SNOWFLAKE: IGVC 2015 DESIGN REPORT

UBC Snowbots

University of British Columbia

Aashish Karna
Albert Lagman
Andres Rama
Andrew Kang
Anmol Jawandha
Anni Wang
Anson Yam
Arjun Sethi
Bardia Beigi
Ben Chow
Calvin Xu
Clarissa Gunawan
Edbert Candra
Edward Li
Emma Tam
Finn Hackett
Jame Lee
Jamie Ye
Jannicke Pearkes
Jarek Ignas-menzies
Jason Raymundo
Jay Dahiya
Jay Paul
Jennifer Ling
Johnny Li
Jonathan Harris
Justin Liang
Kirk Wong
Maxwell Li
Maya Schuller
Nancy Pang
Nicholas Leung
Nicholas Wu
Otto Lau
Rishabh Singal
Ryan McDonald
Simon Jinaphant
Tabitha Lee
Vincent Yuan
Viral Galaiya
Vivian Wong
Winnie Mui
Yuqing Du
Ziyue(Grace) Hu

In memory of Dr. John Meech

INTRODUCTION

Snowflake is a brand new robot designed and constructed by UBC Snowbots this year. *Snowflake* has a square chassis supported by two primary wheels and four omni-wheels. One LIDAR sensor is used for obstacle detection, and three cameras on a tower are used for computer vision. Multiple software strategies are employed to allow *Snowflake* to navigate the IGVC course swiftly and precisely. This report will describe our team's organization, design strategy, and the mechanical, electrical and software elements of our vehicle. The report includes a detailed cost analysis and will end with our aspirations for the upcoming competition.

TEAM ORGANIZATION

UBC Snowbots consists of UBC students from a variety of engineering departments, and the faculty of computer science. There are three main divisions of the team – the mechanical division, the electrical division and the software division as one can see in Figure 1 below. Each division is managed by the team leads. As we are a large team, each division is further split up into sub-divisions and then into project groups. Typically 2-3 students work on a particular project. The entire team meets every week for three to four hours and project sub-divisions meet at additional times as required by their tasks

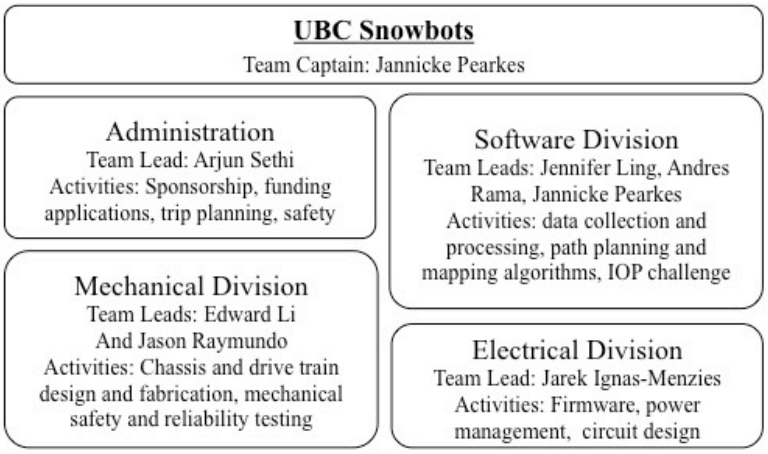

Figure 1. Team Organizational Structure

MECHANICAL DESIGN

The robot was divided into three distinct modules, the drive train, housing and tower. Throughout the design process, ideas were communicated amongst our sub-divisions, and we later unified our assemblies to form the robot. SolidWorks was used to design all components to confirm all design parameters which will be described in detail in the following sections.

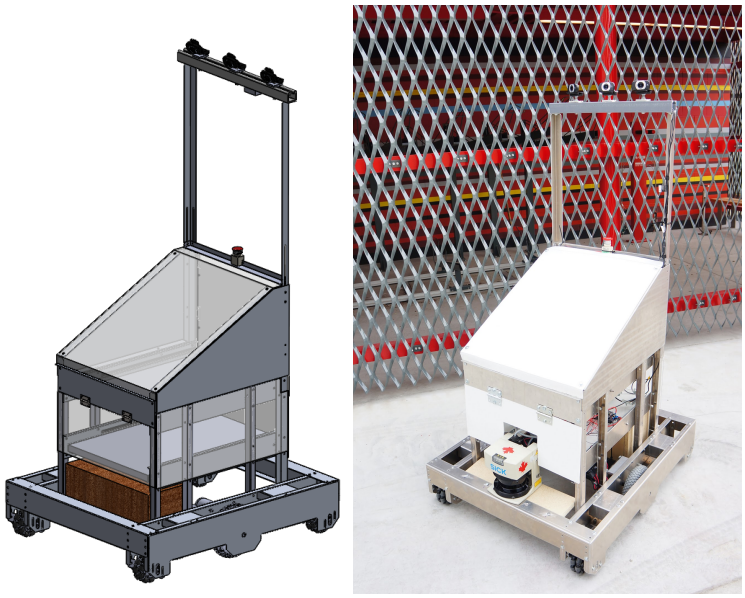

Figure 2. Snowflake - From Design to Reality

MATERIAL SELECTION

The robot is composed of three materials: aluminum 5052 sheet metal, corrugated plastic and acrylic. The chassis of the robot is made from bent aluminum sheet metal. This material was chosen for its machinability, strength and light weight. Certain parts of the robot require weatherproofing and clear acrylic was used to shield the components without obstructing the view. In cases where opaqueness and thermal shielding was desired to protect against direct sunlight, white corrugated plastic was used.

DRIVETRAIN

The drivetrain is composed of two modular sections that are connected via two large brackets. These brackets were carefully designed to handle expected loads, while keeping the overall infrastructure modular. Each drivetrain module consists of two omni-caster wheels and one driven pneumatic wheel.

