2018 Intelligent Ground Vehicle Competition

"Elsa X"

UBC Snowbots

University of British Columbia

Name	Department	Position
Valerian Ratu	Computer Engineering	Captain
Vincent Yuan	Electrical Engineering	Co-Captain/Tech Advisor
Winnie Mui	Engineering Physics	Past Captain
Gareth Ellis	Computer Science	Software Lead
Emma Park	Mechanical Engineering	Mech Lead
Sherry Wang	Mechanical Engineering	Mech Lead
Robyn Castro	Computer Engineering	Software Sub-Lead
Raad Khan	Electrical Engineering	Software Sub-Lead
David Gill	Electrical Engineering	Mech Sub-Lead
Collin Lucas Eng	Applied Science	Mech Member
Arman Zhakypbekov	Applied Science	Mech Member
Geoffrey Hanks	Applied Science	Mech Member
Jacky Sun	Mechanical Engineering	Mech Member
Sharon (Shichen) Fan	Mechanical Engineering	Mech Member
Ivy (Jiayuan) Shi	Mechanical Engineering	Mech Member
Leo Wei	Applied Science	Mech Member
Chris Heathe	Economics and Math	Software Member
William Gu	Computer Engineering	Software Member
Min Gyo Kim	Computer Science	Software Member
Martin Freeman	Mechanical Engineering	Software Member
Marcus Swift	Mechatronics	Software Member
Simon Jinaphant	Computer Engineering	Website Manager
Marinah Zhao	Computer Engineering	Software Member
Remy Zhang	Applied Science	Mech Member

I hereby certify as the faculty advisor that the design and engineering of this vehicle to be entered in the 2018 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

Digitally signed by W Scott Dunbar
Date: 2018.05.16 06:55:30 -07'00'

W Scott Dunbar

Introduction

Elsa X is a vehicle designed and constructed by UBC Snowbots. The vehicle takes its name as part of a previous design from two years ago named Elsa. The goal for this year was to improve on previous designs for ease of manufacturing and shipping, 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 outlines the design decisions made to accomplish this goal and the integrations of the different components of the vehicle.

Team Organization

UBC Snowbots is a multidisciplinary undergraduate team involving students from a variety of engineering departments and the computer science department. The team is divided into three main subteams: the mechanical subteam, the software subteam, and the administrative subteam. The mechanical subteam is responsible for all the fabrication and design of the vehicle as well as designs for the power circuitry. The software subteam is responsible for all sensor firmware and navigational software. The administrative subteam is responsible for outreach, sponsor relations, and managing the team financials. The team meets regularly on Saturdays for at least four hours throughout the school year and additional time is spent throughout the rest of the week on assigned projects. The team structure is displayed in Figure 1.

