WiSE) member (2013 – Present)
Professional Development Team member (2013 – 14)

References

Available upon request

Charles Parsons

121 Kansas City St., Unit 208
Rapid City, SD 57701
605-939-5754
charles.parsons@mines.sdsmt.edu

Education

South Dakota School of Mines & Technology — Rapid City, SD
BS in Computer Science — Expected Graduation: May 2015
GPA: 3.25

Bakersfield College — Bakersfield, CA
Computer Science Major
Fall 1997 – Spring 2002

Skills

Programming Languages	**Database Systems**	**Software/Debugging**
C++ (Proficient), Python, C#, Java	MS SQL Server 2008, MySQL	MS Visual Studio, Eclipse, NetBeans, gdb, valgrind
Operating Systems	**GUI Frameworks**	**Profiling Tools**
Windows, Linux	C++/Qt, Java/Swing	gprof

Projects

- UAV Landing Pad (Senior Design Project) – will have an autonomous UAV take off from and land back upon an autonomous UGV with a landing pad on it.
- ICPC Programming Team 2013 (14th Place in North Central North America Region)
- Windows 8 apps - Match US State Capitals, Alphabet Memory Match, Feed the Hungry Bat
- ACM SVN Repository Management Project 2013-14
- IEEE Robotics Competition-Team 2 Software Lead 2013-14
- Android App – Custom Data Tracking App (Team Lead)

Professional Experience

South Dakota School of Mines & Technology — August 2013 – Present
Rapid City, SD
Tutor - Tech Learning Center
- Tutor students in Computer Science, Physics, and Math

Sencore, Inc. — May 2014 – August 2014
Sioux Falls, SD
Software Engineering Intern
- Improved the calibration application for the DTU-236A, calibration throughput went from 40% to 95%
- Created a windows installer package for RFXpert using WiX.
- Gave an info-session on creating WiX installer packages.

Honors & Activities

Dean's List (Spring 2012, Fall 2012), Student Chapter of ACM (President 2014), Robotics Club

Alex Wulff

404 Denver St, #301
Rapid City, SD, 57701, 970.640.3957,
hsim1912@gmail.com

Education

South Dakota School of Mines and Technology (SDSM&T):

- Computer Science (BS) / Mathematics minor – May 2015
- GPA: 2.9 (SDSM&T)

Past Coursework:

Basic Programming, Data Structures, Programming Languages, Assembly (ARM 11), Parallel Computing, GUI/OOP, Mobile Computing, Database Management Systems, Analysis of Algorithms, Operating Systems, Computer Organization and Architecture, Theory of Computation.

Current Coursework:

Fall 2014: Computer Graphics & Senior Design (Automated UAV/UGV for Search & Rescue applications).
Spring 2015: Natural Algorithms, Differential Equations, Senior Design (Continued).

Related Work History

Software Developer Intern – CHR Solutions, Rapid City, SD - 08/2014 – Present
Developed modified build solutions leveraging existing TFS architecture and continuous integration. Assisted with updates to legacy and current billing software solutions.

Software Engineer Intern – Amazon, WA – 5/2014 – 8/2014.
Assisted in creation of massively scalable architectures leveraging existing Amazon Web Service components (EC2, S3, CloudFront, etc.) to create a virtualized windows solution available on IOS, Android, Windows and Mac hosts.

Research Assistant – SDSM&T, SD – 11/2012 – Present.

- Assisting with Advanced Materials Processing (AMP) in the research, categorization and elimination of wormhole defects found in friction-stir welding. Developed automated counter agent to resolve defects before they form.
- Assisting the Math/Computer Science department with development of a highly scalable parallel research program for multi-dimensional density estimation using wavelets. Usable in signal analysis, atmospheric sciences and security systems.

Software Developer Intern – Innovative Systems, Rapid City, SD - 05/2013 – 05/2014
Developed GUI and back-end for telepohony / IPTV client's servers and set-top boxes. Also assisted with the development of test facilities to help in mass testing and performance analysis of set top boxes and system infrastructure.

Technical Skills

- Programming Languages: C/C++, Java, C#, LISP, ARM11.
- Scripting: Bash, Python 2/3.
- Mark-Ups & GUIs – Xml, Xaml (WPF 4) & QT.
- IDEs / Tools: Linux, Visual Studio 2010/2012, Eclipse, Netbeans, Android Development Kit, Gcc/G++/Gdb, SCONS, Jenkins.
- Commenting methods: Javadocs, Python net-strings, Astyle and Doxygen plugins.
- Profiling and Algorithm Development: (Gprof, Valgrind, timings), Microsoft Visio, VSG (Avizo) and Matlab compilers.
- Source Control – SVN, GIT, and TFS.
- Parallel Extensions – P-threads, B-threads, OpenMP, MPI, (Cilk/++).
- Operating System familiarity: Linux (Ubuntu, Gentoo, BackTrack, Fedora, Arch), Windows (8, 7, & XP), OsX (10.3 – 10.6). Virtual (Xen, ESxI, Virtual Box, and VMware – workstation, hypervisor).