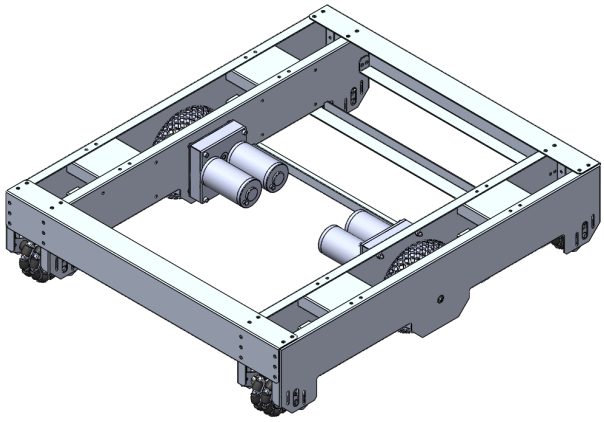

Figure 3. SolidWorks Model of the Drivetrain and Power Train

The pneumatic wheel provides some suspension, which can be adjusted by adjusting the height of the caster wheels. The caster wheel mount was specifically designed to be adjustable and allows for the incorporation of additional suspension in the future.

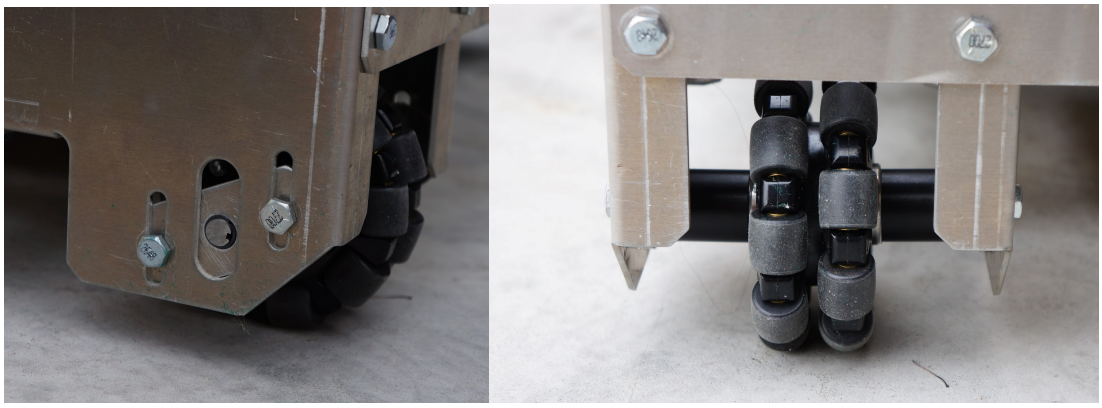

Figure 4. Adjustable Mounting Brackets for the Caster Wheels

POWERTRAIN

The powertrain consists of two gearboxes, each with a gear ratio of 12.75:1. Each gearbox is connected to two brushed 12V DC motors. Calculations were made in order to assess whether the motors and accompanying gearbox would be sufficient to propel the vehicle.

HOUSING

The housing module holds the standard payload and accommodates the robot's electrical system and computer.

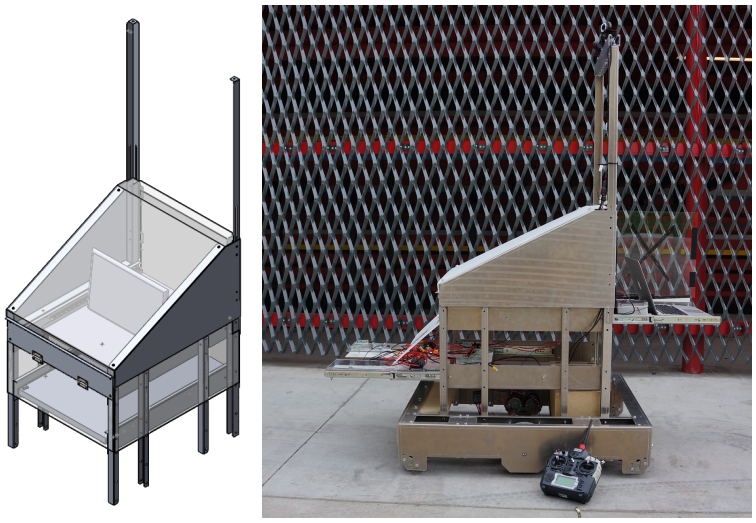

Figure 5. Housing Model and Pop-out Platforms

Since the computer has the highest demand for accessibility, the computer compartment is retractable and was designed with the programmer's seated height in mind.

Figure 6. Comfortable Seating Position for Direct Programming

The electrical compartment is accessible from the opposite end of the robot. Here, our engineers can perform electrical maintenance as needed without disrupting our programmers. To further improve the accessibility to the electrical system, we designed the compartment to be removable. Finally, since most of the human interaction to the robot occurs from the back end, we conveniently placed all of the electrical switches and payload there.

TOWER

The purpose of the tower is to optimize the position of our robot's sensors. This module holds our vision sensing equipment and our GPS antenna. The vision sensors were strategically elevated in order to increase our robot's range of view. We used this necessity to our advantage, as it also allowed us to isolate our GPS antenna from interference. The tower module also has an adjustable height feature to optimize our vision placement.

We chose a strut channel as our platform to mount our sensors. Strut channels allow for easy positioning and secure mounting of sensors such as cameras and antennas. It is made out of fiberglass to eliminate any interference to the antenna that may be caused by nearby metal.

Figure 7. Tower Module with Cameras Mounted

SPEED

To obtain an idealized maximum speed of the robot, a rolling resistance test was conducted and a least squares fit was used to find an approximate linear relation between the pulling force required to maintain the robot at that particular speed.

The rolling resistance and speed of the robot were then converted to angular velocity and torque exerted to the motor shaft using the properties of the drivetrain:

Gear ratio: 12.75 Wheel diameter 0.2m

$$\text{Speed}\frac{km}{h}\times\frac{1000m}{1km}\times\frac{wheel\ cycle}{\pi 0.2m}\times\frac{1h}{60min}\times\frac{12.75\ motor\ cycle}{1\ wheel\ cycle}=336.9\ RPM$$

$$\text{Force to pull robot } N\times\text{wheel radius}\times\frac{0.2m\ \text{wheel diameter}}{2\ \text{wheel radius}}=0.1\ Nm\ \text{exerted on drive train}$$

$$\text{Torque exerted on drive train}\times\frac{(0.5)\ \text{torque to one gear box}}{\text{torque exerted on drive train}}\times\frac{\text{torque from gearbox input shaft}}{12.75\ \text{torque to gear box}}\times\frac{(0.5)\ \text{torque to each motor}}{\text{torque from gearbox input shaft}}=0.01751\ Nm\ \text{exerted to each motor}$$

