Final Report

ARC - Autonomous RC

Computer Science Capstone

Oregon State University

Spring 2017

Tao Chen, Cierra Shawe, Daniel Stoyer

Group 44

June 12, 2017

CONTENTS

1 INTRODUCTION

Research into consumer/hobbyist, high performance RC vehicles was requested by Oregon State University via Mr. Kevin McGrath. The project was requested to determine if it is possible to apply high-speed performance during autonomous navigation and obstacle avoidance to a modified RC car at a cost less than four thousand dollars (USD). Autonomous RC (ARC) sought to push the boundaries of what is possible for autonomous RC vehicles. Our research shows that components are decreasing in cost and increasing in performance. The cost-barrier to autonomous research is decreasing dramatically. Our documentation and parts list provides would-be researchers a launching point to continue the work we started in ARC. Our client was the same person who requested the project, Mr. Kevin McGrath, an instructor at Oregon State University.

1.1 Our Team Members and Their Roles

One of things our group is proud of, is the diversity within our group. Tao Chen is an international student from China, Cierra Shawe is one of the few female students in computer science, and Dan Stoyer is a veteran and served as a medic in the US Army.

Tao Chen

Tao was our software and robotics expert, he worked extensively with our software package and got our car working in simulation, he was responsible for the areas of Motion Model, Path Planning, and Autonomous Algorithms (e.g. obstacle avoidance, parallel parking, etc.).

He will be receiving a bachelors degree in Computer Science - Computer Systems, in June of 2017. After graduation, he will be attending University of Southern California for graduate school, where he will be focusing on a masters degree in robotics.

Cierra Shawe

Cierra was our electronics and hardware expert, she designed all the mounting hardware used to anchor the sensors to the RC car and did all the wiring/soldering. She was also responsible for the Vision Systems, Sensors, and Hardware. She will be graduating in June of 2018, with a bachelors degree in Computer Science - Applied Cyber Security. Summer and Fall term of 2017, she will be working at Rockwell Collins for an engineering co-op. She will be looking to go into a robotics field after graduation.

Daniel Stoyer

Dan was team leader, responsible for making sure the team was on track to hit milestones and Capstone deadlines on time. He was also responsible for overseeing the areas of Image Analysis, User Interfaces, and Radio Communication. He will be graduating in December of 2017, with a bachelors degree in Computer Science - Applied Business Management, as well as a minor in business. Dan will be spending time with his newborn son (who was born during week ten of winter term) and finishing his degree, before he starts working again in tech.

1.2 Our Clients Role in the Project

Kevin gave us an awesome project and provided us with oodles of hardware.

2 Requirements Document

2.1 Requirements Changes

No major or minor changes were made to the requirements document. We found that our initial assessment was accurate throughout the project and our client agreed.

Fig. 1. ARC Fall Schedule Gantt Chart

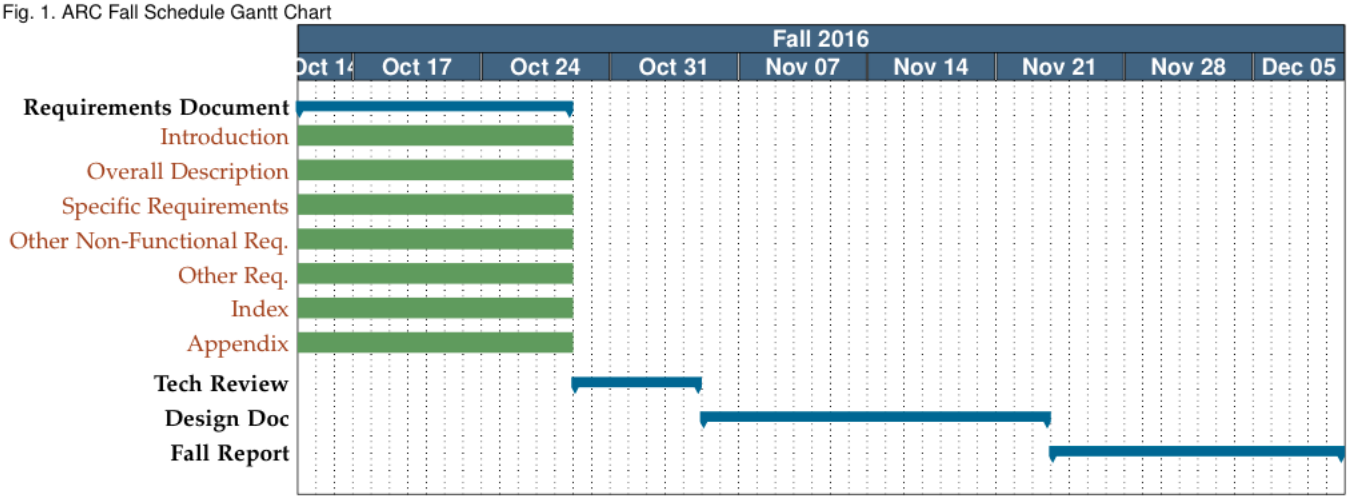


Fig. 2. ARC Winter Schedule Gantt Chart

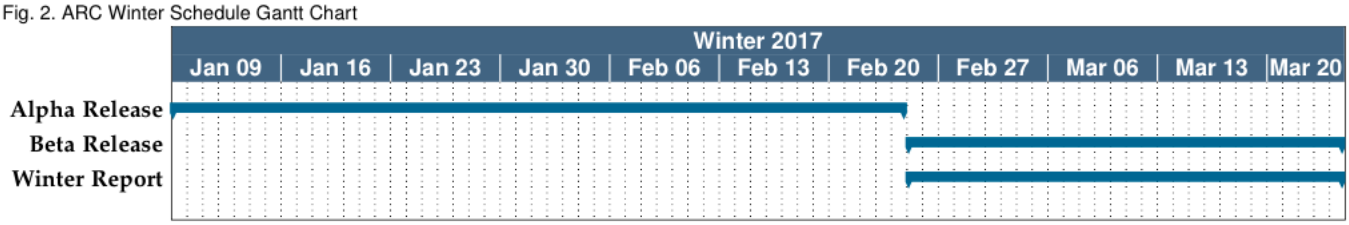
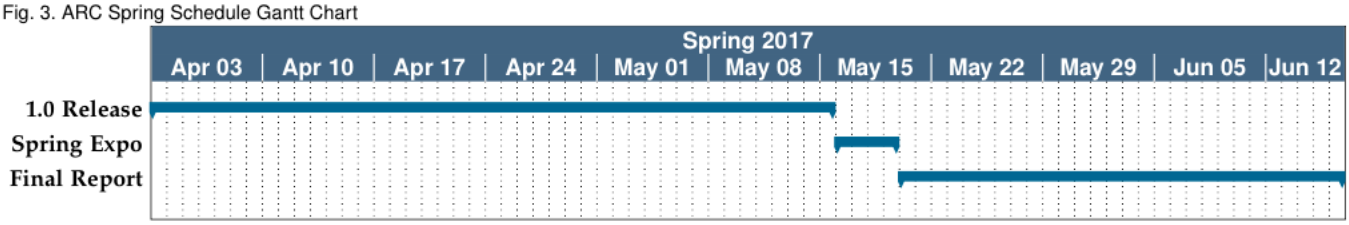


Fig. 3. ARC Spring Schedule Gantt Chart



2.2 Original Requirements Document

(Begins on next page).

Software Requirements Specification

ARC - Autonomous RC

Senior Capstone Project

Oregon State University

Fall 2016

Tao Chen, Cierra Shawe, Daniel Stoyer

Version 1.0

December 1, 2016

D. Kevin McGrath	Date
Tao Chen	Date
Cierra Shawe	Date
Daniel Stoyer	Date

CONTENTS

1 INTRODUCTION

This document aims to provide an overview and list of requirements for group 44, Autonomous RC (ARC). This section provides a purpose and scope description, list of abbreviations and acronyms, and an overview of everything included in this SRS document. As this is a research project, some of the sections within the document may not apply to the project.

1.1 Purpose

A detailed description of the requirements for the "Autonomous RC System" or ARCS will be provided within this document. System constraints, interface decisions, and interactions with other external applications and hardware will also be explained. This document is primarily intended to be a customer proposal for approval and a development team reference for the first version of the system.

1.2 Scope

ARCS is a software-hardware interface designed to retrofit RC cars for autonomous operation, using commodity hardware. The software and hardware specifications should be available free to download and modify at the users will.

Users should be able to purchase and install the specified hardware from major online retailes such as Amazon.com or hobby sites such as Sparkfun.com. Once the vehicle is assembled, it should be able to autonomously navigate to a given destination using GPS, or a within pre-defined boundaries, such as a room. Control software will need to be loaded onto the main processing unit and any other control hardware needed. Other software required will be a control panel to monitor the vehicle and determine it's behavior, such as navigating to a point, or on a track.

Hardware that we expect to need:

- Base station that includes a transceiver
- An RC car with servo steering and DC motors
- Main processing unit (a computer)
- Transceiver to send and receive information on the car
- Controller to send signals to sensors and actuation devices (motors, servos, etc...)
- Vision system to aide in obstacle avoidance
- GPS unit to aide in navigation

ARCS will be expected to be able to receive input from a user base station, and react within the environment based on a destination or path that the system receives. It should also be able to navigate to the destination without user intervention, as fast as possible.

1.3 Definitions, Acronyms, and Abbreviations

1) **IMUs:** Inertial measurement unit. Used to measure acceleration, angular acceleration, and orientation of the vehicle.

2) **Operator/User:** The person who is giving commands such as destination to the system.
3) **Protocol:** Defines the data format to be transferred.
4) **Telemetry Data:** Data that contains the status information of the vehicle, such as speed, temperature, location, battery, etc.
5) **Emergency:** Emergencies include, but are not limited to:
   - The vehicle drifts off course significantly.
   - Vehicle flips upside down.
   - On-board components fall off.
6) **Visual Unit:** Visual components that provide image streams to the main processing unit. The primary computer will extract information from the images, such as road condition, obstacles, and depth.
7) **Actuators:** Motors and servos.
8) **Initialization:** From the system perspective, the initialization process will include self-checking and sensor calibration. From the user's perspective, it contains making sure all the hardware is attached, battery is at least 80% full, and giving the system a destination.
9) **Success:** The vehicle successfully navigates itself to the destination.
10) **ARC:** Autonomous Remote Controlled is our team name
11) **ARCS:** ARC System, which refers to both hardware and software components built for the project.
12) **RC:** Remote Controlled
13) **AV:** Autonomous Vehicle
14) **Main Processing Unit:** The unit which handles all high level decisions and input processing.
15) **Stakeholder:** Any user that will operate ARCS.
16) **Companion Computer:** The computer on board the RC vehicle that performs heavy computation, such as image analysis.
17) **Ground Station:** The remote computer that sends user commands to the vehicle and receives telemetry from the vehicle.

1.4 Overview

The remainder of this document includes three sections and the appendixes.

Section two provides an overview of system functions and intersystem interaction. System constraints and assumptions are also addressed.

Section three provides the requirement specification in detailed terms and system interface descriptions. Different specification techniques are used in order to specify requirements for different audiences.

Section four priorities requirements and includes motivation for the chose prioritization and discusses why other methods were not chosen.

Appendixes at the end of the document include results of the requirement prioritization and a release plan based on the requirements. **[?]**

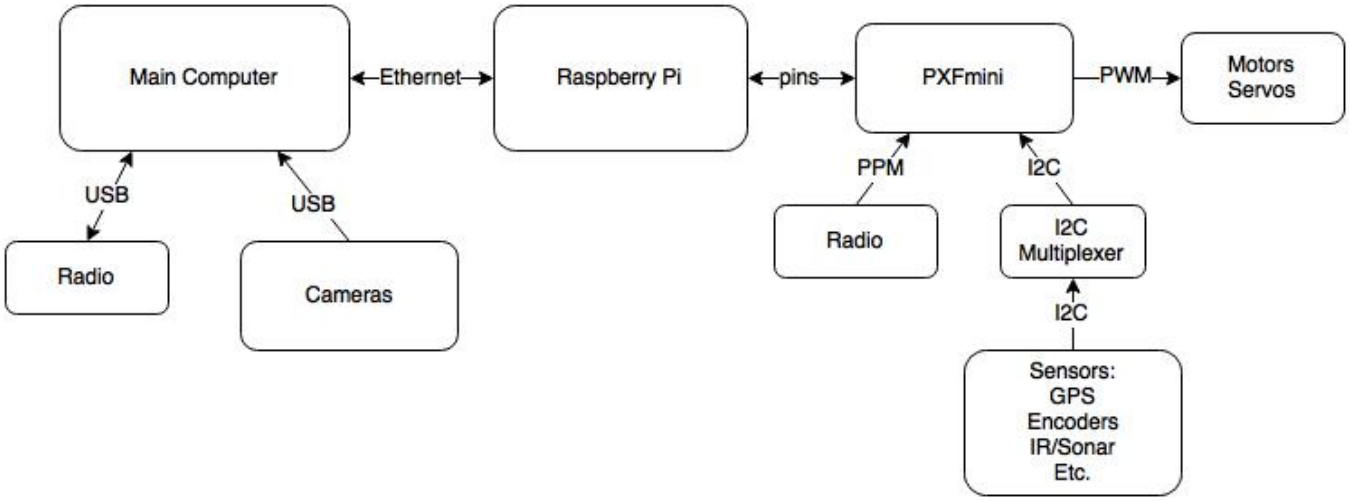


Fig. 1. Block diagram of hardware flow

2 Overall Description

This section will explain the system in its context to create a better understanding of how the system interacts with other systems. In addition, basic functionality will be introduced and addressed. Stakeholders will be defined and functionality for each type of stakeholder will be adressed. Lastly, constraints and assumptions for the system will be presented.

2.1 Product Perspective

ARCS will be designed to integrate into an RC car using commodity hardware, and open to anyone who is interested in using it. This makes ARCS a component of a larger system, namely the RC car.

ARCS will consist of three parts:

1) Base-station used for user interaction
2) Hardware attached to RC car to be able to connect to base-station and operate the vehicle
3) Software implementation for control and communication between hardware and software

In order for the user to interact with the vehicle, commands will be sent via some form of receiver to the car. This will need to be done via a base-station that has software able of providing a way to communicate with the receiver, which then transmits data to the receiver on the car, which is then handled by the on-board computer. The control flow is described in figure 1.

2.1.1 System interfaces

There are a total of 5 system interfaces where the system can communicate with the outside world.

1) Sensors: Sensors will have a two-way communication with a secondary computer unit, where filtering and smoothing will happen before reliable data will be passed to the main processing unit. The programs that reside in the secondary unit will utilize various methods to generate reliable results based on the raw data. At start-up, there will be a script executed by the system to correctly configure and calibrate each sensor. Sensors may include: battery monitor, temperature sensor, GPS, encoders, IMUs, and any other sensors that are needed for autonomous operation.
2) Radio: This is the portal of the system where operator/user can monitor the status of the vehicle. Different protocols will be implemented for telemetry data, which will be displayed to the operator/user. This portal also allows operator/user to take control over the computer in case of emergencies.
3) Visual unit: This is the interface where the visual unit can pass streams of images to the system.
4) Actuator: The system issues commands to the motors and servos via this interface.
5) User interface: This interface is a different interface than the radio interface even though they both allow humans to interact with the system. The user interface will be disable after the vehicle starts maneuvering. This interface allows user to input operation modes and desired destinations into the system.

2.1.2 User interfaces

The user interface will be a graphical user interface (GUI) that can be interacted with by a user. Anyone who knows how to operate a mouse or touch screen will be al. A map makes it easier for users to pin point destinations and view the current location of the vehicle. An error messages will be generated the vehicle and the station exceeds the maximum radio range, or the vehicle is low on battery.

With proper training (in less than 30 minutes), one can understand all the indicators of the system to know the status of the vehicle.

The GUI is a single window/page arrangement. A large portion of the window is dedicated to the map. A section of the GUI will be used to display telemetry data. Error messages will cover at least 10% of the screen in order to alert the user of an issue.

2.1.3 Hardware Interfaces

Two main hardware interfaces currently exist within ARCS.

1) Vision processing
2) Telemetry collection and actuator control

Vision processing, from either a stereoscopic or depth camera, will be interfaced through an i5 or better processor from an on board computer using a control board, which will be used as an interface between a computer that is doing high level computations, the servos, motors, and other telemetry devices.

Telemetry collection and actuator control will be handled by a PXFmini and interfaced through a board with a 40pin connector, such as the Raspberry Pi. This abstracts a lot of control onto the PXFmini.

2.1.4 Software Interfaces

- We require three operating systems. One for a remote PC for user input, one for the primary computer for on-car data analysis, and one for the secondary computer for on-car control.
- We require a user interface to be able to give the car destination commands.
- We require software that takes input from hardware, such as video cameras, and analyze the data
- Sensor analysis software needs to be able to read data from sensors, such IMUs, and generate usable information to pass on to
- Path finding software needs to be able to integrate and analyze data from mapping and localization software products and determine a path to follow in real time.
- The primary and secondary computers will need to talk with one another. The primary computer will need to receive data from the secondary computer, analyze the data and send corresponding commands back to the secondary computer. The secondary computer will need to receive commands from the primary computer, and send data to the primary computer.
- A software interface will required on the secondary computer to convert commands received from the primary computer to instructions usable by pre-existing RC car on-board controller.
- We will require separate interfaces between the secondary computer and the visual/spacial sensors, GPS, speed sensor, and the IMU.

2.1.5 Communications Interfaces

- Radio communication will required to be able to send commands to the car and to be able to receive feedback from the car. The radio frequency will be in 27 MHz or 49 MHz range. We need software that allows us to send and receive radio signals to and from the car.
- LAN communication will be used to transmit data between the on-car computers. Existing network protocols will be utilized to perform the transmissions.
- A software interface will be required to send the commands from the secondary computer to the pre-existing RC car on-board controller.

2.1.6 Memory constraints

Since we are unsure of what hardware will be used, we are not able to set any constraints on the memory requirements that the system has to meet.

2.1.7 Operations

In most cases, operations of the system will be isolated from the user/operator, excluding emergencies and initialization.

1) At initailization, user/operator has to thoroughly check the body of the vehicle, making usre that all components are firmly attached as well as the bettary level is at least 80%;
2) Destination will be given by the user at the station during initialization;
3) User is allowed to take over control from the system in case of emergencies via the remote control;

4) User/operator is resposible for monitoring the status of the system, such as battery level, speed, location, distance, radio signal strength, pre-planned path, etc.

2.1.8 Site Adaptation Requirements

For research and prototype purposes, no site adaptation requirements are relevant.

2.2 Product Functions

This system utilizes resources from more than one computer and controls multiple actuators. The system is designed to minimize user interference during operations, however, monitoring is required. Maintainability of the system is not considered to be user-oriented, which means a user/operator should not be worried about maintaining the system. The system will be made available publicly (open source), however, only users with extensive robotics experience should attempt at modifying the system outside of the given parameters.

2.3 User Characteristics

Our system should be able to be built and operated by a user with at least two years of university engineering coursework and one two years, a user with over five years of technical experience through work in either an electrical or software industry, and a RC hobbyist with over five years of experience.

2.4 Constraints

As this is a research project, constraints may rapidly change or new constraints may be retroactively added to better address problems found or the needs of the system.

Current constraints for the project include:

1) Real time image processing will limit the vehicle's ability to maneuver through obstacles.
2) Telemetry data displayed to the user will be slightly delayed due to long distance. No solutions to this issue are currently available.
3) The vehicle's natural structure will inevitably cause more uncertainty on predictions, which requires the use of more complicated algorithms versus what is required for ideal conditions.
4) Hardware may fall off during operation which will force the system to halt. All components must be securely fastened and adequately protected in the case of an emergency.
5) Telemetry data transfer are limited to radio during outdoor operations.
6) The success rate are set to 80% and above.
7) For research purpose only, the vehicle will only be required to navigate around static objects in an environment. This is to reduce the complexity of the initial system.

2.5 Assumptions and Dependencies

As a research project, the main assumption is currently this is possible.

2.6 Apportionment of Requirements

See next page.

