Vehicle Design Report: UBC Snowbots Avalanche

University of British Columbia

Navid Fattahi, Jarek Ignas-Menzies, Jannicke Pearkes, Arjun Sethi, Jason Raymundo, Edward Li, Andres Rama, Tabitha Lee, Jennifer Ling, Kirk Wong, Grace Hu

Faculty Advisor: Dr. John Meech, cerm3dir@mining.ubc.ca

ABSTRACT

UBC Snowbots is a completely student-run team involved in the design and fabrication of autonomous ground vehicles at the University of British Columbia. This is not part of any technical course and academic credits are not awarded, but the team still sees strong participation from students from various departments of engineering and computer science. Snowbots was founded in 2006 and, since then, has performed very well each year at its main competing event, the International Autonomous Robot Racing Competition. This year, Snowbots is going a step further to compete in the more prestigious and competitive event, IGVC, with its new robot design called *Avalanche*.

INTRODUCTION

Avalanche is a novel vehicle designed and built by UBC Snowbots as the first entry by the University of British Columbia at IGVC. Snowbots prides itself on being innovative, and while we could have continued with well-established vehicle designs used in the past, we decided to experiment with Mecanum wheels on this first entry vehicle. These wheels allow a robot to travel in any direction without requiring any space for a large turning radius. *Avalanche* has a square chassis supported by the four Mecanum wheels, three LIDAR sensors for obstacle detection, and a camera on a tower for computer vision. GPS is also being used for localization and navigation. This report will describe the team organization, the design process, and the mechanical, software and electrical elements of our vehicle. It will also include a cost analysis and then conclude with our aspirations for the competition.

TEAM ORGANIZATION

The Snowbots team consists of UBC students from a variety of engineering programs, computer science, and commerce. The structure of the organization is shown in Figure 1. The team has been divided into two main interactive sub-groups – a mechanical group and a software group. The mechanical division, headed by Mechatronics student, Jarek Menzies, is responsible for the robot hardware such as the chassis, drive chain, and laptop mount. The group also takes care of most of the wiring and firmware on board the vehicle. The software division, led by Electrical Engineering student, Navid Fattahi, is responsible for coding, sensors, and the response testing of the robot. The two team leaders are also responsible for the general organization, managing finances, and coordinating with the sponsor contacts who manage external funding and marketing for Snowbots. The team meets once a week regularly, or as required by the tasks at hand.

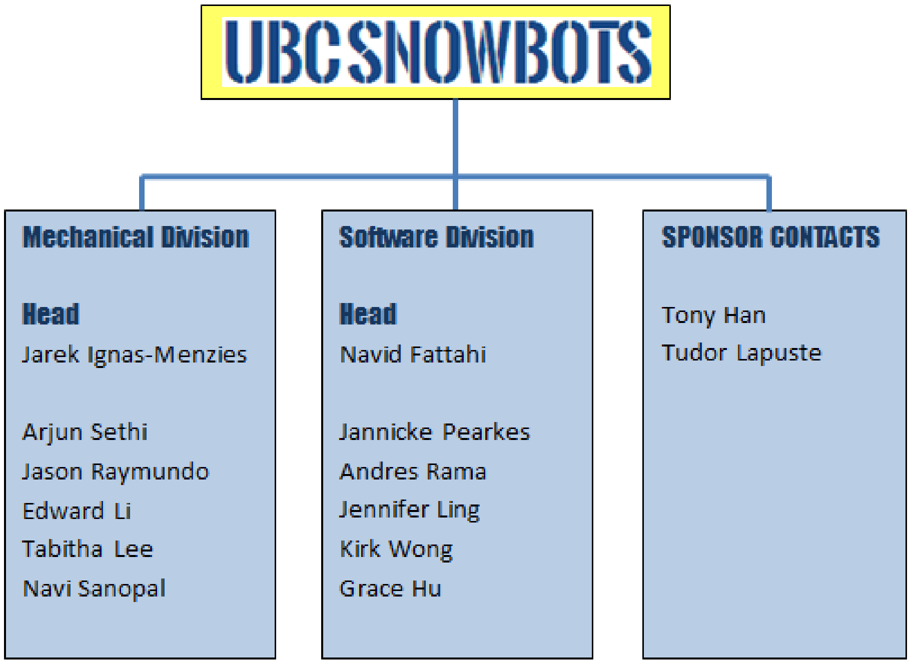


Figure 1. UBC Snowbots Team Organization.

DESIGN PROCESS

The ultimate result of our design process was to address all the objectives presented in the problem statement in the most efficient and effective way. On the technical side, this presented us with two major topics: mechanical and electrical design. The team is split into two major groups that mainly involve individuals with a mechanical or electrical aptitude in order to achieve the major objectives in the competition.

The mechanical and electrical designs were approached as shown in Figure 2.