Achievements / Activities

- Association of Computing Machinery (Spring 2012 to Present).
- Placed 84th at 2012 Regional Inter-Collegiate Programing Competitions.
- Assisted MET / GEO department with teaching courses on XCT and VSG computerized imaging systems.
- Have four research papers either printed or to be published in academic journals.
  Copies available upon request

Relevant Independent Projects

- Development of clothes matching app for Android/ Windows phones with database and GUI (in progress). Will leverage SQL lite databases for both internal and server managed collection of clothing and will coordinate clothing using an approach similar to LINQ.
- Development of personalized “Git-FS” for management of virtual disks. This system uses a change state mapping to update only modified sections of virtual operating systems and will include a convenient front end. Will leverage a C++ server for efficiency and a portable gui and possibly status app for Windows Phone 8 and Android.
- Experimentation with Chess AI-mapping in distributed parallel environment.
- Participation in South Dakota’s Research Experience for Under-graduates (REU), with Micro-CT Scans, 3D extrapolation, and software development and assistance.
- Establishing a method to correctly identify and reproduce fossils that are entirely enclosed by concretions by means of Micro-CT and 3D printing.
- Assistance in graduate-level course leveraging techniques of Micro-CT imaging, image reconstruction and mathematical analysis of various samples.
- Mp3 to sheet music generation.

  Code samples available upon request.

Appendix

7.1 Walkthroughs

Tool Installation Walkthroughs
These instructions assume you are using Ubuntu 14.04.

7.1.1 Installing ROS Indigo

ROS Wiki Page:

http : //wiki.ros.org/indigo/Installation/Ubuntu

1. Setup your sources.list

   sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main"

   > /etc/apt/sources.list.d/ros-latest.list'

2. Set up your keys using:

   wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key

   -O - — sudo apt-key add -

3. Update your system using:

   sudo apt-get update

4. Install ROS Indigo using:

   sudo apt-get install ros-indigo-desktop-full

5. Initialize Rosdep using:

   sudo rosdep init

   rosdep update

6. Setup your environment using:

   echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc

   source ~/.bashrc

7. Get ROS install using:

   sudo apt-get install python-rosinstall

7.1.2 Creating a Catkin Workspace

Catkin Workspace Wiki Page:

http : //wiki.ros.org/catkin/Tutorials/create_a_workspace

1. Source the setup.bash file in your ROS directory by typing the following command in a terminal window:

   source /opt/ros/indigo/setup.bash

2. Make the Catkin Workspace directory and the source subdirectory using:

   mkdir -p catkin_ws/src

3. Change directories into catkin_ws using:

   cd catkin_ws/src

4. Initialize your catkin workspace using:

catkin_init_workspace

5. Change directories back into your catkin workspace directory using:

   cd ..

6. Run catkin_make command using:

   catkin_make

7. Source the new setup.bash file that was created within your catkin workspace using:

   source devel/setup.bash

7.1.3 Installing usb_cam node

1. Clone github repository for usb_cam node into /catkin_ws/src using:

   git clone https://github.com/bosch-ros-pkg/usb_cam.git

2. Build usb_cam node using the following command in the /catkin_ws/ directory:

   catkin_make

7.1.4 Installing PTAM

PTAM Wiki Page:

http : *//wiki.ros.org/ethzasl_ptam*

1. Change directories into catkin_ws/src using:

   cd /catkin_ws/src

2. Clone the ethzasl_ptam repository from github using:

   git clone git://github.com/ethz-asl/ethzasl_ptam.git ethzasl_ptam

3. Get necessary libraries for PTAM using:

   sudo apt-get install freeglut3 freeglut3-dev libblas3 libblas-dev liblapack3 liblapack-dev

4. Build PTAM using the following command in your catkin_ws directory:

   catkin_make

7.1.5 Installing OpenCV

OpenCV Online Reference Manual:

http : *//docs.opencv.org/*

1. Ensure you have the essential build tools needed for OpenCV:

   sudo apt-get install build-essential cmake pkg-config

2. Install required Image I/O Libraries:

   sudo apt-get install libjpeg62-dev libtiff4-dev libjasper-dev

3. Install gtk highgui Libraries:

   sudo apt-get install libgtk2.0-dev

4. Install video Libraries:

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

5. Download and extract OpenCV from the following sourceforge location:

   $http://sourceforge.net/projects/opencvlibrary/files/$

6. Create a build directory within the unzipped file and enter into it:

   mkdir _build

   cd _build

7. Initiate the build process with the appropriate flags:

   cmake -D CMAKE_BUILD_TYPE=RELEASE -D MAKE_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 ..

8. Compile the built source code:

   make

9. Install OpenCV:

   sudo make install

7.2 Acknowledgement

Thanks to:

Matt Richards, original designer of the Control Panel used in this project who also assisted with integrating the program into our API.

Allen Holmquist, who assisted Alex Wulff with the welding of the 6061-T aluminium.

An SDSM&T Student who wishes to remain anyonymous, who performed the machining of the UGV components.

Ian Markon, who assisted with welding.

The inSight Team, for assistance with learning ROS

Devin Kroeber, who helped with wiring the ground vehicle.

Ian Carlson, who also helped with wiring the ground vehicle.

Caleb Jamison, who helped with ROS and vehicle wiring.

Leslie Lamport, for inventing LaTeX.

Without these individuals, our project would not be anywhere near where it is now, so we extend our sincerest gratitude.

Bibliography

[1] ROSCopter. Ros interface for arducopter using mavlink. `https://code.google.com/p/roscopter/`. [Online; accessed March 16, 2015].

[2] ROS.org. Ros indigo installation instructions. `http://wiki.ros.org/ROS/Installation`, 2014. [Online; accessed March 14, 2015].

[3] Scwaber and Sutherland. *Scrum Guide*. Scrum.Org and ScrumInc, 2014. Available at http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf.

[4] Fog Creek Software. Trello. `https://trello.com/`, 2011. [Online; accessed March 14, 2015].