Group 44 - ARC

Tech Review

Senior Capstone Project

Oregon State University

Fall 2016

Tao Chen, Cierra Shawe, Daniel Stoyer

CONTENTS

1 Introduction 4

2 Vision System 5

2.1 Stereo Vision . . . 5

2.2 IR Camera Systems . . . 5

2.3 LiDAR . . . 5

2.4 Vision Choice . . . 6

2.5 References . . . 6

3 Sensors 7

3.1 u-blox Neo-M8N GPS with Compass . . . 7

3.2 PXFMini . . . 7

3.3 UM7-LT Orientation Sensor . . . 7

3.4 Sensor Choices . . . 8

3.5 References . . . 8

4 System Control and Data Processing Unit 9

4.1 Intel NUC . . . 9

4.2 UP Board . . . 9

4.3 Raspberry Pi 3 . . . 10

4.4 System Control and Data Processing Unit Choice . . . 10

4.5 References . . . 10

5 Image Analysis Software 11

5.1 Robotics Operating System . . . 11

5.2 GTSAM . . . 11

5.3 Pushbroom Stereo . . . 12

5.4 Image analysis choice . . . 12

5.5 References . . . 12

6 Telemetry Radio Communication 14

6.1 3DR 915 MHz Transceiver . . . 14

6.2 RFD900 Radio Modem . . . 15

6.3 Openpilot OPLink Mini Ground and Air Station 433 MHz . . . 15

6.4 Telemetry radio choice . . . 15

6.5 References . . . 16

7 User Interface 17

7.1 QGroundControl . . . 17

7.2 Tower/DroneKit-Android . . . 17

7.3 LibrePilot . . . 18

7.4 User interface choice . . . 18

7.5 References . . . 18

8 Control System 20

8.1 Motor Control . . . 20

8.2 Servo Control . . . 20

8.3 Probabilistic Analysis for Motion 21

8.4 Motion Model 22

8.5 references 22

9 Path Planning 23

9.1 Global Path Planning 23

9.2 Local Path Planning 23

10 Other Algorithms 24

10.1 Obstacle Avoidance Algorithm 24

10.2 Parallel Parking Algorithm 24

10.3 References 25

1 Introduction

This technology review covers a subset of technologies needed for use with the Autonomous RC (ARC) system. Each team member researched three technologies and provided three alternative solutions to those technologies along with their choice for which would be selected for use in the ARC system. This document presents the findings of each member with explanations of the technologies researched and provides clear reasoning for the selections made.

Cierra Shawe researched and wrote the sections on Vision System Options, Sensors, and System Control and Data Processing units.

Dan Stoyer researched and wrote the sections on Image Analysis Software, Telemetry Radio Communication, and User Interface.

Tao Chen researched and wrote the sections on Control System, Path Planning, and Other Algorithms.

2 Vision System

For autonomous operation, vision systems are critical. The three main options include stereoscopic cameras, Infrared (IR) based systems such as Microsoft's Kinect, and Light Detection And Ranging (LiDAR) vision systems. With the exception of some forms of LiDAR, all of these methods require what is called a disparity map, which creates a 3D image of the surface, that can be used for telling which objects in an image are closest or farthest away.

2.1 Stereo Vision

Stereo-vision uses two different cameras to create disparity maps in order to create a sense of depth. This is similar to how our eyes work. The biggest benefit to a stereoscopic camera system is the ability to detect objects outdoors, as the cameras are able to function within the visible spectrum and aren't affected by ultraviolet (UV) light. IR LEDs can also be used in order to illuminate an area at night, also allowing for night-time navigation. Another pro to stereoscopic vision is the cost is relatively low, as cameras can be obtained for under $10 a piece. One of the challenges of stereo vision is the computational power required. Another is clarity of the disparity map without post processing of images, which makes real-time operation more difficult[1].

OpenCV [2] contains many examples of how to configure and process stereoscopic images and is one of the largest vision resources. It can be used to synchronize and create the disparity map, which can then be used by another part of our system for decision making.

2.2 IR Camera Systems

Using an infrared laser point map projection, these cameras are able to tell the disparity between the points, which helps in creating disparity maps. The most popular example of an IR camera system is the Microsoft Kinect. A big advantage to IR camera's is the ability to function in low and non-natural lighting conditions, due to using the infrared spectrum, rather than only using the visible spectrum. The biggest problem with IR cameras, the functionality is greatly reduced outdoors, due to the massive amount of infrared waves from the sun. [3] IR cameras don't meet our requirement of being able to use the vision system reliably outdoors.

2.3 LiDAR

LiDAR works by using laser pulses to detect range. By detecting different pulse signatures, it is able to take very precise distance measurements. LiDAR is a great way to form point clouds, however, the cost makes the product unreasonable for most people. A basic SICK LiDAR unit costs around $2,000 USD for a unit with 10m accuracy [4].

A 2D RPLiDAR module is a lower cost alternative, at $449, however, the manufacturer says that it will not perform well in direct sunlight, due to using IR lasers. The RPLiDAR only has accurate measurements up to 6 meters [5].