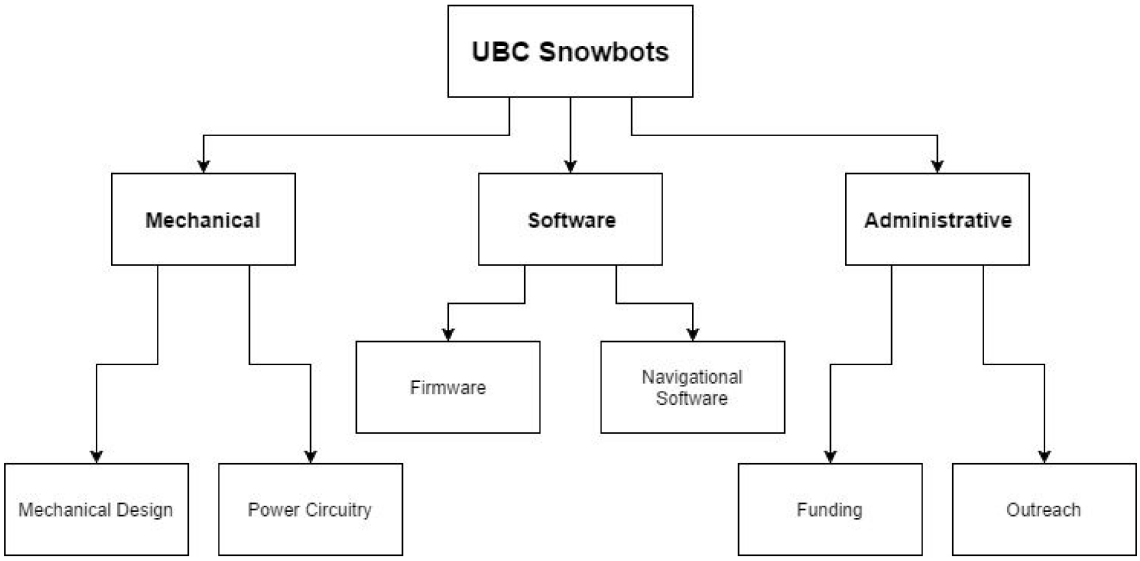


Figure 1 - UBC Snowbots Team Structure

Design Process

The design of this year was focused on mechanical simplicity paired with more sophisticated software design. The design of the robot - both mechanical and software - was done throughout the 2017-2018 school year. We referred to our past years' designs as reference during the build and iterated upon it by assessing pain points encountered and past performances. A large portion of the team is composed of new members and people new to various roles. Thus, a large majority of the design process included teaching and instructions from senior members. The software team and mechanical team maintains parallel development paths with construction completion set at the end of the school year with time for integration during the last month.

MECHANICAL DESIGN

Overview

The goal of this year's design was to create a vehicle that has a simple and compact design to simplify transportation to the competition. Thus we focused on using simple and removable parts with easily machinable materials. The pain point of our previous year's design was the gearbox assembly - which had alignment issues and had a tendency to produce jerky movements. Thus another major focus for this year's design was a more robust and simpler drive train.

Chassis Design

The chassis of Elsa X is a simple square frame, with the core idea of simple manufacturing, assembling, and shipping. The frame is made with aluminium to minimize overall weight but also retain a level of rigidity. Two sliders can open from either side of the robot, as shown in Figure 2, which enables the team members to work on the electrical system and the laptop simultaneously. The bottom of the chassis also holds the payload as well as houses the LIDAR.

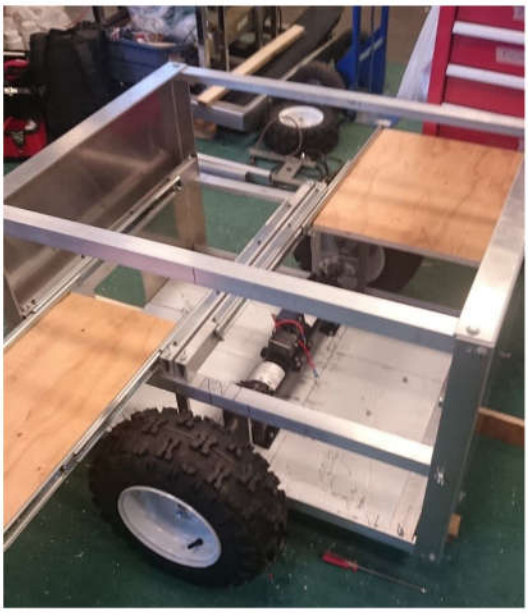


Figure 2 - The chassis skeleton.

The top and sides of the robots is covered with corrugated plastic to further reduce the overall weight. The corrugated plastic is enclosed onto the robot via hinges and velcro, protecting the robot from possible harsh weather and road conditions.

Drivetrain Design

The drivetrain is located at the center of the robot with differential drive. There are two casters at the front and back to provide balance and support. Encoders are attached to the shafts to provide finer input into the software localisation system.

The motors used are a pair of Ironhorse 12V DC motor, which were chosen from torque and speed calculations as well as previous experience using them. The two parameters limiting the robot's performance was maximum and minimum speed given by the competition guideline. The maximum allowed speed was: 5mph. The minimum allowed speed was: 1mph. We estimated the weight of the robot as 50kg and used a cylinder on ramp model to calculate the required torque and speed, estimating the friction coefficient as wood surface. The selected motors are able to provide 60 Nm of torque to provide enough power to reach maximum speed while maintain traction post-ramp.

