2017 Intelligent Ground Vehicle Competition “Jack Frost”

UBC Snowbots

University of British Columbia

Name	Department	Position
Winnie Mui	Engineering Physics	Captain/Mechanical Lead
Vincent Yuan	Electrical Engineering	Captain/Admin Lead
Fion Yu	Commerce	Admin Member
Jacky Sun	Mechanical Engineering	Mechanical Lead
Valerian Ratu	Computer Engineering	Software Lead
Gareth Ellis	Computer Science	Software Lead
David Gill	Engineering	Mechanical Team
Martin Freeman	Mechanical Engineering	Mechanical Team
William Gumboc	Chemical and Biological Engineering	Mechanical Team
Emma Park	Mechanical Engineering	Mechanical Team
Yvonne Ku	Mechanical Engineering	Mechanical Team
Sherry Wang	Mechanical Engineering	Mechanical Team
Tiger Zuo	Computer Science/Physics	Mechanical Team
Ben Smith	Engineering	Mechanical Team
Jay Paul	Mechanical Engineering	Mechanical Team
Ivy Jiayuan Shi	Mechanical Engineering	Mechanical Team
Shichen Fan	Mechanical Engineering	Mechanical Team
Finn Hackett	Computer Science	Software Team
Emmanuel Sales	Computer Science	Software Team
Simon Jinaphant	Computer Engineering	Software Team
William Ou	Computer Engineering	Software Team
Chris Chen	Computer Engineering	Software Team
Robyn Castro	Computer Engineering	Software Team
Kevin Luo	Computer Engineering	Software Team
Daniel Huang	Computer Engineering	Software Team
Jinhao (Luke) Lu	Electrical Engineering	Software Team
Raad Khan	Electrical Engineering	Software Team

I hereby certify as the faculty advisor that the design and engineering of this vehicle to be entered in the 2017 Intelligent Ground Vehicle Competition by the current student team has been significant and equivalent to what might be awarded credit in a senior design course.

W. Scott Dunbar

Scott Dunbar

Introduction

Jack Frost is a vehicle designed and constructed by UBC Snowbots this year. Over 2500 hours were spent between all team members on this project. The goals for this year were to design a vehicle that would be lightweight and compact, allowing easier transportation and increased maneuverability. Multiple sensors including a LIDAR, compass, GPS, and camera will be used to gain environmental data which will then be processed to navigate through the course. This report will describe our team's organization, design strategy, and our mechanical, electrical, and software systems.

Team Organization

UBC Snowbots includes members from a variety of engineering departments, the computer science department, and the faculty of business. The team is divided into three main subteams: the mechanical subteam, the software subteam, and the administrative subteam. The mechanical subteam is responsible for the mechanical system of the vehicle and the power circuitry, the software subteam is responsible for the firmware and navigational software, and the administrative subteam is responsible for connecting with sponsors, managing funds, and outreach. Team meetings are weekly for three to four hours, and our members meet at additional times to complete their projects throughout the week.