$$\text{Force to pull robot [N]}=\ 0.001969\ \text{[Nm]exerted to each motor}$$

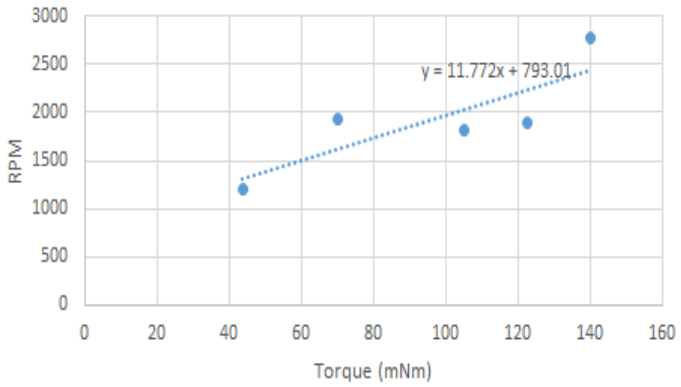

Figure 8. Measured Rolling Resistance Converted to RPM vs. Torque

The obtained rolling resistance in [mNm] is plotted to scale against the manufacturer's motor curve to acquire the ideal case operating speed and load of the robot. Since the manufacturer uses a 12V supply for the motors and our robot uses an 11.1V supply, the motor curve was adjusted appropriately.

The point of interest lies where the rolling resistance of the robot matches the performance curve of our drive motors at the supplied voltage. In addition, two more curves are plotted. These include current draw to each motor and the required angular velocity of the motor at the required speed (1 MPH).

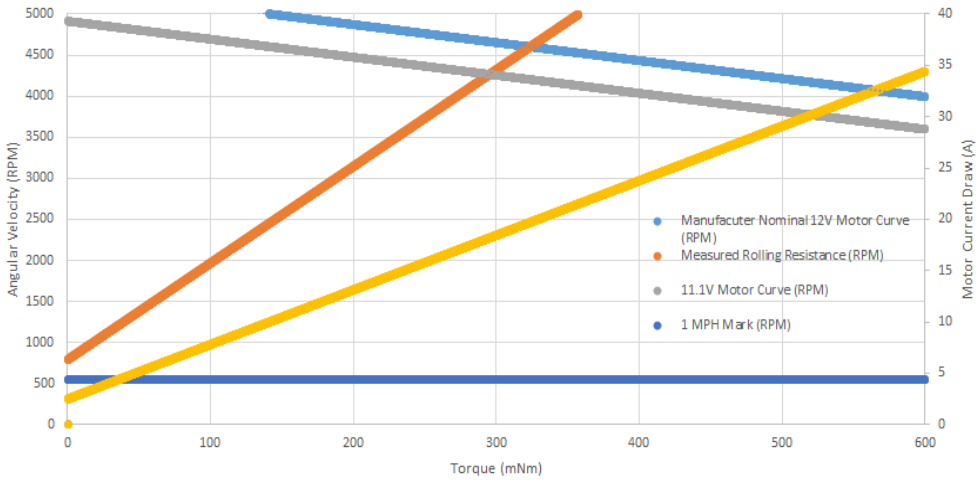

Figure 9. Measured Rolling Resistance Overlaid with Motor Curve

From Figure 8 we can see the absolute maximum speed and current draw that the motors in our drive system can output. The maximum speed of our robot in the idealized case corresponds to a motor angular velocity of 4300 RPM, or 12.8 km/h (7.9 mph). This also corresponds to a current draw of 18 A to each motor. However since our E stop is only capable of carrying 20 A to two switches, we decided to limit the current to each motor at 10 A such that their combined current at most would be within 20 A.

With this design requirement, our robot will operate at a maximum of 7.3 km/h (4.536 mph).

RAMP CLIMBING CAPABILITY

Given that the gradient is to not exceed 15% (~8.5 degrees), and knowing the characteristics of our robot, some simple calculations we performed in order to assess whether or not our robot can traverse through inclines found in the auto-nav course. The following table lists our values used in the calculations.

Parameter	Value
Weight of Vehicle, m	37.6kg
Gear ratio	12.75
Max motor output	0.12Nm
Number of motors	2
Wheel radius	0.1m
Coefficient of Friction, mu	0.35

Table 1. Ramp Climbing Analysis

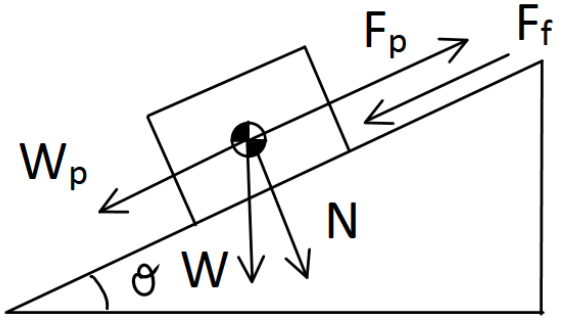

Figure 10. Free body diagram

From our calculations, we have a pushing force of 60N, and a static weight of 55N along the incline. This means that our vehicle is capable of keeping itself static on an 8.5 degree incline. Although it may not be able to propel itself as it cannot overcome static friction, the vehicle will be capable of climbing the incline as long as it approaches it with some momentum.

ELECTRONIC DESIGN

The mandate for this year's electrical division was to produce a professional, reliable and safe design. Fuses, current and voltage monitors are in place to protect from short circuits and over/under-charging the batteries. The wiring is neat and components requiring frequent attention are positioned with a high degree of accessibility.