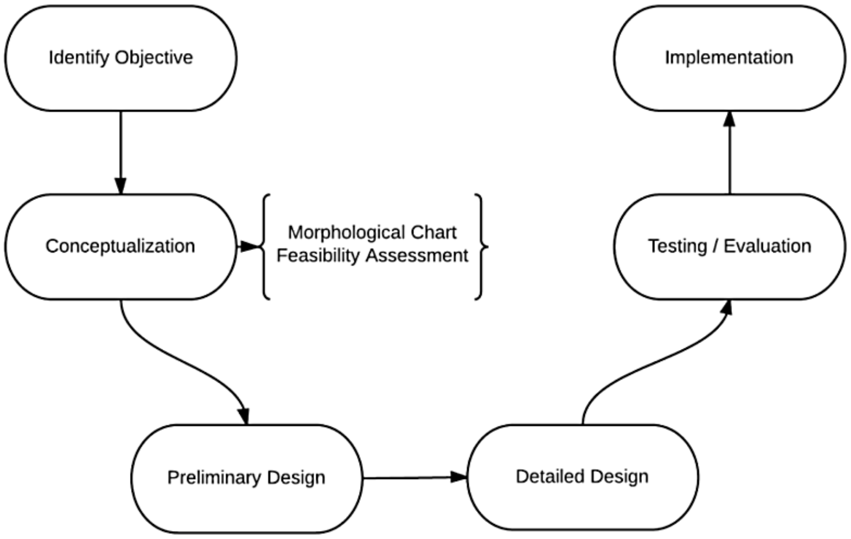


Figure 2. General Design Process.

Although these steps were followed sequentially, an iterative approach allowed us to explore beyond the linear structure shown in Figure 2.

MECHANICAL DESIGN

General Mechanical Design

In terms of mechanical design for the robot, the goal was to obtain a simple, yet effective solution to support the payload while being robust enough to traverse the course of the soccer field. Due to time constraints, we chose to utilize custom-made parts supplied by AndyMark Robot Supplies. This way we could familiarize ourselves with the possibilities of the chassis design without relying too much on our manufacturing skills to perfect the robot's durability and reliability.

In order to ensure compatibility between ordered parts and parts we expected to build ourselves, we modeled all the components in SolidWorks. SolidWorks allowed us to visualize and conceptualize an end-product rather than undergoing a trial and error process when putting the robot together. Modeling in SolidWorks also let us implement new ideas and compare them to previous ideas which reduced the time we spend building, and encouraged us to continuously brainstorm solutions before purchasing and installing hardware. Figure 3 shows a progressive design example of this process.

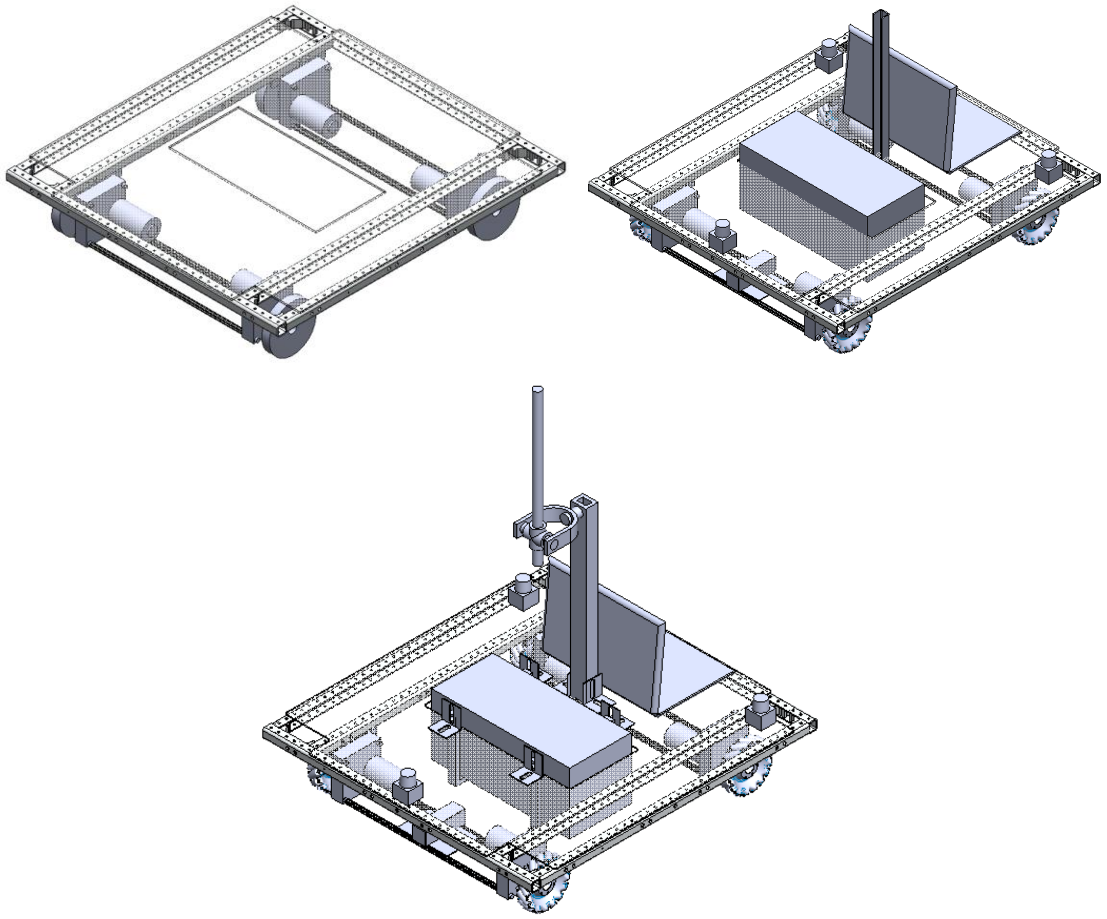


Figure 3. The Progression of the Mechanical Design of the Vehicle.

