UBC Thunderbird Robotics
Snowbots Robot Racing Team

Design and Implementation of *Blizzard*, Autonomous Race Car
Prepared for the 2010 Robot Racing Challenge
University of Windsor
24 July 2010

**Advisor:**
Dr. John Meech – Faculty of Engineering: Professor of Mining Engineering and Director of CERM3

**Team Leader:**
Ian Phillips – Faculty of Science: Computer Science Department

**Design Team:**
Nick Adams – Faculty of Engineering: Electrical and Computer Engineering Department
Navid Fattahi – Faculty of Engineering: First Year
Jarek Ignas-Menzies – Faculty of Engineering: First Year

**Total person-hour expended:** 650 hours
# Table of Contents

1.0 Introduction .................................................................................................................. 1  
2.0 Hardware  
  2.1 Sensors  
    2.1.1 External Sensors  
    2.1.2 Internal Sensors ........................................................................................................ 2   
  2.2 Processors  
    2.2.1 Microcontroller  
    2.2.2 Computer ................................................................................................................... 3  
  2.3 Chassis and Body  
3.0 Software  
  3.1 Platform (ROS) ............................................................................................................... 4  
  3.2 Snowbots’ ROS Architecture  
  3.3 Microcontroller Software  
  3.4 Data Analysis Nodes  
    3.4.1 Path Following and Obstacle Avoidance ...................................................................... 5   
      3.4.1.1 LIDAR (Light Detection and Ranging)  
      3.4.1.2 Computer vision-based path following  
    3.4.2 Computer Vision Lane Following and Object Recognition  
      3.4.2.1 Lane Following ........................................................................................................ 6  
      3.4.2.2 Stop Sign  
      3.4.2.3 Traffic Light  
    3.4.3 Wireless Emergency Stop  
    3.4.4 Localization  
  3.5 Commander Node .......................................................................................................... 7  
  3.6 Driver Node  
4.0 Testing  
5.0 Analysis ............................................................................................................................. 8  
  5.1 Strengths  
  5.2 Weaknesses  
  5.3 Costs  
6.0 Conclusion  
7.0 References
1.0 Introduction

Blizzard is an autonomous vehicle based on a Traxxas E-Maxx remote control car with a custom mounting platform for the equipment. This chassis is a dual motor, four wheel drive vehicle with a top speed of approximately 30 mph. It uses an Acer netbook as its main processor along with a Arduino Duemilanove microcontroller. The sensors comprise of a LIDAR, a webcam, a wireless emergency controller, an Inertial Measurement Unit (IMU) with 9 Degree of Freedom (DOF), Optical odometers, and battery monitors. The sensors are attached on the mounting platform to reduce possible interference and improve readings.

Blizzard’s software has been done on two levels: microcontroller and computer. On the microcontroller side we use an Arduino-custom platform. And on the computer side our software is running on a framework called ROS (Robot Operating System) and based on a UBC Snowbots’ custom-made architecture.

2.0 Hardware

To meet the goals we set in place for the autonomous car, the hardware will have to provide three features: ability to see – or feel– the surrounding environment, ability to deliver its observation to the mind of the robot (processors), and capability to move through various environments. These features can be categorized into three groups: sensors (input), processors (data manipulation), and movement (output).

2.1 Sensors

It is important for the robot to be able to accurately see and interpret its surroundings and know its current state in the world. The sensors supply the robot with information on both the environment around it and its current state.