POWER DISTRIBUTION SYSTEM

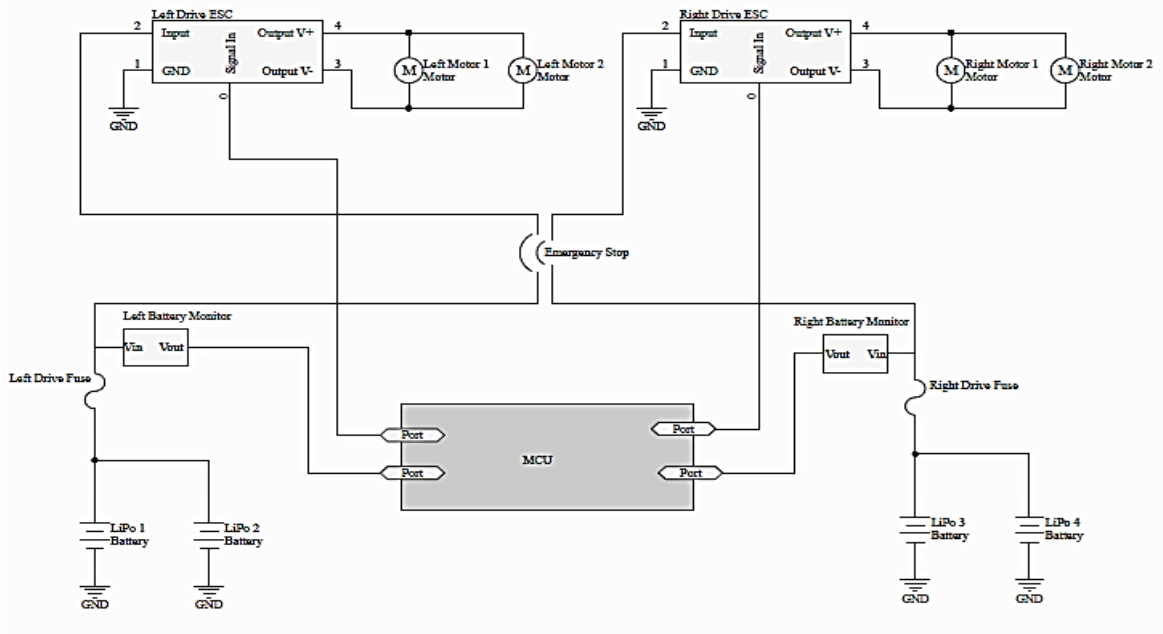

Figure 11. Drive System Layout

The power for the four motors onboard *Snowflake* is provided by four lithium polymer (LiPo) batteries capable of providing 11.1V at 7500mAH. The entire circuit is placed on a platform that is easily detachable from the robot in the event that components need to be repaired or replaced.

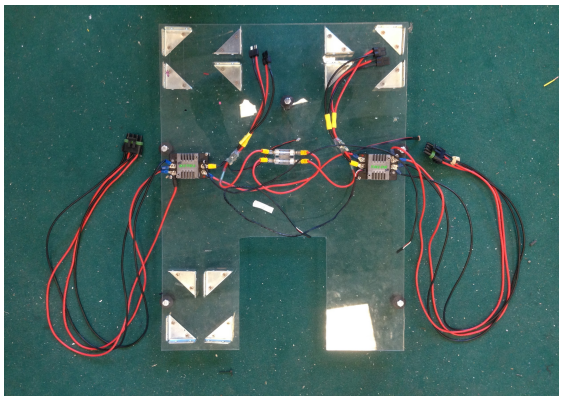

Figure 12. Electrical Platform

Two essentially separate circuits power each of the two motors in one gear box. Each circuit has two LiPo batteries connected in parallel. We used 14 AWG wires which were calculated to be within the range of the maximum current of 20 Amps flowing through the wires.

Before entering the E-Stop, the power is distributed through 20 Amp fuses. The fuses are used to make sure that the battery is protected against short-circuits and excessive currents. We used a fuse block so that burnt out fuses can easily be replaced.

Two additional 11.1V LiPo batteries connected in series power the LIDAR.

Our power system includes an external battery for the laptop to allow for extended use. Our cameras are powered through a USB connection to the USB hub connected to the laptop.

OPERATING LIFE

Our design implements the use of four 7500 mAh 30C LiPo batteries (one for each motor). A LiPo battery has a relatively steady voltage over discharge, and a steep voltage drop past the 80% discharge. To keep our LiPo batteries operational, we recharge them after 80% (6 Ah) discharge.

To simplify calculations, the analysis has been performed on one motor with the assumption that current is equally drawn by all 4 motors.

Battery life at minimal speed:

$$1 \text{ motor} \frac{1 \text{ battery}}{1 \text{ motor}} \times \frac{7.5\ Ah}{1 \text{ battery}} \times 0.8\ discharge = 6\ Ah \text{ charge available}$$

$$\text{Current to move robot at 1 mph} = 2.5\text{A}$$

$$6\ Ah \text{ charge available} \times \frac{1}{2.5A\ discharge} = 2.4\ hours \text{ availble}$$

Battery life at maximum speed:

$$\text{Current to move robot at 7.9 mph} = 10\text{A}$$

$$6\ Ah \text{ charge available} \times \frac{1}{10A\ discharge} = 0.5\ hours \text{ available}$$

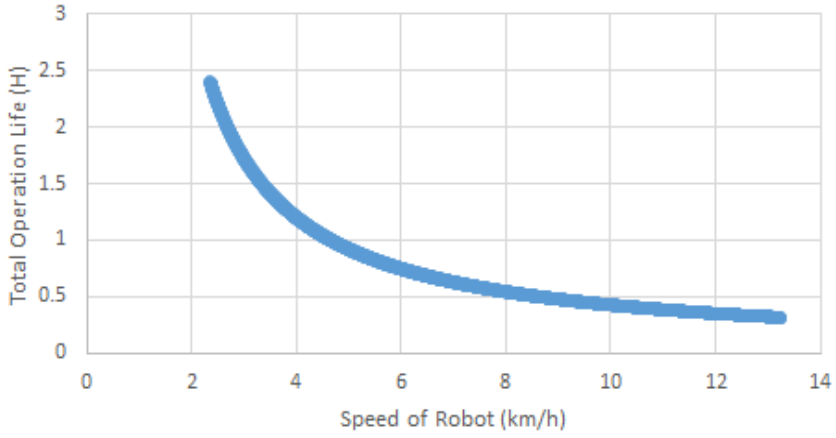

Figure 13: Estimated Battery Lifetime at Different Speeds

EMERGENCY STOP SYSTEM

Our emergency stop system on *Snowflake* is implemented by using a 1NO+1NC emergency push button located at the center of the robot's width and at a height of approximately 3.5ft from the ground. Our emergency stop switch is manual and does not involve any software to stop the vehicle.