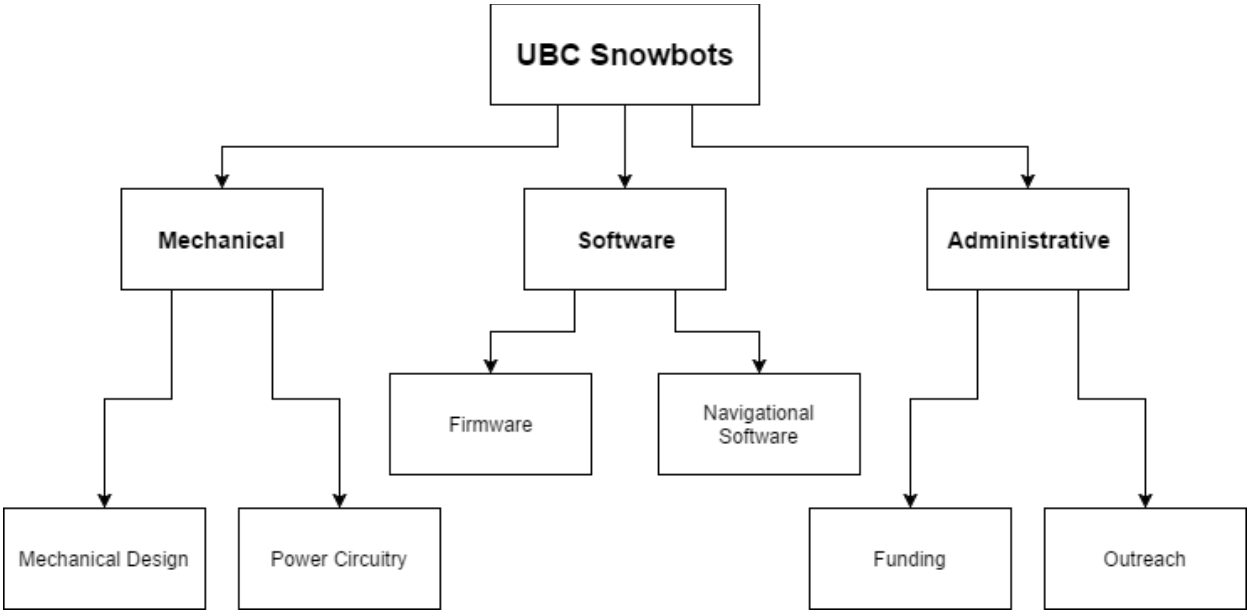


Figure 1 - UBC Snowbots team structure.

MECHANICAL DESIGN

Introduction

The goal of this year's design was to create a vehicle that was lighter and more compact to simplify transportation to the competition. In the process, we also worked to better integrate sensors into the design of our system, along with increasing the maneuverability of the vehicle. As a student team, the main focus through the design process has always surrounded ease of machinability. The effects of this are shown in our material selection and frame design.

Material Selection

Chassis manufacturing was chosen to be done in house, which lead to the decision of using aluminum square stock and metal sheets as the main material for our vehicle this year. Harder materials such as stainless steel were intentionally avoided due to machining difficulties. All stronger parts were designed to be made with commercially available parts. The parts of the vehicle with high mechanical load were reinforced with steel coated in paint to prevent rusting.

Drivetrain Design

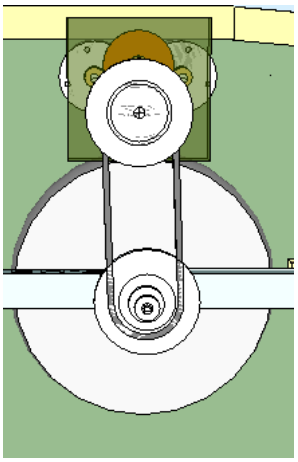


Figure 2 - Drivetrain design.

To satisfy one of our main goals of reducing the volume of the vehicle for easier transportation to the competition, we decided the best solution for utilizing space within the vehicle was to decouple the motors from the wheels. To achieve this, our drivetrain uses a roller chain drive system so that the belly of the vehicle would be free of obstacles to hold the payload. Each wheel is connected to the roller chain, which is connected to our gearboxes. Commercial gearboxes were modified to include encoder and sprockets for our new drive design. The calculations to determine whether the gear ratio would be sufficient to propel the vehicle can be seen in the ramp climbing ability section.

Chassis Design

Our chassis was designed to carry items in layers to best utilize the space. The bottom layer consists of the heaviest items to ensure stability of the vehicle. This includes the battery, payload, and LIDAR. The middle layer consists of a drawer capable of sliding out to hold the laptop, and the top layer

consists of the electrical box. The chassis features various openings to allow easy access to its components. The top opens for access to the electrical box and its components, with minimal disturbance to wiring. The front of the vehicle flips up to allow for access to the battery and LIDAR. The back can also be opened to access the payload and the laptop drawer.

Tower

Figure 3 - Tower with camera mount.

A tower was necessary to ensure that our camera is elevated and can gain a wide, bird's eye view of the field. It is securely mounted to the center of the vehicle to ensure that the view itself is centered. The tower design allows for the camera to be tilted to change the angle of view, as well as moved up and down along the tower to adjust the height of its view.