In following the design process, most of the time was allocated to the three design phases:

- Preliminary Design
- Detailed Design
- Testing/Evaluation

The first two phases were addressed early within the mechanical design process of the robot and covered areas of concern that included maneuverability, durability, and safety. Some solutions that proved to be very rewarding were the implementation of Mecanum wheels and the pre-design of the chassis frame. After these objectives were identified, the team began brainstorming solutions to combine these ideas using the iterative process involving the 3 phases mentioned earlier. Key features of our mechanical design are discussed in further detail below.

Drivetrain

We decided on an innovative approach to this year's competition with the use of AndyMark Mecanum wheels. Directly coupled to an AndyMark Toughbox, we are able to obtain the proper gear configuration required to have a simple and dynamic drivetrain. With such a simple drivetrain, we have a safe and reliable system where we can easily diagnose and fix problems.

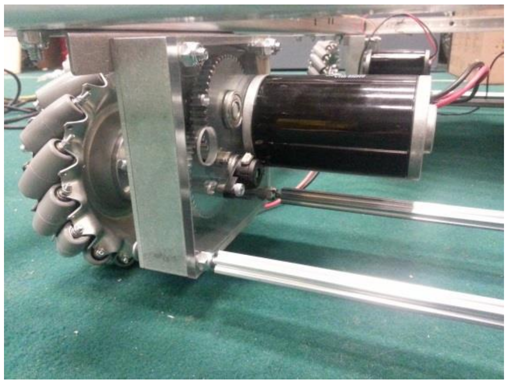


Figure 4. Drivetrain with Mecanum Wheel and Motor

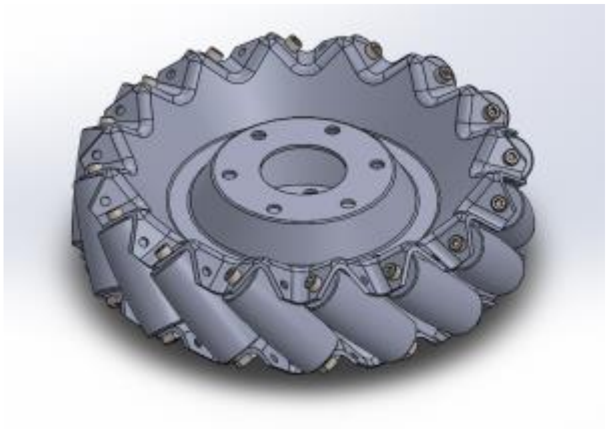


Figure 5. CAD Model of a Mecanum Wheel.

By using the Mecanum wheels, our robot is capable of strafing in order to have a net sideways direction to avoid a close obstacle, rather than going through the ordeal of reversing and advancing forward. This is achieved by rollers on the wheels oriented at 45 degrees from the wheel's drive shaft. Forward and backward movement is achieved by running all four wheels in the same direction. By running the wheels in different directional combinations as illustrated in Figure 6, sideways and rotational movement of the robot is achieved. Furthermore, combinations of all these wheel motions allow the robot to move in any direction.

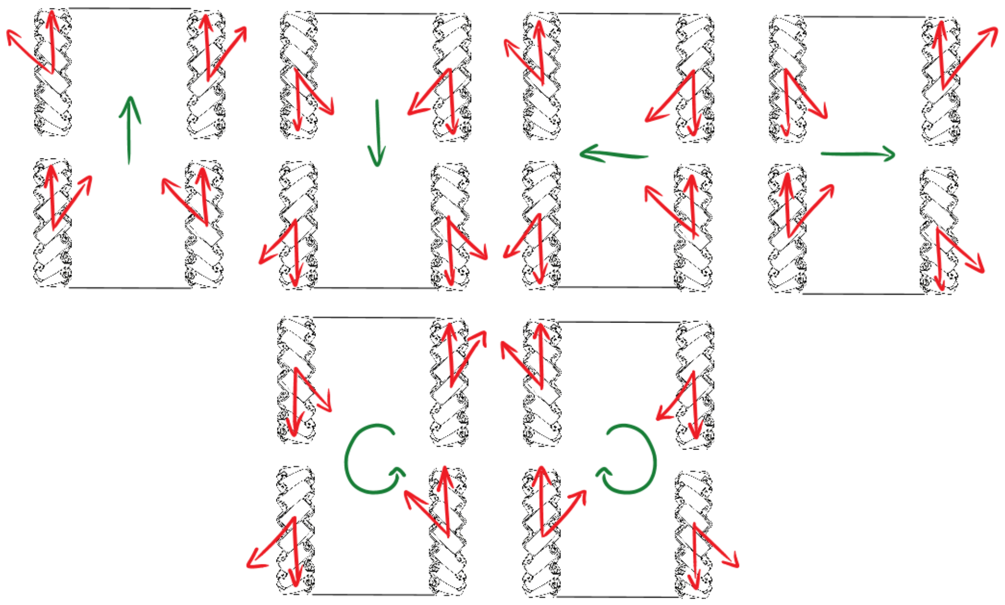


Figure 6. Force Balance in the Mecanum Wheels to Achieve Desired Direction.