FIRMWARE

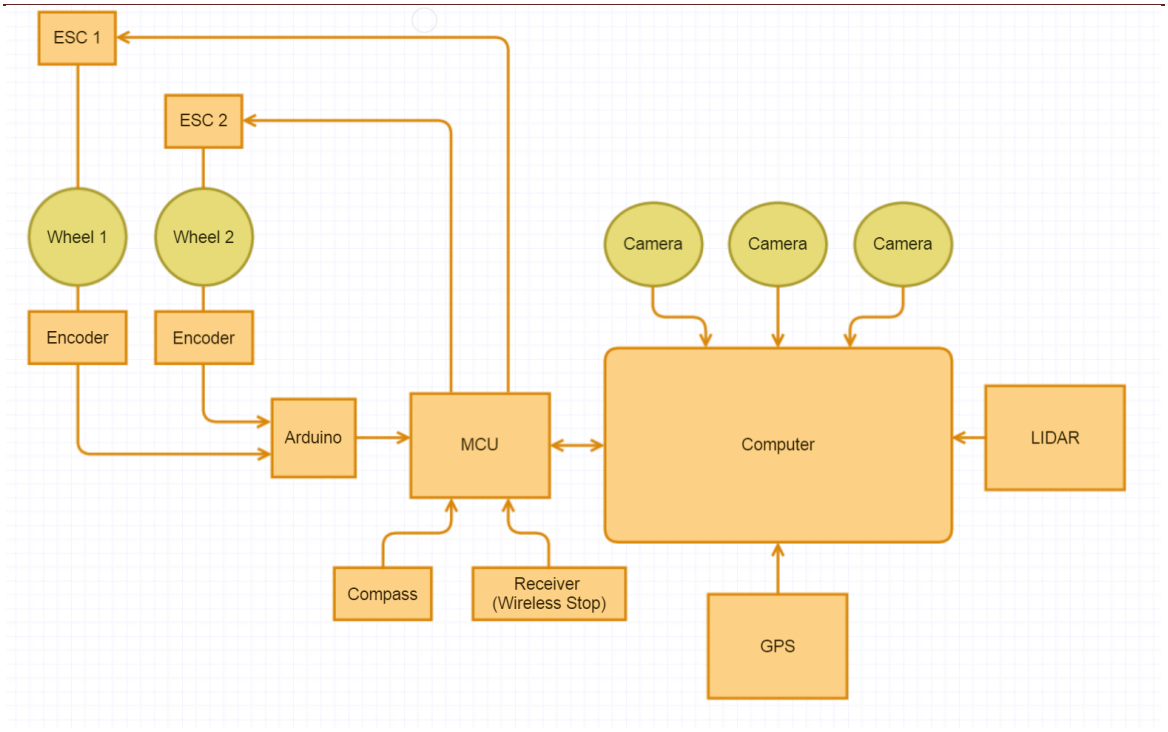

Figure 14. Integration of Sensors with Electrical System

An APM 2.6 microcontroller is used for controlling and communicating with the electrical systems. Ardupilot, an autopilot based on Arduino was used to write the firmware controlling the vehicle's motion.

The vehicle has three main states: autonomous, remote controlled, and stopped, which can changed through the radio controller. When it is set to autonomous mode, serial communication is used to send a twist signal from the laptop to the microcontroller. Under remote control, a wireless communication signal is sent from the radio controller to the microcontroller's receiver.

The signals are then processed by the microcontroller using functions from the hardware abstraction library (HAL). After processing, the microcontroller sends Pulse Width Modulation (PWM) signals to the Electronic Speed Controls (ESCs) which control the two wheels independently, thus moving the robot in its desired direction.

Encoders are used to calculate the speed of the robot. Due to the high resolution of our encoders, the encoder output is first processed by an Arduino Uno, and then using I2C protocol, the position is sent to the microcontroller. In addition, the microcontroller also receives information from the compass, battery monitor, and wireless stop.

The compass and encoder count are combined to calculate the velocity of the vehicle, which is then sent back to the computer to be processed as feedback.

As a safety precaution, the two wheels will stop if the battery monitor detects the battery voltage to be below 9V, if a current above 19.5A is drawn, or if the wireless stop is activated.

LED ALERT SYSTEM

Green LED strips are mounted on the front, back, and sides of the tower to ensure that the safety light can be easily viewed from all directions. A separate circuit utilizing discrete logic drives the LED's operation modes.

SOFTWARE STRATEGY

The software has been developed in C++, using Robotic Operating System (ROS), Open Computer Vision (OpenCV) and Open Simultaneous Localization and Mapping Libraries (OpenSLAM) libraries.

The software system receives three fundamental inputs: GPS information from the GPS, LIDAR scans from the LIDAR, and video streams from the cameras. The data from these inputs is distributed via ROS messages to the system, which processes and updates the speed and direction of the robot.

This year UBC Snowbots has implemented two software strategies: an advanced mode and a basic mode. The advanced mode incorporates sophisticated mapping and path finding algorithms, whereas the basic mode utilizes much simpler algorithms for navigation.

BASIC MODE

In basic mode, velocity vector and a priority are created by each processing node. The priorities are then compared to determine a final velocity vector for the robot.

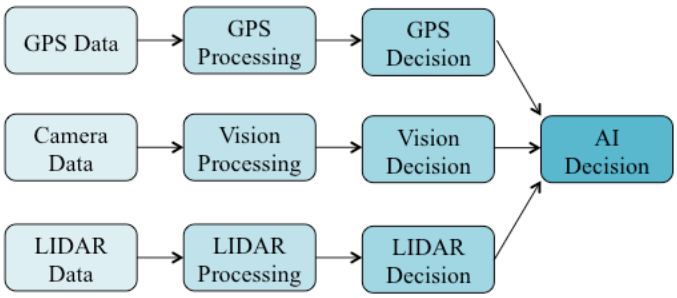

Figure 15. Basic Mode Block Diagram

ADVANCED MODE

The primary strategy (advanced mode) combines the data from all inputs to create a two dimensional map of the competition space. A path finding algorithm is then applied to the map to determine the velocity vector of the robot.

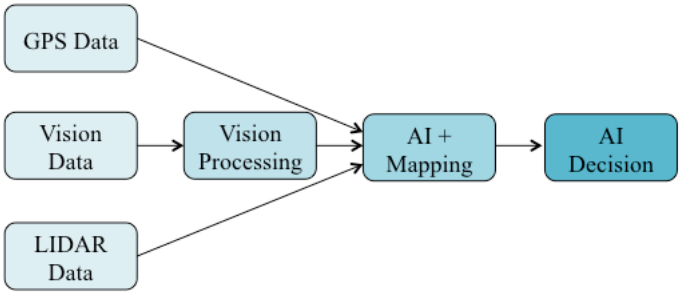

Figure 16. Advanced Mode Block Diagram

VISION SYSTEM

The vision system of the robot uses three camera inputs which from which the shape of the path painted on the field is constructed. This shape is then transformed into a bird's eye view of the field for use in the path-finding and mapping algorithms.