Tower

A simple metal tower is designed to house the camera and the GPS antenna. The camera is held in place with simple tripod base and a metal plate acts as an extended ground plane for the antenna. The tower is held in place by inserting it into a hole in a metal bar attached to the chassis and secured on either end to minimize swaying and instability.

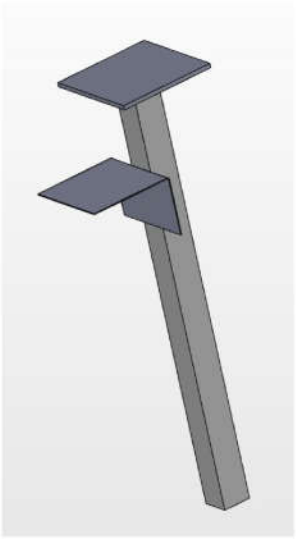


Figure 3 - The tower design

Power System (Electrical Design)

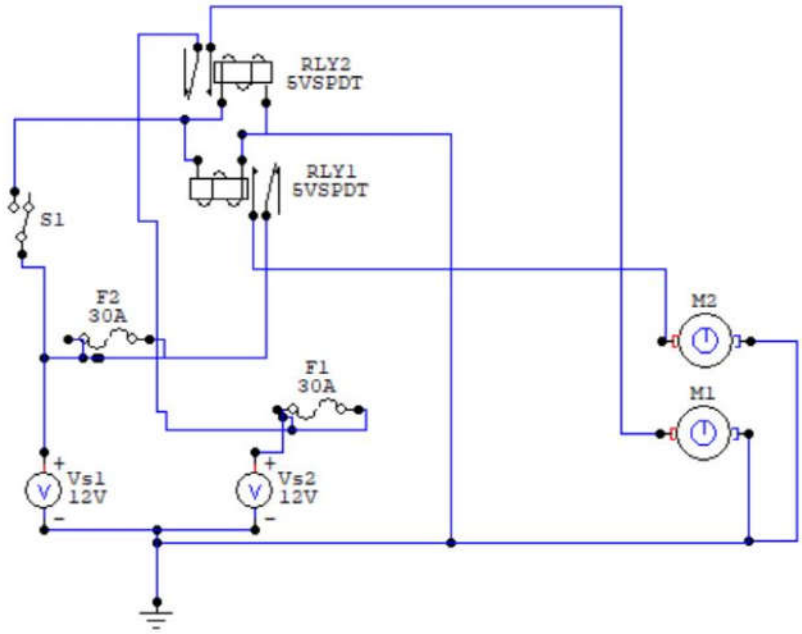


Figure 4 - The power system

The robot is powered through two 12V LiPo batteries. There are fuses in place to reduce the risk of shorts and overload in the circuit.

Emergency Stop

There are both a physical and wireless emergency stops attached to the circuit as per regulation. The wireless emergency stop is controlled via a wireless remote and the physical emergency stop is connected to a push button on the top of the robot. The emergency stops are in series so that either would be able to cut off power from the entire system.

LED System

An LED system is also integrated to show the mode of the robot as per specifications. The lights are designed to turn on when power is available to the motors and blinks during auto-nav procedures.