A third option for LiDAR has yet to come to market. The Sweep LiDAR unit by Scanse will be released in January of 2017 [6]. The Sweep unit claims to be a LiDAR unit available for all. Costing $349, the unit has the ability to scan 360 degrees, create points up to 40m away, and is able to function in ”noisy” environments, such as outdoors. The Sweep’s distance capabilities and ability to be used almost any lighting condition, including outdoors, would make this a great candidate for our project. The caveat to the Sweep is it will not be released until January.

The RPLiDAR LiDAR system that would be available to our group, would not meet the requirement of being able to function consistently in an outdoor environment, and others do not qualify as commodity hardware.

2.4 Vision Choice

Due to the need to be able to navigate in outdoor environments, our team will start out attempting to use stereoscopic imaging as our primary vision system. We will do this using the OpenCV library to analyze the images and create the disparity map that can be used for other purposes, such as obstacle avoidance. If we have the computational power to post-process disparity maps in real time, we will attempt to do so.

If time allows, given we are able to obtain a unit, our team could research the use of the Sweep LiDAR system for navigation in place of stereoscopic imaging.

2.5 References

[1] M. Brown, D. Burschka and G. Hager, ’Advances in computational stereo’, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 8, pp. 993-1008, 2003.

[2] Docs.opencv.org, ’OpenCV: OpenCV modules’, 2016. [Online]. Available: http://docs.opencv.org/3.1.0/. [Accessed: 13- Nov- 2016].

[3] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter and R. Siegwart, Kinect v2 for Mobile Robot Navigation: Evaluation and Modeling, 1st ed. Zurich, Switzerland: Technical University of Munich, Autonomous Systems Lab of ETH Zurich, 2016. [Online]. Available: http://e-collection.library.ethz.ch/eserv/eth:48073/eth-48073-01.pdf. [Accessed: 15- Nov- 2016].

[4] Sick.com, ’2D laser scanners’, 2016. [Online]. Available: https://www.sick.com/. [Accessed: 15- Nov- 2016].

[5] Robotshop.com, ’RPLIDAR A2 360 Laser Scanner’, 2016. [Online]. Available: http://www.robotshop.com/en/rplidar-a2-360-laser-scanner.html. [Accessed: 15- Nov- 2016].

[6] Scanse.io, ’Meet Sweep. An Affordable Scanning LiDAR for Everyone.’, 2016. [Online]. Available: http://scanse.io. [Accessed: 14- Nov- 2016].

3 SENSORS

Sensors such as encoders, global positioning systems (GPS), and inertial measurement units (IMUs), in addition to the vision system, are key to helping a robot navigate within its environment. The main sensors needed are a GPS and IMU unit. A GPS and compass unit is needed to locate the vehicle on a map. An IMU is needed to measure acceleration, rotation, and velocity.

Other sensors that we will consider, but are not covered in this review, are infrared or sonar sensors for distance and encoders to track wheel rotation. These items will be added on an ”as needed” basis.

3.1 u-blox Neo-M8N GPS with Compass

The u-blox Neo-M8N GPS with Compass will be used to help the vehicle navigate to a point on a map and provides a built-in compass. [1] The biggest pros of the Neo are:

- Availability - the Neo is currently in hand
- Usage - libraries exist and the PXFmini is pre-configured to work with the Neo
- Detached module - the Neo can be placed away from our motors and other hardware to minimize electromagnetic interference
- Price - the Neo only costs $23.87 (Amazon.com as of 10/16/2016)

The only con is that the Neo uses half of the ports on the PXFmini (a UART and I2C), leaving only one I2C port left for other usages. One solution to this problem is using an I2C multiplexer board in order to extend functionality if needed.

3.2 PXFMini

Erle Robotics includes a triaxial gravity sensor, gyroscope, and digital compass, as well as a barometer, temperature sensor, and an analog to digital converter(ADC) on the PXFmini [2]. The gravity sensor, gyroscope, and digital compass are contained on a chip called the MPU-9250, made by InvenSense [3]. As there are no added components, due to the PXFmini already being used for handling PWM and sensor communication, there is no added monetary cost after the unit has been purchased. The cons to the PXFmini is that the sensors are more likely to be affected by electromagnetic interference, which could produce unpredictable readings from the sensors. There is also minimal data on the typical accuracy of the internal sensors, which makes it difficult to determine whether the PXFmini will be precise enough for our application.

3.3 UM7-LT Orientation Sensor

The UM7-LT Orientation Sensor (UM7) IMU is produced by Red Shift Labs and ”combines triaxial accelerometer, rate gyro, and magnetometer data using a sophisticated Extended Kalman Filter to produce attitude and heading estimates”.