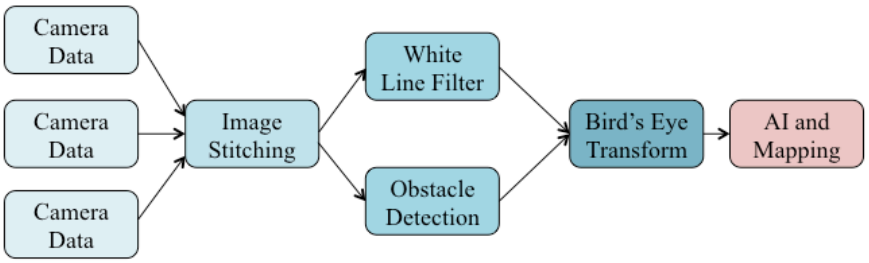

Figure 17. Advanced Mode Vision System

CAMERA

Three Logitech C615 HD Webcams are being used to collect data for the vision system. These cameras were chosen due to their economical price and 74° field of view. The cameras are placed with their fields of view overlapping by 10° which gives us a resultant field of view of 182°. The images are merged using an image stitching algorithm.

Figure 18. Mounted Cameras

IMAGE STITCHING

The image stitching algorithm receives images from the three cameras as input. The images are then stitched together using the "Stitcher" function defined within OpenCV. This function utilizes the overlap between adjacent images to create a single image. A delay of 5 ms is added to the function, which produces an output of 200 fps.

FILTERING

The goal of the filtering process is to extract the white lines painted on the grass from the surrounding environment. This is achieved by first applying a Gaussian blur to the image to remove high frequency noise. The image is then transformed into HSV colour space and separated into Hue, Saturation and Value channels. A threshold is then applied to the Hue channel to extract the white from the background.

To mask the white appearing on obstacles such as cones another filter is applied in parallel to the image. The orange from the cones is filtered and a mask is applied around the region of interest.

The two binary images are then combined using a logical AND to remove the interference from white lines appearing on obstacles.

Figure 19. Extraction of White Lanes from Background

BIRD'S EYE VIEW PROJECTION

For the bird's eye view transformation, the input video is converted into a bird's eye view so that AI can map the vision information. The program takes the four corners of the plane of the ground in the input video and transforms them to the corners of the output video.

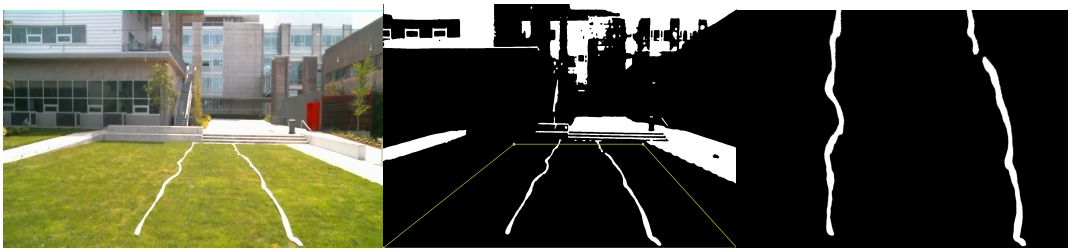

Figure 20. Bird's Eye View Transformation of Lanes in a Practice Area

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 robot and has a scanning angle of 180 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 do the following:

- Publish the data to the Mapping node where the LIDAR data can be combined and processed along with other data, or
- Process the LIDAR data and makes decisions based on object distance and the robot state, then publish the decisions in form of a Twist Message directly to the robot's driver node.

GPS

The Piksi RTK Kit by Swift-Navigation is used to provide GPS information. The Piksi GPS Module includes two Piksi OEM boards, two GPS antennas, and 915 MHz two robotics radios to support RTK (Real Time Kinematics) functionality. This GPS was chosen because it can have a positioning accuracy of less than 10cm. The Piksi GPS development boards included within the kit consist of a GPS receiver and an on-board microcontroller that supports both USB and UART for more flexible design

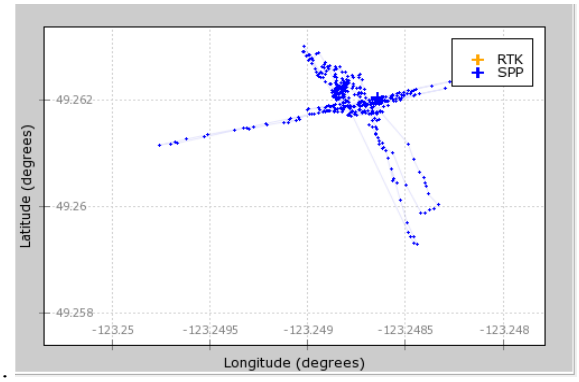

Figure 21. GPS Outputting Single Point Position Solutions While Walking Around an Intersection

The sb_gps ROS node provides hardware data acquisition with the help of GNSS library and SBP library by Swift-Navigation. After acquiring data, the node also performs the calculations providing the system with direction and angle from the current location to the next way point.

NETWORKING SYSTEM

Networking was developed using the JAUS++ framework (from ACTIVE-IST). 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.

These commands are as follows:

- Local Waypoint Driver (ability to travel toward a single GPS waypoint)
- Waypoint List Driver (ability to travel along a list of local GPS waypoints)
- Velocity State Sensor: axis-aligned linear velocity (forward only), yaw rate
- Local Pose Sensor (reports local pose relative to a global pose)
- Local Pose contains position in Lat/Lon format, as well as Yaw.

DESCRIPTION OF MAPPING TECHNIQUE

The main purpose of our mapping system is to generate a local environment map that is used to update an internal global environment map. The local map generation will be implemented in our custom ROS mapping node. The mapping node will then pass the processed local map to a path-finding node, which will map the local map onto the global map, and execute the algorithm, which uses the global map. This entire process is looped through until the ROS system is aborted.

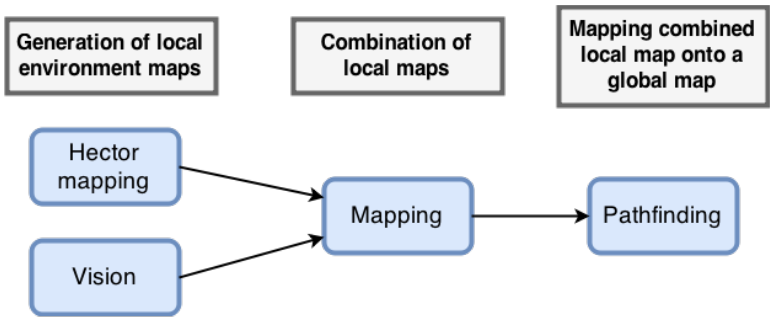

Figure 22. An Overview of the Mapping System: The Blue Nodes Represent ROS Nodes.