Fig. 2. ARC Fall Schedule Gantt Chart

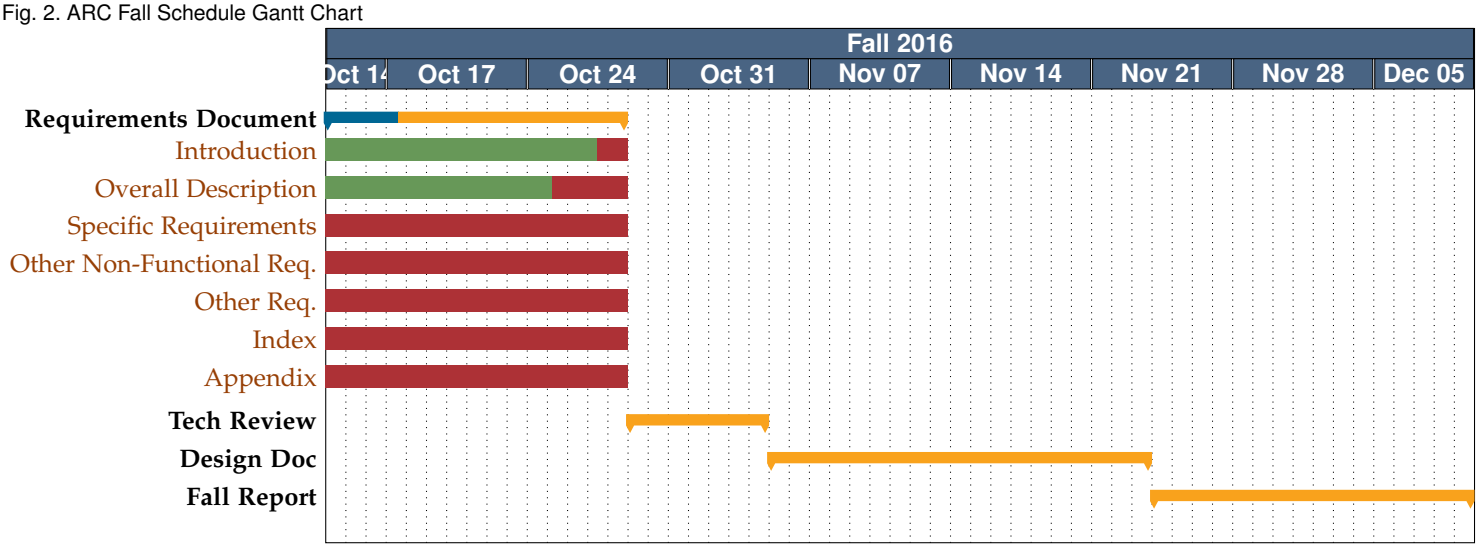


Fig. 3. ARC Winter Schedule Gantt Chart

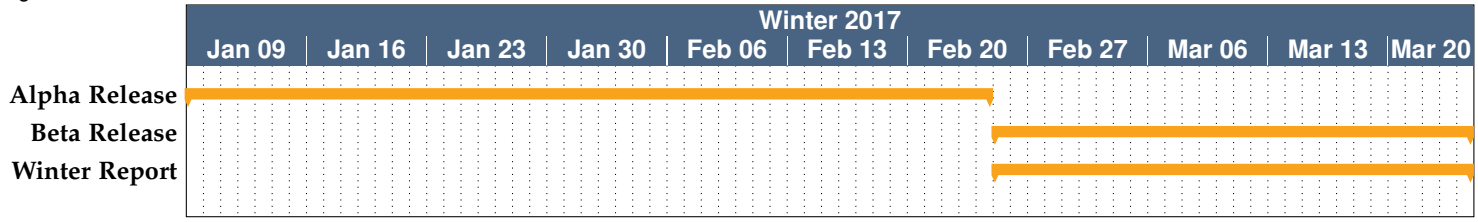
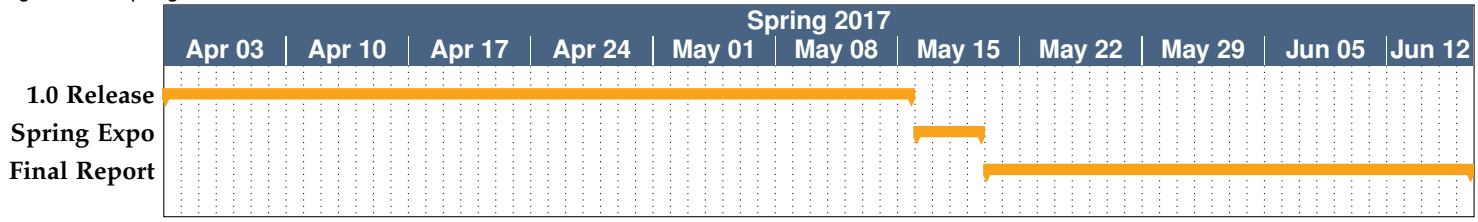


Fig. 4. ARC Spring Schedule Gantt Chart



3 SPECIFIC REQUIREMENTS

This section describes the hardware, software, and performance requirements necessary for the ARC system to function. It lays out requirements for input and output of the system through interfaces. It also covers the major systems features and functional requirements for those features.

3.1 External interface requirements

This section describes all of the interfaces required for input to and output from the ARC system.

3.1.1 User interfaces

There should be a remote station (commonly referred to as a ground station) where all information about the vehicle can be seen. The ground station should allow user control of the vehicle including setting way-points, adjusting parameters for vehicle behavior, and direct control over vehicle movement.

3.1.2 Hardware interfaces

There should be hardware interfaces for manual control of the vehicle. This interface should be able to override software control so that users can safely commandeer control of the vehicle from the autopilot. Other hardware interfaces should be between a network interface between the autopilot and the companion computer on the vehicle. There should also be a hardware interface between the autopilot and the vehicle control system. We will use existing protocols for communication between the autopilot and the companion computer and the vehicle.

3.1.3 Software interfaces

There should be software interfaces between the user interfaces and the hardware interfaces. These interfaces will be handled by the operating system on both the ground station and the companion computer.

3.1.4 Communications interfaces

3.2 System features

Describes what the ARC system can be expected to do in terms of its major features.

3.2.1 System Feature: Image Analysis

3.2.1.1 Functional requirement image analysis 1:

1) ID: FR-IA1
2) Title: Fast Image Processing.
3) Description: Image analysis needs to be at a rate of around 15 or more frames per second.
4) Rationale: In order for the vehicle to move fast, images will need to be acquired and processed very fast.
5) Dependencies: N/A

3.2.1.2 Functional requirement image analysis 2:

1) ID: FR-IA2
2) Title: Depth finding
3) Description: Image analysis needs to determine how far away objects are.
4) Rationale: The proximity of objects will determine timing for speed and turning.
5) Dependencies: FR-IA1

3.2.1.3 Functional requirement image analysis 3:

1) ID: FR-IA3
2) Title: Object height
3) Description: Objects that are required to be detected, must be at least 12″ tall.
4) Rationale: The car should be able to avoid obstacles that are taller than the car itself.
5) Dependencies: FR-IA2

3.2.2 System Feature: Sensors

3.2.2.1 Functional requirement sensor 1:

1) ID: FR-SN1
2) Title: GPS Data Collection
3) Description: When outside, GPS coordinates should be able to be obtained from a GPS unit and be accurate up to 10 meters, in an open space with no trees within 50 meters.
4) Rationale: Within an open field, this accuracy should be able to be obtained in order to provide reasonable locational accuracy to be used in navigation.
5) Dependencies: FR-HW1, FR-HW2

3.2.2.2 Functional requirement sensor 2:

1) ID: FR-SN2
2) Title: GPS Data Processing
3) Description: Data collected from the GPS unit will need to be relayed to the user through the main computational effort.
4) Rationale: A user needs to be able to view the location of the car at any time.
5) Dependencies: FR-HW1, FR-HW2, FR-SN1

3.2.3 System Feature: Navigation

The ARC vehicle needs to navigate a course autonomously. The following functional requirements describe what is required to achieve this.

3.2.3.1 Functional requirement navigation 1:

1) ID: FR-NAV1

2) Title: Motor Control
3) Description: To limit the uncertainty caused by the motor under 25% when going forward and backward.
4) Rationale: Maximize the accuracy of motor control to make prediction easier.
5) Dependencies: Probabilistic Analysis for motion

3.2.3.2 Functional requirement navigation 2:

1) ID: FR-NAV2
2) Title: Servo Control
3) Description: To limit the uncertainty caused by the servo under 25% when turning.
4) Rationale: Maximize the accuracy of servo control to make prediction easier.
5) Dependencies:

3.2.3.3 Functional requirement navigation 3:

1) ID: FR-NAV3
2) Title: Probabilistic Analysis for motion
3) Description: Use probability to estimate the current location of the vehicle with less than 25% error.
4) Rationale: Tires may slip and wobble, which will cause uncertainty on the location of the vehicle.
5) Dependencies: Motion Model

3.2.3.4 Functional requirement navigation 4:

1) ID: FR-NAV4
2) Title: Motion Model
3) Description: Motion model is significant as it decides what commands should be given by the computer under various circumstances.
4) Rationale: This is like learning how to drive a car. Knowing the configuration of the car is critical for the computer to know how to control it. Different combinations of speed and turning angle will result in different paths. Under different surface and weight distributions, the vehicle will also act differently.
5) Dependencies: Motor Control & Servo Control

3.2.3.5 Functional requirement navigation 5:

1) ID: FR-NAV5
2) Title: Global Path Planning
3) Description: When operating outdoor, the vehicle needs to go from one location to another following a legal path that is determined by algorithms using sensor data.
4) Rationale: A legal path is a path that goes on concrete surfaces and does not run into any objects.
5) Dependencies: Obstacle Avoidance, Sensors, Probabilistic Analysis for Motion

3.2.3.6 Functional requirement navigation 6:

1) ID: FR-NAV6
2) Title: Local Path Planning

3) Description: When operating indoor and given the map of the indoor area, the vehicle needs to go from one location to another within the area following a legal path that is determined by algorithms using sensor data.
4) Rationale: A legal path is a path that does not run into any objects.
5) Dependencies: Obstacle Avoidance, Sensors, Probabilistic Analysis for Motion

3.2.3.7 Functional requirement navigation 7:

1) ID: FR-NAV7
2) Title: Obstacle Avoidance
3) Description: When approaching an object, the vehicle needs to decide whether to turn left or right and by how much. The algorithm can overwrite the path planned by the path planning algorithms.
4) Rationale: When operating both indoor and outdoor, crashing into objects needs to be avoided.
5) Dependencies: Sensors, Control System, Probabilistic Analysis for Motion

3.2.3.8 Functional requirement navigation 8:

1) ID: FR-NAV8
2) Title: Parallel Parking
3) Description: Given sensor data and the estimate of the current location, the algorithm outputs the next optimal action (turning the front wheel a certain angle and go forward/backward at a certain speed).
4) Rationale: A dedicated algorithm for parallel parking is necessary because path planning algorithms do not promise the orientation of the vehicle. The parallel parking algorithm will make sure the vehicle is align with the curb/wall when finished.
5) Dependencies: Sensors, Control System, Probabilistic Analysis for Motion

3.2.4 System Feature: Hardware mounting

3.2.4.1 Functional requirement 1:

1) ID: FR-HW1
2) Title: Minimize Port and Hardware Exposure
3) Description: Seal any unused ports, encase electronic components to minimize environmental exposure.
4) Rationale: Minimizing dust and other elements to electronic components to reduce wear and tear.
5) Dependencies: N/A

3.2.4.2 Functional requirement 2:

1) ID: FR-HW2
2) Title: Secure Hardware Mounting
3) Description: Hardware should be mounted in such a way, that in the event that the vehicle rolls over or high-speed impact, hardware remains in place and moves no more than 1 cm from its original location.
4) Rationale: In the event of an impact or rollover, the hardware should remain in place, and not detach from the vehicle.
5) Dependencies: FR-HW1

3.2.5 System Feature: Communications

3.2.5.1 Functional Requirement Comms 1:

1) ID: FR-CM1
2) Title: Telemetry
3) Description: Telemetry needs to be transmitted from the vehicle to a ground station.
4) Rationale: Users need to see the current state of the vehicle at all times.
5) Dependencies: N/A

3.2.5.2 Functional Requirement Comms 2:

1) ID: FR-CM2
2) Title: Vehicle Control
3) Description: Control signals, such as ”start”, ”stop”, ”go to way-point”, etc. needs to be sent to the vehicle. Signals should be received and processed in under 2 seconds.
4) Rationale: Users need to have a way to command the vehicle remotely.
5) Dependencies: N/A

3.2.5.3 Functional Requirement Comms 3:

1) ID: FR-CM3
2) Title: Emergency Stop
3) Description: When sent a signal to stop, the car needs to stop within 2 seconds from when the command is sent.
4) Rationale: A failsafe must be in place if the car is ”going rogue”.
5) Dependencies: FR-CM2

3.3 Performance Requirements

This section specifies numerical requirements placed on ARC system:

- The system only supports one user at a time.
- The system only provides one interface for user interaction at any single moment.
- The system supports no more than 5 waypoints as inputs.
- The system is capable of driving the vehicle at maximum 10 miles per hour.
- The system is capable of navigating in an indoor environment of area no larger than 500 square feet.
- The system is capable of navigating outdoor without space constraints.

3.4 Design constraints

The ARC RC vehicle will be guaranteed to communication with the ground station within 2km due to limitations of the telemetry radio.

3.5 Software System Attributes

As this is a research project, requirements found along the way will be appended onto this section.

3.6 Other Requirements

As this is a research project, requirements found along the way will be appended onto this section.

3 DESIGN DOCUMENT

3.1 Changes in Design

We had to make several changes to our design throughout the life of the project. We moved away from stereo vision for obstacle avoidance and went with LiDAR instead. This had a cascading effect on other parts of our design. Since we didn't use stereo vision, our image analysis design also changed. We didn't need to worry about communicating images over the network so our communication protocols became much more simple.

3.2 Original Design Document

(Begins on next page).

Design Document

ARC - Autonomous RC

Tao Chen, Cierra Shawe, Daniel Stoyer

Version 1.0

December 2, 2016

CONTENTS

1 Overview

This document will provide an overview of current design and milestones for the ARC system (ARCS).

1.1 Scope

This document is issued on December 2, 2016. At this time, system design is not final, due to the nature of a research project, and shall be modified as necessary.

1.2 Purpose

Each section of the document will provide an overview of the different systems contained within ARCS.

1.3 Intended Audience

The intended audience for this design document is the development team. As the design is not final, technical users are currently not taken into account.

2 System Interfaces - Cierra

The main computational unit (MCU), an Intel NUC SkullCanyon, is used as a starting point to test the feasibility of stereo-vision for a vision system as per FR-IA1. The NUC runs Ubuntu 14.04 to provide a graphical user interface for observing inputs such as the vision system and sensor input. Ubuntu 14.04 is also a standard used to run ROS and handle input from other nodes. All of the data processing will be done on the MCU.

The MCU interfaces in the following ways:

- Telemetry radio via a USB connection
- Raspberry Pi 2 via an ethernet connection
- Vision system via two USB ports.

19V of power is the manufacturer's rated usage during peak performance for the Intel NUC. Testing must be done to see if the NUC can run on 12V power, which is the RC battery standard, while running complex computations such as processing images into a point cloud.
The NUC should be able to process at least 15 frames a second without powering off while using 12V power. If the NUC is unable to run on 12V power, a new method for powering the NUC is needed, or a less powerful board, such as the UPBoard or a NVIDIA Jetson must be used. Using a less powerful board limits the speed in which vision is processed.

The Raspberry Pi interfaces with the PXFmini autopilot through a 40pin connection. Power is supplied via a micro USB port, or through the PXFmini, which has the option to be powered through a pass-through connection to main LiPo battery. The Raspberry Pi runs Debian Jessie ("cyan"), which is provided by Erle Robotics and is designed to process input from the PXFmini. The Raspberry Pi and PXFmini control motors and sensors, and collect sensor data.

Data is passed via a client-server model to the MCU using a TCP connection over an Ethernet connection. Packets are sent as JSON if the Raspberry Pi cannot be configured as a node within ROS.

The PXFmini provides sensor data and can directly interface through UART and I2C connections with a ublox Neo GPS with Compass unit and provides built-in support. The GPS unit can also interface through an IMU for more complex and accurate motion information. An IMU unit requires adapters to interface with the new Drone Foundation standard. Servos and a motor speed controller functions interface through the pulse width modulation pins on the PXFmini. The PXFmini abstracts the majority of motor and servo functions.

The system will be tested in incremental steps to ensure that all of the devices are talking to each other. The first step is to establish a connection between the MCU and the Raspberry Pi via TCP. Then a ping packet will be sent, with an expected response of pong.

With a working connection, the first goal is to control the direction of a motor from a command sent from ROS to the PXFmini. The motor should be able to spin one way when told to go forward, and reverse when a reverse command is sent. Then servo control will be tested.

After motor and servo control is established, sensor data will be collected and sent to the MCU. Two servos should be able to run synchronously to simulate steering. Data should be able to be displayed on to the user through the ROS interface. At this point, the system can be interfaced into the car with extreme caution. Once the servos and speed controller from the car has been connected to the PXFmini, all functions should be tested on minimum speed for the case of motors or the using least amount of travel for servos, to ensure all actions are as expected.

3 Vision System - Cierra

ARCS uses a stereoscopic vision system to aid in navigation and obstacle avoidance. Vision systems are the ”eyes” of the vehicle and are what allow the vehicle to operate autonomously. The vision system relates to requirement FR-IA1 and is a precursor to requirements FR-IA2 and FR-IA3. FR-IA1 is relevant because images must be captured and processed at a rate of 15 frames per second or higher to allow the vehicle to navigate quickly without colliding with objects. The vision system uses input from two USB cameras to create a depth map to detect objects at least 8” high to avoid collisions with objects that the vehicle is unable to drive over. OpenCV and ROS are used to process images from two different USB cameras, plugged into the MCU. ROS handles depth maps using a SLAM library in either a point cloud or occupancy grid map format, and the output is required for FR-NAV6.

The first step to testing whether the vision system works is ensuring input is received from both cameras simultaneously through visual observation on a display. Then after input is successfully received, the cameras must be calibrated using OpenCV. The calibration process involves using a checkerboard and taking images from various locations. The MCU plugged into a display, must then display a depth map based on the disparity between the left and right images. Once a depth map is created, frame rate must be compared to find the optimal resolution to process images successfully at 15 frames a second, with 60 frames being an ideal number. Frames per second are outputted onto a display for the user to see in order to ensure the frame-rate is above the target.

Currently, the optimal distance between cameras is not known, so testing needs to be done to see how far apart the

cameras must be placed to view objects between two and 20 feet away. The optimal range would be 6 inches to 100 feet of range. However, this goal is unlikely due to the restraints in image quality and the ability of the MCU to process the disparity maps in real-time. Using lower quality images provides fewer data points for OpenCV to compare, with a tendency towards unclear pixels on the edges of the images, unclear images when objects are very close to the cameras, and poor depth calculation when pixels are too far apart.

Other sensors, such as sonar or LiDAR, can be used alongside the stereo-vision cameras if the stereo cameras are unable to process objects within three feet of the vehicle. Other vision sensors are only used when it is determined stereo-vision alone is not enough to detect obstacles at close range.

Testing for the optimal distance for the two USB cameras is done by calibrating the cameras and holding a white piece of paper in front of the cameras from different distances, and then determining when the cameras can no longer detect the depth. After the camera distance have been calculated, the two cameras are mounted either via a 3d printed case or integrated near the car's headlights.

4 Sensors - Cierra

Sensors provide critical information that is used to provide feedback for where the vehicle is within a space. Sensors are a pre-requisite for all navigation criteria. FR-SN1, FR-SN2, FR-SN3 are the requirements for all sensor input. Sensors required for the vehicle are the following:

- Accelerometer to calculate acceleration and deceleration
- Gyroscope for orientation
- GPS with a compass for location
- Barometer for altitude
- Ultrasonic sensors for parallel parking and forward collision avoidance
- Encoders to measure wheel rotations (as needed)

The accelerometer and gyroscope are included on the PXFmini as a 9-axis sensor, or the data can be pulled from the external IMU. The GPS unit can be plugged directly into the PXFmini using one of the UART and I2C ports. Barometer data is not strictly required. However, the PXFmini has a built-in sensor, so it can be used as needed.

Sensor data is sent through the Raspberry Pi to the MCU, where it is processed by path planning and control algorithms. Obtaining sensor data is required before these algorithms are able to be implemented.