The board is comparable in size to a US quarter. The specifications for the UM7-LT are as follows [5]:

- +/- 2-degree typical static pitch/roll accuracy
- +/- 4-degree typical dynamic pitch/roll accuracy
- +/- 5-degree typical static yaw accuracy
- +/- 8-degree typical dynamic yaw accuracy
- 0.5-degree angle repeatability
- 0.01-degree angular resolution

The pros of the UM7 include the ability to take in GPS input in order to provide more accurate velocity and motion readings, more precise sensor readings that include less noise due to the Kalman Filter. The sensor can also be positioned within the car for optimal readings, versus h The cons for the UM7 include price, added component complexity. The UM7 is priced at $129.95 on the CH Robotics website. Overall, using the UM7 would provide clearer sensor input and provide pass through GPS input.

3.4 Sensor Choices

For a GPS and compass unit the u-blox Neo-M8N GPS with Compass will be used, due to the ease of use with the PXFmini and availability.

Before a final decision is made on whether or not the UM7 will be added, tests will be done by comparing outputs of both pieces of hardware to gain a better understanding of real world performance. Ideally, the PXFmini internal sensors will be used to start, without the use of the UM7. The reasoning behind this decision is that the overall system complexity and number of components required will be reduced. The UM7-LT Orientation Sensor will be added if the PXFmini is unable to produce sufficiently accurate results, or there is a significant boost in clarity and performance.

3.5 References

[1] u-blox, 'NEO-M8 series', 2016. [Online]. Available: https://www.u-blox.com/en/product/neo-m8-series. [Accessed: 17- Nov- 2016].

[2] Erlerobotics.com, 'PXFmini', 2016. [Online]. Available: http://erlerobotics.com/blog/pxfmini/. [Accessed: 15- Nov- 2016].

[3] U. Board, "UM7-LT Orientation Sensor — CH Robotics", Chrobotics.com, 2016. [Online]. Available: http://www.chrobotics.com/sh
lt-orientation-sensor. [Accessed: 17- Nov- 2016].

[4] Invensense.com, 'MPU-9250 — InvenSense', 2016. [Online]. Available: https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/. [Accessed: 16- Nov- 2016].

4 System Control and Data Processing Unit

The ARC system will need to have a computer handling the transmission and computation of data between the user, vision system, and other inputs. The computer will need to be able to handle many computations from the different systems in parallel, which means ideally it will have at least two physical cores. Being able to process data in real-time (15 frames a second) will also be key in being able to navigate an environment.

4.1 Intel NUC

The Intel NUC6i7KYK (pronounced 'nook'), or NUC Skull Canyon[1]is a small form computer that has a quad-core i7 6770HQ quad-core processor, at 2.6GHz with up to 3.5GHz boost, Iris Pro Graphics 580, and uses 64-bit architecture. The NUC can have up to 32GB of DDR4 RAM at 2133MHz. Intel's NUC has a similar processor, to Georgia Tech's "GT AutoRally" platform[2], which was successful in creating a high speed drifting autonomous RC truck. The biggest difference is Georgia Tech uses a dedicated EVGA 750 GTX Ti video card for rendering. The pro to the NUC is that is is very powerful and can theoretically perform upwards of a billion operations a second. The NUC also has an option of an external graphics card, if data is unable to be processed in real-time.

One of the biggest constraints is the form factor of 8.3" x 5.6" x 1.1", takes up a lot of space within the vehicle. The NUC is also supposed to be running at 19W, which would require a separate battery to prolong the operation time. Another large constraint is the cost of the Intel NUC. The base model, without RAM or a solid state drive (SSD), costs $650, which is the most expensive of the three options.

Overall, the NUC would be a good starting point to not have resource limitations, as we could always scale up.

4.2 UP Board

The UP Board [3] uses an Intel Atom quad-core CPU, up to 1.44GHz CPU with a 64-bit architecture and Intel HD 400 graphics up to 500MHz. It also features up to 4GB DDR3L RAM and up to 64GB eMMC storage.

The physical dimensions of the board are 3.37" x 2.22" (roughly the size of a credit card), which is advantageous due to it not taking up very much room within the assembly. The UP Board also supports Windows 10 and various Linux distributions, which is a plus in terms of flexibility. The cost of the highest specification UP Board is $149, which is over a quarter of the cost of the bare bones Intel NUC.

The biggest drawback to the Up Board is not knowing if it will have the computational power that we need to handle the flow of data. Benchmark testing for vision processing would need to be done against the more powerful Intel NUC, to see if it would be feasible to navigate at speed, and process at least 15 images a second.

4.3 Raspberry Pi 3

There is a possibility that computations could be reduced down to the level of a Raspberry Pi 3 [4], however, it would have to be a stretch as we need to be able to process at least 15 images a second. Being able to run everything on the Raspberry Pi 3 would mean that our project would be affordable enough for even a college student to replicate our implementation.

The pros of a Raspberry Pi are that it's inexpensive, has many available libraries, and has direct hardware access.

The cons of the Pi are it isn't as powerful as the other two options, which may be a major limitation.

4.4 System Control and Data Processing Unit Choice

Our starting point will be the Intel NUC, assuming that it should have enough processing power to handle what we need. Another reason is it has the ability for an external graphics card if rendering is not able to be done on the integrated graphics in a timely manner.