HECTOR MAPPING

Our mapping system uses the *hector_mapping* ROS node, which utilizes the *SLAM (Simultaneous Localization and Mapping)* method. This node primarily relies on data from the LIDAR instead of odometry, as the LIDAR's fast update rate can provide more accurate and precise results. The node publishes the gathered data in the form of an *occupancy grid*, along with useful localization data such as position and orientation. In addition, the *hector_mapping* node provides us with the ability to customize the map resolution and update thresholds, making it a versatile system.

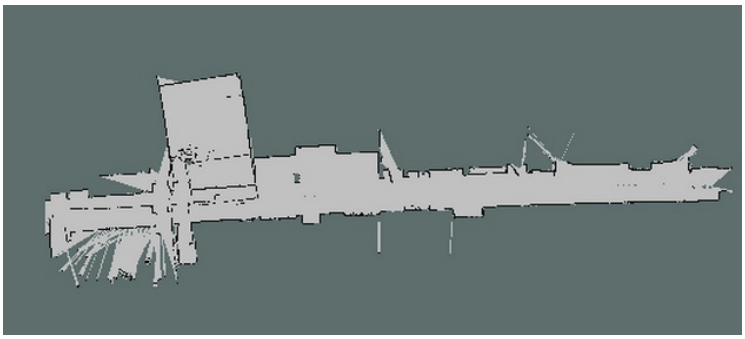

Figure 23. Sample Map of a Hallway in the UBC Civil and Mechanical Engineering Building Created Using Hector Mapping

LOCAL MAP GENERATION

The main local map generation is implemented in a custom ROS mapping node that processes and combines local environment data from the cameras and LIDAR. While the cameras' images are processed by our ROS vision node, the LIDAR scans are processed by an open-source ROS node called *hector_mapping*. The mapping node combines the occupancy probability values from the two local map grids into a single local map that is sent to the path-finding node, where our internal global environment map is initialized and maintained.

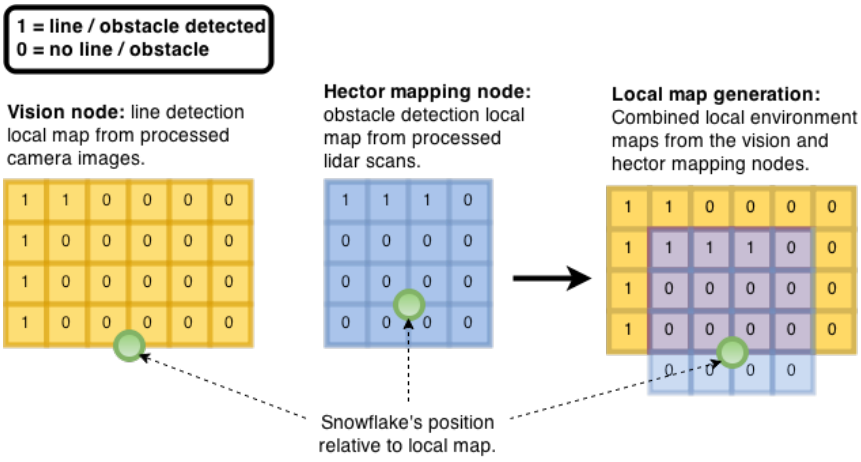

Figure 24. A Summary of the Local Map Generation

LOCAL TO GLOBAL MAP

The path-finding algorithm requires a global map, regardless of how incomplete the map may be. The algorithm also needs to know which area of the global map is being updated to avoid recalculating the path over the entire global map. During every ROS loop iteration, the local map received from the mapping node is mapped onto a global map that is initialized in the path-finding node. This operation is performed by transforming the local map matrix into the orientation and position on the global map with the assistance from the GPS compass as well as relative position changes based on Hector Mapping data. Upon

consolidating this information, the map is then ready for the path-finding algorithm to update its current path.

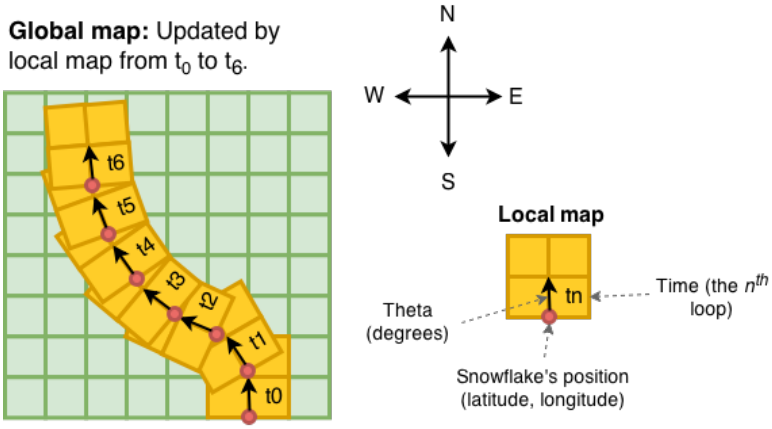

Figure 25. The Two Local Maps are Combined to Generate a Single Local Environment Map, Which will be Sent to the Path-finding Node to Update the Global Environment Map.

SYSTEMS INTEGRATION

The following section describes the algorithms in place to lane following, waypoint navigation and collision avoidance. The basic is used in the event that the advanced mode fails and thus allowing the robot to have a back-up software system.

BASIC MODE

BASIC LANE FOLLOWING ALGORITHM

The basic lane following algorithm extracts the slope of the white lines close to the robot using a least squares fit. The slopes of the lines are then used to extrapolate the lines to their intersection point. The deviation of their intersection point from the centre of the image is then used to update the robot's velocity.