The mechanics of the Mecanum wheels allows the robot to be much more agile than using conventional wheels, and simplifies the CPU control for motor actuation. The benefits of the Mecanum wheels remove the requirement for steering systems such as the Ackerman Steering System or a rack and pinion. By reducing the number of parts in the drive train, we minimize any energy losses that might occur between the motor output and the shaft output.

Chassis

With our minds set on using Mecanum wheels, we decided to use a square chassis in order to easily balance the Mecanum wheel force vectors. We chose to use a chassis kit from AndyMark to have the convenience of modularity while sacrificing some innovation from building a homemade chassis. This chassis kit allows us to have a strong foundation to design around while at the same time assuring reliability and durability to meet the needs set out by the competition objectives.

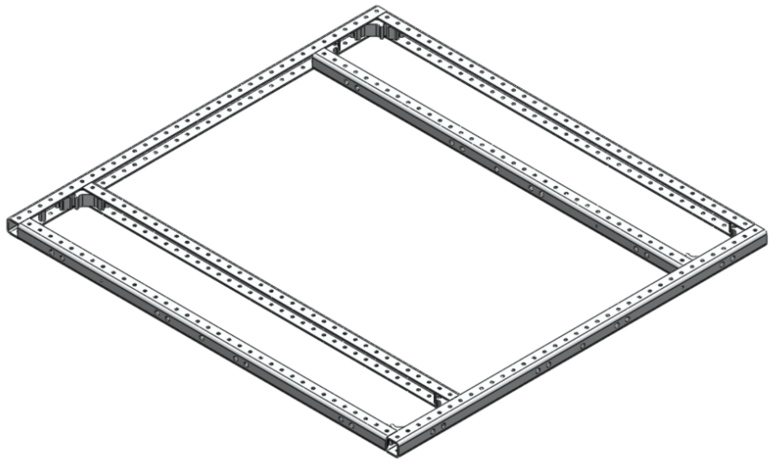


Figure 7. Square Chassis Assembled From AndyMark Products.

Steadicam

The Steadicam is a solution to reduce vibrations experienced by the webcam on the robot. From past experience, we understood that vibrations caused by the soccer field terrain proved to be an issue when attempting to render a quality video for feedback to the computer. There are many solutions to solve this issue; however, the simplest idea we conceptualized without significantly compromising our current design was to directly dampen the vibrations leading to the arm suspending the webcam. This can be done by suspending the webcam on a two-arm lever in which one arm has a counter weight to allow rotation about two axes.

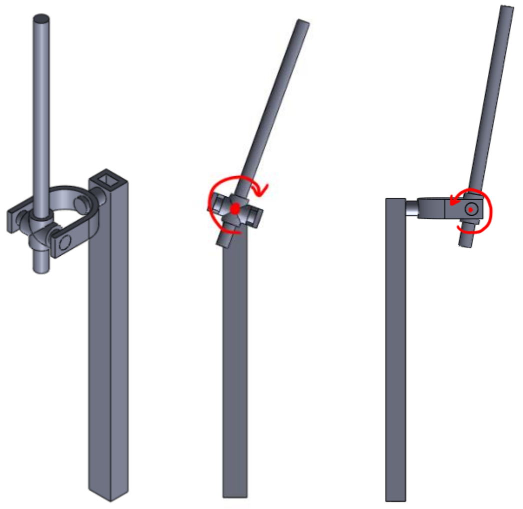


Figure 8. Diagram of the Steadicam showing the Degrees of Freedom

The bottom portion of the second arm acts as a counter weight and dampens sudden movement, or resists movement due to small vibrations about the axes shown above. A downfall to this feature is that it does not address vertical vibrations which would require a much more complicated apparatus and level of manufacturing in which we could not afford to invest time.

A picture of *Avalanche* during assembly is shown below.

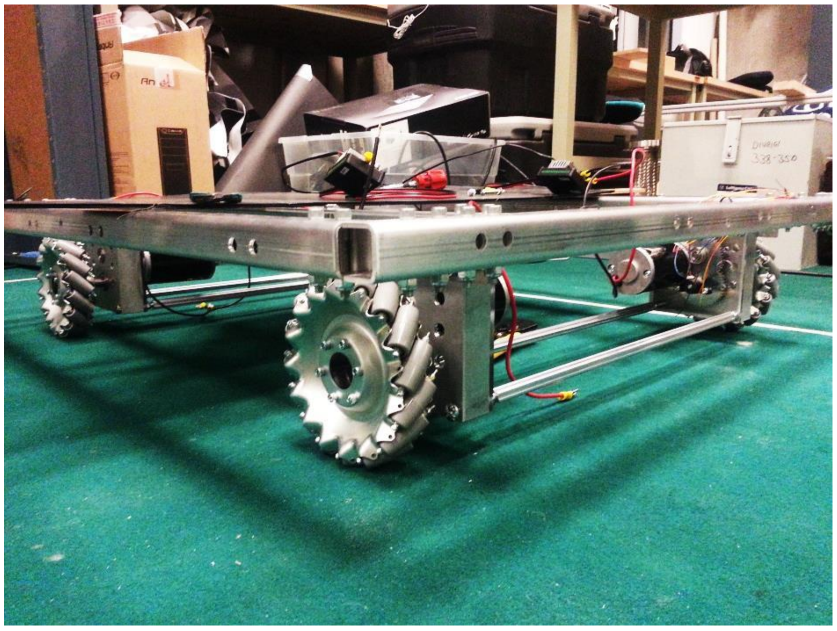


Figure 9. ***Avalanche*** **Assembly**

ELECTRICAL DESIGN

The vehicle is powered through two separate power systems; one system powers the motors, while the other powers all sensors. This design was chosen to prevent noise from the motors interfering with the rails powering the sensitive electronics. The design also allows for easy implementation of the emergency stop requirements.