An I2C port will be multiplexed to support additional sensors if needed. Each sensor will be assigned a UID to be accessed through the multiplexed port.

Testing sensor output will be done by displaying values to the user to ensure the sensors are sending information to the MCU. The 9-axis sensor will be placed on a level sensor to ensure that it is properly calibrated.

5 HARDWARE - CIERRA

All hardware components will be tied down and encased with the minimal amount of port access required for the car to function as per FR-HW1. 3D printing and laser cut cases will be used, and venting will be used as necessary. This is to protect the hardware from rollovers as per FR-HW2.

Hardware layout will first be done on cardboard to ensure components are in the correct placement on the board. After the layout is determined, cases for objects will be modeled and printed, then secured onto an acrylic or polycarbonate laser-cut plate.

6 MOTION MODEL

Each iteration, the system issues a series of commands to the actuators, which are the motor and the servos in this project. Actuators then actuate the commands for the duration of one iteration. The time taken by each iteration may vary due to the different amount of information the system has to process. A control scheme/method must be needed to avoid the inconsistency in the actuators' behaviors caused by the inconsistent duration.

The speed control package from the AutoRally project is used to satisfy the speed consistency requirement. The system controls the speed by publishing a speed message to the listener, same as the steering.

6.1 Milestones:

1) Complete all simulations in ROS.
2) The motor can physically spin forward and backward according to instructions given by the system.
3) Servos can physically turn clockwise and counterclockwise according to instructions given by the system.
4) The speed of the vehicle is consistent with the speed issued by the system regardless of the road condition.

Motion model determines the correct commands to be issued by the system. A person has to adapt to the acceleration and steering of a particular car. The ARC system also has to adapt to the vehicle's configuration, such as weight, steering ratio, etc.

Besides, to follow a given path produced by the path planning algorithms, the system requires another control scheme, to ensure the vehicle strictly follow the path. When operating on a rough surface, drifting/slip off path happens. Converging back to the path also requires the same control scheme. The adaptation of the system to the vehicle is done by entering a set of parameters that include:

- Center of gravity.
- Weight.
- Turning radius to steering ratio.
- Maximum speed.
- Minimum turning radius
- Note: We need to keep it simple given that we have not even started any testing. These 5 parameters will satisfy the basic operation of the vehicle. The system will change the control strategies based on the parameters set.

6.2 Milestones:

1) Test the vehicle and record all necessary data.
2) Analyze the collected data by hand and simulate different configurations in ROS.
3) Derive equations from the results returned by simulations.
4) Polish the equations by doing more simulations in ROS.
5) Test the results on the actual vehicle with the minimum amount of hardware mounted on it.
6) Test the results on the actual vehicle with fully loaded hardware.

The PID (proportional–integral–derivative) control model is used to make sure that the vehicle is always on track. The PID model makes sure that, when the vehicle runs over bumps and drift off paths, it converges back to the desired path without overshoot or undershoot. When operating on a bumpy surface, the PID model is extremely critical to keep the vehicle on the paths.

6.3 Milestones:

1) Fully understand the PID control model.
2) Customize the model to adapt to our vehicle.
3) Simulate the model in ROS.
4) Test the model on a smooth surface and regularly changing the paths. The vehicle should converge to the new path every time.
5) Test the model on a rough surface. The vehicle should converge back to the planned path every time it drifts off paths.

The system has to know the location of the vehicle at any given moment. After each iteration when a series of actions are taken, the uncertainty of the vehicle location increases. Thus, the system has to utilize probabilistic methods with sensor measurements to minimize the uncertainty. The inputs to this modules are the previous location of the vehicle, actions taken in the previous iteration, and the sensors measurements, such as GPS, wheel encoders, vision system. With Bayes filter, the module produces reliable results.

6.4 Milestones:

1) Fully understand Bayes filter so that we can modify the library according to our need.
2) Being able to gather sensor measurements and feed them to the module.
3) Simulate the module on ROS.
4) Optimize the module so that the computation can be completed within a normal iteration.
5) Test the module on the actual vehicle.

7 PATH PLANNING

7.1 Global path planning:

When operating outdoor in an open area with a pre-downloaded satellite map, a global path planning module is required for the vehicle to travel from start to destination as desired. Dijkstra's based A* algorithm is used. The inputs to the module are the start and end locations (GPS coordinates), and a satellite map. The pre-processing unit generates valid waypoints, edges and edge costs by analyzing the map. Those waypoints and edges then get passed to the algorithm, which makes sure the shortest path is found. Since the implementation of Dijkstra's algorithm is not complicated, it is fully customized to accommodate the inputs/data types specified above. The heuristic for the algorithm is defined to be the straight-line distance from the waypoints to the destination, calculated using the GPS data.

7.2 Milestones:

1) Being able to pull data from the GPS unit.
2) Being able to generate waypoints on map images provided by OpenStreetMap.
3) Implement the algorithm
4) Test the algorithm
5) Test the algorithm with customized data structs.
6) Test the algorithm on the vehicle outdoor.

7.3 Local path planning:

When operating indoor within a closed area (with pre-defined map and boundaries), local path planning is required for the vehicle to travel from start to destination. Rapidly-exploring random tree is the algorithm used by the system. The inputs to the algorithm are start location (known), destination, the map with boundaries but without marked features/obstacles. The start location is either the very start location where the vehicle initializes and that is input by humans, or the subsequence locations estimated by the motion analysis. The vehicle also maps the space while it is traversing within it. It detects obstacles as it goes.

7.4 Milestones:

1) Customize the algorithm so that it does not generate samples within the vehicle size.
2) Test the algorithm by giving it a starting point, a destination point, and a map with marked features.
3) Integrate the algorithm into the system.
4) Simulate the algorithm in ROS.
5) Optimize the algorithm so that it plans out a path within the shortest amount of time. It needs to be short enough that human can not notice any delay when it needs to reconstruct a new path.
6) Test the algorithm on the vehicle.

8 Other Algorithms

8.1 Obstacle avoidance:

Obstacle avoidance is required for both local and global operations. Obstacle avoidance module grabs data packages from the vision system and tries to generate new paths on the go. Data included in the packages is defined by the vision system. There are two situations where the module may act differently. The first situation is when the vehicle has to come to a complete stop, due to high speed and path generation delay. The second situation is when the new path generation process is completed before the system decides to come to a complete stop.

8.2 Milestones:

1) Decide the data structs contained in the packages used to transfer data to the module.
2) Implement the module. There are existing packages that can be used. However, modifications must be required.
3) Integrate the path planning modules into the obstacle avoidance module.
4) Test the module in Simulation on ROS.
5) Optimize the algorithm.
6) Test the module on the vehicle.

8.3 Parallel Parking:

Parallel parking requires two special path planning algorithms. The first algorithm detects valid parking spots. This algorithm relies on images analysis results produced by the vision system. The second algorithm is a path planning algorithm specifically designed for parallel parking maneuvers. A similar approach to the Autonomous Parallel Parking RC Car project done at Cornell University is used. The software package provided by Cornell University is modified to use image information instead of readings from ultrasonic sensors.

8.4 Milestones:

1) Modified the software package so that in can be integrated into the system.
2) Test the algorithm by doing simulations on ROS.
3) Find out any optimizations that can be made to the algorithm. For example, minimize the number of turns it takes to complete the tasks.
4) Optimize the algorithm.
5) Set up an environment for testing.
6) Test the algorithms on the vehicle in the pre-set environment.

9 Image Analysis

Author: Dan Stoyer

The ROS image_transport (http://wiki.ros.org/image_transport) library should be used to pass images from the sensors into rtabmap_ros i. rtabmap_ros. The rtabmap_ros library handles depth-finding and environment mapping.

Milestone #1: Detect an object Use ROS image_transport (http://wiki.ros.org/image_transport) to pass images into rtabmap_ros (http://wiki.ros.org/rtabmap_ros). rtabmap_ros analyzes the images to detect objects. proj_min_cluster_size (int, default: 20) is used to control the size of object detected.

Milestone #2: Detect an object while moving After determining that rtabmap_ros is detecting an object while the vehicle is stationary, object detection should be tested while moving.

Milestone #4: Depth finding Once objects are detected, the distance to the object should be found by passing the images to rtabmap_ros's left/image_rect (sensor_msgs/Image) and right/image_rect (sensor_msgs/Image). The depth analyzed from the images should be tracked using subscribe_depth (bool, default: "false").

Milestone #5: Hand off analysis The object detection and depth information should be handed over to the path-finding algorithm.

10 User Interface

Author: Dan Stoyer

The user interface (UI) allows the user to issue commands and see the state of the vehicle. There are two UI platforms at play: the ground station and the companion computer. The companion computer UI is the most important to get working right away.

Milestone #1: CLI for companion computer. The robotics operating system (ROS) provides CLI for vehicle control. This interface is on the companion computer only.

Milestone #2: See vehicle status on remote terminal. The companion computer's UI should be transmitted over wifi through screen-share software. This approach allows immediate viewing of what is happening on the companion computer which allows time to work on more advanced remote UI solutions.

Milestone #3: Enter CLI commands on the ground station. The simplest place to start with remote commands is from the command line. The ground station will be running either Linux or Windows. In the case of Windows, a third-party terminal (such as MinGW or Cygwin) should be used.

Milestone #4: See vehicle status via ground station GUI. To see the vehicle status, a GUI is needed on the ground station. The ground station will run either a Linux distribution or Windows 7,8,10. ArduPilot runs on both and should be the first GUI tested. If ArduPilot does not Telemetry data should be sent with MAVLink protocol. Image data should be sent to rtabmap_ros using image_transport (http://wiki.ros.org/image_transport).

11 Radio Communication

Author: Dan Stoyer

Milestone #1: Send/Receive telemetry data The companion computer should send telemetry data by radio using the MAVLink protocol. The ground station should receive the telemetry data by radio using the MAVLink protocol.

Milesone #2: Send commands The ground station should send commands. Testing for seding commands should start with simple ″start″ and ″stop″ commands and gradually increase in complexity up to enter waypoints for autonomous navigation.

12 APPENDIX

12.1 Annex A Bibliography

4 TECH REVIEW

4.1 Changes in Tech

As our research and development process matured, we discovered different ways of doing things and changed direction on some of the technology we used for the project. We changed from stereo vision to LiDAR for depth-finding and obstacle detection. We changed from using the PXFMini autopilot with the Raspberry Pi 3 to using a PWM controller with an Arduino to control motors and servos. We want to use the Q Ground Control GUI to control the vehicle remotely, but had to scratch that when we abandoned the PXFMini. We switched to using RVIZ (visualization package for ROS) for vehicle control

4.2 Original Tech Review Document

(Begins on next page).

Group 44 - ARC

Tech Review

Senior Capstone Project

Oregon State University

Fall 2016

Tao Chen, Cierra Shawe, Daniel Stoyer

Abstract

CONTENTS

1 Introduction

This technology review covers a subset of technologies needed for use with Autonomous RC (ARC). 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 ARC. 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.

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

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

2 Vision System Options

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 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.

2.2 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.

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 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.

2.4 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.

3 Sensors

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

3.1 Internal Measurement Unit (IMU) - External

3.2 IMU - PXFmini

4 System Control and Data Processing

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.

4.1 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.

4.2 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.

4.3 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.

4.4 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.

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]. Avaliable: 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/
[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 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.

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 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. [3] 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: [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].

5 Weekly Blogs

5.1 Cierra's Blog Entries

5.2 Fall

Fall Week 4

Worked On

- This week we met with Kevin, and he asked us to add a section called "motivation" to our problem statement. Although he was mostly content with what we were saying, he wanted it to be clear from the very first paragraph, WHY this project matters, and why someone would even bother supporting it. Apparently, it went poof in his inbox, since he couldn't find it.

Problems Encountered

- Our biggest "problem" this week, was not knowing the status of our project statement.

Plans for next week

- Next week the goal is to get our project statement signed, on Monday in our meeting, and complete a draft of our Requirements Document.

Fall Week 5

Worked On

- Filling out the introduction section and other various parts of the SRS document.
- Overall layout of the general control flow.

Problems Encountered

- I think our biggest problem is not having a strong understanding of specifics of our project. It's also hard to have specifics for what we will be able to do, and what we'll have, and how things will play together.

Plans for next week

- Plans for next week include getting more specifics on the what and how portions of our requirements will work and should be within the document.

Fall Week 6

Worked On

- Finding out about what technology is available.
- Filling out sections of the SRS (hardware spec).

- Got an extension for the SRS.

Problems Encountered

- The biggest problem has been finding a time where we can all get together and finish the document.

Plans for next week

- Our main plan for next week is solidifying what our requirements are, and attempting to finish the SRS document, along with assigning ”roles” or requirements of the project.

Fall Week 7

Worked On

- Researched different options for vision systems, and their respective advantages and disadvantages. There is a lot of research in this field, but it’s hard to find tangible examples instead.

Problems Encountered

- The biggest problem is sorting through different options, and figuring out what is reasonably implementable for our project. It’s also difficult to gauge how much computing power we would need.

- Time has been a big factor for us in the past few weeks. The SRS has been very difficult without the Tech Review, as we don’t know about most of the technologies that we would need. If this was a purely software based project, the SRS would have been fine because we have a general idea of what to do, versus this project has a heavy hardware focus, which makes it harder for us poor CS majors to have a crash course in basic ECE concepts. It feels as if we keep pushing deadlines, due to the overhead that this project requires in terms of material that needs to be learned. I’m not sure if I can speak for everyone, but I feel that we would all like to be able to finish our assignments in a timely manner, it just doesn’t feel realistic due to all of the components required.

Plans for next week

- The goal for next week is to finish out the tech review and finally be able to complete the SRS document. At this point, without the tech review, it doesn’t seem feasible to get the SRS done without finishing the tech review first.

Fall Week 8

Worked On

- Worked on completing the Tech Review and updating the SRS document. Overall, the research has been slow, as documentation is a lot of information to parse through. Plus, I find research papers interesting to read, so it’s hard to not get lost in all of the cool information available on different topics.

Problems Encountered

- The biggest problem has definitely been getting the SRS done this term. We're relatively unsure of what it's supposed to look like when our entire project is based on finding if these things are even possible. The Tech Review was is also very time-consuming, since there have been a lot of unknowns within our project.

Plans for next week

- Well, we still need to finish the SRS and start the design document. I am hesitant much will get done over Thanksgiving.

Fall Week 9

Worked On

- Worked on requirements in the requirements doc. Started to discuss the design document. Overall, not a lot has been completed due to the holiday.

Problems Encountered

- Worked on requirements in the requirements doc. Started to discuss the design document. Overall, not a lot has been completed due to the holiday.

Plans for next week

- Finish the design document and then start the end of term project write-up and video. We also still need to turn in our SRS.

Fall Week 10

Worked On

- Finishing SRS and design documents, basically, we want to pass this term.

Problems Encountered

- The SRS document has been really difficult to do without the tech review and design, we really weren't able to finish the SRS. We did finally make some headway, and everything should be in by the end of the term.

Plans for next week

- Ensure everything is turned in, and then enjoy a well needed break. Probably do some research in that time.

Fall Week 11

Worked On

- Finals and turned in all of our documents.

Problems Encountered

- Not enough sleep.

Plans for next week

- Absolutely nothing - yay break!

5.3 Winter

Winter Week 1

Worked On

- Tried to obtain software for our PXFmini.
- Worked the design for our stereo vision system. The idea is that the mount will be able to have adjustable slides in order to allow for us to test what the optimal distance for the cameras is. This first revision will be just to make sure to that the stereo vision is working correctly.
- We managed to find times that worked with all of our schedules, which was a big improvement over winter term. Now, we are meeting twice a week from 4pm until we are ready to finish. We also identified that Sunday afternoons are a time where we are mostly free.

Problems Encountered

- Obtaining the software has been unsuccessful.

Plans for next week

- My plan is to draft a CAD drawing and identify the hardware and software that we're missing. As a group, we'll start to identify what it will take to get Kevin's approval to start using the car.

Winter Week 2

Worked On

- Created a model to be 3D printed for our stereo-vision mount.
- Researched more about stereovision.

Problems Encountered

- We are still waiting on hardware and software. This has been a huge hangup.

Plans for next week

- Try to obtain hardware and software, and see if we can get it working together. Also, print the case for our stereo camera.

Winter Week 3

Worked On

- Obtained PXFmini software - however, the original image was corrupt.
  - Raspberry Pi 3
  - PXFmini software (Corrupt and working version)
  - Motor and speed controller for testing
  - Charger for a 3-cell LiPo battery
  - Alligator clips to allow us to power the speed controller.
- Printed camera assembly, there is a lot of room for improvement.

Problems Encountered

- Our original PXFmini image was corrupt and we didn't have a cord to connect it to a display.
- The 3D print turned out well, but I've identified some things that will need to be changed: fix flanges on the assembly to either clip in, or have a better fit that accounts for different tolerances within 3D printing.
- I need to get a better understanding of ROS, as I'm not quite sure how to interact with it.

Plans for next week

- My goal will be to revise our stereo mount, get it assembled (and obtain teeny, tiny screws) to keep the cameras in place.
- Work on getting a better understanding of ROS in order to be able to help more with the software side of the project. I've mostly been focusing on the hardware aspect of it, and I'd like to be able to help more with the software in ways other than chasing it down.

Winter Week 4

Worked On

- This week I worked on reassembling the camera enclosure and looking more into RTABmap. I had the opportunity to see a friend's project while I was in Seattle that was using a ZED and RTABmap and ROS.
- I also started to solder the Serial cable we needed to the IMU. After setting this up, I was able to get data from the IMU on a windows computer.

Problems Encountered

- When doing the initial wiring diagram, I was going TX to RX, which should be correct; however, ultimately I needed to have TXI go to TXO and RXI go to RX0
- We're still waiting on more hardware, and once we have that, I can start trying to use the new LiDAR unit Kevin ordered from us.
- I'm worried that the rolling shutter cameras will cause problems in the future.

- I also need to find a ROS package for the IMU. While the Windows GUI is nice, it does not help us for integrating the IMU into our car.

Plans for next week

- Working the ROS tutorials and attempting to calibrate the cameras. Once that is done, I would like to be able to test the camera with RTABmap.
- Talk to Kevin about other options for vision systems.

Winter Week 5

Worked On

- Attempted to get the stereo vision working through OpenCV.
- One of the options we are now considering is the LeddarM16 for our vision system. This would be paired with another LiDAR unit to provide a better range of view.
- I found an IMU package for ROS.
- I talked with Kevin, and we will try to us a LeddarM16 moving forward, to replace the stereo cameras. This will give us thousands of updates a second, in comparison to about 5. The Leddar is also really cool, because it works in direct sunlight, due to the solid state LED based system that they use.

Problems Encountered

- I was able to view the two stereo images on the NUC, but the lag between them was almost half a second. Basically, that meant our stereo vision set up will not work for the project. If the two images are not able to be synced, then the computer will never be able to create a disparity map that is of any use to ROS. All in all, stereo vision will not work for our project with the current hardware.
- Even though I was able to find a ROS package for the IMU, which is what we need, I was not able to find any C code that would have allowed us to interface with it. Also, when I tried to watch the serial port, I was only able to see gibberish, no matter what baud rate I used.
- Still blocked on hardware. Even though Kevin had ordered the Leddar unit, we don't have access to the documentation and there isn't a lot about it.

Plans for next week

- A lot of the week will probably be focused on getting the midterm together.
- Try to get the motors moving.

Winter Week 6

Worked On

- Complete the midterm report and presentation.
- The Leddar should be here next week.

Problems Encountered

- We still can't talk with the NUC.

Plans for next week

- Hopefully work with the Leddar.

Winter Week 7

Worked On

- Important links:
  - How to enable i2c on the pi
    https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c
  - How to use the PWM board library:
    https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/using-the-adafruit-library
  - Pi 3 pin layout:
    http://pi4j.com/pins/model-3b-rev1.html
- How to move the motors and servos (as of Friday)
  - Plug in the PWM usb cable to 5v power supply (ideally 2.5A).
  - Connect Display cable
  - Start the pi by plugging in the power supply (using ideally a 5v 2.5A/3A power supply)
  - Navigate to /Adafruit_Python_PCA9685/examples
  - Plug in the ESC to PWM port 1 and servos into PWM ports 2 and 3
  - Run "sudo python esc_test.py"
  - Press "t" to set the esc to the top speed
  - Plug in the battery to the esc
  - Wait a second and then hit b to set the esc to the lower speed limit. Now you can use the arrow keys to accelerate, reverse, and turn.
  - Hook up peripherals

Problems Encountered

- Just a lot of progress.