If time allows, testing will be done to see if it is possible to scale down the system to use either an UP Board or Raspberry Pi 3.

4.5 References

[1] Intel, 'Intel NUC Kit NUC6i7KYK Features and Configurations', 2015. [Online]. Available: http://www.intel.com/content/www/u
kit-nuc6i7kyk-features-configurations.html. [Accessed: 15- Nov- 2016].

[2] Georgia Tech, 'Build your own AutoRally Platform', 2016. [Online]. Available: https://autorally.github.io/build/. [Accessed: 13- Nov- 2016].

[3] Up Board — Power Up Your Ideas!, 'Specifications', 2016. [Online]. Available: http://www.up-board.org/up/specifications/. [Accessed: 15- Nov- 2016].

[4] RaspberryPi.org, 'Raspberry Pi 3 Model B', 2016. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 16- Nov- 2016].

5 Image Analysis Software

Image analysis, for the ARC project is the processing of visual data received from cameras into deterministic information, such as path-finding, or spacial awareness. This is the primary means for our autonomous vehicle to assess its surroundings and find its way to a given way-point while avoiding obstacles. We require software that is freely available for use (via fairly liberal open source licensing), known to be correct (works well) with little modification needed, and has relatively easy to use API libraries.

5.1 Robotics Operating System

The Robotics Operating System (ROS) is a general-purpose framework for writing robot software.[1] It contains obstacle avoidance packages and navigation options. [2] Based on the examples given on the ROS website, it appears that the obstacle avoidance included with ROS is intended for low-speed, controlled environments. ROS is a mature software platform with many libraries and a strong on-line community. There are many tutorials and examples of ROS implementation on rovers and UAVs. Even if the image analysis packages in ROS are not adequate for our purposes, this might be a platform we could use in a broader application for ARC.

Pros: does have obstacle avoidance built-in, has strong user-base, many examples and tutorials.

Cons: does not appear to be directly compatible with MAVLink or the PXFMini, the obstacle avoidance packages appear to be for low speed, indoor environments.

5.2 GTSAM

The Georgia Tech Smoothing and Mapping library is a set of C++ classes that implement smoothing and mapping for robotics and computer vision. [3] Georgia Tech AutoRally used this library, along with ROS (see above) in their autonomous RC project. [4] The AutoRally project is open source and freely available. GTSAM seems to have a fairly robust library with example code for implementation. [5] We would need to build the API to bridge GTSAM to our platform of choice. Since AutoRally used ROS with GTSAM, we might be able to modify their implementation to suit our purposes. This is probably a very time-consuming option.

Pros: integrates GPS, IMU and computer vision for mapping, has large library of C++ classes and example uses, was used by Georgia Tech in an autonomous RC project and we have access to the code, has a Python API.

Cons: Very probably time-consuming to create and API for our project.

5.3 Pushbroom Stereo

Pushbroom Stereo is a software library for high speed (up 20 mph) navigation in cluttered environments. [6] It uses a stereo vision algorithm that is capable of obstacle detection at 120 frames per second on an ARM processor. [7] The class files are provided but there is no support or community base to rely on. Further, this implementation is for flying UAVs, so it would need to be reimplemented for ground applications.

Pros: allows high speed obstacle avoidance

Cons: no support, small library (two classes), needs to be reimplemented for rovers

5.4 Image analysis choice

There is no clear winner for our image analysis selection. Our research has led us to conclude that obstacle avoidance is still in its technology infancy and is not widely available to the open source community. The added criteria for operation at high speeds further limits our options.

Pushbroom Stereo has the functionality of high speed obstacle detection, but it not well supported, has no examples of use and would need to be reimplemented for ground use. We probably would not have time to get that to work this year.

ROS might be a good option, but the built-in obstacle avoidance is not intended for high-speed application. All the examples of outdoor use on the ROS site did not use the ROS object avoidance library.

At this time we will attempt to duplicate AutoRally's implementation of GTSAM with ROS. GTSAM has a decent library to work from and ROS is a mature platform with large library collections and strong user support.

5.5 References

[1] ROS.org, 'About ROS', 2016. [Online]. Available: http://www.ros.org/about-ros/.
[Accessed: 15- Nov- 2016]

[2] ROS.org, 'navigation', 2016. [Online]. Available: http://wiki.ros.org/navigation.
[Accessed: 15- Nov- 2016]

[3] Georgia Tech Borg Lab, 'GTSAM' [Online]. Available: https://bitbucket.org/gtborg/gtsam.
[Accessed: 15- Nov- 2016]

[4] ROS.org, 'The AutoRally Platform'. [Online]. Available: http://www.ros.org/news/2016/06/the-autorally-platform.html.
[Accessed: 15- Nov- 2016]

[5] GTSAM, 'examples'. [Online]. Available: https://bitbucket.org/gtborg/gtsam/src/a738529af9754c7a085903f90ae8559bbaa82e75/e
[Accessed: 15- Nov- 2016]