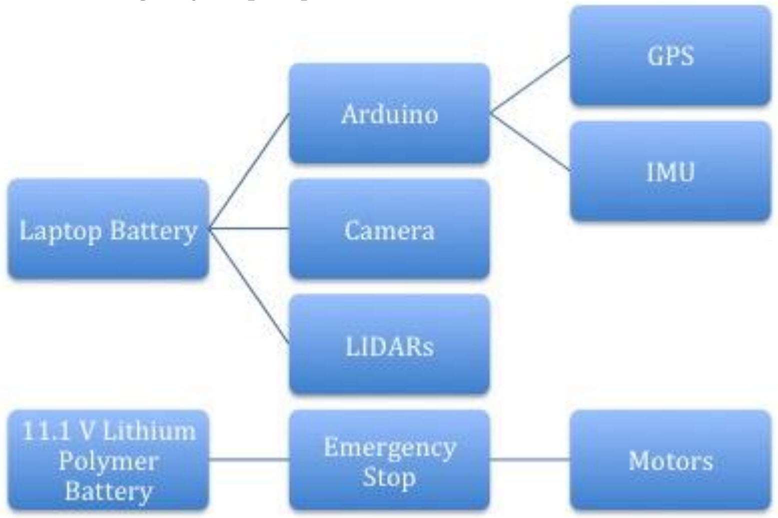


Figure 10: Electrical Layout

Power to the motors is supplied using an 11.1V lithium polymer battery. We chose these batteries as they are extremely light and able to supply a sufficient, steady current.

As the sensors typically require very little power to operate, they are powered through the laptop's internal lithium-ion battery. In addition, the laptop we chose operates using a solid state drive which typically requires less power than traditional hard-drives.

SOFTWARE DESIGN

Software Strategy

The software required to drive *Avalanche* has to accomplish the following goals:

- Observe the surrounding environment and scan for the relevant factors
- Evaluate and analyze the incoming data, producing appropriate suggestions
- Make decisions on the next move, based on the performed analysis

To satisfy the required goals, our custom software architecture is implemented using the Robot Operating System (ROS). ROS is an open-source framework that manages communications between different parts of the software allowing for a high level of modularity. Each software part is referred to as a node in this framework. Nodes can talk to each other via messages on different topics. A standard node called *master* manages the communication between all the nodes.

Software Architecture and Systems Integration

A custom-designed high-level software architecture has been designed and adopted by UBC Snowbots for ease of development, expansion, and to increase modularity and efficiency. At the level closest to the hardware layer, a *driver* node is responsible for sending and receiving information to and from the microcontroller. The data coming from the microcontroller (as well as the serial buses) is passed to the *data analysis* nodes; these nodes receive the sensory inputs and process the information. Every analyzer node then sends its processed suggestions to the *commander* node where a final decision will be made based on a set of priority-based criteria. The *commander* node then sends its final decision back to the driver node for execution and motion.

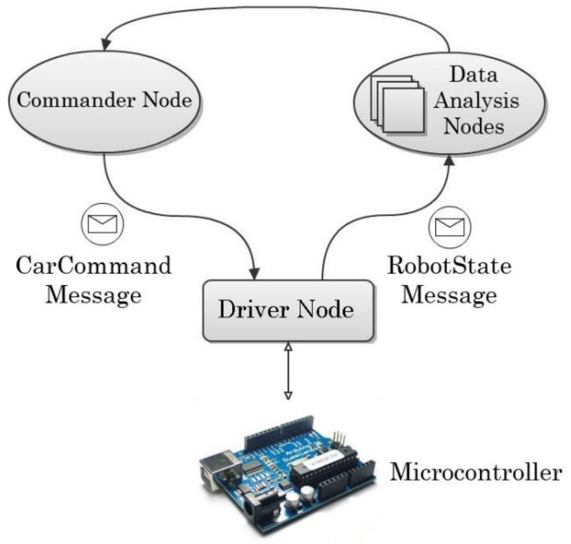


Figure 11. Software Strategy

Mapping Technique

Avalanche dynamically generates its own two-dimensional map of the track using odometer, waypoint, and vision data. Two maps are used: a global map and a local map. The global map is *Avalanche*'s saved copy of the course as it remembers. The local map is entirely dependent on *Avalanche*'s position and what it sees or detects. If *Avalanche* goes on the same path more than once and the local map detects some inconsistencies, it can easily update the global map and path re-planning will take place locally. The both maps are built like a grid, and each grid unit is composed of 4 variables: its location (waypoint), the probability of whether it contains a white line or an obstacle, whether the robot is detected on the unit, and its index on the global map.