Plans for next week

- Integrate motor control into ROS
- Look into GPS

Winter Week 8

Worked On

- Leddar! Looked into the Leddar API and Tao and I were able to get it to created a point cloud. There is a LeddarM16 ROS package, nifty!

Problems Encountered

- There is almost no actual documentation on the Leddar.

Plans for next week

- See about integrating both the RPLiDAR and Leddar in ROS.

Winter Week 9

Worked On

- We were able to get the Leddar and LiDAR working together. The Leddar really doesn't like glass, and will give wonkey readings.

Problems Encountered

- Leddar doesn't like glass.

Plans for next week

- Report.

Winter Week 10

Worked On

- Starting to get the presentation and paper together for the end of the term.
- Worked with Tao to get the motors turning, we are also able to control it while hard wired with a joystick. We also mapped the different outputs of the PWM controller. This was using an Arduino and a PWM controller.

Problems Encountered

- Dan's wife had a baby. Bye bye, Dan. It's not really a problem, we just don't know when he'll be back, and will have to adjust the project accordingly for the rest of the year.

Plans for next week

- Record and submit video.
- Finish and submit term report.
- We have finals and I leave for Seattle on Wednesday, so probably not a lot after that.

Winter Week 11

Worked On

- Recorded end of term video.
- Submitted end of paper.

Problems Encountered

- Still no Dan. Still not a problem, but we really have no idea how this will affect us moving forward.

Plans for next week

- Europe :D

5.4 Spring

Spring Week 1

Worked On

- Worked on wiring the 12v connection for the Leddar.
- Worked with Dan on getting the IMU and ”LiDAR” sensors working in ROS.
- Made our schedules available to everyone so we could start planning when we will meet during the term. We also set up our meeting time with Kevin.

Problems Encountered

- The Leddar has appeared to have died or encountered a problem with USB. We need to talk to Kevin and see what can be done to fix this.

Plans for next week

- Try to start moving our software onto the hardware. This will allow us to test if the simulation Tao has been working on will work.
- We also need to make sure to register for Expo and really figure out how to implement the hardware and software solutions.
- Need to get radios from Kevin.

Spring Week 2

Worked On

- Tried various computers to see why the Leddar wasn’t working.
- Finally created the 12v cord to power the Leddar.
- Dug through forums and Leddar documentation (or lack thereof) to see why our unit isn’t recognized.
- Started to see how to wire the Leddar using rs485, and obtained the box to convert from RS485 to usb.

Problems Encountered

- Found out the Leddar wasn’t working.

- Found out that there is a jumper that needs to be changed for 12v operation of the Leddar, but have no idea how to change it. We still might have killed it.

Plans for next week

- Start 3D modeling cases so we can print and cut out our soon to be sweet looking car.
- Help with system design and integrating software to hardware.
- Continue to investigate why the Leddar isn't working - follow up with support if no solution is found.
- Do poster.

Spring Week 3

Worked On

- Worked on hardware mounting. We now have a pretty box that I found in the hallway.
- I can now run the entire electrical system off of a LiPo battery.
- Performance tested the NUC. We were able to get about 1:20 minutes off of the LiPo while running the stress tests.

Problems Encountered

- Not a lot.

Plans for next week

- Re-design the box so it doesn't hit.
- Start working on GPS integration.

Spring Week 4

Worked On

- Obtained the new Leddar from Kevin.
- A lot of homework for other classes, trying to get caught up and ahead for this last push.

Problems Encountered

- Time. Time is a problem.

Plans for next week

- Design and print a case so we can have a snazzy car.
- Help with anything else I can.

Spring Week 5

Worked On

- Monday - Wired the pins to the GPS so we could read the data. I also worked with Dan and Tao on finally getting the GPS data. After a few hours, Tao and I were able to read the data. The problem with this, as of right now, is we are using the 3.3v power from the IMU in order to power the GPS. I will work on fixing this later, as I don't feel right using a $180 sensor as a voltage regulator.

Problems Encountered

- Monday - We couldn't figure out why the changes we were making to the GPS config weren't changing, and then I realized that I never soldered an RX pin to the board, so it wasn't able to read new data. To fix this, I simply held a pin in place while we made the changes, that way we had a known configuration. This also allowed us to have faster update rates.
- A worry with this is that the updates don't seem to be accurate enough. We will need to see if there is an orientation that both the sensor and the antenna need to be orientated.

Plans for next week

- Submit everything we have for the code freeze...

Spring Week 6

Worked On

- Focused on getting all of our electrical components working. Was able to start reading GPS data from the GPS.
- As a group we started writing the report and progress report for the project.

Problems Encountered

- The IMU was being used as a $180 power source.
- Somehow all three of our group members misunderstood the poster submission. We submitted the poster and are hoping for the best.

Plans for next week

- Start building cases. Get a new power supply.
- Make the box look pretty for Expo.
- Finish the video.
- Check in to make sure our poster will be done.

Spring Week 7

Worked On

- Printed cases for electrical components.
- Spray painted box and made it look cooler
- Finished the video, project write up draft, and midterm report.

- Worked with Tao on getting the servo's to be able to control the steering via the simulation.
- Survived Expo... Maybe not with my sanity, but definitely survived.

Problems Encountered

- What is sleep?
- The ESC requires a very specific PWM signal pattern on start-up in order to program it. Figuring out how to do this has made it so we have been unable to get the car to be controlled via software. This is the last step we need to overcome in order to get the car moving on its own.

Plans for next week

- Sleep.
- Sleep some more.
- Start working on final report.
- Get caught up on other classes that have been ignored (looks at technical writing)

Spring Week 8

If you were to redo the project from Fall term, what would you tell yourself?
I would tell myself to avoid stereo-vision all together, because while it might seem like a good option, it is fairly difficult to correctly implement. Another thing would have been to avoid using the PXFmini, as trying to get it working took up over half of a term that we could have spent implementing other features. Finally, I would tell myself to take any time estimates, and quadruple them. Nothing will go as planned, so you need to be able to adapt to whatever is thrown your way.

What's the biggest skill you've learned?
As a whole, I have gained a lot of knowledge on how to interface hardware and software together. As an applied computer science major, I never had to take ECE classes, so things like wiring and soldering were new to me. While I had a lot of robotics experience from high school, interfacing software and hardware was always incredibly abstracted, and made as easy as possible. It was a lot different when we were given a pile of hardware and I had to start reading technical specification sheets in order to understand how it worked.

What skills do you see yourself using in the future?
I can't think of one skill that I will not use in the near future. Being able to understand technical specifications and documentation has made it so I will no longer have a hurdle of software-hardware interfaces. I have a few embedded projects I would like to pursue in the near future and this project has really broken down the walls of "I do not know where to even start."

What did you like about the project, and what did you not?
I feel incredibly fortunate to not only have had an awesome project, but an awesome team as well. It is hard not to enjoy working with a massive RC car, that has a ton of power. This project also made me really admire the autonomous vehicles that are currently being developed.

The only thing I did not like about the project, was the lack of actual budget. I really appreciate all of the hardware that Kevin provided us (there is a lot of it), it would have just been nice to have been able to try out some other hardware options. Some examples are a Pixhawk Mini, ZED stereo-vision camera, NVidia Jetson TX1, and more than one GPS and IMU. I can by no means complain about the hardware we were given, as we have access to some incredible things; however, I can not help but wonder if our implementation fell short of the goal because of the hardware or software. I also think that in order to determine if this is possible, or what configuration works best, would require being able to compare different pieces of hardware.

What did you learn from your teammates?

If you were the client for this project, would you be satisfied with the work done?

As a whole, yes. We started from nothing and ended up creating our own ”autopilot” ROS setup, we have the car working in simulation, and we are able to turn the physical wheels on the car. Also, we were waiting on hardware until week 8 of winter term, which means a lot of our actual implementation was done within 10 weeks. If we had been able to start implementing and making significant progress in week 1 of winter, then I have no doubt that we would have been abbe to get the project working.

This project was aimed at reasearching the question of ”is it possible,” and as a whole we did determine it could be given more time.

If your project were to be continued next year, what do you think needs to be worked on?

The biggest part would be to get the motors working. Now that the platform is almost completely built, another group could focus on getting it rolling and implementing the high-speed navigation. I also think it would need at least one more GPS and IMU, along with wheel encoders, in order to give the car better odemtry. I also think it would be interesting to see a group test different hardware, and see what provides the best performance for the price.

Speak a little about your expo experience.

Expo was incredibly stressful for me, but not for the reasons most people would expect. I really do not mind talking with people, but I do have a hard time with loud areas. The group next to us had many people with very loud voices, the group next to them was blasting music most of the day, and the tents amplified the noise even more. These three things made it very overwhelming when trying to talk to people. It was also hard to not be inside the tents, as it was hot and the sun was beating down on us the entire day. One positive to our location was the tub with water and ice was right next to us.

Overall, I had a lot of fun getting to show off our project and present in front of the advisory board. It was cool to see the overwhelmingly positive responses we were getting and the kids really liked the demo we put out with the 360 LiDAR.

5.5 Tao's Blog Entries

5.6 Fall

Fall Week 4

Worked On

- Worked on the problem statement performance measure section.

Problems Encountered

None.

Plans for next week

None.

Fall Week 5

Worked On

- Have not contributed so far.

Problems Encountered

None.

Plans for next week

None.

Fall Week 6

Worked On

- The SRS document.
- Have a better and clear view of the project.
- An abstract block diagram that depicts the structure of the project.

Problems Encountered

- Still a lot of unclear/unspecified aspects of the project.

Plans for next week

- Individual writing.

Fall Week 7

Worked On

- SRS document.
- Tasks breakdown list.

Problems Encountered

- Still a lot of uncertainties on what hardware/interface we are using.

Plans for next week

- Finish SRS document.
- Finish tech review.

Fall Week 8

Worked On

- Worked on the SRS document and the Tech Review. Did research on efficiency of specific algorithms. Since our vehicle will eventually run in different environments, possibly not predefined, algorithms must be robust as well. Thought about the project as a whole. Realized that we could only have a more and more thorough understanding of the different aspects of this project.

Problems Encountered

- We still need more hardware. The SRS document hasn't been finished yet. The research we are doing currently may turn out to be in the wrong direction.

Plans for next week

- Tech review and the SRS document.

Fall Week 9

Worked On

- Tech review and the SRS document.

Problems Encountered

- Haven't started the design document yet.

Plans for next week

- Start the design doc ASAP. Get more hardware in hand.
- We'd better have a clear view (graph/list) of we need to accomplish along with potential solutions.

Fall Week 10

None.

Fall Week 11

None.

5.7 Winter

Winter Week 1

Worked On

- Ros on the NUC is fully functional.
- Tested the LiDAR unit. It worked great.
- Went through some tutorials on Ros.

Some thoughts on the project

- Start our development on the NUC and see if it is possible to migrate all functionalities to the Intel board at the end.
- Start with the LiDAR to implement obstacle avoidance.
- The two cameras we were given provide two separate feeds. I think we need two video streams in one feed such that we only need to dedicate one port to it and it will provide a better interface to interact with the images.
- Need to mount the two cameras on something rigid.

Winter Week 2

Worked On

- Ros tutorials.

Problems Encountered

- A lot of software only supports certain versions of Ros.

Plans for next week

- Car model on Gazebo.

Winter Week 3

Worked On

- More Ros tutorials. RC car model on Gazebo. Simple shape with a rectangular box as the chassis and four wheels.

Problems Encountered

- Ros Indigo only works with Gazebo 2.X, which doesn't have the model editor built-in. So we have to code our URDF file.

Plans for next week

- Finish the RC car model over the weekend. Make it move in Gazebo. Add sensors to it.

Winter Week 4

Worked On

- Modified the AutoRally car model.
- Added a LiDAR unit and a camera unit to the model.
- Retrieved data from the two sensors and display results in real time in Gazebo successfully in simulation.
- Migrated part of the AutoRally codes to our own package.
- Tried to figure out the dependencies between nodes/packages within the AutoRally project so that we can reuse some of them.
- Working on a documentation as well.

Problems Encountered

- AutoRally is a huge project and there is a lot of physics involved. It might take a little bit longer than expected to go through all the nodes/package.
- The LiDAR unit worked flawlessly in Gazebo. However, the point cloud produced by the camera unit was rotated 90 degrees up. Worked on it for one day, trying to rotate it so that it actually reflects the reality. No luck.

Plans for next week

- Keep on working on migrating codes. Plan to finish it by the end of this week. And then we can add our own stuff to it.
- Try to get the car moving in the simulation.

Winter Week 5

No entry

Winter Week 6

No entry.

Winter Week 7

No entry.

Winter Week 8

No entry.

Winter Week 9

No entry.

Winter Week 10

No entry.

Winter Week 11

No entry.

Note:

None indicates no blog posts were made for those weeks.

No entry means blogs and notes were made but did not get posted and will appear on later blog posts.

5.8 Spring

Spring Week 1

Worked On

- In the last two weeks of winter term I didn't do the whole lot mainly because of finals. However, I managed to get the car to follow paths. It turned the path planning algorithm didn't generate paths for the car to follow. Instead, it generates commands to drive to car. That made things very easy. All I did was translate the commands for the controller. So that's done. I have decided to use the teb_local_planner as our path planning package. I integrated it into the smart_driving package and set it to work on car-like robot. According to the ros wiki for the car-like setup, the configuration is only experimental. Just something to keep in mind. It may sometimes output unpredictable behaviors. Since the teb_local_planner also does obstacle avoidance, obstacle avoidance was handled.

Problems Encountered

- The steering commands oscillate a lot. The oscillation makes the movements of the car unnatural. We need to look deeper into the code to make sure the steering is smooth.
- Obstacle avoidance requires a lot of computing power. It requires so much that it's undoable on this laptop. The algorithm does obstacle avoidance by computing the costmap each time it sees a new objects. (detected by laser scans) A costmap is basically a 2D grid that describes the environment and helps to calculate the most efficient path. On this computer, a new costmap takes on average 3 seconds to compute. When there's no objects, the algorithm works fine on this computer. Note that the costmap is only updated when objects are detected. Not sure if the NUC can handle it. Need to find out.
- The software can be simplified. When I have been doing is just adding new nodes. I didn't consider if the nodes are actually necessary. It's because I didn't want to spend time on going over the AutoRally packages. For example, when I need to translate some data, I just make a node to do the task. In fact, I could have just remap the message

name in the launch file. That will reduce a lot of overhead. It will possibly make the software run faster. I will make a table to figure out the data flow and delete all unnecessary nodes.

Plans for next week

- Continue working on the navigation.

Spring Week 2

Worked On

- We decided to move away from AutoRally a little bit because it was too complicated. We tried ROS stage, which was a 2D simulation platform. It's simpler than Gazebo. We quickly got the teb_local_planning working because we already had it working with AutoRally.

Problems Encountered

- I am not sure if stage can simulate gps and IMU. It will only simulate laser scan right now. If we couldn't get the IMU and gps simulated on stage, it is fine. We can do it directly on the car.

Plans for next week

- Get the IMU and gps working.

Spring Week 3

Worked On

- Got the odometry in Stage working.
- Worked on the poster.

Problems Encountered

- Stage doesn't simulate IMU.

Plans for next week

- Keep working on Stage and the poster.

Spring Week 4

Worked On

- Didn't do a whole lot because I was at a point where if I couldn't get the IMU data I couldn't move forward in integrating the navigation stack. Dan and I were working on it. But we didn't make much progress. I had a lot homework to do. So I left it to Dan. Looks like he hasn't gotten it working yet.

Problems Encountered

- The IMU driver provided by ROS outputs IMU data. However, the state estimator receives a constant pointer to the data. We need to find a way to transmit the data.

Plans for next week

- Keep working on the IMU. If that's done. Move on to the GPS.

Spring Week 5

Worked On

- Worked on the GPS. The data from the GPS was not reliable.
- Looked for other options for state estimation, because we couldn't the way around the constant pointer.
- Found one package that seemed promising.

Problems Encountered

- The AutoRally state estimator was a perfect package to use. But we didn't have the right hardware. The new package we found was intuitive but had a lot of parameters that needed to set up.
- We are still not able to set it up.

Plans for next week

- Keep trying the new package.
- Start preparing for Expo (demo).

Spring Week 6

Worked On

- Midterm progress report.
- IMU and GPS.
- Code Freeze. Documentation.
- simulation.

Problems Encountered

- Our software is very environment-dependent, meaning that it might not be able to run on another computer if the environment was not set up right.
- GPS data was still not reliable. Can't just work with the IMU.
- We might need wheel encoders.

Plans for next week

- Keep preparing for the expo.
- Keep experimenting the new package.

Spring Week 7

Worked On

- Prepared the system for expo demo.
- Expo pitch.
- Expo.

Problems Encountered

- A lot of software was installed on the laptop.
- Wasn't able to run the Leddar during Expo, which was kind of embarrassing.

Plans for next week

- Relax.
- Relax more.
- Relax even more.
- Relax a little bit more.
- Then start working on the final write up.

Spring Week 8

If you were to redo the project from Fall term, what would you tell yourself?

- I would tell myself that my team would win the first prize at the end of the year.
- I would tell myself to get as many sensors as possible.
- I would tell myself this project was one of the coolest projects so do your best.

What's the biggest skill you've learned?

- Using ROS.
- Communication.

What skills do you see yourself using in the future?

- Since I will be doing robotics, I think knowing how to use ROS will greatly benefit me.
- I want to do research or development, which means I will most likely be working in a team. Good communication skill will also benefit me lot.

What did you like about the project, and what did you not?

- I like everything about this project.
- I don't like the fact that we were given so many writing assignments.

What did you learn from your teammates?

- I think my understanding of sarcasm improves a little bit.
- Most importantly, it's collaboration. I was working in a team with more than 15 people. I didn't really learn a lot from that team. A three person team is a good starting point. We can distribute the work load evenly and still have a complete picture of the project.

If you were the client for this project, would you be satisfied with the work done?

- I will be satisfied but a little disappointed because the car can't drive itself.

If your project were to be continued next year, what do you think needs to be working on?

- To make the project a little easier, I think we need to get another car first, or make one ourselves. We didn't have much information about the car we have right now. and that car is too much for our purposes. I want the wheels to be completely rigid or just with thin tires. I don't want the car to have less complicated suspensions and drive train.
- With a new customized car, we can add wheel encoders as well as arrange the hardware in a more elegant way. With the rigid wheels and the new suspension and drive train, we can gain more accuracy to the location estimate.
- If we have more time, we could possibly customize our own self driving package.

Speak a little about your expo experience.

- It was very hot. And I didn't get enough free soda.
- We got a big table, which was nice.
- We were at a good location too.
- People seemed to enjoy our project and presentation.

5.9 Dan's Blog Entries

5.10 Fall

Fall Week 4

Worked On

- Created the problem statement tex template.
- The problem statement proposed solution section.

Problems Encountered

- Not getting feedback on the revised problem statement as quickly as I would have liked.

Plans for next week

- Create LaTeX template for the SRS.
- Research LaTeX Gantt charts.
- Start on SRS.

- Start on filling out a Gantt chart.

Fall Week 5

Worked On

- Created an SRS tex template.
- Did not use IEEEtran.cls because it does not format the SRS properly.
- I found a template online that has the proper formatting and modified it to follow the IEEE 830 documentation.
- https://github.com/Eisenbarth/SRS-Tex
- Created a new 'srs' git branch.
- Has a new 'srs_template' folder with the template tex file, makefile, and supporting resources.
- Worked on the SRS.
- Filled out Software Interfaces and Communications interfaces.
- Found templates for LaTeX Gantt charts.

Problems Encountered

- Trying to write a document that assumes we know what components we will need and will be using without really knowing what those things are.

Plans for next week

- Work on SRS final draft.
- Work on / finish Gantt chart.
- Start thinking about the tech review.