Ramp Climbing Capability

To determine whether the vehicle would be able to climb a ramp of 8.5 degrees (the maximum specified in IGVC rules), a force analysis was performed to determine if the vehicle would slip down a ramp of that incline. If the vehicle does not slip down the ramp, then the vehicle will be able to climb the ramp provided that there is sufficient momentum prior to beginning the climb. The pushing force was calculated to be 60N, while the static weight along the incline is 55N, therefore the vehicle will not slip down the ramp.

Mass of vehicle (kg)	40
Gear ratio	12.75
Wheel radius (m)	0.127
Number of motors	4
Angle of Inclination	15% / 8.5°
Coefficient of Friction	0.35 mu

Speed

To determine the maximum possible speed of the vehicle, the torque of the vehicle was calculated and the performance curve of the motor was used to determine the corresponding rotations per minute. From a force analysis of the system, 137N is required to move the vehicle, this corresponds to 17.4Nm torque at the wheel, which corresponds to 0.34Nm at the output shafts of the motor. From the motor performance curves, the rpm can be determined and used to calculate the maximum speed. The final maximum speed is estimated to be 2.378 m/s. However, due to the roller chain in the drive train, the inefficiencies of the drive train have increased, and the maximum speed is estimated to be 80% of the calculated maximum, or 1.9 m/s, which corresponds to 4.26 mph, which is under the speed limit.

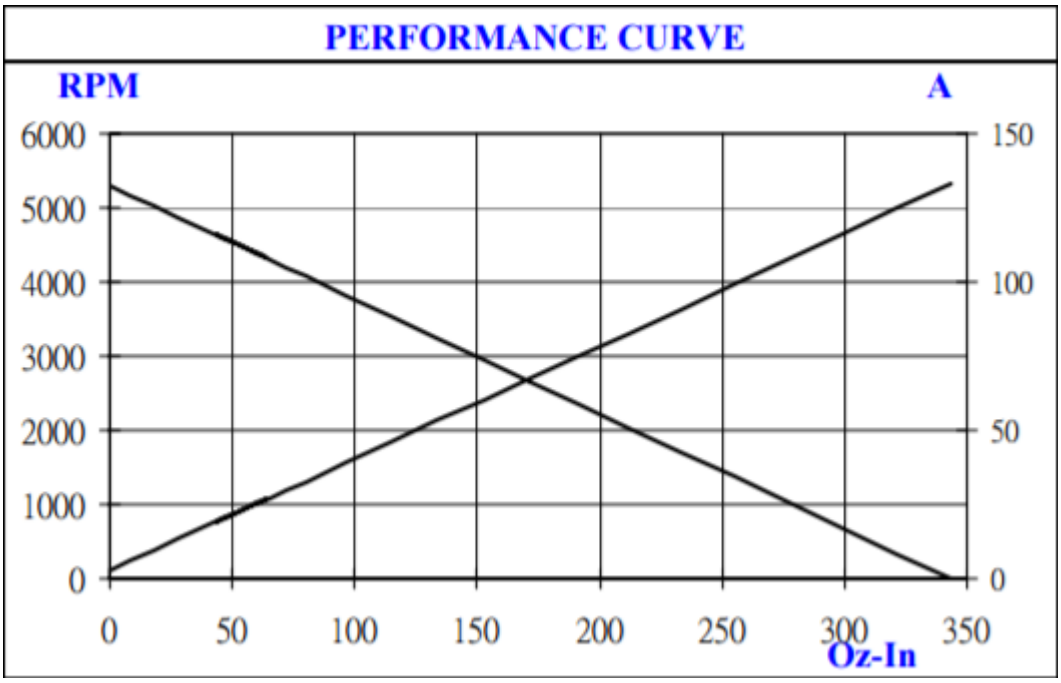


Figure 4 - Motor performance curve.