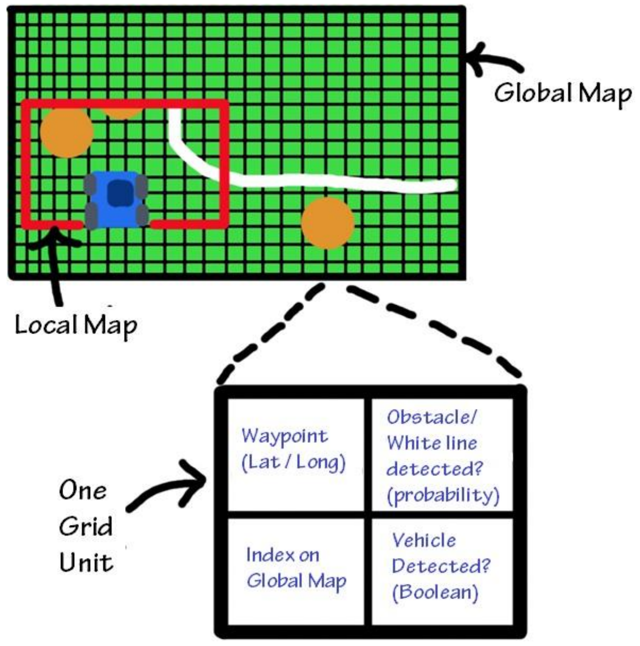
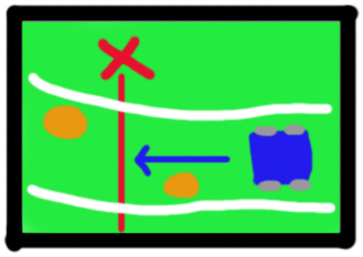


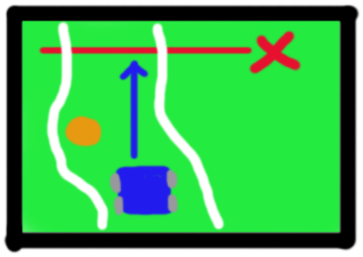
Figure 12. ***Avalanche's*** **Mapping.**

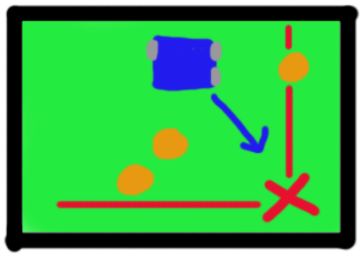
Navigation and Path Finding

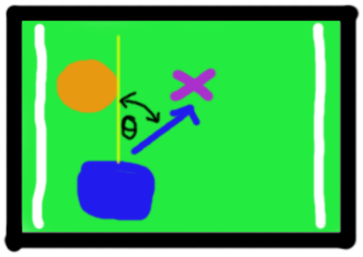
At first, *Avalanche* will have no knowledge of the course and will depend heavily on the local map for navigation. It will first attempt to find a white line to confirm its boundaries. Once it detects it, the angle of the boundaries relative to the latitude will be used to determine the general directions it can go in. If the angle implies a more horizontal movement (east and west), *Avalanche* will be guided in the direction of the goal's longitude value. If the angle implies a more vertical movement (north and south), then it will move towards the goal's latitude value. If the boundaries indicate free movement in all directions, *Avalanche* will simply take the most direct route towards the goal.

When *Avalanche* encounters an obstacle, it will calculate the smallest possible angle relative to its forward position, or normal, it needs to turn at in order to avoid the obstacle. This prevents a 'zigzagging' movement during its constant attempt to move towards the goal. Once it passes the obstacle, it will change its direction parallel to the boundary lines again.


A horizontal track. Since Avalanche cannot move vertically, it will attempt its best to move towards the longitude value of the goal.


A vertical track. Since Avalanche cannot move horizontally, it will attempt is best to move towards the latitude value of the goal.


A track without line boundaries. Avalanche is free to move in any direction it desires, and will attempt to move directly to the goal waypoint.



To avoid obstacles, Avalanche will move towards a point closest to the obstacle without collision.

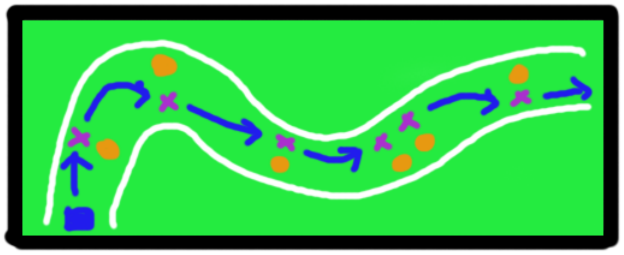

Path planning in an unknown environment. Ideally, there will not be too much 'zig-zagging' while Avalanche is navigating the competition course.

Figure 13. *Avalanche's* Navigation.

The global map, once mapped, will be running a simple pathfinder. Since *Avalanche* knows where the previously detected obstacles are along the paths, it will plot points beside the obstacles where it is sure that it can go directly to without hitting an obstacle. Points will be plotted where there may be drastic changes in direction along the path as well. In general, the pathfinder will plot a path that is as linear as possible in case *Avalanche* requires its use for future reference.