Fall Week 6

Worked On

- Corrected our SRS tex document to use IEEEtran.cls.
- Corrected the heirarchy format to be numeric, instead of the default Roman numeral.
- Gantt chart:
  - Created LaTeX document for rendering a Gantt chart:
  - Added files and folder for Gantt chart to repo
  - Added comments explaining how to set up groups and tasks.
  - Filled out SRS group with sub-tasks.
  - Created system for groups to track sub-task progress.
  - Fixed formatting.
  - Finished basic layout for Gantt chart
  - Integrated gantt_chart.tex into arc_srs_template.tex

Problems Encountered

- Getting the Gantt charts to render in LaTeX. It took around 9 hours, and it's not even close to scale-able.

Plans for next week

- Work on finishing SRS final draft.
- Determine what my part of the tech review is.
- Work on tech review.

Fall Week 7

Worked On

- Tech Review: Researched image analysis software, telemetry radios, and user interfaces for drones.
  - Image analysis software
  - Found DroneKit-Python, ArduPilot, and LibrePilot.
  - Telemetry radios
  - Found 3DR 915 MHz Transceiver, RFD900 Radio Modem, and OpenPilot OPLink Mini Ground and Air Station 433 MHz
  - User interfaces
  - Found QGroundControl, DroneKit-Android, and LibrePilot.
  - Worked on the write up for these components.

Problems Encountered

- The documentation for the drone software is rather vague when it comes to information on path-finding and image analysis capabilities.
- DroneKit documentation was somewhat confusing. At times it seemed like it was its own project, but at other times it referenced ArduPilot, making me think that DroneKit is a Python API wrapper on top of ArduPilot.

Plans for next week

- Finish SRS document.
- Finish Tech Review document.
- Start:
  - Planning the design document.
  - Planning the progress report and presentation.
  - Planning the poster.

Fall Week 8

Worked On

- Completing tech review and additions to the srs document.
- Researched hardware for telemetry radios.

- Researched software for UI to be able to communicate with the vehicle.
- Researched software for image analysis for environment mapping and depth finding based off of sensor data.

Problems Encountered

- Not too much different that other weeks: the document formats require the writer to have concrete examples and plans for where the project is going. Since this is a research project, and has not been done before, our work-flow really requires some experimentation before we can establish what is going to be used.

Plans for next week

- Work on creating a plan that will be used in the Design Document.
- Create the design document.

Fall Week 9

Worked On

- Filled out specific requirements section of the SRS.
- Wrote ”Radio Communications”, ”Image Analysis”, and ”User Interfaces” sections of the the tech review.

Problems Encountered

- Having been delayed on the previous documents, we are about 1 to 1.5 weeks behind schedule. We need to finish up the srs and tech review still. This makes starting on the design document difficult, if not impossible.

Plans for next week

- Write the design document.

Fall Week 10

Worked On

- Finishing the SRS and design documents.
- Wrote up how we will test our implementation of the ARC project. My part of the progress report is to synthesize my comments in these progress posts into a coherent subsection of the report. For the presentation, I provided summaries on power point slides, these included graphics and diagrams.

Problems Encountered

- Our implementation for many components of ARC is currently unknown. More time is required to know exactly what specific software will be used on our project. We need to dive into the APIs used and see how compatible they are and see if we can write our own limited API (meaning simple conversions). So far we have only really had time to look into what open source resources are available at a higher level. This makes writing a detailed design document (in terms of actual code implementation) difficult. So, the design document is being written in terms of

experimental plan. We have milestones that our project needs to meet. After succeeding in the first milestone, we continue to the next.

Plans for next week

- Write up my section of the progress report.
- Record the progress report presentation.

Fall Week 11

Worked On

- Progress Report Created LaTeX template. Wrote the purpose and goals, weekly summaries for weeks 1-6.

Problems Encountered

- It's finals week, enough said.

Plans for next week

- It's Christmas break, but I plan to start looking into implementation, how the APIs will work.

5.11 Winter

Winter Week 1

Worked On

- Setting up the development environment for Linux and ROS.
  - We will be using the Robotics Operating System (ROS) for communication and control of our vehicle. Linux is the operating system of choice for the ROS platform.
  - I worked through orientation tutorials for ROS to become more familiar with the platform.

Problems Encountered

- No real problems encountered in week 1.

Plans for next week

- Get Linux installed on my laptop to dual boot, also get Linux running in VMware for further testing/development capabilities.

Winter Week 2

Worked On

- Installed Linux to dual boot on my laptop.

- Got ROS running in Ubuntu 14.04 and 16.04.
- Got GT Autorally working in 14.04 and 16.04 in both ”native” Ubuntu and in a VM.

Problems Encountered

- My laptop died on Tuesday, so I ordered a new one and got it set up on Thursday/Friday.

Plans for next week

- Start digging into how Autorally uses ROS and Gazebo (a simulator) to see how we can use it in our project.
- Look into direct computer-to-computer communication.

Winter Week 3

Worked On

- Continued with development environment setup.
- More work in ROS.
- Found software library for our IMU.
- Started looking into computer-to-computer direct communication.

Problems Encountered

- The research I have done stated that new computer should be able to do direct ethernet communication without a crossover cable. I was not able to get the computers to communicate. I need to get a crossover cable to see if that is the issue, or if some other setting is incorrect.
- We received a corrupt image for the PXFMini, so we will need a new one sent to us.
- We were hoping we could find a ready-made computer model for our RC car for ROS or Gazebo, we have not found one as yet. This means that we might need to spend quite a bit more time building a model for simulation, if we want to go that route for development.

Plans for next week

- Get computers talking to each other directly over ethernet.
- Get telemetry data from the RC to a remote computer.
- Get command from remote computer to RC car.

Winter Week 4

Worked On

- Got computers talking with each other directly over Ethernet.
- In Ubuntu on both systems:
  - Edit wired connection
  - set ipv4 to manual

  - set the IP address
  - subnet mask, and gateway
  - The networking service may need to be started:
  - sudo service network-manager restart
  - Got Raspberry Pi3 talking with computers over Ethernet as well.
- Worked on tutorials for ROS publishers and subscribers
- Got ROS publisher on computer A talking with ROS subscriber on computer B.
- This test is between my personal laptop and the Raspberry Pi 3.
  It is important that /.bashrc have the following lines:
  For computer running ROS master (this is the Raspberry Pi, and note that IP addresses are the same):
  export export ROS_MASTER_URI=http://192.168.1.2:11311
  export ROS_IP=192.168.1.2
  For client computer (note ROS_IP is local IP address):
  export ROS_MASTER_URI=http://192.168.1.2:11311
  export ROS_IP=192.168.1.3

- Got the NUC talking to the Raspberry Pi 3 in ROS following a publisher/subscriber tutorial:
  http://docs.erlerobotics.com/robot_operating_system/ros/basic_concepts/examples/publisher_and_subscribers
- Able to get internet working over Ethernet. This will allow us to update the Raspberry Pi3, install needed ROS packages and other library dependencies.

Problems Encountered

- The keyboard layout on the Raspberry Pi 3 was set to UK by default.
- Fixed the layout permanently with: sudo raspi-config
- Other settings can be configured for the Pi 3 from the same config menu.
- WiFi is not working on the Raspberry Pi 3, internet is not working, in general.
- Able to get internet working over ethernet.
- The PXFMini did not power on.
  Discovered that the PXFMini cape was connected to the Raspberry Pi 3 incorrectly.
  After connecting correctly, the Rpi3 boots, but the PXFMini LED status lights show a code that tells us that it could not launch. This could be normal, or caused by an error. The documentation says that a vehicle selection should appear when the Rpi3 boots up, but I do not see that.
- Upgraded the Raspberry Pi 3 using sudo apt-get dist-upgrade and then the rpi would not boot any more.
  - It turns out that the upgrade somehow wiped out the required kernel7.img needed for the OS.
  - Copied a kernel7.img from another system image, along with other missing files, the rpi now boots, and WiFi is now working too!
- Discovered that our power supply (USB power from the NUC) is under-powered for the Raspberry Pi 3, especially with the PXFMini attached. We will need to try using the DC power converter and the passthrough to the Rpi. We will need to get a voltage-meter to make sure our power is correct (need 5V, 2.5A).

Plans for next week

- Get proper power to the Raspberry Pi 3.
- Get the PXFMini runnning properly.
- Make servos and motors run via the ROS and the PXFMini.

Winter Week 5

Worked On

- Got the PXFmini launched and armed with instructions for installing binaries for APMrover2.
- Got the Raspberry Pi 3 (RPi3) connected to the internet via ethernet using the original Erle OS image.
  - Steps for connecting to internet using ethernet:
    In /etc/network/interfaces, uncomment the eth0 section.
  - Steps for getting WiFi internet working with home router you will need to edit two files, interfaces and wpa_supplicant.conf:
    - Back up original /etc/network/interfaces file.
    - In /etc/network/interfaces, comment out the lines for wlan0 and add the following lines
      auto wlan0
      iface wlan0 inet dhcp
      netmask 255.255.255.0
      wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
    - Back up original /etc/wpa_supplicant/wpa_supplicant.conf
      Edit /etc/wpa_supplicant/wpa_supplicant.conf:
      Comment out current network block
      Add new network=... block containing the following lines:
      ssid=”¡name-of-network-ssid¿”
      psk=”¡network-password¿”
    - Reboot the Rpi3.
  - I was able to publish to mavros topics and confirmed that the subscribers did get the information. So far it has not caused the actuators to fire.
  - I was also able to confirm that the pxfmini is talking with the rpi, I can see the IMU data being transmitted and changing when I move the unit around.

Problems Encountered

- Continued to struggle with getting internet working using WiFi on the RPi3 and the original Erle image. This is important to get the proper software installed.
- I can’t compile any of the code that is given in the examples because dependencies are missing... I’ve gotten one script to actually run, but it doesn’t do anything for the pxf. and the documentation is so incomplete, it just assumes that ”everything” (whatever that is) is set up and ready to go to launch the given script. And in places where it tells you to do something, it doesn’t explain what is necessary for it to run, or even what the code is doing. When I try to compile a cpp file (ground_rover.cpp) with include¡ros/ros.h¿ the g++ compiler says it can’t find the file, which

means that the ros library wasn't installed into the system include directory. I can compile using an include option and point to the correct directory for the ros include libraries, but then it errors out on a different file not found in one of the included headers of ground_rover.cpp.

- The instructions from erle-robotics in "first steps" show a mavros node 'raspicam_node when running rosnode list, that node is not listed after going through their instructions. If I run rostopic list I get a similar output, but rostopics and rosnodes are two different things.
- Getting WiFi working for the internet breaks launching rostopics. The wifi settings must be set back to original to restore rostopic settings. – Use erle-reset.
- After talking with our client, Kevin McGrath, we have determined that something is probably not set up properly, or strangery of some kind is afoot. The rpi3 and Erle PXFmini should work with the erle-image "out of the box", but it does not on our rpi3, for an as-of-yet unknown reason. Kevin and I are going to try to figure out what is going on.
- Well, I am now able to publish (via command line) to any of the mavros topic subscribers. I have only published to few of them, but I know how to do it now. I am also able to subscribe to topics to see what they are sending. The pxfmini is "working", in the sense that is communicating with the rpi3. I can see the IMU data, which changes when I move the pxfmini. I don't know if the numbers or correct, or even usable, but at least they are numbers...
- Unfortunately, nothing I have tried has gotten the actuator to move. When I arm the pxfmini, the actuator fires erratically until I send a message via mavros/rc/override, then the motor stops, but further messages do not cause the motor to do anything.

  At this point, I feel like I could spend the rest of the term figuring this out. Obviously that is not an option.

  I'm talking with Kevin about this problem. He's also having a hard time getting the pxfmini to work via software.
- Progress is going very slowly due to very poor documentation for the pxfmini and mavros (an api for RC communication).

Plans for next week

- It's week 6 and the midterm report and presentation is due.
- Get our required One Note project environment setup and filled out.
- Make revisions to the documentation for my areas of focus.
- Make my portion of the video presentation.

Winter Week 6

Worked On

- Got the OneNote workbook up.
- Added document and presentation PDFs to OneNote.
- Added revision table templates to OneNote
- Added "template" or empty revision tables to documents (Problem Statement, Tech Review, SRS, Design).
- Made revisions to:
  - tech review

  - design doc
  - Looked at srs and updated gantt chart.
  - Created template for winter midterm report.

Problems Encountered

- None.

Plans for next week

- Install QGroundControl on my system and see if we can control the pxfmini.
- Try to get ”gtest” (a library necessary for one of the erle tutorials) installed.
- make a determination if we want to continue with trying to use the pxfmini autopilot or switch to a simple controller.

Winter Week 7

Worked On

- Got the ”Teleoperating” tutorial installed. The tutorials allows control of the RC vehicle using arrow keys on the keyboard. Kevin McGrath figured out how to install it:
  - mkdir -p /erle_ws/src
  - cd /erle_ws/src
  - git clone https://github.com/erlerobot/gazebo_cpp_examples
  - git clone https://github.com/ros-perception/vision_opencv
  - git clone https://github.com/ros-perception/image_common
  - cd ..
  - cd /usr/lib/arm-linux-gnueabihf/
  - sudo ln -s libboost_python-py34.so libboost_python3.so
  - sudo apt-get install libgtest-dev
  - catkin_make –pkg ros_erle_cpp_teleoperation_erle_rover
- The package now installs, and the keyboard changes the input to the rostopic /mavros/rc/override, but the motor does not activate.
- Able to set the parameter SYSID_MYGCS, this is needed to override RC in and enable computer control of the autopilot.
  - To set using $ rosrun mavros mavparam set SYSID_MYGCS 1, the autopilot must be launched and armed.
- Able to use /mavros/rc/override to publish data to /mavros/rc/in. This is very important because it is necessary for controlling motors. The motors are still not moving, however.
- Installed GCS software (ArduPilot) and connected telemetry radios. Now able to see attitude of the autopilot in a gimble. But nothing else is connecting, not able to control motors at all. Cannot disarm/arm the autopilot from GCS.

Problems Encountered

- Using the above ”Teleoperation” tutorial, I am able to change the channel values for steering and throttle to /mavros/rc/in. I can see these values change on the GCS via the telemetry radios. This means that data is being sent over mavros and the telemetry radios are communicating properly. So, the problem is somewhere between /mavros/rc/in and the PXFmini. At this point we have spent almost 4 weeks trying to get the PXFmini to work. This was supposed to work ”out of the box”. We will move on to a different autopilot.

Plans for next week

- Work out the kinks while installing AutoRally.

Winter Week 8

Worked On

- Went through uninstalling and reinstalling Autorally.
- The autorally softare package has a few different parts to it and odd/obscure errors can occur during installation. My goal was to find errors that arise and document how to fix/correct them so that users will have a better installation experience and get the system up and running faster.
- Found the following error:

```
CMake Error at autorally/autorally\_control/CMakeLists.txt:19 (find_package): By not providing

Could not find a package configuration file provided by "Eigen3" with any of the following name

Eigen3Config.cmake
eigen3-config.cmake

Add the installation prefix of "Eigen3" to CMAKE_PREFIX_PATH or set "Eigen3_DIR" to a directory
```

- This was resolved by changing the calling CMakeLists.txt file:
  Original: find_package(Eigen3 REQUIRED)
  Changed to:
  find_package(PkgConfig)
  pkg_search_module(Eigen3 REQUIRED eigen3)

- I was able to reproduce the above error and verify the above fix worked consistently. I have not been able to test this on other systems.
- Worked through the uninstall process and identified where files were install to, this mostly had to do with GTSAM installation. Autorally itself is fairly localized and doesn’t install any libraries to /usr/..
- I have a fairly solid grasp of the uninstall/reinstall process for autorally, which means that as we move forward with integrating our own software we will be able to provide a more smooth experience for other users through documentation, and perhaps automation of the installation process.

- Still not sure if the Eigen3 error is something that will occur on a first-time install. I don't recall seeing the error before.

Problems Encountered

- It took a few days to figure out what all needed to be cleaned out of the system to be able to reinstall the software. Then it took another day or two to figure out how to resolve installation errors that we hadn't seen before.

Plans for next week

-
- Get autorally installed with the new files Tao is working on.
- Get Telemetry radios working without the pxfmini...

Winter Week 9

Worked On

- Integrating Tao's implementation in to the installation process.
- Able to launch rviz using a custom configuration file. This allows us to load a specific configuration on different computers which will eliminate the need for tedious setup of the rviz environment.

Problems Encountered

- None this week.

Plans for next week

- Continue with building our installation process.

Winter Week 10

Worked On

- None

Problems Encountered

- My wife had a pregnancy-related emergency and our son was born via emergency c-section.

Plans for next week

- None.

Winter Week 11

Worked On

- None.

Problems Encountered

- I was in the NICU in Eugene.

Plans for next week

- None – Spring Break

Spring Week 1

Worked On

- Talked through the complexity of AutoRally with Tao and Cierra. We determined that AutoRally is too complex to try to integrate into our project in the time-frame we have left. We decided to drop AutoRally and focus solely on ROS for the navigation stack.

Problems Encountered

- The physical USB port on one of your telemetry radios broke. We cannot do radio communication until it is fixed.

Plans for next week

- Get the IMU, LiDAR, Leddar, and GPS drivers installed on the NUC.

Spring Week 2

Worked On

- Installed the IMU and LiDAR drivers on the NUC.
- Got rostopics for the IMU and LiDAR publishing data and able to visualize the data in RViz.
- Worked with Tao on figuring out how to get another simulation (Stage) working with our hardware packages.

Problems Encountered

- The Leddar unit is not being recognized in Windows or Linux via USB. It was working earlier, but isn't now. We will try the RS485 interface to see if the Leddar unit is bad, or just the USB interface.

Plans for next week

- Integrate IMU and GPS.
- Get simulation stable using GPS to smooth out the IMU calculations.
- Refine hardware mounting on the car

Spring Week 3

Worked On

- Edited the outcomes and findings for the poster.
- Continued to try to get IMU and GPS data converted into something the AutoRally state estimator will accept.

Problems Encountered

- Still having no luck converting the data we need.

Plans for next week

- Help Tao with the simulations and Cierra with the hardware mounting.

Spring Week 4

Worked On

- Attempting to convert our ROS-published IMU data to something that IMUGPSestimator can use. IMUGPSestimator is from GT AutoRally, we need it to integrate our GPS and IMU data to determine the position of the car in the world.

Problems Encountered

- We need the IMU and gps data to accurately track the position of our car in the world. Integrating that data is not trivial. This is absolutely required to do any autonomous navigation in the real world. It turns out that this very problem is what was really helpful in having an autopilot. The autopilot does all this integration for you. Without the autopilot we have to come up with an integration and interpretation solution on our own.
- We do not have time to develop our own software to integrate GPS and IMU data to accurately track the position of our car in the world.
  - We are trying to use the estimator from GT AutoRally to integrate GPS and IMU data. However, IMUGPSestimator subscribes to a different data type (IMUConstPtr) than the data type published by by our IMU (IMU data type).
  - We are trying to resolve this difference by converting IMU to IMUConstPtr. This has proven very challenging. There is no documentation on what IMUConstPtr actually is. The definition shows us that it's super type is shared_ptr from the Boost library.
  - shared_ptr seems to be a generic type used to track pointer usage and garbage collect automatically when the pointer is no longer used. It does not seem to be relevant to our need to convert from IMU to IMUConstPtr.
- So far, we have not been able to convert the data.

Plans for next week

- Do as much as we can to get our software running the car.
- Get the poster more-or-less finalized and approved by our client.
- Get all the code we have been working on/with uploaded to our Github repo.
- Submit our poster for print before May 1.

Spring Week 5

Worked On

- Continued researching how to convert our IMU data into the constant pointer data structure that AutoRally wants.
- Reviewed the poster put together by Cierra and Tao while I've been caring for my wife and newborn son.
- Worked with Tao on finding different simulations to possibly use during Expo.

Problems Encountered

- Adjusting to a newborn baby is rough!
- Getting accurate state estimation for the vehicle is proving problematic.
- We found out that we misunderstood poster submission requirements. We did not submit the printing job to Printing Media Services within the allotted time-frame for Engineering Expo posters.

Plans for next week

- Continue to hack away at getting sensor data integrated and experiment with our navigation stack in simulation.
- Work on our mid-term submission requirements.
  - Written progress report
  - Progress report video
  - Expo pitch
  - Final project write up rough draft.
- Practice project presentation for the advisory board.