Electrical Design

Power Distribution System

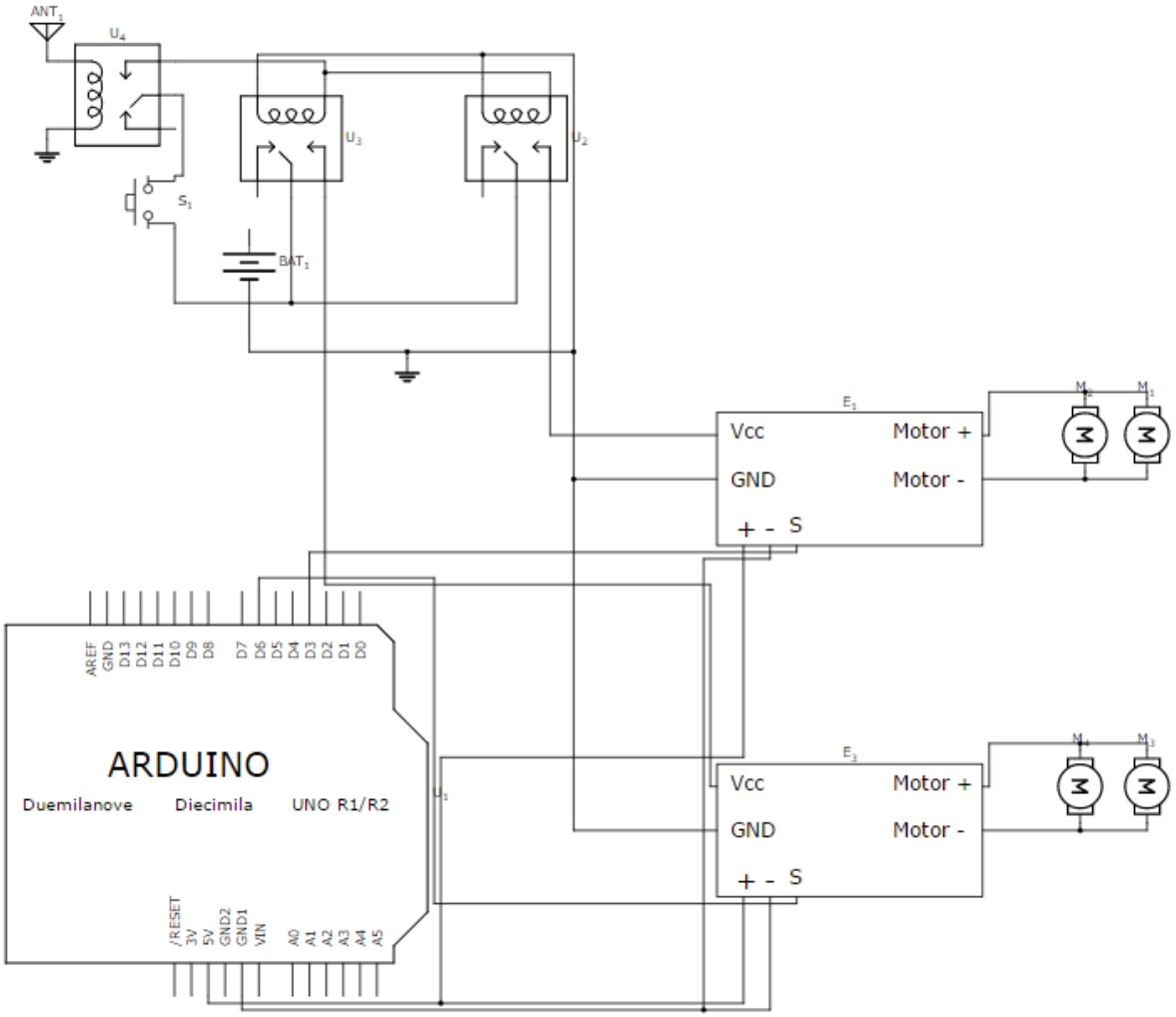


Figure 5 - Power circuitry of Jack Frost.