2.1.1 External sensors

The first group of sensors are supposed to give the robot an understanding of its surrounding environment. The first sensors we tried in this category were ultrasound distance detectors. The main issue with them, however, is their lack of accuracy outdoors under two conditions: windy weather and rugged surfaces. We instead chose to use infrared sensors as alternatives due to their improved performance outside. Again, there was a potential draw back with this solution: using infrared sensors, we would get small number of data points and non-negligible inaccuracies. As an alternative solution we moved to using LIDAR (light detection and range finding) to give us longer range, more accurate, and an increased number of data points. While LIDAR can be used for most of the navigation, visual identification of traffic light and stop sign are being done by a webcam attached to a pan tilt servo unit.
2.1.2 Internal sensors

The second group of sensors are used to give the robot information about its current state and movement. This group comprises of two optical encoders, an accelerometer, a gyroscope, a magnetometer, and battery monitors. The optical encoders use a photo transistor that returns a voltage dependent upon the amount of light reflected off the inside of the wheel rim. The rim is covered with black and white strips to change the reflectivity of the surface. Using a voltage comparator this signal is turned into a digital signal that the micro-controller can interpret as an interrupt. The encoders are mounted on both back wheels to get more accurate measurements. To obtain information about the robot's orientation, a 9 DOF IMU has been placed included. The IMU uses three MEMS sensors: an ITG-3200 3-axis gyro-scope to measures the angular acceleration\[^1\], a BMA-180 3-axes accelerometer for acceleration measurement\[^2\], and a 3-axis magnetometer to measure the magnetic field\[^3\]. These three sensors are mounted on individual breakout boards that are in turn mounted on a breadboard. There are also simple voltage dividers used to measure the current voltage on the batteries.

2.2 Processors

Blizzard uses a netbook computer and a microcontroller as its processors. While some processing is being done on the microcontroller itself, the netbook also uses it to connect to all the external devices that are not using USB ports. The sensors that use USB ports are LIDAR, webcam and Bluetooth e-stop; the rest of the sensors and motors are using the Arduino to communicate with the computer.

2.2.1 Microcontroller

Blizzard uses an Arduino Duemilanove\[^4\] to receive inputs from sensors and send output commands to the motors. The Arduino uses I2C communication, analogue inputs, digital interrupts and PWM waves (for controlling both the ESC – electric speed control – and the servomotors) as its means of communication to the computer and sensors.

2.2.2 Computer

An Acer aspire one netbook is being used as the main processor. The reason it was chosen over a separate processor, was because of its small size included screen, keyboard and touchpad. The netbook, however, has the disadvantage of having a weaker processing power; although, this was only an issue while taking into account the heavy vision processing power needed. This problem was solved by software optimization.
2.3 Robot Chassis and Body

A Traxxas E-Maxx remote controlled monster truck [5] has been used as the Blizzard’s chassis. The E-Maxx provides a stable platform capable of navigating around the course with a fairly high speed and power (maximum of 30 mph).

A mounting platform made out of corrugated plastic sits on the chassis as the body. The mounting platform has a two level compartment on the back half that holds the majority of the electronics. The Arduino, breadboard, LIDAR battery and LIDAR voltage regulator are on the bottom in recesses designed to reduce space wastage. The netbook is placed on top of the electronics to keep it close to the electronics it connects to. On the back of the platform, there is a tail that holds the mechanical emergency stop. The nose of the platform holds pan-tilt server unit on which the webcam is sitting. LIDAR sensor is placed underneath the nose to insure that it does not see over the cones; it also has a sun cover to prevent possible sunlight effects on the readings.

The chassis also allows for operating on rough terrain and in wet conditions while the mounting platform is keeping the electronics safe from damage. In case of any possible software failure, the robot can be stopped immediately using the emergency stop button on the tail-end of the robot. Overall, the mechanics of the robot has been designed for achieving the best usage of the space and it seems to be providing us with the maximum amount of efficiency.

3.0 Software

The software required to drive Blizzard had to accomplish three main goals; read the sensor information, evaluate all the relevant sensor information to make a navigation decision, and take action controlling the motors and servos. On the microcontroller side there is an Arduino Duemilanove. The computer is running Ubuntu Linux with Robot Operating System (ROS) as the main communication platform; and the software is primarily written in C++ and Python.

The software that Blizzard is running can be divided into four main categories: Arduino microcontroller software, data analysis nodes, commander node and driver node.
3.1 Platform (ROS)

ROS (Robot Operating System)\[6\] is a platform on which all our software is running on. It is a framework, developed and maintained by Willow Garage, which manages the intercommunications between different parts of software and allows for a very high level of modularity. Each part of software is referred to as a node on 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.