[6] A. Barry, 'flight'. [Online]. Available: https://github.com/andybarry/flight.
[Accessed: 15- Nov- 2016]

[7] A. Barry, R. Tedrake, *Pushbroom Stereo for High-Speed Navigation in Cluttered Environments* Unknown, http://groups.csail.mit.edu/ro
center/public_papers/Barry15.pdf. p. 1

6 Telemetry Radio Communication

In this section we will examine three different telemetry radios, comparing and contrasting them and making a choice on which radio we will use for ARC. Telemetry is simply the transmission of measurement data (velocity, angle, rotation, etc.) by radio to some other place. [1] This data allows the user to know the current state of the vehicle. This is especially important for autonomous operation, as the vehicle may not be operating within line of sight. Telemetry transmission is well-established, so we will not be comparing vastly different transmission technologies, such as long range (MHz radio frequencies) versus short-range (blue-tooth) where the advantages of ranges of 2-15+ kilometers obviously outweigh ranges of 20-100 meters.

The main criteria for consideration are:

- Cost

  One of our main goals with ARC is to keep the costs low.
- Power consumption

  We have limited power available, therefore we need power consumption to be low.
- Ease of use

  The radio needs to be easily integrated into the autopilot system. This means it needs to have a developed API with little no modification required.
- Form factor

  The size and weight needs to be small and light. If it is too bulky, we might not have space on the vehicle. If it is too heavy, more power will be required to operate the drive system and will drain the battery faster.

6.1 3DR 915 MHz Transceiver

The 3DR 915 MHz telemetry radio has a cost of $39.99 USD for two radios. It is powered by the autopilot telemetry port (+5v) which means it has low power consumption. This radio transceiver uses open source firmware, has a robust API, and is fully compatible with PX4 Pro, DroneKit, and ArduPilot, using the MAVLink protocol. These features will allow us to implement telemetry transmission with little to no modification of the API, should we use one of those autopilot systems. The form factor has dimensions of 25.5 x 53 x 11 mm (including case, but not antenna) at 11.5 grams (without antenna). [2]

The range of this transceiver is from 300 meters to several kilometers, depending on the antenna arrangement.

Pros: inexpensive, small form factor, low power consumption.

Cons: range out of the box could be as low as 300 meters.

6.2 RFD900 Radio Modem

The RFD900 Radio Modem has a cost of $259.99 USD for two radios. [3] It requires separate +5v power for operation which means that it has high power consumption. This radio has open source firmware, a robust API, and is fully compatible with PX4 Pro, DroneKit, and ArduPilot, using the MAVLink protocol, which will allow us to implement telemetry transmission with little to no modification of the API, should we use one of those autopilot systems. [4]

The form factor has dimensions of 70 x 40 x 23mm (including case, but not antenna) at 14.5 grams (without antenna). The range of this transceiver is 25+ kilometers.

Pros: ultra long range.

Cons: expensive, large size.

6.3 Openpilot OPLink Mini Ground and Air Station 433 MHz

The OPLink Mini Ground Station has a cost of $26.59 USD for two radios. [5] It requires input voltage of +5v and can be powered off the autopilot telemetry port which means that it has low power consumption. This radio has open source firmware but is only compatible with the OpenPilot RC control system. The form factor has dimensions of 38 x 23 x mm (including case, but not antenna) at 4 grams (without antenna). [6] The range of this radio is not known, but based on the power requirements and frequency it likely has less range than the 3DR 915 MHz radio.

Pros: smallest size and weight (only 4 grams), lowest cost ($26.59 USD)

Cons: Only works with the LibrePilot control system.

6.4 Telemetry radio choice

The 3DR 915 MHz Transceiver is our selection for the telemetry radio. While the OPLink Mini Ground Station was significantly smaller, lighter, and cheaper than the other two, its implementation being tied solely to LibrePilot was a deal breaker (more information on LibrePilot can be found in the User Interface evaluation). The RF900 Radio Modem would have been a good choice, it has fantastic range and all the API options we were looking for. But it had a significantly larger form factor, required a separate power supply, and was quite expensive at $259.99. Put together, these facts eliminated the RF900 as a viable option. The 3DR 915 MHz Transceiver is a good balance of cost, performance, and size. The cost of $39.99 for two radios, the ability to power the autopilot off the telemetry port, and the portability of its APIs and their ease of use, puts the 3DR 915 MHz at the top of our list and the clear choice for the telemetry radio going forward.

6.5 References

[1] Merriam-Webster.com, 'telemetry', 2016. [Online]. Available: http://www.merriam-webster.com/dictionary/telemetry. [Accessed: 15- Nov- 2016].

[2] 3DR, '915 MHz (American) Telemetry Radio Set', 2016. [Online]. Available: https://store.3dr.com/products/915-mhz-telemetry-radio. [Accessed: 15- Nov- 2016].

[3] jDrones.com, 'jD-RF900Plus Longrange', 2016. [Online]. Available: http://store.jdrones.com/jD_RD900Plus_Telemetry_Bundle_p/ [Accessed: 15- Nov- 2016].