Operating Life

The operating life of the system is dependent primarily on the power consumption of the motor and battery capacity. Though there are other components of the vehicle that consume power such as the LEDs, microcontrollers and LIDAR, their effect on battery life is negligible in comparison to that of the motors. The following parameters were used for the operating life calculations:

Effective Battery Capacity (Amp hours)	32
LIDAR Power Consumption (Amp Hours)	1
Motor power consumption (Amp Hours)	23
Battery life (hours)	1.33

As our battery cannot be fully discharged, the effective battery capacity was estimated to be 80% of the total capacity. Using the specifications for our motors, the motor power consumption is estimated to be 23 W when running at 4 mph. The final estimated battery life is 1.33 hours of driving full speed. Since the vehicle does not run at full speed the entire run, the operating life while navigating the course should be longer than 1.33 hours. During test runs of the vehicle the battery life of the vehicle was timed to be 1.5 hours, this variance is most likely due to differences in run speed and efficiency of the actual vehicle.

Emergency Stop System

Our emergency stop system has three components. The first component is firmware-based. Before executing any commands from the laptop, the microcontroller will check that the radio controller is both on and set to autonomous mode. The next component is a wireless relay that will cut power to relays which will stop power to the motors when pressed. The final component is entirely hardware based, and is our emergency stop button located at the back of our vehicle which will physically cut power to the vehicle when pressed. To ensure safety, all power relays and switches are normally open.

Firmware

Arduino Mega microcontrollers are used for control and communicating with the electrical systems. The vehicle has three main states: autonomous, remote controlled, and stopped, which can be changed through the radio controller. When the vehicle is set to autonomous mode, USB serial communication is used to send velocity commands from the laptop to the microcontroller. Under remote control, all velocity commands are sent wirelessly from the radio controller to the microcontroller's receiver. The signals are then processed by the microcontroller which sends Pulse Width Modulation (PWM) signals to the Electronic Speed Controls (ESCs) which control the wheels independently, thus moving the vehicle in the desired direction.

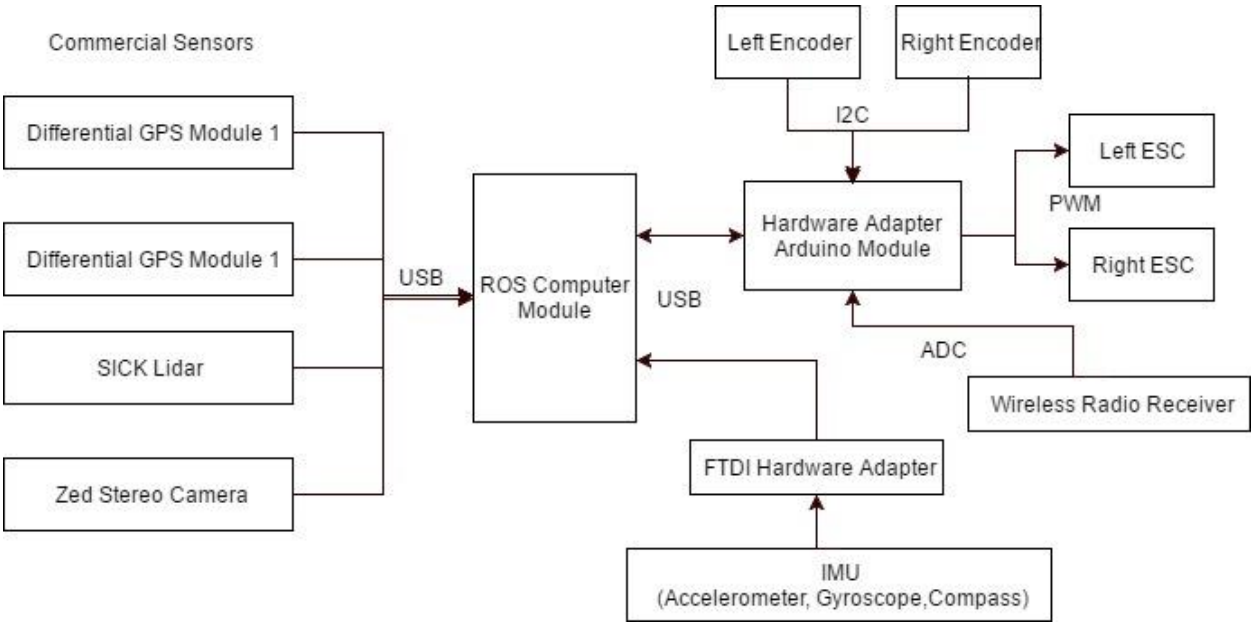


Figure 6 - Software sensor diagram.