SOFTWARE DESIGN

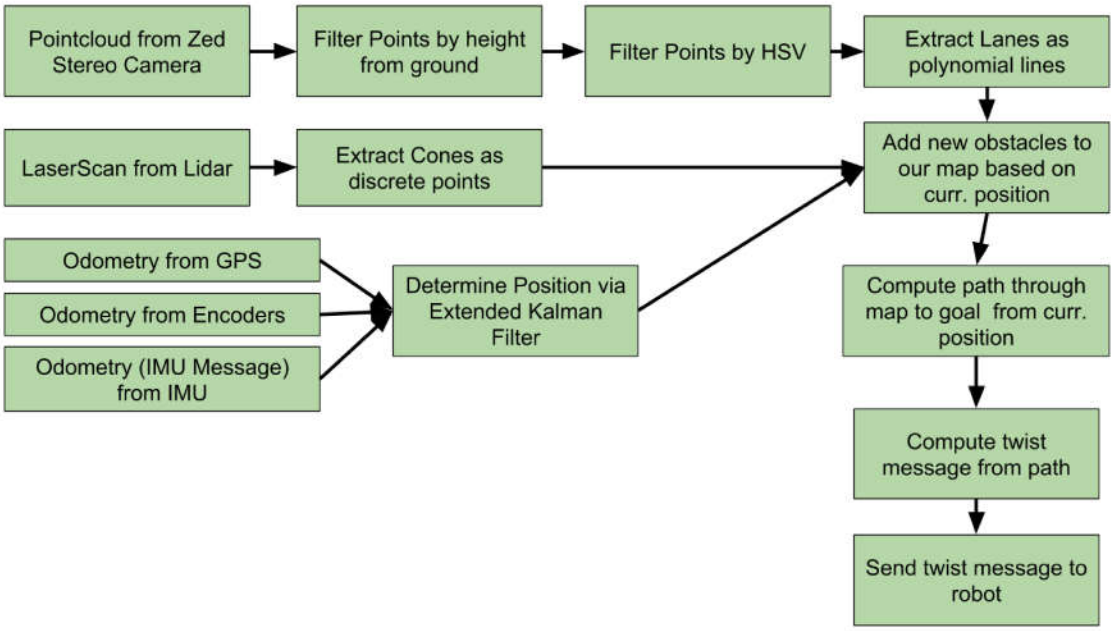


Figure 4 - The software architecture

The software system is built on the Robot Operating System (ROS) Framework. This allows for independent development of a distributed system whereby individual executables can communicate via pre-designated messages over a TCP/IP protocol.

PointCloud Stack

To track lines in the vicinity of the robot, we use a stereo camera to generate a pointcloud of the robot's surroundings, then extract the lines from said pointcloud. We initially used the ZED stereo camera (as it was already available to us), but moved to the Intel Realsense D415 part way through the year. The Realsense uses IR and offloads the image processing from the computer to the camera itself, allowing us to dramatically increase battery life of our computer and run the system on laptops without a large discrete GPU. This pointcloud is then passed through various filters to find white lines on the ground.

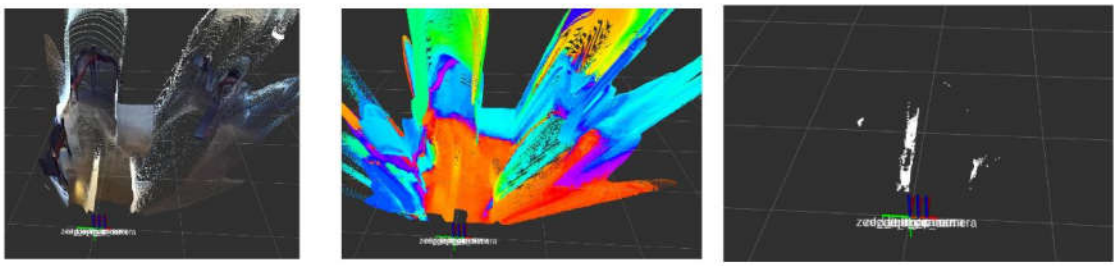


Figure 5 - The line extraction process

Once the point cloud has filtered out most points that do not represent the white lines (Figure 6), the next task is to extract the lines out of the point cloud. Density-based Spatial Clustering of Applications with Noise (DBSCAN) is performed to group the points into clusters, where each cluster represents a line (Figure 7). DBSCAN is a clustering algorithm that clusters points that are densely grouped together, and uses a hyperparameter to define the minimum density required for a series of points to be clustered together. DBSCAN was chosen for two main advantages; it can cluster convex shapes and it is robust to outliers since it filters out points that are not closely packed together.

After performing DBSCAN, each cluster is then used to compute a best polynomial line of fit using linear regression (Figure 8). This allows us to extract discrete lines from the extracted pointcloud data and feed it into our mapping system (Figure 9).

Figure 6 - The initial state of a filtered point cloud

The point cloud contains two dense lines and some sparse outliers.