Spring Week 6

Worked On

- Worked with Tao testing a simulation package called Stage. It is very light-weight and, used with RVIZ, it allows use to control and environment and display our car much more simply than Gazebo allows. We will likely use this combination, Stage with RVIZ, to present at Expo.
- Wrote my section of the mid-term progress report.
- Tao, Cierra, and I recorded the mid-term progress video.
- Wrote up my section of the write-up rough draft, which was very similar to the progress report.
- We worked on our Expo pitch and project presentation. I am doing the intro, Tao is talking about what we did, and Cierra is wrapping things up in the conclusion.

Problems Encountered

- We are still struggling with getting the data from the navigation stack to be usable on actual hardware. The main problem is that the simulation is using "perfect" data for the sensors, in real life the sensor give somewhat inaccurate data. This can be countered by adding redundant sensors to increase noise, which smooths out the data, and/or

add an odometry encoder to the car to track distance traveled which, added to the data from the sensors, will give the software a more accurate estimation of the state of the car.

Plans for next week

- Last minute work on expo presentation.
- Build stand for the car.
- Give project presentation to the advisory board.
- Do Expo.

Spring Week 7

Worked On

- Built stand for the car.
  - Designed base and legs to raise and support the vehicle off the ground.
  - Painted the stand with a gloss finish coat.
- Gave project presentation to the advisory board.
- Presented our project at Expo.

Problems Encountered

- No major problems encountered this week.

Plans for next week

- Continue to try to implement functionality for hardware.
- Build out some of the interfaces to allow for future projects to implement the hardware.
- Continue working on the instructions/documentation so that future projects will be able to get to the point we are at very quickly and go from there.
- Work on the Project Write Up, add more detail and refine the findings and conclusion sections.

Spring Week 8

If you were to redo the project from Fall term, what would you tell yourself?

- Identify critical points of failure. Things that could possibly set us back significantly if they were to fail.
- Test those points early, if possible.
- Be more proactive about getting help for problems. Don't struggle with a problem for more than a few days before getting help.

What's the biggest skill you've learned? I learned about the Robotics Operating System (ROS), a general purpose, open source platform for robotics communication and operation. ROS provides a wide range of libraries for communication and control of many kinds of robots, from mechanical arms to rovers, like the car we worked on during Capstone. This project was complicated enough and covered a long enough period of time that I was able to research ROS in depth.

I went through beginner tutorials to understand the basics of how ROS works and then applied some of the concepts from the tutorials to our specific needs in our project. I was able to go through a fairly complete learning cycle to become comfortable with how ROS works. In addition, I had no prior experience with autonomous vehicles, this project gave me exposure to the world of autonomy and gave me a greater appreciation and understanding of what is involved for vehicles to operate autonomously.

What skills do you see yourself using in the future?

- Computer Science is about solving problems (or trying to...), this project required us to answer the question, "Is high performance autonomous navigation possible on inexpensive hardware?" We had to identify the problem we were trying to solve, research possible techniques/approaches for solving the problem, test our solutions, make adjustments and try again, communicate with a client and conform to requirements, work as a team for a "long" period of time, then arrive at a conclusion and present our findings and conclusion.
- These experiences help develop/refine a wide range of skills, both technical and relational, that apply directly to the CS field and life in general.

What did you like about the project, and what did you not?

- I liked my client and my team. Cierra and Tao were great teammates and always positive, eager to work on the project. Kevin was a great client, giving us pointers and keeping expectations within reason.
- I liked that we had several terms to work on the project, this allowed us to do something fairly significant and complex, versus only having one term as would be the case with a normal course.
- I liked the project itself, learning about autonomous vehicles and the current state of the hobbyist-level in this field was very interesting and it was fun to try to get our car working.
- I was not a fan of having to do so many "Progress Report" videos. They were quite time-consuming and a little frustrating. Each progress report took around a week to put together, that's about 4 weeks taken from Fall and Winter terms. I realize that accountability needs to be maintained, perhaps the written report would suffice? It seemed like our weekly meetings with our TA were like a running version of the video.
- I did not really like the "Wired" article. I get that the point was to practice our expo pitch and interact with others, but grading us on the formatting and style of writing was a bit much.

What did you learn from your teammates? I learned from them how to have fun with the process. I think at the beginning of the project my demeanor was too serious. Their enthusiasm and general positivism helped me lighten up and have more fun (I don't know if they would agree, haha). I learned more about electronics from Cierra and more about software simulations from Tao.

If you were the client for this project, would you be satisfied with the work done? I think I would be satisfied. Our objective was to determine if it's possible to make a high performance autonomous RC car with inexpensive hardware and open source software. We put more than sufficient work to come a informed conclusion on the matter.

If your project were to be continued next year, what do you think needs to be working on? A team would need to take our hardware and software platform and work on creating a state estimator with the sensors and data we are using. The state estimator is need to keep the car in control during autonomous operation.

Speak a little about your expo experience. Expo was a fun experience for me, overall. We gave a presentation of our project to the industry advisory board and then talked with attendees during the event. It was fun explaining what we did and why we did it. I talked to quite a few kids and it was really fun seeing them interested in the project and answering their questions.

6 PROJECT POSTER

ARC - AUTONOMOUS RC

Autonomous Obstacle Avoidance and Navigation

Project Introduction and Background

Introduction

Autonomous RC (or ARC) is a research project aimed at seeing if autonomous navigation, obstacle avoidance, and parallel parking can be achieved, at a 1/10th scale. A significant goal is to use commodity hardware that is obtainable through major retailers (online as well as brick and mortar), for a reasonable cost.

Our implementation follows the Robot Operating System (ROS) standards. ROS is a widely-used platform for robotics applications that provides consistent protocols and guidelines for data communication, which makes integration of multiple modules trivial.

Our implementation enables our the RC car to go from point A to point B autonomously while avoiding obstacles. The RC car is also able to parallel park if the surrounding environment is suitable.

ARC is designed to be open-source. With slight modifications, it should be able to run on different types of car-like vehicles. Students and hobbyists are welcome to contribute to the project by improving upon the software.

RC Car V2

Project Background and Motivation

Millions, if not billions of dollars are invested in autonomous vehicles each year. Most of these systems are cost prohibitive for the average user, looking to replicate them at a small-scale. Our motivation for the project includes the following:

- Seeing if commodity hardware can be used to help reduce the cost of autonomous systems.
- Provide an affordable test platform for scaled high-speed obstacle avoidance with GPS navigation.
- See if there are solutions that can be scaled up from the RC car, for consumer DIY projects or commercial products.

Project Description and Outcomes

Project Description

Using ROS and RVIS, we were able to simulate our software package before testing on the car. Simulation allowed us to move forward with our software implementation while we also worked on the hardware side of the project. Using the software simulation also allowed us to ensure that our code was working properly, and the car would have the expected behavior.

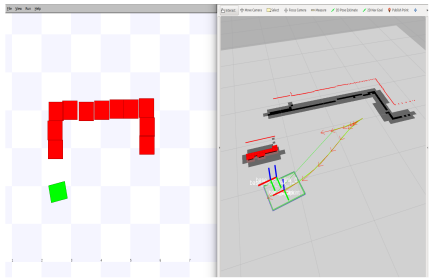
Software features:

- GPS waypoint following
- Obstacle avoidance
- Parallel parking
- Remote control

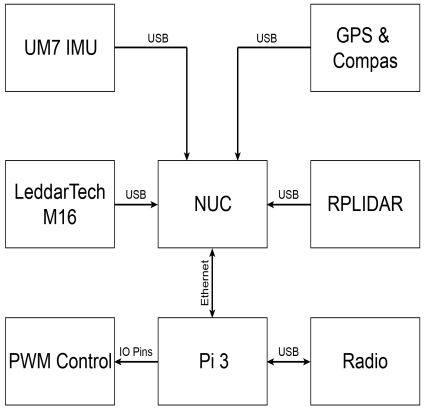
Hardware components:

- Traxxas© Summit RC car
- Intel® NUC Skull Canyon
- Rasberry Pi 3 LeddarTech M16 and RPLiDAR
- Sparkfun Venus GPS with TFDI Serial Breakout
- UM7 Inertial Measurement Unit (IMU)
- Voltage Boosters and Regulators
- 3DR SIK 915mHz Radios

We originally wanted to modify GT AutoRally project and integrate it with ROS to fit our software needs; however, as research progressed, we determined that the complexity of GT AutoRally made modifying to our project too complicated and time-consuming. We changed from integrating GT AutoRally, to strictly using ROS for our navigation stack.



Car Parallel Parking using Path Planning in RVIS



System Hardware Layout

Project Outcomes and Results

With our research and hardware choices, we determine that it is possible to build an autonomous RC with a budget of approximately $2000.

Throughout the entire development process, we focused on using open-source software packages for almost all autonomous features, with the goal of having a combination of ROS packages that would be simple to understand and modify.

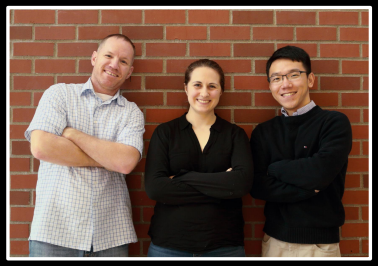
The most challenging part of our project was finding packages that would allow us to have flexible hardware options so that users would not be required to use our hardware configuration.

Another challenge is having the car navigate in sunlight, due to the UV saturation that causes our RPLiDAR unit to give incorrect data. Using the LeddarTech M16 allows us to have forward facing vision in direct sunlight, up to 150'; however, it does not provide a 360º view of surrounding the car needed for functionality such as parallel parking.

Environments that include a lot of glass are very challenging to navigate in (such as Kelly Engineering Center) as the car is unable to receive stable signals from the Leddar and LiDAR sensors.

We also have no implementation for low object detection for items such as curbs and drop-off detection for cliffs. The car can drive over a 4" curb; however, it occasionally will crash if the curb is between 6" and 8".

Computer Science Capstone
Group 44



Group Members

Daniel Stoyer
stoyerd@oregonstate.edu

Cierra Shawe
shawec@oregonstate.edu

Tao Chen
chentao@oregonstate.edu

Client and Affiliation

D. Kevin McGrath
Oregon State University

Conclusion

In conclusion, we implemented basic obstacle avoidance and path planning within our simulation software. Using the the obstacle avoidance, we are also able to parallel park the car by specifying the cars end position. The simulation has also been implemented on the hardware; however, the functionality is still very limited and has room for further improvement in the future.

Our research and documentation, along with our hardware, lay the foundation for others to further improve upon our software platform, by adding functionality, such as autonomous navigation at speed.

Oregon State UNIVERSITY

7 PROJECT DOCUMENTATION

ARC is designed to be modular. We chose the Robotics Operating System as our primary software platform because it has a large number of libraries supporting many different types of sensors. You should be able to swap the sensors we chose (such as GPS, IMU, LiDAR units) with another of your choosing. Consider our parts list a minimum requirement for operation. More sensors (even duplicate sensors) can be added for greater accuracy in data.

7.1 Structure

The structure of ARC has two parts, software and hardware. The primary software platform is ROS Indigo. We also use Gazebo and Stage for simulation. We used AutoRally as a learning tool and provide instructions for how to install and use it. We moved away from AutoRally, but feel it is still a valuable tool to learn about vehicle simulation and ROS. Hardware-wise, we have a main computational unit that handles the heavy lifting for data collection and path-planning, a secondary computer for PWM control or autopilot communication (depending on which you go with), and GPS/IMU sensor integration.

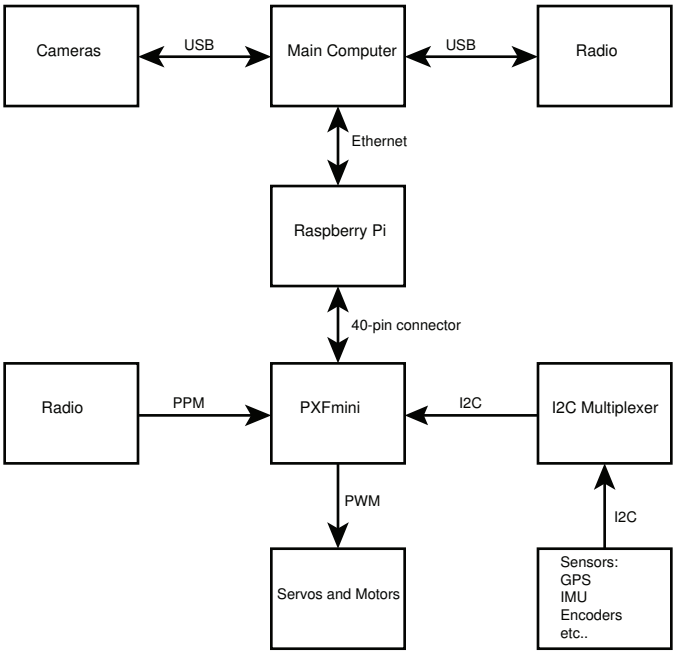


Fig. 1. Main Project Components

7.2 Software Installation

7.2.1 GT AutoRally Instructions

Step-by-step instructions on how to run our implementation of the GT AutoRally simulation platform.

1) Go to https://github.com/AutoRally/autorally and follow the instructions to install the AutoRally platform. Be sure to follow the instructions carefully and completely, failure to do so may result in an incomplete installation

that will require extended time to troubleshoot.

NOTE: It is assumed from this point on that you have already gone through the ”Autonomous Driving in simulation” instructions at GT AutoRally.

The following instructions are intended for after completing the GT AutoRally setup and simulation instructions above.

2) At this point you should be able to run GT AutoRally
3) Keep all simulation configuration according to the AutoRally specifications.
   e.g. stateEstimator’s InvertY and InvertZ values.
4) Navigate to the work space where you put the AutoRally package. And then go to the src/autorally directory.
   NOTE: All files/folders modified and provided by ARC for AutoRally are found in the arc_autorally folder.

5) Under autorally_description/urdf, replace the autoRallyPlatform.urdf.xacro file with file of the same name that we provide.
6) Copy the world folder we provide to autorally_description/models.
7) Under autorally_description, there is a file named empty_sky_AR.world. Copy the code below to the end of the file (before the last ””):
   model://urdf/models/world
8) Under autorally_gazebo/launch, there is a file named autoRallyTrackWorld.launch, comment out the last node (named spawn_track).
9) Copy the autorally_smartdriving folder to src/autorally.
10) Open a terminal, navigate to your work space. The directory that contains devel/, src/ and build/.
11) Use catkin_make to compile the package.
12) Open the bashrc by typing in gedit /.bashrc.
13) At the bottom of the file, add these two lines: source workspace_directory/devel/setup.bash source workspace_directory/src/au
    (Replace the workspace_directory with your actually work space directory)
14) Copy the run_autorally.sh we provide to your work space.
15) Open another terminal, navigate to your work space and run the run_autorally.sh script.
16) run_autorally.sh script automates the initialization processes of the GT AutoRally simulation.
17) A single terminal window opens opens with sub-divided windows using tmux.
18) Applications necessary for AutoRally will open next, including Gazebo and rqt_publisher. These applications along with other processes are launched by the script within different tmux sub-windows in the main terminal window.
19) Find the application window (seperate from the terminal window) titled ”rqt_publisher”.
20) Go to rqt_publisher and make sure the topics created during initial setup are still there:
    /runStop /chassisState /constantSpeedController/speedCommand
21) If rqt_publisher does not have those three topics in the main part of the window, go back to the setup instructions and add them.
22) Make sure /runStop is set to true and speed (under /speedCommand) is set to 3. Then check the boxes to the left of those three topics.
23) Run these two commands in two of the tmux windows:

roslaunch autorally_smartdriving autorally_configuration.launch roslaunch autorally_smartdriving move_base.launch

24) Go back to the rqt_publisher window and uncheck the constantspeedcontroller message.
25) Terminate the waypointfollower and the constantspeedcontroller.
26) Copy the rviz config file we provide.
27) roslaunch autorally_smartdriving auto_turn.launch
28) run rviz with the rviz config file. rosrun rviz rviz
29) Now you can set goals through Rviz and the car should go towards the goal autonomously.

7.2.2 ROS Indigo Installation

If you are not installing AutoRally (which links you to the ROS installation instructions), then you will need to install ROS before doing anything else with ARC.
Go to http://wiki.ros.org/indigo/Installation/Ubuntu and follow the official instructions for installing ROS Indigo.

7.2.3 Installing Sensor Packages

ROS has pre-built packages for the sensors we used in this project. The instructions for installing those packages are below. If you use different sensors, you will need to install packages corresponding to your sensors, or write your own packages if none are available.

- SlamTech RPLiDAR

  - At the terminal command prompt:

  $ sudo apt-get install ros-indigo-rplidar-ros
- LeddarTech Evaluation Kit:

  - Go to the ROS-Leddar GitHub repository (https://github.com/mcgill-robotics/ros-leddar, download (clone) the LeddarTech package and follow the instructions on that page.
- UM7 IMU:

  - At the terminal command prompt:

  $ sudo apt-get install ros-indigo-um7
- Ublox GPS:

  - Clone the Ublox repository at https://github.com/KumarRobotics/ublox and floow the instructions on that page.

7.2.4 Stage Installation Instructions

All files modified by ARC for use in the Stage simulator are found in the arc_stage folder provided.

1) Extract the NewArc compressed file. In it there will be two packages named ”smart driving” and ”stage_launch”.
2) Create a ROS work space, and copy the two packages to the src folder inside the new work space.
3) On the root of the new work space, run ”catkin_make”.
4) On a new terminal, run ”roscore”.
5) On the original terminal, run ”source devel/setup.bash”.

6) On the original terminal, run ”source devel/setup.bash”.
7) Run ”roslaunch stage_launch arc_stage.launch”.
8) A stage simulator and Rviz should show up.
9) The red blocks on the simulator can be moved around.
10) Use the ”2D Nav Goal” on Rviz to tell the car where to go.

7.3 Special Considerations

- Ubuntu 14.04 was used on the primary computer and is required for the version of ROS (Indigo) that we used.
- ARC requires use of ROS Indigo, specifically. Newer versions of ROS might work, but have not been tested.
- If you go with a PXFMini, you will need to get a custom Raspberry Pi 3 image from them.

7.4 User Guides, API documentation, Etc.

- MAVLink API: https://github.com/mavlink
- Traxxas Summit owner’s manual: https://traxxas.com/sites/default/files/110829-R02-5607-manual.pdf
- PXFMini documentation: http://docs.erlerobotics.com/brains/pxfmini

7.5 Parts List

These are the final components we used for our vehicle. Note that all components in this list may be swapped out for different brands/technologies as long as the corresponding ROS packages are modified accordingly.
This list does not include all components we tested during research and development it is only the components on the final version of the RC car.

Description	Price
Autopilot	
Raspberry Pi 3 (RPi3)	$45
PXFmini	$77.73
OR	
PWM Controller	$3.95
Arduino	$39.90
Ublox Neo-M8N GPS with Compass	$8.85
UM7-LT Orientation Sensor	$139.95
3DR radios (2)	$100.00
LeddarTech Leddar Evaluation Kit	$299.99
Intel NUC Skull Canyon With 16GB DDR4 and 480GB SSD	$994.98
SlamTec RPLiDAR	$199.00
Continued on next page.	

Component Sub-Totals	
Components with PXFMini/RPi3	**$1,865**
OR	
Components with PWM/Arduino	**$1,785.63**
Traxxas Summit RC car	$549.95
Totals with RC car	
Using PXFMini/RPi3	**$2,414.46**
OR	
Using PWM/Arduino	**$2,335.58**

8 New Technology Learned

As a whole, every piece of technology we touched within the project was new to us, which was one of the hardest parts of the project.

What web sites were helpful?

- http://wiki.ros.org
- https://github.com/AutoRally/autorally

What, if any, reference books really helped?

We did not use any reference books.

Were there any people on campus that were really helpful?

Our client was Kevin, who was also our instructor. So it was very helpful. He provided us with all the hardware and gave us instructions on how to use them.

9 What We Learned

We experienced how a research project was approximately carried out.

We got to work in a diverse team, which helped us improve our communication skills.

When working in a small team, the relationship between teammates became more intimate, which make us become more considerate people.

We learned a lot on how to use ROS.

ROS was a big part of this project, and we only had little to no experience working with it.

What we were able to accomplish was incredible.

MORE TO ADD.

9.1 Cierra

What technical information did you learn?

Reading technical specifications for a product:
The first time I tried to read a product specification document, I had no idea how to even find information about it. Being able to read these types of documents was incredibly important throughout the project, as I needed to see how were were supposed to wire all of our components together.