Sensors

Encoders, GPS modules, and an IMU are used to calculate the speed and position of the vehicle. These modules run through Arduino Megas and Unos, which in turn communicate with the main computer via USB serial. The computer, upon deciding upon the best direction to move, then sends velocity commands back over USB serial to another Arduino, which converts them into Pulse Width Modulation (PWM) signals to send to the Electronic Speed Controls (ESCs) which control the wheels independently, moving the vehicle in the desired direction.

LED Alert System

A LED light is mounted directly behind the front windshield of the vehicle for visibility. The Arduino Mega drives the LED alert system according to its mode, with remote-controlled as a steady light, autonomous as a blinking light, and off as off as per the rules.

Software Strategy

The software has been developed in C++ and Python, using the Robotic Operating System (ROS), Open Computer Vision (OpenCV) library, and a custom Simultaneous Localization and Mapping Library (SLAM) library developed in-house.

In order to localize itself, the vehicle receives input from encoders, GPS modules, and an Inertial Measurement Unit (IMU) (consisting of an Accelerometer, Gyroscope, and Magnetometer). The vehicle can also generate a local map of the environment from the pointclouds received from our Zed stereo vision camera, and from our LIDAR. Once localized, this map can then be superimposed onto our global map, thus updating it. Pathfinding can then be performed over this global map to come up with the best path to the destination waypoint. From this path, a velocity vector is then computed to send to the wheels.

Vision System

The vision system revolves around the Zed stereo camera. The output from the camera is fed to a filter which picks out white lines and obstacles which can then be consumed by the mapping system.

Zed Stereo Camera

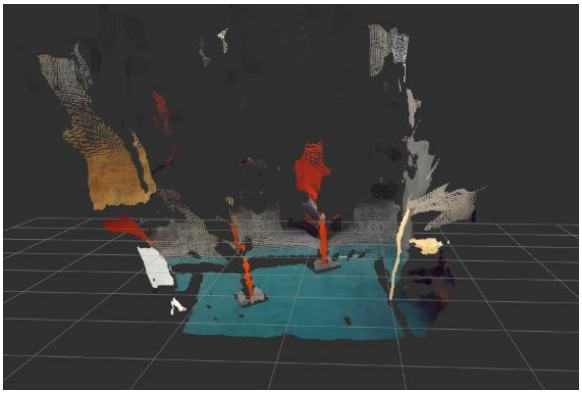
The Zed Stereo Camera is a cutting edge stereo camera package which comes with a well-developed SDK. The Zed Stereo camera's main output is the point cloud generated from the image disparity between each camera. This data consists of the xyz position of each pixel of the image in meters, relative to the camera, and the color of said pixel.

To use this data, we created the filter which analyzes the point cloud for consumption by our mapping system. The filter operates by first flattening the point cloud onto one plane - the xy plane. This creates an easy surface to analyze with manageable margins of error. Each point's RGB values are then converted into the HSV (Hue, Saturation, Value) colour space which leads to improved and more tolerant colour filtering. Filtering for a specific colour on the RGB colour space is made difficult

due to changes in lighting - which occurs outdoors frequently due to time of day, weather, and changing cloud patterns. The HSV colour space handles this more effectively as its parameters are better tuned to the aforementioned changes in the environment.

The end output of the filter is a localized point cloud in the xy plane with only the colours of interest. For example, a filter which is configured to detect white will output a point cloud which corresponds to the white lines on the field as well as any white obstacles.

We create multiple instances of the filter which independently consume the same raw point cloud data. Each instance corresponds to a certain colour of obstacle and the output of each instance is processed independently by the mapping node.



An example of point cloud produced from the Zed

Mapping Technique

LIDAR

The SICK LMS 291-S14 LIDAR uses a rotating laser beam to measure distances to obstacles by analyzing the time of flight of the reflected beam. The LIDAR is mounted in the front of the vehicle and has a scanning angle of 90 degrees and a range of 30m.