[4] ArduPilot Dev Team, 'RFD900 Radio Modem', 2016. [Online]. Available: http://ardupilot.org/copter/docs/common-rfd900.html. [Accessed: 15- Nov- 2016].

[5] Banggood.com, 'Openpilot OPLINK Mini Radio Telemetry', 2016. [Online]. Available: http://www.banggood.com/Openpilot-OPLINK-Mini-Radio-Telemetry-AIR-And-Ground-For-MINI-CC3D-Revolution-p-1018904.html [Accessed: 15- Nov-2016].

[6] HobbyKing.com, 'Openpilot OPLink Mini Ground Station 433 MHz', 2016. [Online]. Available: https://hobbyking.com/en_us/op oplink-mini-ground-station-433-mhz.html [Accessed: 15- Nov-2016].

7 User Interface

In this section we will examine three user interfaces, comparing and contrasting them and making a decision on which one we will use with ARC. A user interface (UI) is required to allow the user to command the vehicle. The UI must be open source and have easy-to-implement API libraries. We are looking for a UI package that will work with both the control station (the user computer) and the companion computer (the computer on board the vehicle). It is preferable that the UI be a combination of graphical UI (GUI) and command line UI (CLI). Note that though our project is a land vehicle (rover) the following software is primarily used for UAV flight control and is referenced in such a way. If possible, we would like to use software that can be configured to control a rover, or easily modified to do so.

7.1 QGroundControl

QGroundControl (QGC) is a full flight control and mission planning GUI software package that is compatible with any MAVLink enabled drone. [1] It is open source and is configured for use with ArduPilot and PX4 Pro. QGC runs on Windows, OS X, Linux, and iOS and Android tablets. QGC has video streaming with instrument overlays, allows mission planning including map point selection, rally points, and even a virtual fence to keep the drone from going beyond a specified area. QGC is a mature software package that has excellent libraries and support with very good documentation. [2] Additionally, QGC works with ArduPilot which is known to work with the PXFMini, the flight controller we intend to use and can be configured for rovers. [3] QGC appears be GUI only with no CLI functionality.

Pros: easy to use, great documentation, compatible with MAVLink, tested on the PXFMini. Can be used on all major pc and mobile platforms. Supports rovers.

Cons: does not appear to have CLI support.

7.2 Tower/DroneKit-Android

Tower is a Android mobile app that works with most drones that use the MAVLink protocol. Tower allows basic map point selection and allows drawing a flight path on the tablet. [4] It is based on the open source DroneKit-Android framework. DroneKit-Android has good documentation providing code snippets with working example code. [5] Because of the modularity of Android development and access to Tower source code, adding feature and interfaces to the existing app should be relatively easy. Tower is not configured for rovers, so we would have to write the functionality into it. DroneKit-Android and Tower are only available on Android.

Pros: is easy to use, has basic map point selection, adding features should be relatively straight-forward.

Cons: is only available on Android, configuring for rovers requires writing code for support.

7.3 LibrePilot

(https://www.librepilot.org/site/index.html)

LibrePilot (LP) is a full flight control and mission planning software package. It is open source and operates via GUI and allows map point selection. LP is compatible with OpenPilot control system exclusively. It does not work with any other hardware but OpenPilot hardware and does not have rover support. It has some helpful documentation, such as Windows build instructions, but the information is very basic. The source code does seem to have decent commenting which could help since we would need to heavily modify the code base for rover support. LP runs on Linux, Mac, Windows, and Android. [6]

Pros: Has a nice GUI for map point selection, mission planning, and vehicle control. If used in the OpenPilot ecosystem, it should communicate well. Runs on most major pc platforms.

Cons: Is locked in to the OpenPilot ecosystem. Does not have rover support out of the box which will require extensive coding. Does not run in iOS.

7.4 User interface choice

QGroundControl is our user interface choice.

LibrePilot has similar features to QGC but being locked in to the OpenPilot ecosystem is a deal breaker. We need to be able to use the PXFMini and LibrePilot cannot do that. LP is also not configured for rovers, which would require extensive coding.

Tower is the most modest of the user interface options. It is only available on Android, does not have rover support and has limited options for navigation and vehicle control. For these reasons we reject Tower as a viable option.

QGroundControl runs on all major pc and mobile platformsis configured to run rovers, and uses the MAVLink protocol. It has a nice GUI and allows map point selection and advanced mission planning. It is known to work with the PXFMini flight controller, a component we want to use as part of the ARC build. These features give us a platform that is meets our needs and is flexible, should our needs change. Therefore, QGroundControl is the clear user interface choice for the ARC project.

7.5 References

[1] QGroundControl.com, Unknown. [Online]. Available: http://qgroundcontrol.com/.
[Accessed: 15- Nov- 2016].

[2] QGroundControl.com, Unknown. [Online]. Available: https://donlakeflyer.gitbooks.io/qgroundcontrol-user-guide/content/.
[Accessed: 15- Nov- 2016].