Reading a wiring diagram:
This was really important, as many of the things we were given did not have headers or ways of attaching the wires to

our other components. While someone from ECE might think it was a trivial task, it took a little while to understand what the different things meant.

Working with ROS:

Unlike the rest of my group, I did not have quite as much exposure to ROS throughout the project, as I ended up taking a primarily hardware role. I did learn the basics of how it is used through the tutorial and through my teammates. I hope to be able to spend more time learning the details of ROS, as it definitely is a useful tool for robotics, as most things are already implemented for you.

Soldering:

The only thing I had ever soldered in my life before this project was a little light-up pin one day at the fair. While the idea is simple, I learned a lot about how to properly splice wires, how to de-solder (and how you want to avoid this at all costs), and how to ensure soldered components will not easily come loose. I also gained a lot of respect for the ECE students, as it is really hard to make a clean looking board.

Voltage conversion (and how to not fry components):

This was incredibly important, as a lot of our components required odd voltages. In order to convert from 12V to whatever else we needed, I used voltage boosters to be able to use the NUC and leddar at 19V and 24V respectively. I also learned that separating components (such as the PWM board) on to their own power system is important, as we kept dropping the Arduino and Raspberry Pi too low, so they should shut down. With each component we added, I always made sure to read the specifications of what it needed, and then ensured that it was being provided the correct voltage and current with a voltmeter. I also learned just because something says it can take either 12V or 24V, does not mean it can take both, it means that somewhere on the poorly documented board, there could be a jumper that allows you to change from 12-24v. Providing something 12V that is looking for 24V, could lead to it frying the component. This is in reference to our Leddar, which we still do not know why it broke.

How to control servos and motors with a Raspberry Pi and Arduino:

While both the Arduino and Raspberry Pi are able to directly control servos, it was a lot easier to use a motor control board. One thing I noticed, is that the was considerable lag in the system when the Arduino or Raspberry Pi power dropped too low.

Working with LiDAR systems:

Before the project, I had never actually see a point cloud created before. Now I have a much stronger understanding and appreciation for LiDAR devices, as they can do some pretty incredible things. The cool part about using ROS, is all of these implementations became trivial, as it is a built in feature of a ”LaserScan” topic.

What non-technical information did you learn?

I have always struggled with trying to explain my thoughts and ideas, especially through writing. Before this year, writing papers always seemed dreadful. As much as I disliked the WIC inclusion into Capstone, I feel it has improved my ability to create documents and critically think about a problem before starting. It is incredibly satisfying to be able to sit down and write a paper, without it taking a million years. I am now able to break down a problem into more manageable pieces, instead of being overwhelmed by everything at once. Also, ten pages seems like a trivial amount of

work now, which has made all of my other classes a lot easier when we had to write various project reports.

What have you learned about project work?

Planning is essential if you expect a project to go well. I would say as a whole we did a really good job creating good guidelines, but we fell short in estimating and having back up plans for when we found out things just would not work.

Being able to track tasks with some sort of management software would have been nice as well. We had tried to start using an agile method fall term, but it was really difficult considering everyone was also having to juggle their other classes and there were not a lot of times we could sit down and get together to break things into specific tasks. Instead, a lot of the task breakdown was done by hangouts, which often made it difficult to go back and figure out who was responsible for what parts of a paper or the project.

What have you learned about project management?

Of of the biggest things I learned, was always assume things will take more time than what you originally expect. One of the biggest setbacks we had within the project, was not having allocated enough time in the beginning. As I was the one who submitted the majority of our work, I also learned that plenty of time should be left around a deadline to account for potential problems that may occur.

Communication is also key, and I feel like I am able to take a step back and say ”This is what we need to do, now how are we going to get there?” Since at the beginning of the project, we had no experience with anything we were going to be using, stepping back and trying to break the problem down into smaller pieces was key.

What have you learned about working in teams?

If you could do it all over, what would you do differently?

If I could start over, I would have made sure to not get hung up on something for too long. If we had been able to accept that the PXFmini was not going to work for our situation, we would have been able to start implementing things earlier in the term. Finally, I would take any time estimates and quadruple them. Nothing will go as planned, so you need to be able to adapt to whatever is thrown your way.

9.2 Tao

What technical information did you learn?

- A basic structure of a project.
- Using ROS.
- Version control.

What non-technical information did you learn?

- Communicating with teammates.

- Explaining concepts and reporting to client.
- Recording progress.
- For me personally, this project made me like to think about problems with the big picture while focusing on the individual components.

What have you learned about project work?

- Coding could be easy if the project was planed out nicely.
- To make a good plan, one must view the project from different perspectives, and listen to others' opinions.
- Keeping track of progress is extremely important.
- Stick with the plan, and yet quickly switch other solutions when the original does not work out.

What have you learned about project management?

- Lists are useful for keeping track of hardware.
- Version control is important but be careful when using it.

What have you learned about working in teams?

- Good teammates are key. It depends on luck.
- Always try to create a positive atmosphere even things aren't working so well.
- Helping teammates out is not sacrificing. You gain more than you lose.

If you could do it all over, what would you do differently?

- I'd probably prioritize things differently and look for other options when something didn't work out. For example, we could have worked on the software and hardware simultaneously, which I think was our original plan. But then we hit a road block that totally stalled our work on hardware from progressing. Other than that, I don't think I would do much different.

9.3 Dan

What technical information did you learn?

- How to develop for robots:
  The Robotics Operating System (ROS) is an open source general-purpose software platform for developers working with all manner of robots. ROS allows the developer to use its extensive libraries to implement custom code for controlling robots and provides communications and navigation libraries that are very useful for autonomous vehicle applications. We wrote some programs in C and python using Ros libraries to communicate with the vehicle.
- How to set up simulators:
  I used both Gazebo and Stage are simulation environments that we used on this project. I learned how to set up and run both.

- RC cars and Autonomy

  I had never worked with RC cars, or tried any sort of robotics, so diving into the field of autonomous vehicles really stretched me. There is *so much* that goes into research and development of autonomous vehicles. The computer has to either know the exact state of the vehicle via sensor data, or be able to estimate the vehicle's state closely enough to operate.
- CMake

  CMake is a platform for build and testing software. It is open source. ROS uses CMake as part of its package installation platform, called Catkin.
- I learned more about bash and Linux, creating a script to launch AutoRally, spawning multiple terminals and processes.
- I learned about LiDAR and point-clouds. I had never seen LiDAR in action before. I learned that there are different types of LiDAR. We used IR and some sort of LED technology.

What non-technical information did you learn?

- I learned that ”autonomous” has many different meanings when related to vehicles. My assumption was that it meant that cars could navigate pretty much anywhere on their own. I found out that a car can be considered ”autonomous” if it performs *any* operation on its own. For instance, Georgia Tech's AutoRally project claims their car is ”autonomous” and the car does go around a track without a user controlling the car directly. However, the car does guide itself, it follows predetermined way-points and performs no obstacle avoidance what-so-ever. The barriers around the track were in place to merely keep the car from going completely off the rails should something go wrong. The car did self-correct steering around the turns and kept the car under control will power-sliding around corners. So, the car was autonomous, just not in the way that I had though.

What have you learned about project work?

- Plan for setbacks and do not be disappointed when they happen.
- It's really hard to work on a complicated project part time while taking other classes.
- Ten hours of trial and error will save an hour of good design planning.

What have you learned about project management?

- It would be very helpful to use project management software.

  My team did *not* use any sort of project management software for this project, by ”project management software” I mean packages such as MS Project, or Redmine. Project tracking using manually implemented Gantt charts was incredibly tedious and not really used as a practical tool.
- Milestones need to be clearly identified and consistently communicated throughout the life of the project.

What have you learned about working in teams?

- I was the team leader. It was difficult to balance keeping the team focused on our goal and not squashing enthusiasm when attempting to redirect tangent ideas back on track.

- Good communication is key to keeping a team operating smoothly.
- When everyone pitches in and they are enthusiastic about the project, the work is quite fun.

If you could do it all over, what would you do differently?

- Identify critical points of failure. Things that could possibly set us back significantly if they were to fail.

  Test those points early, if possible.
- Be more proactive about getting help for problems. Don't struggle with a problem for more than a few days before getting help.
- Start with a rover that has an autopilot integrated into it, possibly even already has some sensors on it.

  Our team had *no* experience going into this project. Trying to start from scratch was probably a bit too much, if the goal was to implement high-speed performance. It would have been more realistic to attain if we had started with a vehicle that had some sort of autonomous capability already and then we "upgrade" it to perform at a high rate of speed.
- I would use project management software.
- I would set an expectation of a required weekly meeting time at the beginning of the project. Our team agreed to do that, but I did not really follow through on it well.

10 ARC Findings and Conclusions

ARC Findings and Conclusions

CS Capstone - CS 463

Oregon State University

Spring 2017

Cierra Shawe, Tao Chen, Daniel Stoyer

June 10, 2017

Abstract

Group 44's *ARC Findings and Conclusions* provides an overview of the Autonomous RC (ARC) project, the methods used to conduct research and implement software, findings of the research, and a synthesis of the findings. ARC is designed to used commodity hardware and uses the Robot Operating System. It was determined that with more time and resources, it is possible to create an autonomous RC car.

CONTENTS

1 Introduction

The purpose of ARC is to determine if it is possible to create a small-scale autonomous hardware and software implementation that is able to be retrofitted onto a Remote Controlled (RC) car. The goal of ARC is to have a vehicle capable of navigating from point A to point B while implementing obstacle avoidance. The vehicle should also be able to parallel park itself when told where the final location of the vehicle should be. ARC is also designed to cost under $4,000 to make the platform available for education, hobby, and commercial purposes. The components within the projects are also designed to require minimal fabrication and electrical experience. Obstacle avoidance and the low cost are what differentiate ARC from other autonomous vehicle projects, as implementations such as Georgia Tech's AutoRally platform cost upwards of $40,000 to build. This makes ARC an order of magnitude cheaper than something such as the AutoRally platform. By providing access to a low cost autonomous platform, ARC would provide a way to test different autonomous features without having to use a full scale vehicle, lowing development costs and making autonomous vehicles more accessible to a wider range of users. The ARC software platform uses the Robot Operating System (ROS) in order to provide an easy interface for users to contribute to the project. ROS is a standard within robotics and it provides users the ability to customize the packages being used with minimal coding experience.

2 Methods

2.1 Trial and Error

We used the trial-and-error method to develop our software, because we had technically unlimited options. We could write our own software from scratch or there was a limited number of packages that were available from the internet. And we did both. We explored multiple open-source software, integrated with our own code. For instance, to develop the navigation functionality, we tried building another layer on top of the AutoRally project that will generate new waypoints to the "WaypointFollower". We did not choose to go this route because it was difficult for us to integrate obstacle avoidance to the layer, which would require a lot more time and resources. On the other hand, we discovered an open-source software that would do both navigation and obstacle avoidance, and issue commands to the control modules directly. However, we did not have too many options for some of the parts. This brings us to adaptation.

3 Adaptation

In the real world, engineers are required to work around the hardware that they were given. After an unsuccessful attempt to use an autopilot to handle the motor control and sensors, we were required to implement the functionality of the autopilot using using the GPS, IMU, Leddar, and Lidar that were provided. All of the components had ROS libraries that we were able to utilize, which made them easier to use; however, the GPS was not as accurate as we had hoped.

Custom component housings needed to be designed for all the hardware for items such as the GPS, IMU, Raspberry Pi, power converters, and motor control board.

ROS was a well-known platform for building robots. We did not have other options but to use it. Luckily, it worked very well; however, the learning curve was very steep at first. So it took us some time to learn how to use it.

3.1 Iterative/Modular Development

Using our requirements as a guide, we focused on getting every component to work individually before moving onto the next requirement. This allowed us to ensure that everything was able to work individually before attempting to integrate everything together.

4 FINDINGS

4.1 Image Analysis

4.1.1 Fast Image Processing

We required image processing to be at least 15 frames per second (fps) for the vehicle to move at the speeds we desired. We found that image processing using stereo vision was too computationally expensive to run on our hardware. Much more expensive, super-computer-level hardware would be required to make image processing of stereo vision attain 15 or more fps.

4.1.2 Depth Finding

Our requirements to detect objects at least 6 feet away and display the objects in a GUI was met using a combination of a 360 degree Lidar unit and a front-facing Leddar unit. The LeddarTech M16 unit detects objects up to 50 meters in front of the car with a precision of 5cm at 50m. The RPLiDAR unit detects objects at 360 degrees up to 6 meters away with a precision of 1cm at 6m. Objects are displayed with the RVIZ visualization package.

4.1.3 Object Height Detection

We required our sensors to detect objects of 8 inches or more in height, from ground level. We have achieved that requirement using the front-facing LeddarTech M-16 which can detect a four inch object from six feet away from its position on the car. This exceeded our required object detection height of 12 inches.

4.2 Sensors

The main reason for needing GPS and IMU data is to determine the car's position in the world, termed localization. Localization can be achieved by using a combination of odometry, GPS, and IMU data, it can also be achieved with odometry. We wanted to omit odometry because retro-fitting a RC car with an odometry encoder is time-consuming and very technically/mechanically difficult. We discovered that achieving localization accurate enough for autonomous navigation requires GPS and IMU hardware much more precise than that available to us. Moreover, such hardware is quite expensive and would increase our costs substantially. Therefore, we see a needed trade off: localization can be achieved using a combination of lower-precision GPS and IMU hardware with an odometry encoder retro-fitted onto the RC vehicle. The advantage of this configuration is that it is rather inexpensive. The disadvantage is that retro-fitting the encoder is very time-consuming and requires mechanical expertise. On the other hand,localization can be achieved using only GPS and IMU hardware. The advantage is that you avoid retro-fitting an odometry encoder, the disadvantage is that such hardware is very expensive.

4.2.1 GPS Data Collection and Processing

Our requirements specified GPS accuracy to 10 meters outdoors in an open space with no trees within 50 meters. Our GPS unit meets the requirements, however we found that localization based off of GPS data requires much more accurate (and more expensive) equipment.

4.2.2 IMU Data Collection and Processing

IMU data was required to be sent to the main computer at a rate of at least 20 measurements per second and visualized a some sort of GUI. Our research revealed that it is standard to transmit such data at rates around 10 Hz, or 10 packets a second. We are able to send IMU data using ROS at 10 Hz, which is sufficient for our application. We have IMU data visualized through RVIZ and CLI.

4.3 Navigation

4.3.1 Motor Control

We required at least 75% accuracy when going forward and backward. We learned that odometer encoders were necessary to determine the accuracy of the motor control. Since we do not have odometer encoders, we are not able to determine motor control accuracy. We are able to visually confirm forward and backward operation of the motors.

4.3.2 Servo Control

We required a 25% margin of error with respect to the commands sent by the main computer at any instance. We are able to control servos on command, however we found that determining the exact margin of error of physical performance with respect to commands given requires more research with the physical hardware.

4.3.3 Probabilistic Analysis for Motion

We required that our filtered instance location estimation be of higher accuracy than our unfiltered instance location. We were unable to determine instance location accuracy on our physical car. Further research, implementing software state estimation on the hardware, will be needed to determine this requirement.

4.3.4 Motion Model

The commands sent by the computer were required to adapt to the vehicle's configurations. We were to confirm the correctness of the motion model on the vehicle through observation. As our vehicle is not currently in motion, we are unable to determine whether the motion model is working correctly for the physical car. The motion model does work within the simulation with our general car object.

4.3.5 Global Path Planning

We required a point-to-point global path via setting way-points. Our initial concept was to set multiple way-points, but it became clear that our global path needed to be a straight line, local path planning would handle obstacle avoidance and bring the car back on line with the global path. We currently have this functionality working in simulation only.

4.3.6 Local Path Planning

Our requirements for path planning were to generate a path of way-points that best fit avoiding local obstacles bringing the car back to the global path. We were able to achieve local path planning in simulation. Further time is needed to implement local path planning on the physical platform.

4.3.7 Obstacle avoidance

The requirements for obstacle avoidance were to avoid all obstacles the image analysis system could detect. Within the simulation, the car is able to avoid all obstacles, unless it senses that there is not enough room to reasonably get itself unstuck, in which case it will remain in the same position. The obstacle avoidance running in simulation should apply to the car running in real life, as the simulation accounts for the use of our LiDAR and Leddar sensors.

4.3.8 Parallel Parking

We required the vehicle to parallel park itself when given a open spot. We achieved this requirement in simulation being able to have the car park into a spot that is 2.5 times it's length. This is done by selecting the point, and direction, in which the user would like the car to end up. The smaller the space, the longer it will take to be in the proper position.

4.4 Hardware Mounting

Hardware mounting design is complete with CAD models. The mounts on the vehicle are currently in the prototype phase, we are waiting on facilities/resources to make 3D prints of the mounting models.

4.4.1 Minimize Port and Hardware Exposure

All ports and sensitive hardware needed to be isolated from the elements. We have a prototype body constructed and have all ports and points of exposure sealed with electrical and duct tape.

4.4.2 Secure Hardware Mounting

Hardware was required to be mounted securely in the event of collision or rollover. We currently have hardware mounted with zip ties and screws. As our mounting system is still in the prototype stage, it cannot withstand collision or rollover. We do have mounting hardware design in CAD software, but did not have time to have them 3D printed.

4.5 Communications

Radios were required for transmitting telemetry and basic commands from and to the vehicle. During research/testing we did have successful radio communication. However, once we abandoned the PXFMini autopilot we lost the API libraries that enabled our radios to function. More time and research is needed to build a software framework for sending and receiving radio data, presumably using MAVRos, a MavLink library for ROS. Further research is needed to determine how to implement an interface with the radios.

4.5.1 Telemetry, Vehicle Control, Emergency Stop

We required telemetry to be radioed from the car to a ground station. This was initially accomplished but is currently unmet, as explained above. Logically, vehicle control and emergency stop are not enabled.

4.6 User Interface

4.6.1 Graphical User Interface (GUI)

We required a GUI to see the current state of the vehicle and the sensors perceptions of its surroundings. The GUI was to also allow us to interact with the car and issue commands, such as way-point destinations. We are using RVIZ as our GUI, it shows us the state of the vehicle, the sensor data of the surroundings, and allows us to set way-point destinations.

4.6.2 Command Line Interface

A CLI was required to be able to enter way-points, and start and stop the vehicle. We did not meet this requirement. Our research into ROS tells us that such functionality can be written in either Python or C++, but more research and time is needed to implement the interface.

4.6.3 Way-Point Selection

We required way-point selection via either CLI, GUI, or both. We achieved way-point selection via GUI and the RVIZ software package. Vehicle Information We required a GUI to see all information regarding the vehicle. We are using RVIZ to accomplish this requirement.

5 Performance Requirements

5.1 Support of one user and setting 5 way-points

This is fulfilled using RVIZ, only one way-point is set, which meets the requirement of the global path. Capable of driving 10 mph max in straight line. We do not have tested information on this requirement. Our estimates from simulation suggest that the vehicle will be able to achieve 10 mph when travelling straight.

5.2 Capable of navigating indoor environment of 500 sqft.

It is undetermined whether the physical car will be able to navigate a room. From tests conducted on the vision system, the walls must not be made of glass.

5.3 Capable of navigating outdoor environment without space constraints.

The system is able to operate without constraints within our simulation.

5.4 Capable of parallel parking in under a minute.

Within our simulation, the car is able to parallel park within a space that it at least 2.5 times its length, in under a minute.

6 Design Constraints

6.1 Radio Communication within 2 km.

We were not able to test the radios, however the radio manufacturer (3DM) claims the radio range is up to 15km.

6.2 Autonomous navigation at 3 mph.

Our research through simulation suggests that this is very achievable. Further research is needed to implement the navigation stack on the physical platform.

6.3 Hardware Mounting

We needed to have mounting surfaces fixed to a pre-existing RC vehicle chassis. We have prototype mounting surfaces fixed to the vehicle. We have designed mounting parts using CAD models which need only be printed out with a 3D printer.

7 Conclusion

After close to 7 months of hard work, we can conclude that it is feasible to use only commodity hardware to build a platform that can autonomously drive a RC vehicle. As a research project, we hate to say that we have come to our conclusion without a working prototype; however, we made a list of things that we still need to implement based on what we have currently to make the platform work, and reasons why we cannot get to them. So far, we have had the software working on simulation. The software consists of 4 parts:

1) Descriptions of the vehicle
   a) Dimensions
   b) Max velocity, Max acceleration, Max steering angle

c) Etc...