3.2 Snowbots’ ROS Architecture

A custom-designed ROS structure has been adopted by UBC Snowbots team for ease of development and expansion and more modularity and efficiency. At the level closest to the microcontroller, there is a driver node whose job is to send and receive information to and from the microcontroller. The data coming from the microcontroller is passed to the data analysis nodes; these nodes will analyze the inputs and give appropriate suggestions to the commander node on how to act. The commander node, then, sends its final decision back to the driver node in order to execute them.

3.3 Microcontroller Software

The Arduino is connected directly to the motors, servos and sensors on the robot. It sends sensor information to and receives driving instructions from the computer via serial communication. The data is being transferred between the two via strings of bytes. The majority of the sensor information is raw voltage values when it is put into a string and sent to the computer. The only sensor data that is being processed within the microcontroller before it gets passed to the computer is the IMU data. The reason for this is that by doing some simple calculations in Arduino, we would prevent sending a large volume of unnecessary data to the computer.

3.4 Data Analysis Nodes

In our software architecture the job of data analysis nodes is to receive inputs, process them and generate appropriate suggestions regarding different tasks. They receive information either via messages of type RobotState or directly via USB ports. Because of the essence of this architecture, any of these nodes can be switched in or out of the system at any time. The nodes necessary to run the robot in the robot race competition have been described in details below:
3.4.1 Path Following and Obstacle Avoidance

3.4.1.1 LIDAR (Light Detection and Ranging)

Blizzard’s LIDAR sensor is a Hokuyo product\(^7\) with a 240° field of view and a maximum measurement of 5.2 meters. It has a resolution of 0.36°. The LIDAR is connected directly to the computer via USB but is powered separately. Blizzard’s LIDAR algorithm uses the concept of “center of mass” to navigate around. In this notion, the further away the objects are from the LIDAR, the more “mass” is located between them. Therefore when we add all our data points together, the center of mass tends to approach towards the more “empty” space. By figuring out the coordinates of the center of mass on the plane, and following it, we would essentially always move towards the more “open” space. When the amount of “free space” in front of the robot is too small, a suggestion for backing up is being issued. These suggestions are being passed to the commander node via a ROS standard Twist message that includes both steering and throttle values.

3.4.1.2 Computer Vision-based path following

The computer vision navigation algorithm is very similar in design to the LIDAR navigation algorithm except that it uses a camera and scaled pixel distance as a unit of distance. The pixels are summed for every pixel moving outward from the centre and up the image until the hue and saturation values are different enough, from a start point about 5 cm ahead of the vehicle. That would determine where the road has ended. Once the algorithm determines that a pixel is not on the road, any further points radially outward from that pixel relative to the start point are excluded and not used in the calculation. This algorithm runs at 10Hz and sends the same set of navigation coordinates to the commander as the LIDAR node.

3.4.2 Computer Vision: Lane Following and Object Recognition

Computer vision seems to be the best way to accurately detect the presence of a stop sign and identify a traffic light. It is the only sensor powerful enough to identify what an object is as opposed to what it looks like or how far away it is. These object recognition nodes are running at 1Hz as they are much more computationally expensive than other nodes. It is worth mentioning that we have taken advantage of openCV libraries\(^9\) in all our vision software.
3.4.2.1   **Lane Following**

The basics of the lane following node is essentially the same as that of vision path following. The camera looks ahead of the robot and detects the white line on the ground using saturation and hue filters. It then sends the commander node its recommendation on where to go next. This, likewise to other navigation nodes, is done via a Twist message.

3.4.2.2   **Stop Sign Detection**

The stop sign algorithm takes a frame and looks in it for a cluster of red pixels (in a certain color-range). If this cluster is big enough (meaning close enough to the robot), the software calculates how many sides it has using the pixels on the boundaries. If it is has 8 sides then there is a good chance that the object is a stop sign. After the processing is done, the node publishes a message that contains a “degree of certainty” regarding the existence of a stop sign.