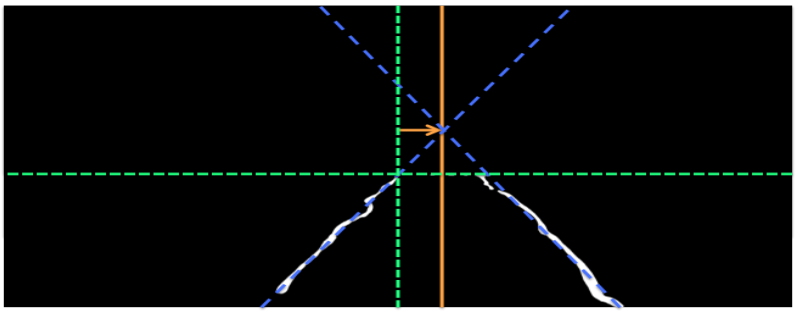

Figure 26. Lane Following Algorithm: Robot Direction is Adjusted so that Centre of the Image (Green) is Aligned with the Centre of the Path (Orange) at the Horizon (Green)

BASIC GPS WAYPOINT NAVIGATION ALGORITHM

The algorithm used for calculations mainly surrounds the Haversine formula shown below that calculates the distance across a sphere provided the longitudinal and latitudinal coordinates along with the radius of the sphere we are traversing.

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

BASIC COLLISION AVOIDANCE ALGORITHM

The LIDAR node utilizes the LIDAR to determine the throttle and steering of the robot using force field navigation. For each object that a laser beam detects, a repellent force is assigned. The magnitude of the repellent force is inversely proportional to the distance between the robot and detected obstacle, and its direction is pushing against the robot.

$$F_{rep} = \frac{-1}{distance}$$

This force is used to calculate the x-total and y-total, which is the sum of all of the repelling force vectors acting on the robot (x-total being the magnitude, y-total being the direction). The robot then makes steering and throttle decisions based on these values.

The LIDAR also uses safety zones to determine the throttle value. There are three zones: green zone for when the robot is a safe distance away from any obstacles, orange zone when a robot is approaching an obstacle, and red zone in which an obstacle is in the robot's immediate vicinity. The robot responds to the three zones as: go straight, slow down and turn, or stop then turn, respectively. This method has a simple logic and ensures a fast response time for the robot.

BASIC COMMANDER

The control unit for the robot in basic mode is a finite state machine designed for the basic course at IGVC. The commander keeps track of the number of waypoints passed, indicating whether the robot is in no man's land or on the path and uses a set of priorities to create a weighted sum of the velocities calculated by the vision, LIDAR and GPS nodes.

ADVANCED MODE

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 only 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 robot traverses the environment and discovers more obstacles. For this reason we have chosen to implement the D* Lite algorithm.

8.6.2 COMMAND FOR ROBOT MOVEMENT

Once D* Lite finds the current optimal path to the goal waypoint, a new linear and angular velocity will be sent to *Snowflake* to direct it. To determine *Snowflake*'s 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.

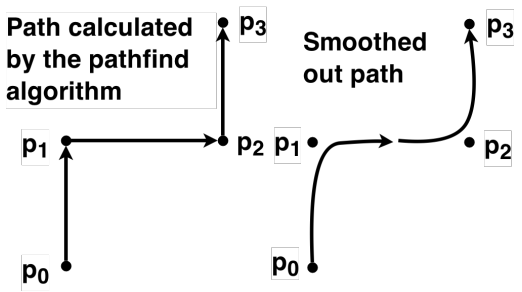

Figure 27. When Reaching a Point on the Path with a Sharp Direction Change, Snowflake's Linear Velocity Will Decrease as its Angular Velocity Increases to Accommodate for a Smoother Movement.

ATTENTION GIVEN TO SAFETY, RELIABILITY, DURABILITY AND FAILURE MODES

FAILURE POINTS IDENTIFICATION AND RESOLUTION

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

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.

DRIVE WHEEL CONTACT

Since our robot maneuvers using 6 wheels and does not incorporate a spring damping mechanism, there is a possibility that it may bottom out with the driven wheels off contact from the ground. To prevent this, the 4 caster wheels can be adjusted beforehand, and the driven wheels can be deflated to provide some suspension.

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 achieved by constantly polling the rotary encoder when power is supplied to the motors.

BATTERY MONITORING

The voltage of the LiPo batteries is monitored by the firmware using an inline LiPo battery monitor. This monitor automatically stops the robot if the voltage in any of the battery packs goes below a threshold voltage of 3.3V.

CHALLENGING OBSTACLES

In the case of a challenging obstacle (dead end or island) *Snowflake* is programmed to slowly back up and try again. Upon a third failed attempt the robot will switch to basic mode, and if that fails *Snowflake* will slowly start turning to find a different path.

COST ANALYSIS

The estimated overall cost of *Snowflake* was as follows.

	Retail Price	Cost to Team
Electronic Speed Controllers	$160.00	$160.00
Motor Encoders	$60.00	$60.00
APM Microcontroller	$160.00	$160.00
Logitech Webcams	$200.00	$200.00
Sick LIDAR	$4,500.00	-
Piksi GPS Moldule	$1,235.00	$1,000.00
ASUS Laptop	$2,000.00	-
USB Hub	$50.00	$50.00
Laptop Power Pack	$120.00	$120.00
Electrical Hardware	$100.00	$100.00
Sheet Metal and Machining	$1,161.00	$461.00
Brushed DC Motors	$112.00	$112.00
Gear boxes	$132.00	$132.00
Acrylic & Plastic	$120.00	$120.00
Wheels	$258.00	$258.00
Mechanical Hardware	$700.00	$700.00
	Total Cost	$11,068.00
	Total Cost to Team	$3,633.00

CONCLUSION

Over 50 members of UBC Snowbots worked on the design and construction of *Snowflake* this year. Our design has greatly improved over last year's entry with a brand new mechanical and electrical design, and a more sophisticated software strategy. The mechanical design greatly emphasizes adaptability, accessibility and innovation through its modular design and unique features. The electrical design emphasizes both reliability and safety. It uses feedback systems to monitor voltages, currents, and position to minimize the damage that could be caused by over-discharged batteries and stalled motors. Safety mechanisms such as the E-stop and fuses are also in place in case of a failure. Both the mechanical and electrical systems have been extensively tested and are performing very well. This year's software strategy is sophisticated and customizable, allowing the robot to utilize a number of different strategies to complete the Auto-Nav challenge. New sensors such as a LIDAR with a 30m range, a GPS with 20 cm accuracy and a camera system with a 180 degree field of field have also greatly improved our performance. The team is looking forward to bringing *Snowflake* to this year's IGVC and excited to see the results of the competition.