2) Description of the world
   a) An empty space where the vehicle always starts at the center of the world.
   b) Transformation of the vehicle to the world
   c) Max velocity, Max acceleration, Max steering angle
   d) 200 meters * 200 meters
3) Navigation Stack
   a) Path planning
   b) Obstacle avoidance
   c) Command issuing
4) Control
   a) Translation from commands issued by the navigation stack to the right values to control the motor and servos

We have everything else ready except 3a. “Transformation of the vehicle to the world” means the estimated location of the vehicle with respect to the world. Ex, the car always starts at the center of the world, which means the initial coordinate of the vehicle will always be (0, 0). Assuming a normal x-y coordinate system where y points to north and x points to east, if the vehicle move north for 10 meters, the location of the vehicle will become (0, 10). In simulation, the transformation is perfect because it is given by the simulation platform Stage. When we leave the simulation, and need to gather location information from sensors, it introduces huge uncertainties to the transformation. We only have one GPS unit and one IMU unit, which are the exactly opposite of “abundant”. We believe if we have a system that can produce accurate and reliable location estimation, our platform will work. This brings us to “commodity hardware”. We would like to elaborate on “commodity hardware”. We have stuck with our project proposal where it limits us to only use commodity hardware. So far, every component we have used can be bought easily online. Prices may vary depending on the websites. We believe 2,000 - 3,000 US dollars will be sufficient (including buying the RC car) to build such a platform from scratch. As a research group, we did not realize that the research budget was different than the actual cost of the platform. The 2,000 - 3,000 US dollars we mentioned above was the estimated total cost to build the platform. But we needed more than that for the research, because we had to experiment with different components to get the best performance, without paying too much, which we did not do. We did not really have a research budget. Time is another reason why we could not build a working prototype. We know this is cliché, but it’s really a factor. We got caught on some hardware that we found out later was useless to our project. We spent almost a term and a half on learning the ROS system and the AutoRally project. Also, we recently just realized that wheel encoders were highly recommended for doing location estimate for car like robot. We knew that wheel encoders were an option; however, we thought GPS and IMU were sufficient to give us accurate estimate, which was not true. Things like this delay our progress tremendously. We feel disappointed that we won’t have the car ready for demo; however, we are confident our conclusion holds true.

11 APPENDIX 1

Essential Code Listings. You don't have to include absolutely everything, but if someone wants to understand your project, there should be enough here to learn from. If you worked within a larger project, something like a patch file might be a good way to go.

```
/*********************************************
* @file  LiDARDetection.cpp
* @author Tao Chen <chentao@oregonstate.edu>
* @date Feburary 21, 2017
* @copyright 2017 Oregon State University
* @brief ROS node to generate new waypoints based on LiDAR scan
*
* @details When an obstacle is detected, it alters the original
* set of waypoints to go around the obstacle. This node subscribes to
* the laser_scan message and the current_list_of_waypoints message and
* publish the new_waypoints message.
*********************************************/

#include "LiDARDetection.h"

namespace autorally_smartdriving{
    LiDARDetection::LiDARDetection(){
        l_LiDARSub = l_nh.subscribe("/autorally_platform/laser_scan", 10, &LiDARDetection::gatherLiDARData,
            this);
        l_LiDARPub = l_nh.advertise<sensor_msgs::LaserScan>("scan", 10);

        samples = 2000;
        laser_frequency = 10;
        scan.ranges.resize(samples);
        scan.intensities.resize(samples);
    }

    LiDARDetection::~LiDARDetection(){}

    void LiDARDetection::gatherLiDARData(sensor_msgs::LaserScan data){
        int counter = 0;

        ros::Time scan_time = ros::Time::now();

        scan.header.stamp = scan_time;
```

```
        scan.header.frame_id = "LiDAR";
        scan.angle_min = data.angle_min;
        scan.angle_max = data.angle_max;
        scan.angle_increment = data.angle_max / samples;
        scan.time_increment = (1 / laser_frequency) / samples;
        scan.range_min = data.range_min;
        scan.range_max = data.range_max;

        for(counter = 0; counter < samples; counter++){
            scan.ranges[counter] = data.ranges[counter];
            scan. intensities [counter] = data. intensities [counter];
        }

        // publish scan to scan topic
        l_LiDARPub.publish(scan);
    }
};

int main(int argc, char** argv){
    ros :: init (argc, argv, "LiDARDetection");
    autorally_smartdriving::LiDARDetection LiDARDetection;
    ros :: spin() ;
}
```

Code Example 1. Example Custom ROS Node

```
<launch>

  <!-- ************** Global Parameters *************** -->
  <param name="/use_sim_time" value="true"/>

  <!-- ************** Stage simulator *************** -->
  <node pkg="stage_ros" type="stageros" name="stageros" args="$(find stage_launch)/stage/empty.world">
    <!-- <remap from="base_scan" to="scan"/> -->
    <remap from="base_scan" to="base_scan_0"/>
  </node>

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find smart_driving)/config/costmap_common_params.yaml" command="load"
        ns="global_costmap" />
```

```
    <rosparam file="$(find smart_driving)/config/costmap_common_params.yaml" command="load"
        ns="local_costmap" />
    <rosparam file="$(find smart_driving)/config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find smart_driving)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find smart_driving)/config/base_local_planner_params.yaml" command="load" />

    <param name="base_global_planner" value="global_planner/GlobalPlanner" />
    <param name="planner_frequency" value="1.0" />
    <param name="planner_patience" value="5.0" />

    <param name="base_local_planner" value="teb_local_planner/TebLocalPlannerROS" />
    <param name="controller_frequency" value="5.0" />
    <param name="controller_patience" value="15.0" />

    <param name="clearing_rotation_allowed" value="false" />
  </node>

  <node name="map_server" pkg="map_server" type="map_server" args="$(find
      stage_launch)/maps/empty_world.yaml" output="screen">
    <param name="frame_id" value="/map" />
  </node>

  <!--<node pkg="amcl" type="amcl" name="amcl" output="screen">
    <rosparam file="$(find stage_launch)/cfg/amcl_params.yaml" command="load"/>
    <param name="initial_pose_x" value="0" />
    <param name="initial_pose_y" value="0" />
    <param name="initial_pose_z" value="0" />
  </ode>-->

  <!-- <include file="$(find Leddar)/launch/Leddar.launch">
    <arg name="serial" value="AJ04071" />
    <arg name="frame" value="base_laser_link_0" />
    <arg name="fov" value="45" />
    <arg name="range" value="50" />
  </include> -->

  <include file="$(find stage_launch)/robot_localization.launch"/>

  <node name="odom" pkg="autorally_smartdriving" type="odom" />
```

```
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find stage_launch)/stage/rviz_navigation.rviz"/>
</launch>
```

Code Example 2. Example Custom Launch File for Stage

```
/*********************************************
* @file  odom.cpp
* @author Tao Chen <chentao@oregonstate.edu>
* @date Feburary 25, 2017
* @copyright 2017 Oregon State University
* @brief pulishes odometry data to the navigation stack
*
* @details This node subscribes to odometry data from /autorally_platform
and publisher the data to  the navigation stack
*********************************************/

#include "odom.h"

namespace autorally_smartdriving{
  Odom::Odom(){
    o_odomSub = o_nh.subscribe("/base_pose_ground_truth", 1, &Odom::gatherOdomData, this);
    o_odomPub = o_nh.advertise<nav_msgs::Odometry>("odom", 1);
    o_PoseArrayPub = o_nh.advertise<geometry_msgs::PoseArray>("particlecloud", 1);
    o_poseCovariancePub = o_nh.advertise<geometry_msgs::PoseWithCovarianceStamped>("amcl_pose", 1);
  }

  Odom::~Odom(){};

  void Odom::gatherOdomData(nav_msgs::Odometry data){
    tf :: TransformBroadcaster odom_broadcaster;
    ros :: Time time_stamp = ros::Time::now();

    //publish transform over tf
    geometry_msgs::TransformStamped odom_trans;
    geometry_msgs::TransformStamped map_trans;

    odom_trans.header.stamp = time_stamp;
    odom_trans.header.frame_id = "odom";              //base_link
    odom_trans.child_frame_id = "base_footprint";     //odom

    odom_trans.transform.translation.x = 0;
```

```
37  odom_trans.transform.translation.y = 0;
38  odom_trans.transform.translation.z = 0;
39  odom_trans.transform.rotation.x = 0;
40  odom_trans.transform.rotation.y = 0;
41  odom_trans.transform.rotation.z = 0;
42  odom_trans.transform.rotation.w = 1;
43
44  /*odom_trans.transform.rotation.x = data.pose.pose.orientation.x;
45  odom_trans.transform.rotation.y = data.pose.pose.orientation.y;
46  odom_trans.transform.rotation.z = data.pose.pose.orientation.z;
47  odom_trans.transform.rotation.w = data.pose.pose.orientation.w;*/
48
49  map_trans.header.stamp = time_stamp;
50  map_trans.header.frame_id = "map";
51  map_trans.child_frame_id = "odom";       //base_link
52
53  map_trans.transform.translation.x = data.pose.pose.position.x;
54  map_trans.transform.translation.y = data.pose.pose.position.y;
55  map_trans.transform.translation.z = data.pose.pose.position.z;
56  map_trans.transform.rotation.x = data.pose.pose.orientation.x;
57  map_trans.transform.rotation.y = data.pose.pose.orientation.y;
58  map_trans.transform.rotation.z = data.pose.pose.orientation.z;
59  map_trans.transform.rotation.w = data.pose.pose.orientation.w;
60
61  /*map_trans.transform.rotation.x = 0;
62  map_trans.transform.rotation.y = 0;
63  map_trans.transform.rotation.z = 0;
64  map_trans.transform.rotation.w = -1;*/
65
66  //send the transform
67  odom_broadcaster.sendTransform(odom_trans);
68  odom_broadcaster.sendTransform(map_trans);
69
70  //Publish pose array
71  geometry_msgs::PoseArray poseArray;
72  poseArray.poses.resize(1);
73  poseArray.header.stamp = time_stamp;
74  poseArray.header.frame_id = "map";
75
```

```
76      for(int i = 0; i < 1; i++){
77        poseArray.poses[i].position.x = data.pose.pose.position.x;
78        poseArray.poses[i].position.y = data.pose.pose.position.y;
79        poseArray.poses[i].position.z = data.pose.pose.position.z;
80
81        poseArray.poses[i].orientation.x = data.pose.pose.orientation.x;
82        poseArray.poses[i].orientation.y = data.pose.pose.orientation.y;
83        poseArray.poses[i].orientation.z = data.pose.pose.orientation.z;
84        poseArray.poses[i].orientation.w = data.pose.pose.orientation.w;
85      }
86      o_PoseArrayPub.publish(poseArray);
87
88      //Publish pose with covariance
89      geometry_msgs::PoseWithCovarianceStamped PoseWithCovariance;
90      PoseWithCovariance.header.stamp = time_stamp;
91      PoseWithCovariance.header.frame_id = "map";
92      PoseWithCovariance.pose.pose.position.x = data.pose.pose.position.x;
93      PoseWithCovariance.pose.pose.position.y = data.pose.pose.position.y;
94      PoseWithCovariance.pose.pose.position.z = data.pose.pose.position.z;
95
96      PoseWithCovariance.pose.covariance = data.pose.covariance;
97      o_poseCovariancePub.publish(PoseWithCovariance);
98
99      //add frame_id and child_frame to odom message
100     data.header.stamp = time_stamp;
101     data.header.frame_id = "map";
102
103     data.child_frame_id = "odom";
104     o_odomPub.publish(data);
105   }
106 };
107
108 int main(int argc, char** argv){
109   ros::init(argc, argv, "OdomNav");
110   autorally_smartdriving::Odom odom;
111   ros::spin();
112 }
```

Code Example 3. ROS odom/transform node (not including the .h file. The .h file is located in our github repository)

```
1 <launch>
```

```
  <include file="$(find autorally_core)/launch/hardware.machine" />
  <!-- Run the map server -->
  <node name="map_server" pkg="map_server" type="map_server" args="$(find
      autorally_smartdriving)/launch/test_map_empty.yaml"/>

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find autorally_smartdriving)/config/costmap_common_params.yaml" command="load"
        ns="global_costmap" />
    <rosparam file="$(find autorally_smartdriving)/config/costmap_common_params.yaml" command="load"
        ns="local_costmap" />
    <rosparam file="$(find autorally_smartdriving)/config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find autorally_smartdriving)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find autorally_smartdriving)/config/base_local_planner_params.yaml" command="load" />

    <param name="base_global_planner" value="global_planner/GlobalPlanner" />
    <param name="planner_frequency" value="1.0" />
    <param name="planner_patience" value="5.0" />

    <param name="base_local_planner" value="teb_local_planner/TebLocalPlannerROS" />
    <param name="controller_frequency" value="5.0" />
    <param name="controller_patience" value="15.0" />

    <param name="clearing_rotation_allowed" value="false" /> <!-- our vehicle can't rotate inplace -->
  </node>
</launch>
```

Code Example 4. Example launch file

```
<?xml version="1.0"?>
<package>
  <name>autorally_smartdriving</name>
  <version>0.0.0</version>
  <description>This package contains all the necessary files to enable the car to detect and avoid obstacle using
      the teb_local_planner package.</description>

  <maintainer email="chentao@oregonstate.edu">Tao Chen</maintainer>

  <license>BSD</license>

  <url type="website">https://github.com/cshawe/cs46x</url>

```

```
13   <author email="chentao@oregonstate.edu">Tao Chen</author>
14
15   <buildtool_depend>catkin</buildtool_depend>
16
17   <build_depend>roscpp</build_depend>
18   <build_depend>rospy</build_depend>
19   <build_depend>std_msgs</build_depend>
20   <build_depend>tf</build_depend>
21   <build_depend>nodelet</build_depend>
22   <build_depend>sensor_msgs</build_depend>
23   <build_depend>geometry_msgs</build_depend>
24   <build_depend>nav_msgs</build_depend>
25   <build_depend>autorally_msgs</build_depend>
26   <build_depend>move_base</build_depend>
27   <build_depend>my_odom_configuration_dep</build_depend>
28   <build_depend>my_sensor_configuration_dep</build_depend>
29   <build_depend>my_tf_configuration_dep</build_depend>
30
31   <run_depend>roscpp</run_depend>
32   <run_depend>rospy</run_depend>
33   <run_depend>std_msgs</run_depend>
34   <run_depend>tf</run_depend>
35   <run_depend>nodelet</run_depend>
36   <run_depend>sensor_msgs</run_depend>
37   <run_depend>geometry_msgs</run_depend>
38   <run_depend>nav_msgs</run_depend>
39   <run_depend>autorally_msgs</run_depend>
40   <run_depend>move_base</run_depend>
41   <run_depend>my_odom_configuration_dep</run_depend>
42   <run_depend>my_sensor_configuration_dep</run_depend>
43   <run_depend>my_tf_configuration_dep</run_depend>
44
45
46   <!-- The export tag contains other, unspecified, tags -->
47   <export>
48     <!-- Other tools can request additional information be placed here -->
49
50   </export>
51 </package>
```

Code Example 5. Example package config file

```
# footprint is like the shadow of the vehicle if the light is shining from directly above.
# Assume that the center of the vehicle is (0, 0)
footprint: [[−0.3, 0.15], [0.3, 0.15], [0.3, −0.15], [−0.3, −0.15]]

transform_tolerance: 0.5        # undecided
map_type: costmap

obstacle_layer:
  enabled: true
  # MaxIMUm range sensor reading that will result in an obstacle being put into the costmap
  # 8.0 means the robot will only updata its map with info about obstacles that are within
  # 8.0 meters of the base.
  obstacle_range: 8.0
  # 5.0 means that the robot will attemp to clear out space in front of it up to 5.0 meters
  # away given sensor reading.
  raytrace_range: 5.0
  # This value determines how far the vehicle will stay away from obstacles.
  inflation_radius: 1.0
  # from the tutorial
  track_unknown_space: false
  combination_method: 1
  # we are only using laser scan
  observation_sources: laser_scan_sensor
  # marking and clearing mean that the sensor data will be used to mark and clear obstacle info
  # on the costmap
  laser_scan_sensor: {sensor_frame: LiDAR, data_type: LaserScan, topic: scan, marking: true, clearing: true,
      expected_update_rate: 0.5}

inflation_layer:
  enabled: true
  cost_scaling_factor: 10.0    # exponential rate at which the obstacle cost drops off
  inflation_radius: 1.0        # max distance from an obstacle at which costs are incurred for planning paths.

static_layer:
  enabled: true
map_topic: "/map"
```

Code Example 6. Example yaml (package params) file

```
cmake_minIMUm_required(VERSION 2.8.3)
project(autorally_smartdriving)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  sensor_msgs
  geometry_msgs
  nav_msgs
   tf
)

catkin_package(
  INCLUDE_DIRS include
)

include_directories(
  include ${catkin_INCLUDE_DIRS}
)

add_subdirectory(src/LiDARDetection)
add_subdirectory(src/Odom)
add_subdirectory(src/Global_Path)
add_subdirectory(src/Drive)
add_subdirectory(src/IMU)

 install (DIRECTORY launch/
    DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}/launch
    FILES_MATCHING PATTERN "*.launch" PATTERN "*.machine"
)

 install (DIRECTORY include/${PROJECT_NAME}/
    DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
)

 install (DIRECTORY cfg/cpp/${PROJECT_NAME}/
```

```
    DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
    FILES_MATCHING PATTERN "*.h"
)
```

Code Example 7. Example cmakelist file

```
include "arc_robot.inc"
include "block.inc"


define floorplan model
(
  # sombre, sensible,  artistic
  color "gray30"

  # most maps will need a bounding box
  boundary 1

  gui_nose 0
  gui_grid 0
  gui_outline 0
  gripper_return 0
  fiducial_return 0
  laser_return 1
)

resolution 0.02
interval_sim 100  # simulation timestep in milliseconds

window
(
  size [ 600.0 700.0 ]
  center [ 0.0 0.0 ]
  rotate [ 0.0 0.0 ]
  scale 60
)

floorplan
(
  name "empty_world"
  bitmap "../maps/empty_world.png"
```

```
  size [ 200.0 200.0 2.0 ]
  pose [ 0.0 0.0 0.0 0.0 ]
)

# throw in a robot
carlike_robot
(
  pose [ 0.0 0.0 0.0 0.0 ]
  name "robot"
)

# adding blocks
block ( pose [2.0 4.0 0.0 0.0] color "red")
block ( pose [4.0 4.0 0.0 0.0] color "red")
block ( pose [6.0 4.0 0.0 0.0] color "red")
block ( pose [8.0 4.0 0.0 0.0] color "red")
block ( pose [10.0 4.0 0.0 0.0] color "red")
block ( pose [12.0 4.0 0.0 0.0] color "red")
block ( pose [14.0 4.0 0.0 0.0] color "red")
block ( pose [16.0 4.0 0.0 0.0] color "red")
block ( pose [18.0 4.0 0.0 0.0] color "red")
block ( pose [20.0 4.0 0.0 0.0] color "red")
block ( pose [22.0 4.0 0.0 0.0] color "red")
```

Code Example 8. Example stage config file

```
define laser ranger
(
  sensor
  (
    range_max 6
    fov 360
    samples 2000
  )

  color "black"
  size [ 0.06 0.15 0.03]
)

define Leddar ranger
(
```

```
  sensor
  (
    range_max 50.0
    fov 45
    samples 16
  )
  color "black"
  size [0.06 0.15 0.03]
)

define carlike_robot position
(
  pose [ 0.0 0.0 0.0 0.0 ]

  odom_error [ 0.03 0.03 999999 999999 999999 0.02 ]

  size [ 0.56 0.46 0.4 ]
  origin [ 0.2 0.0 0.0 0.0]
  gui_nose 1
  color "green"

  drive "car"
  localization "gps"
  wheelbase 0.38

  #laser(pose [-0.1 0.0 -0.11 0.0])
  laser(pose [0.1 0.0 -0.11 0.0])

  Leddar(pose [0.2 0.0 -0.11 0.0])
)
```

Code Example 9. Car description for stage

12 APPENDIX 2

Anything else you want to include. Photos, etc.