Obstacle Detection

We utilize the LIDAR's sensing capabilities to detect obstacles around our robot. The robot avoids these obstacles using a type of force field navigation called "forceNav". ForceNav works by assigning every obstacle a repellent force where its magnitude is inversely proportional to the distance between the robot and the obstacle. The robot's steering and throttle are controlled and adjusted as a proportional relationship to the sum of all force vectors acting on the robot. Designating the angle between the robot and obstacle with the variable 'θ' (where an angle of zero represents the direction straight ahead) and the magnitude of the force with 'm', a single obstacle's force vector upon the robot is calculated from the following matrix:

$$\begin{bmatrix} -m\cos\theta \\ -m\sin\theta \end{bmatrix}$$

The robot avoids obstacles by allowing the repellent force to push it away from obstacles.

Lane-Following

The lane-following algorithm filters the image obtained by the webcam and then analyzes it to determine the heading direction. The image is first blurred using a Gaussian blur filter to reduce noise. Next, histogram equalization is applied to the image to improve the performance of the lane detection under different lighting conditions. The image is then passed through a binary thresholding filter to isolate the white lines.

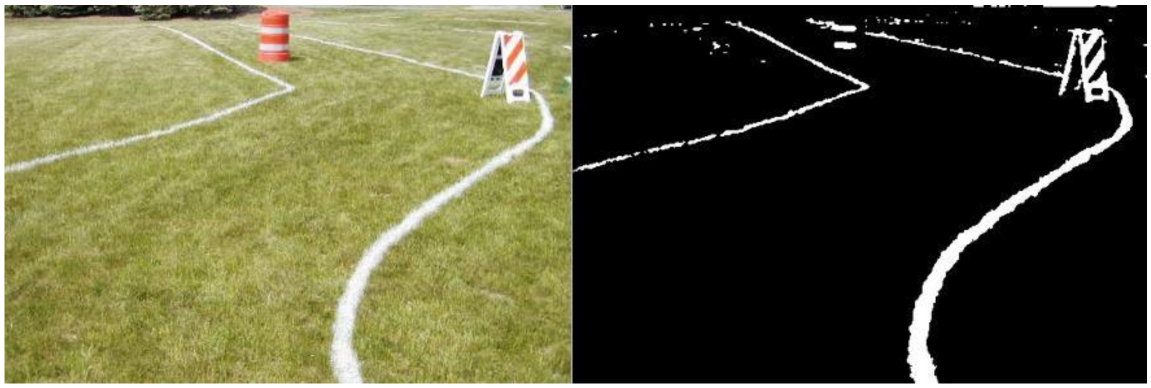


Figure 14. Before and After Filtration

Once isolated, the image is scanned horizontally to detect pairs of white lines in the near field of view. The average midpoint of the lines is used to determine a heading direction. This direction is then combined with input from the LIDAR to obtain a global heading direction.

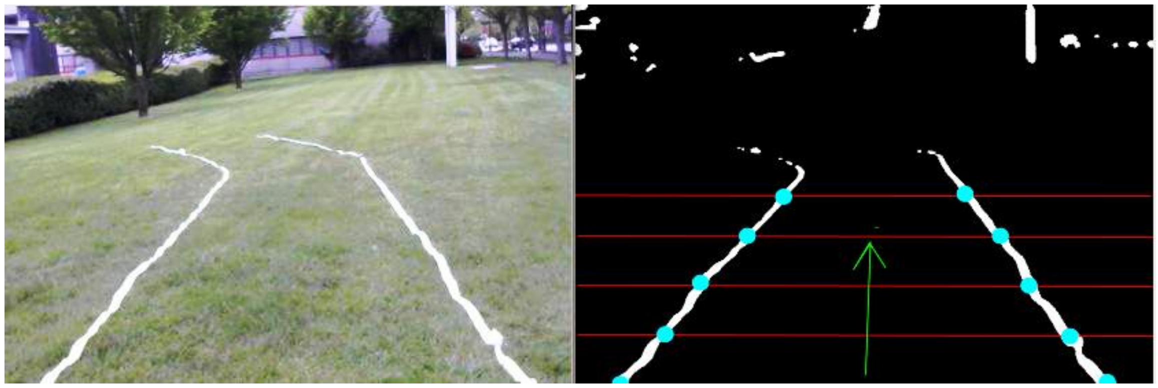
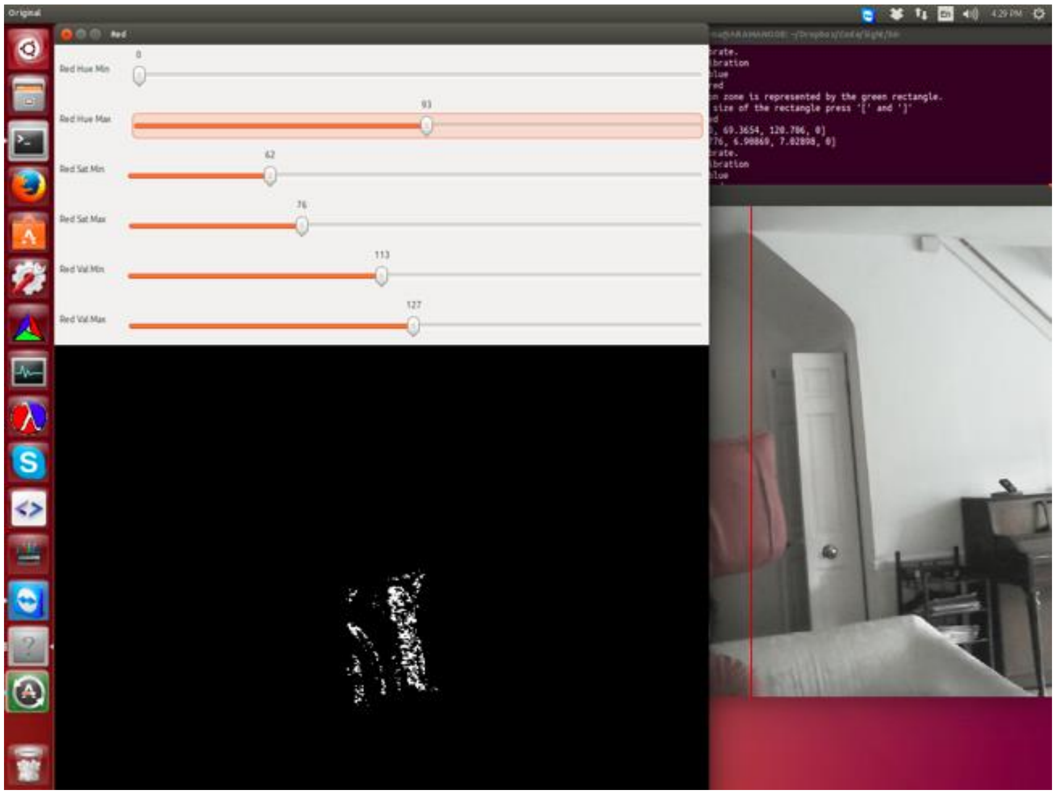