The ROS node for LIDAR runs concurrently with the sicktoolbox_wrapper, which translates raw data from the LIDAR into useful data that can be read by the LIDAR node. The LIDAR node then takes the translated data and can publish the data to the Mapping node where the LIDAR data can be combined and processed along with other data.

GPS

Commercially available GPS modules (Swift Piksi's) were obtained for this year's competition with 2 qualities in consideration: Differential GPS capabilities and Antenna extension capabilities. From previous experience, single point precision with on board antennas provided us with a 10m accuracy based off of longitudinal and latitudinal fluctuation from collected data. Hypothesis of fluctuation was based off of further research into antenna propagation and comparison of other on hand GPS modules. A secondary GPS module was also added to the system to further reduce error.

Encoders

High precision 1024 PPR (Pulses Per Rotation) encoders have been attached to each wheel, giving us an accurate picture of vehicle movement, particularly over short distances, where error has little time to accumulate. They are also used to correct vehicle movements, as detailed in the *PID Control* section below.

Localisation

The position of the world in our global frame is computed from the encoders, GPS, and IMU, combined together via an EKF (Extended Kalman Filter). The filter is implemented in the *robot-pose-ekf* ROS library. With this, we can achieve highly accurate results, up to +/- 30cm relative to our initial position. To improve estimation of our absolute position, (in terms of longitude and latitude), before a run the localisation system will gather a large (200-300) sample of readings from our GPS's, and average them to produce a more accurate absolute starting position.

Mapping

The mapping node is based off the *grid_map* package created by the Autonomous Systems Lab in ETH Zurich. We have created our own mapping system based off the data structures provided in this package, allowing us to arbitrarily manipulate and combine "layers" computed from different data sources dependent upon the elements of interest in the world.

At a high level, our global map is a multi-layer two dimensional array. Each layer has been designed to correspond to a single sensor or filter output. Once a local map has been computed from sensor or filter outputs, we use the current vehicle position and orientation obtained through the EKF to superimpose this local map onto its global counterpart stored in each layer. Thus each layer contains the history of outputs from a given sensor. By keeping the sensor data in separate layers, we can reduce computational load by storing layers that are likely to only have larger objects (such as cones seen by the LIDAR) with a lower layer resolution as compared to obstacles such as white lines on the ground, which require a higher resolution. It also allows us to easily produce global maps using subsets of, or all, layers, which can then be used for navigation.

Pathfinding

Path Finding Algorithm

Popular algorithms, such as Dijkstra and A*, are capable of determining the shortest path in an obstacle filled environment; however, these algorithms are more suitable for a fully mapped environment where nothing will change during the path calculation. Our situation required an algorithm that can determine an initial path based on the limited data provided by the LIDAR and vision system, and update its resultant path as the vehicle traverses the environment and discovers more obstacles. For this reason we have chosen to implement the D* Lite algorithm.

Command for Vehicle Movement

Once D* Lite finds the current optimal path to the goal waypoint, a new linear and angular velocity will be sent to *Jack Frost* to direct it. To determine this new linear and angular velocity, a local path (essentially the first portion of the optimal path) will be estimated and smoothened out to mimic a more realistic vehicle movement.

PID (Proportional-Integral-Derivative) Control

In order to ensure that the commands sent to the wheels are accurately reflected in the vehicle's movement, we use a heavily modified version of the *ros_arduino_bridge* library to account for error in the vehicle's movement, by counting the number of encoder pulses received, and comparing them to the expected number of pulses for a given movement command. Corrections are then computed and sent to the speed controllers. To reduce latency and ensure accuracy and responsiveness, PID corrections are performed in firmware, directly on the Arduino controlling the motors.

Networking

The networking module's purpose is to ensure JAUS compliance of the system, and to map from required JAUS commands to internal commands using the ROS framework.

The system is implemented in the Python programming language using the asyncio framework in order to meet two implementation goals: ease of troubleshooting and debugging, and to provide a simple and easy to understand implementation.