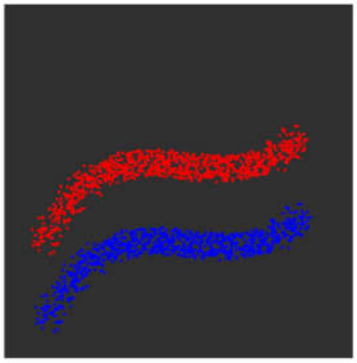


Figure 7 - Clustering of the point cloud

The point cloud is clustered into two lines with some given hyperparameters on density, and the outliers are discarded.

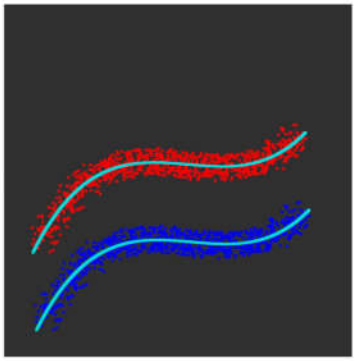
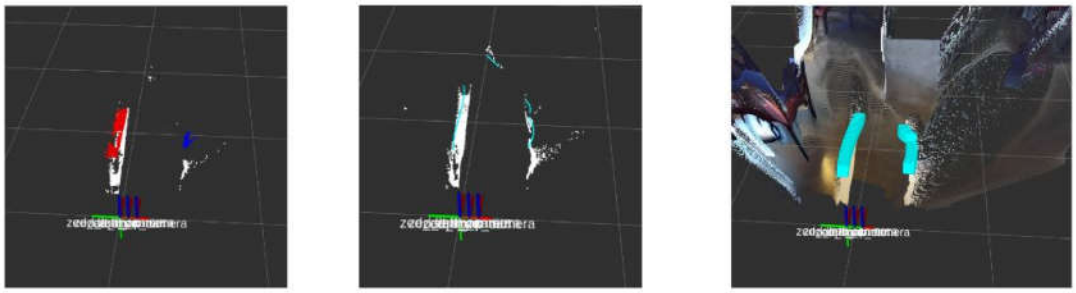


Figure 8 - Linear Regression

Linear regression is performed on each cluster to generate a best polynomial line of fit.



Point clustering | Line extrapolation | Line overlay

Figure 9 - Line extraction in practice

Cone Detection

The cone detection stack includes a cone extractor node that uses LIDAR laserscan data to detect cones of a certain radius in the robot's vicinity. We first sweep the laserscan from the minimum to maximum angle to detect groups of laserscan points close to each other. Then for each group, we split potential clusters of cones (that will appear as one line made of points from the laserscan perspective) into single cones, by finding the local minima of angles formed by two lines converging on some point in the group. Finally, we use a hyper-fit circle fitting algorithm to calculate the center coordinates and radius of some cone, based on the laserscan points that hit its edge. Cones with radii that match closely with expected values are kept, while other objects that do not fit the radii are discarded. Again, this allows for the extraction of discrete obstacles, which may be then fed into our custom mapping system.

Localisation Stack

Localisation of the robot is performed via an Extended Kalman Filter (EKF) which merges sensor data from our GPS, IMU, and wheel encoders to get an accurate estimate of the robots current state. We use the ROS *robot_pose_ekf* package for this purpose, as it was determined that there would be almost no benefit to writing our own implementation for the amount of work that would have been required.

Mapping

Based on experiences from previous years with localisation error due to our lower cost sensors, we have developed a mapping system that allows for accurate and robust mapping of our local environment. By discretizing obstacles (cones and lines) in our environment, we can account for drift in the state of the robot by updating and moving known obstacles based on new sensor readings, rather than “cluttering” up the map with many obstacles that are actually the same, but appear in slightly different relative positions due to state drift.

PathFinding

The pathfinding module receives an occupancy grid where each cell represents either a free or occupied space. The occupancy grid is in a frame that is defined relative to the world frame, and it has several properties such as resolution, width, and height that determine the conversion of a continuous space into a finite number of discrete cells.