[3] ArduPilot Dev Team, 'PXFmini Wiring Quick Start', 2016. [Online]. Available: http://ardupilot.org/rover/docs/common-pxfmini-wiring-quick-start.html.
[Accessed: 15- Nov- 2016].

[4] Fredia Huya-Kouadio, 'Tower', 2016. [Online]. Available: https://play.google.com/store/apps/details?id=org.droidplanner.andro
[Accessed: 15- Nov- 2016].

[5] 3D Robotics Inc., 'DroneKit', 2015. [Online]. Available: http://dronekit.io/.
[Accessed: 15- Nov- 2016].

[6] LibrePilot, 'Open-Collaborative-Free', 2016. [Online]. Available: https://www.librepilot.org/site/index.html.
[Accessed: 15- Nov- 2016].

8 CONTROL SYSTEM

A good control system can improve the accuracy of the estimated current location of the vehicle. If we choose unreliable control scheme and motion model, it will produce a lot of overhead for our main computer and the results won't be as precise. Power consumption may also increase due to the excessive amount of calculation done by the computer. In conclusion, motor control, servo control, and motion model are methods to minimize the computation the main computer has to undergo.

8.1 Motor Control

There are two types of motor control schemes that are practical for our project. I would not say that they are technologies. They are just two ways to decide how the vehicle should move forward and backward:

1) Time Critical: This motor control scheme tells the motor to spin forward/backward for a certain amount of time at a certain speed.
2) Distance Critical: This motor control scheme tells the vehicle to go forward/backward for a certain distance.

When an iteration is finished under time critical, the system will decide whether to keep the speed for another time period or slow down or accelerate. In order to reduce jerkiness, the rate of acceleration and deceleration will be low.

When an iteration is finished under distance critical, the system will decide what to in the next cycle. It can either switch to time critical or move for another distance.

In the final implementation, we may switch between the two methods based on real-time conditions. When the vehicle is operating at high speed in an open environment, such as a parking lot, with GPS cooperating, time critical better suits my purpose. When operating indoor without a predefined map, in other words, the vehicle is exploring the environment, distance critical works better. When the system detects an approaching obstacle, whether it is operating indoor or outdoor, the system should always switch to distance critical.

8.2 Servo Control

Servos control the steering of the vehicle. Servo control is similar to motor control. There are also two scheme under which the system operates the servo:

1) Time critical: The system tells the servo to keep a certain angel for a certain amount of time.
2) Angel critical: The system tells the vehicle to steer left/right for a certain angel.

Time critical makes drifting possible, which it is one of the goals of this project. When the car is operating at high speed, time critical will be easier to harness because over-steering will happen, and angel critical will cause unexpected maneuver when over steering happens.

Angle critical may do a better job in obstacle avoidance. The angle of turning is defined as the angle between the driving direction of vehicle before and after the turn.

Again, in the final implementation, both schemes will be implemented and the system will switch between them based on real-time situation.

8.3 Probabilistic Analysis for Motion

A couple of options for representing the location of the vehicle are x, y, θ coordinate, and grid representation. Since we are dealing with a car that can go at anything directions, and has a sufficient size, we will not consider grid representation as our model.

1) Bayes Filter: [1]
   Bayes filter is based on Bayes Rule. Bayes filter constructs the posterior probability density function by incorporating all available measurements. Using the log-odd operation and assuming Markov property holds true in our set up, we can turn multiplication into addition, which will increase the accuracy of the prediction since computers are generally not good at doing floating point multiplications.
2) (Extended) Kalman Filter: [2]
   Kalman filter also uses Bayes rule but with Gaussian noise. It uses the prior knowledge of state to predict the future. It then incorporates measurements to make the predicted state more accurate. Extended Kalman filter is the extension of Kalman filter that can handle nonlinear systems, meaning that the output of a system is not proportional to its input. [3] Our system is a good example of a nonlinear system. The results are not proportional to any of the measurements and prior states.
3) Particle Filter: [4]
   Particle filter uses sampling method to estimate the probability of states. It can keep track of multiple assumptions. It updates the probability density function iteratively. Each time new movement and measurement arrive, it shifts the particles accordingly and uses the measurements to update the probabilities. It generates new particles at each iteration at locations where the probabilities are the highest. It can also handle nonlinear systems.

For our interests, we will be using particle filter because it is good model for localization. It is easy to implement, fast, and accurate. The only drawback is that going from an initial uniformly distribution to an acceptable prediction might take longer if the configuration of the space is complex.

8.4 Motion Model

In this context, the motion model states the behavior of the vehicle under different combinations of speed and steering. A concrete and precise motion model is important to the control system. The control system operates the vehicle based on this pre-defined motion model. For example, if the system needs to turn the vehicle 90 degrees right, it will have to output a sequence of actions by applying the motion model to the status of the vehicle, such as speed and center of gravity.