In order to achieve these goals, the *pytest* unit testing framework was used to test both the functionality of individual modules and the overall compliance of the system using simulated network events.

Elements of the JAUS standard that have been implemented so far include:

- JUDP transport: the low-level transport specification
- Liveness: a basic heartbeat pulse
- Events: request scheduled reports from other services
- Management: set emergency shutoff, query platform state
- Access Control: request exclusive control over the platform
- Discovery: query the name and components of the platform
- List Manager: manage a linked list structure in the platform's memory
- Local Pose Sensor: report the position and orientation of the platform in local coordinates
- Local Waypoint Driver: execute move-to-waypoints commands
- Local Waypoint List Drivers: move to a list of waypoint in sequence (managed by the list_manager service)
- Velocity State Sensor: report the platform's velocity

It may be of interest that development of the networking module has led to the development of a generic binary data parsing library which aims to make possible declarative parsing of arbitrary data structures, especially those like JAUS messages which mix bit fields with variable-length arrays. We hope that this pushes us further toward to ease of use and understanding criterion, since it removes the usual requirement for ad-hoc parsing code and instead allows other program logic to simply see objects with named attributes.

Failure Points Identification and Resolution

The following potential failure points of our vehicle were identified and a corresponding solution was found to minimize the possibility of failure.

Emergency Stop System

As described in the emergency stop section, our emergency stop systems are designed to be normally open to account for any possible failure in the stop system. If any of the stop systems fail, their default state is to leave an open circuit, cutting power to the vehicle.

Radio Interference

Two different wireless switches have been implemented, a soft switch in software and a hard switch by mechanical relay that controlled by a small radio button. In cases where the soft switch signals are out of range or receive interference, the vehicle goes into default stall mode such that the vehicle can be physically switched off either by the wireless relay or physical kill switch.

Overcurrent To Power Transmission Cables

The power supplied to the brushed DC motors is monitored by voltage and current meters and is controlled by PWM from the ESC's. In case overcurrent arises due to a programming error going undetected, fuses are placed at the bottleneck wire gauge at its rated ampacity.

Operation In Rainy Conditions

As the housing of our vehicle is made with sheet metal and held together with bolts, water may seep into the gap between the sheet metal and bolts. In order to prevent this, O-rings, weather stripping and silicone caulk were used to weather seal bolts, doors and crevices respectively.

Stall Motors

Although preventing a stalled motor beforehand is difficult, we can reduce the severity of this occurrence by minimizing the amount of time for which the motor is stalled. This is accounted for in firmware, via the aforementioned PID controller.

Wheel Slippage

In cases where the wheels are caught or slipping, software can detect this by comparing rotary encoders to SLAM results. Proper response actions are then carried out to reattempt previous action or wait in place for human assistance.

Challenging Obstacles

By maintaining a global map of the course, challenging obstacles such as switch-backs can be easily navigated by re-computing the optimal path as we see more of the obstacle.

COST ANALYSIS

The estimated overall cost of *Jack Frost* was as follows.

	Cost
Electronic Speed Controllers	$200.00
Motor Encoders	$110.00
Arduino Microcontrollers	$160.00
ZED Stereo Camera	$700.00
Sick LIDAR	$4,500.00
Compass	$40.00
Mechanical Hardware	$700.00
Piksi GPS Module	$1,235.00
ASUS Laptop	$1,600.00
USB Hub	$50.00
Laptop Power Pack	$120.00
Electrical Hardware	$100.00
Sheet Metal and Machining	$1,300.00
Brushed DC Motors	$100.00
Gear boxes	$200.00
Acrylic & Plastic	$150.00
Wheels	$200.00
Total Cost	$11,465.00

CONCLUSION

Over 20 members of UBC Snowbots worked on the design and construction of *Jack Frost* this year. Our team has experienced a change in leadership in the last year, and has worked hard to rebuild a strong core team along with rebuild the vehicle from scratch. Our design has improved greatly with a lighter and more compact mechanical design, more sophisticated software strategy, and cleaner electrical system. The team is looking forward to bringing *Jack Frost* to this year's IGVC and excited to see the results of the competition.