In addition to the occupancy grid, the module also receives a starting point and an end point represented in the world frame. It transforms them into the frame of the occupancy grid in order to find which cell in the obstacle grid they fall into. The starting point is guaranteed to be inside any grid produced by the mapping module since the starting point is always assigned to be the location of the robot; which means the information regarding obstacles around the starting point will always be available thanks to all the sensors equipped on the robot. On the other hand, the goal point may be far out of reach from the sensors, resulting in cases where the goal point is located outside of the occupancy grid. In order to cope with such scenarios, this module assumes that all cells outside of the occupancy grid are free. The intention behind this approach is that the module recalculates the map and path by the time the robot reaches the once unknown part of the map.

Given a starting and end point as well as an occupancy grid, the module calculates the shortest path between the two points using the A* algorithm with diagonal movements allowed. The path is represented as an ordered list of positions, and it is first calculated in the frame of the occupancy grid, then converted back into the world frame.

Navigation

To generate the robot's instantaneous velocity and turn rate, we use the path calculated by the path finding module. By generating vectors from waypoints in the given path message, we can take their weighted average (with vectors closer to the origin given a higher weight) to calculate a velocity and turn-rate.

Firmware

The firmware for Elsa X is a ROS node run from an Arduino Mega board. The node receives a twist message from the navigation node and an odometry message from the encoders. The node uses a PID library to calculate the output velocity using the navigation twist message as the set value, and the odometry message as the input value. Finally, the node takes the PID loop’s output of a linear and angular velocity and converts it to left and right motor values compatible with Elsa X’s differential steering system before sending appropriate PWM commands to the electronic speed controllers governing the motors.

Failure Point Identification and Resolution

Mechanical

With the design focused on implementing differential steering for ease of computation on the software portion of our design, balance of the robot was important. Previous experience has lead us to believe that castor balance for differential drive was a main failure point which caused wheel slippage and stalled motors. As a result, with this year's design, the drive wheels have been centered with two castors added respectively on either side of the drive axis. This design allows the robot to have three points of contact in any orientation should there be uneven terrain infront or behind the robot.

Further it has been identified that the main failure point of the robot would be the belly plate where all load of the robot is transferred to the drive shafts. As a result, 3/16" aluminum sheet was chosen to to hold our drive motors, with ⅜" aluminum bars added for further support of bending perpendicular to the drive shaft axis, between the motor mount points.

Due to the vertical loading on our robot, to prevent any possible buckling, bending and shearing of our bolts and support shafts, angle aluminum has been chosen to be the main shape to support vertical load along the robot.

Electrical

Main source of failure for our electrical system would be over current and heating of the system under high load and extreme heat from the sun. To protect the system, fuses have been placed between the batteries and the main circuit to prevent any over current from shorting or motor stalling.

Second, the electrical system is subject to EMF interference from motor back current through the circuit as well as magnetic fields from permanent magnets in our drive motors. As a result, EMF sensitive sensors have been chosen to be mounted outside of the aluminum robot chassis that acts as a Faraday cage to our sensors. The circuitry for sensors and drive motors have also been decoupled and separated.

Software

We employed several layers to help detect and prevent points of failure in the software system. This year we have strived to automate as much failure point detection as possible. Alongside our various layers of simulation and testing (see below), we have made it a requirement that all code added to the code base must go through a full code review process by at least one senior member of the team before we permit it to be merged. Furthermore, to increase code readability, and hence make it easier to review, we have added enforced code formatting. All checks for formatting and code review are performed automatically on *Github*, preventing any code that has not been reviewed, formatted, and validated from entering the repository.

Simulations and Testing Employed

Unit Testing

We have made it an absolute requirement that all code added to the code base be thoroughly unit tested via *gtest*. This requirement has been made an integral part of our training process, helping ensure everyone full understands the level of testing required, and is furthermore enforced in code review, where code coverage is a major point the review of any pull request.

Regression Testing

For larger systems involving several ROS nodes, or even complex systems with a single ROS node, we have made *ROSTests* an absolute requirement. *ROSTests* allow us to test larger sections of our system then unit testing would allow, by providing input (such as specific images or lidar scans) to the system, and checking the output (such as paths, detected cones and lines, or generated maps). This allows us to validate the interaction between the individual components of our system in a more deterministic manner than simulation would allow.

