Group 44 - ARC

Tech Review

Senior Capstone Project

Oregon State University

Fall 2016

Tao Chen, Cierra Shawe, Daniel Stoyer

Abstract

CONTENTS

1 Vision System Options - Cierra

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.

1.1 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 with vast amounts of 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. [**?**] OpenCV [**?**] 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.

1.2 IR Camera's such as Kinect and RealSense

Using an infrared point map, 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 massive amount of infrared waves from the sun. IR cameras don't meet our requirement of being able to use the vision system reliably outdoors.

1.3 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 unite costs around $2,000 USD for a unit with 10m accuracy. citesick https://www.sick.com/us/en/product-portfolio/detection-and-ranging-solutions/2d-laser-scanners/tim3xx/c/g205751 A 2D RPLiDAR module is a lower cost alternative, at $449, however, the manufacture says that it will not preform well in direct sunlight, due to using IR lasers. http://www.robotshop.com/en/rplidar-a2-360-laser-scanner.html The RPLiDAR also only has accurate measurements up to 6 meters.

A third option for LiDAR, has yet to come to market. The Sweep LiDAR unit by Scanse will be released in January

of 2017. 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 LiDAR systems that would be available to our group, would not meet the requirement of being able to function consistently in an outdoor environment, or do not fall under the category of commodity hardware.

1.4 Our 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. 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 investigate the use of the Sweep LiDAR system for navigation in place of stereoscopic imaging.

2 Sensors - Cierra

As our group has already been given a PXFmini and a UM7-LT Orientation Sensor.

2.1 Internal Measurement Unit (IMU) - External

2.2 IMU - PXFmini

3 System Control and Data Processing - Cierra

Our system will 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 in parallel, which means ideally it will have at least two physical cores. The three options chosen for consideration were the Intel NUC (Skull Canyon), UP Board, and the Raspberry Pi 3, in order from most powerful to least powerful.

3.1 Intel NUC

The Intel NUC6i7KYK (pronounced ’nook’), or NUC Skull Canyon, is a small form computer that has a quad core i7 6770HQ quad core processor, at 2.6GHz with up to 3.5GHz boost and Iris Pro Graphics 580. The NUC can have up to 32GB of DDR4 RAM at 2133MHz. One of the biggest constraints is the form factor of 211mm x 116mm x 28mm takes up a lot of space within the vehicle, and the NUC is supposed to be running at 19W.

3.2 UP Board

The UP Board uses an Intel Atom Quad Core CPU, with 2M Cache, 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", 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.

3.3 Raspberry Pi 3

There is a possibility that we could minimize all of our computations down to the level of an Raspberry Pi 3. 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.

3.4 Our choice

Our starting point will be the Intel NUC, knowing that it should have enough processing power to handle what we need and because it has the ability for an external graphics card if the computer can't render fast enough.

4 Image Analysis Software - Dan

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.

4.1 Option 1

more to follow...

4.2 Option 2

more to follow...

4.3 Option 3

more to follow...

4.4 Image analysis choice

ArduPilot is our choice for image analysis software. The documentation for the image analysis software researched was rather vague across the board when it comes to information on path-finding and image analysis capabilities. So, while we are going with ArduPilot to start with, we will not really know its effective capability at obstacle avoidance and path-finding until we have tested it.

4.5 References

5 Telemetry Radio Communication - Dan

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.

5.1 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.

5.2 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.

5.3 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.

5.4 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.

5.5 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].

6 User Interface - Dan

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.

6.1 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.

6.2 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.

6.3 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.

6.4 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 platforms, is 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.

6.5 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].

7 CONTROL SYSTEM - TAO

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 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.

7.1 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.

7.2 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.

7.3 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:
   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:
   Kalman filter also uses Bayes rule but with Gaussian noise. It uses the prior knowledege of state to predict the future. And then it incoperates 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 propotional to its input. Our system is a good example of a nonlinear system. The results are not propotional to any of the measurements and prior states.
3) Particle Filter:
   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.

7.4 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.

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 Path Planning - Tao

8.1 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.

8.2 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.

9 Other Algorithms - Tao

9.1 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) 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.

9.2 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:

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.