Figure 15. Direction Estimation.

Flag Detection

Avalanche is outfitted with a camera to detect and respond to flags placed along the course. To do this we created a C++ program that uses the OpenCV library to process the images coming from the webcam in real time and provide a vector for the robot based on what it sees. The program achieves this by:

1. Capturing an image from the webcam;
2. Changing the color format from RGB (three channels: Red Green Blue) to HSV (three channels: Hue Value Saturation), as this format is less susceptible to false positives as well as being easier to use;
3. Creating a binary (one channel image) by setting any pixels that aren't red to 0 (black in Figure 14) and setting the others to 1 (white in Figure 16); achieved by filtering out colors that aren't in the user-defined range (by moving the sliders or by using the calibration function on the object being tracked);
4. Calculating the totals of each individual column of pixels in the image;
5. Determining which column has the highest total (and thus where the flag is likely to be). This column is highlighted in the window showing the original image;
6. Producing a 1D vector based on how close the line is to the center;
7. Repeating steps 2 through 5 with blue in place of red;
8. Using both 1D vectors to create one 1D vector to guide the robot left/right.



Red Hue Min 0
Red Hue Max 93
Red Sat Min 62
Red Sat Max 76
Red Val Min 113
Red Val Max 127

```
rate.
ibration
blue
red
n zone is represented by the green rectangle.
 size of the rectangle press '[' and ']'
d
, 69.3654, 120.786, 0]
776, 6.98869, 7.02898, 0]
rate.
ibration
blue
```

Figure 16. A Screenshot of the Program Tracking a Red Pillow

COST ANALYSIS

Avalanche was built from a scratch and is very different from previous vehicles built by UBC Snowbots. Our objective was to design an innovative yet economic vehicle which could be managed well within our budget for the current year. The LIDAR sensors used are from our previous vehicles, and the only costly components for this new vehicle were the Mecanum wheels and the new laptop.

Table 1. Cost Analysis.

Part	Qty	Cost (Canadian $)
LIDAR sensors	2	$3,000
Laptop	1	$1,149
Mecanum wheels and gearboxes	4	$1,108
Microcontroller	1	$240
Chassis Frame	1	$155
Battery and Charger	1	$120
Polycarbonate sheet	1	$114
Steadycam kit and Velcro	-	$80
Encoder	2	$74
GPS Sensor	1	$40
Angle Bracket	6	$6
Encoder Mounting Pad	4	$4
Cross Member	4	$4
Total (Canadian $) =		$6,084

CONCLUSION

This report has discussed the team organization of UBC Snowbots and the mechanical, electrical, software, and economic elements of our robot *Avalanche*. This year, we are aiming to perform well in terms of maneuverability through our innovative use of Mecanum wheels and path planning algorithms. Great care was taken to ensure that our vehicle is light, reliable and on-budget. We intend to perform well in this competition and to improve in the years to come.

ACKNOWLEDGEMENT

UBC Snowbots would like to take this opportunity to thank the people without whom this project would not have been possible. We would first like to thank our faculty supervisor Dr. John Meech from the UBC Norman B Keevil Institute for Mining Engineering for his guidance and for giving us the resources to carry out our operations. We would also like to thank the UBC Professional Activities Fund, Shell Canada, Walter Gage Memorial Fund, and the Association of Professional Engineers and Geoscientists of British Columbia for their monetary support. Pololu, Skookum, and Cellula also backed our team with support for spare parts. Lastly, we would like to thank past and current members of UBC Snowbots who are not here for the competition but have contributed significantly to our endeavors.