Simulation

To allow the software team to test the entire system without having to have a complete and functioning robot, we have developed a custom robot model in *Gazebo* that allows for the simulation of the entire system, all the way from sensor input to robot movement. This allows us to rapidly test different scenarios that might be difficult to replicate in the real world, increases ease of development, and allows the mechanical team more slack in their timeline were a full physical robot required to test the entire system.

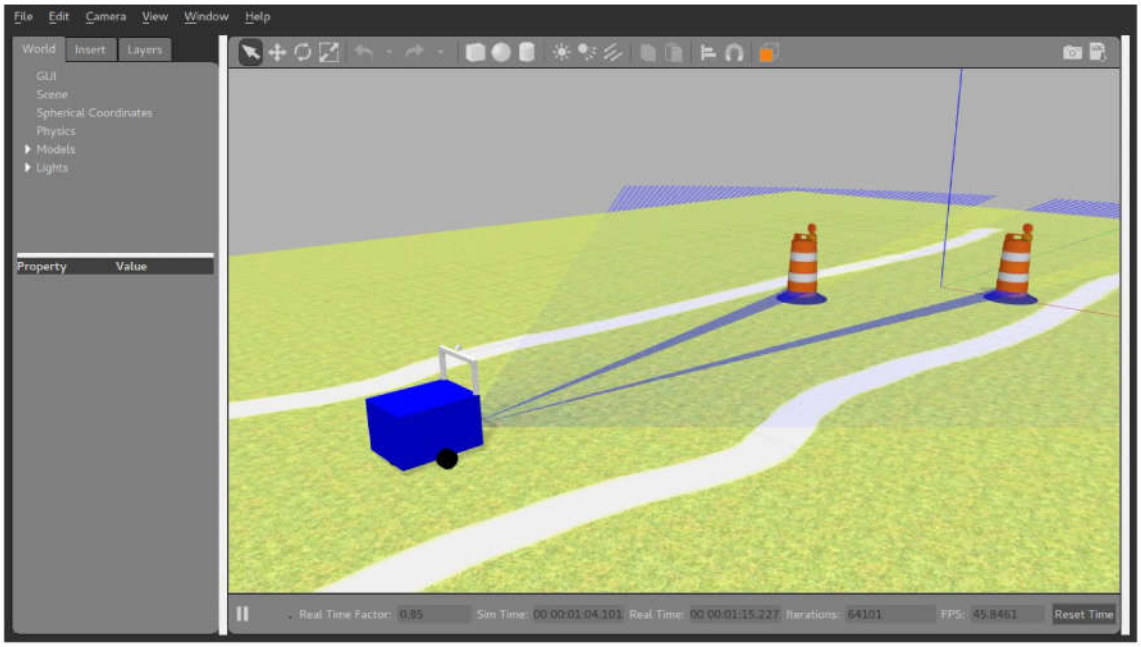


Figure 10 - Software Simulation

Physical Pre-Robot Testing

To assist with sensor integration and performance testing, as well as to allow for various sub-systems to be tested individually in the physical world before the completion of the full robot, we have developed a test cart that allows for quick and easy testing of sensor-dependent systems in real world environments. The cart can be equipped with all the sensors that we have on the final robot.

Figure 11 - Test Cart

Vehicle Cost

The overall cost for Elsa X is defined below:

Item	**Price (CAD)**
Motors	$745.00
Metal	$291.44
LiPo Batteries	$500.00
Laptop	$1,600.00
GPS	$1,235.00
SICK Lidar	$4,500.00
Intel Realsense D415	$200.00
Wheels	$296.70
Encoders	$264.28
Electronic Speed Controllers	$223.98
Total:	**$9,856.40**

Conclusion

Members of UBC Snowbots worked hard on the design and construction of Elsa X this year. Our team has gained a lot of new members and each have worked hard to learn and pick up the required skills in order to meet the competition goals. The team is looking forward to bring Elsa X to IGVC 2018 and are excited to compete this June.