Different vehicle will have different configurations of motion model. When our vehicle is fully loaded with hardware, it will also have a different configuration of motion model than when it is unloaded. Thus, we will have to conduct multiple experiments to create the motion model of our vehicle. Alternatively, ROS provides a very nice simulation environment, where we can have a script that describes the configuration of the vehicle and have the simulator run the vehicle. [5]

Potentially, as the development progresses, we may find the motion model to be obsolete, because I have a feeling that we can get around without a pre-defined motion model. But for now, a large part of me still believes it is necessary.

8.5 references

[1] School of MIME, Oregon State University, 'Bayes Filter'. [Online]. [Accessed: 16- Nov- 2016].

[2] Michael Rubinstein, Computer Science and Artificial Intelligence Laboratory, MIT, 'Introduction to recursive Bayesian filtering'. [online]. Available: https://people.csail.mit.edu/mrub/talks/filtering.pdf.
[Accessed: 16- Nov- 2016].

[3] Larry Hardesty, MIT News, 'Explained: Linear and nonlinear systems'. [Online]. Available: http://news.mit.edu/2010/explained-linear-0226.
[Accessed: 16- Nov- 2016].

[4] School of MIME, Oregon State University, 'Particle Filter'. [Online]. [Accessed: 16- Nov- 2016].

[5] Ros.org, 'Robot Model Tutorials'. [Online]. Available: http://wiki.ros.org/robot_model_tutorials.
[Accessed: 16- Nov- 2016].

9 Path Planning

9.1 Global Path Planning

Global in the perspective of our vehicle will be an area about the same size of a basketball court. We will predefine maps with multiple way points for the system to navigate itself through.

- Breath-first Search:
  This algorithm promises to output the optimal path for start to destination in terms of nodes visited. In reality, the cost of going from one node to another should be considered when planning, so this algorithm will not be considered.
- Depth-first Search:
  Similar to breath-first search, this algorithm guarantees to find the optimal path for start to destination in terms of nodes visited. In reality, the cost of going from one node to another should be considered when planning, so this algorithm will not be considered.
- Dijkstra's algorithm:
  This is algorithm and breath-first search are very much alike. This algorithm takes into account the costs between nodes and promises to find the shortest path.
- A* Heuristic search:
  This algorithm uses an extra variable, the heuristic, to improve the performance of the Dijkstra's algorithm.

In conclusion, we will be using the A* Heuristic search algorithm as our global path planning algorithm. It produces reliable results and is the fastest algorithm among all the above-mentioned algorithms. Our computer should handle the computation no problem.

9.2 Local Path Planning

Local in the perspective of our vehicle will be an area about the same size of a classroom. The vehicle may have to build the map itself by exploring the space or be provided with a predefined map.

- Rapidly-Exploring Random Trees (RRTs):
  This algorithm guarantees to find a path from start to goal as the number of sample points goes to infinity.
- RRT*:
  This algorithm is an extension of RRTs. It promises to find the optimal path from start to goal as the number of sample points goes to infinity.

These two algorithms are both feasible given the map is small, because they require a large amount of data to be stored in memory. This is why we did not consider the two to be our global path planning algorithms.

Depending on the computer memory and time constraints, we will use any one of them that suits our purpose. If the system is asked to output a valid path as fast as possible, RRTs will be used. If we were to ask the system to output the shortest path it can generate given all the computational resources, RRT* will be used.

10 Other Algorithms

10.1 Obstacle Avoidance Algorithm

In the global and local path planning algorithms sections, the underlying assumption is that the world is static (the map does not change). However, our team wants the vehicle to respond properly when unexpected objects are blocking the planned path.

There are two ways to accomplish this. We can: [1]

1) Diverge from the original path as little as possible and converge back to the preplanned path as soon as possible.
2) Reconstruct an entirely new path.

While re-planning may be more intelligent overall, it is infeasible in a global setup due to the large amount of computation the system has to redo. Rarely the construction of a path will take less than noticeable period time. Our team's vision of the system is to enable the vehicle to drive itself as if a rational human is driving it. New plan construction that causes suspension of movements is unacceptable. Thus, we will go for the first option and will not consider reconstructing new paths.

10.2 Parallel Parking Algorithm

By going through a research project carried out by students at University of Minnesota, parallel parking requires the system to perform following tasks: [2]

1) Parking Spot Detection
2) Parking
3) Exiting the spot

For simplicity, we don't need to perform parking spot detection if the system is given a map. The algorithm will output a trajectory to the control system on every iteration. The system will be using multiple sensors, such as IR/sonar sensors, to scan surroundings, adjusting the trajectory accordingly. The control system must make sure the vehicle follows the trajectory precisely.

10.3 References

[1] School of MIME, Oregon State University, 'Planning'. [Online]. [Accessed: 16- Nov- 2016].

[2] Lisa S, Christopher W, Mun Hoe Sze Tho, Trenton P, Joel H, University of Minnesota, 'Autonomous Parallel Parking of a Nonholonomic Vehicle'. [Online]. Available: http://www-users.cs.umn.edu/ joel/_files/Joel_Hesch_EE4951.pdf. [Accessed: 16- Nov- 2016].