3.4.2.3   **Traffic Light Detection**

The traffic light algorithm detects the rectangular box that is the traffic light case. Once the box has been identified it is simple to determine which of the two lights is on. The algorithm divides the box in two parts horizontally and then determines which of the two halves is brighter. The half with the greater value is the light that is on, top means light is red and bottom means light is green. The output of this node is identical to that of the stop sign detection node.

3.4.3   **Wireless Emergency Stop**

The emergency stop node receives a signal from the emergency stop controller and immediately sends the stop signal to the vehicle causing the robot to stop until the emergency stop button is released back again. The emergency stop controller is either a Bluetooth connected Wiimote or a computer or any other device capable of SSH connection to the same network. The type of the recommendation of this node is a Boolean value: on or off.

3.4.4   **Localization**

Using the odometer and IMU our vehicle can determine approximately where it is on the course to aid in navigational recommendations. Our algorithm has a map containing unique features on the course such as specific sharp corners, or the traffic light, or the tunnel. Using the known position of these specific features, the odometer and the IMU we can approximate where we are on the course and thus the general direction in which we want to travel in. Similar to the other two navigation nodes, this node sends its recommendation to the commander node via a Twist message too.
3.5  Commander Node

The commander node receives all the data from the data analysis nodes and evaluates their overall recommendations for consistencies and discrepancies. There are three main nodes (LiDAR, vision and localization) which return a recommended direction; therefore unless there is a specific reason not to, the majority agreement is taken as the new heading. The commander is running at 10 Hz giving the vehicle a reaction time of 0.1 second. The human readable motor control values are passed via a CarCommand message to the driver node to be converted to PWM values and passed to the Arduino.

3.6  Driver Node

The driver node acts as the intermediary between the Arduino and the computer. It communicates to the Arduino through serial communication and to the commander and data analysis nodes via ROS messages. One of its tasks is to convert human understandable servo values into PWM values and vice versa. The purpose of this node is to further modularize the software so that the vehicle can be programmed without a detailed understanding of how the hardware works or how the hardware communicates with the software. This node need not be changed or re-written unless the hardware we are using changes significantly.

4.0  Testing

An important part of the design process was the facilitation of early testing. This was done by having the hardware and software modular enough to allow for components to be tested in section instead of as a whole. This process involved making proxies for the missing components that allowed the testing of independent components. For example, sensors were tested by streaming a feed of their results in a terminal window; or the LiDAR navigation system was tested on a ROS-based simulator called “Stage” [8]. The frequent testing at the early stages of development allowed for multiple generations of designs for most of the hardware and software components.
5.0 Analysis

5.1 Strengths

Modularity is probably the most important strength of Blizzard. On the hardware side, we have tried to use as much off-the-shelf pieces as we can; and instead of re-inventing the wheel, try to optimize it. Likewise, on the software side, the design is modular to the extent that every single piece can be tested on its own or using custom-made proxies.

5.2 Weaknesses

The main issue with Blizzard, at the time of writing this paper, is lack of enough testing. Although different pieces have been tested separately and successfully many times, we still need to do a full test or preferably run the robot in a setting that simulates the competition.

5.3 Costs

The estimated duplication cost for Blizzard is about $3630 CND. This relatively high cost is mainly due to the high price of the LIDAR sensor which costs about $2600 CND alone. Note that the duplication cost does not take into account the work labour.

6.0 Conclusion

The Blizzard Robot demonstrates the ability for autonomous navigation systems to successfully operate without the use of expensive computers or sensors. The use of mostly off-the-shelf parts allows for rapid duplication of the robot and quick replacement of broken parts. The modularity of design also allows for scalability of the driving system only reacquiring a modification of steering and motor controls. This meets the design goal of developing methods of autonomous navigation that is inexpensive and easily scalable. We believe that Blizzard’s solid hardware design and optimized artificial intelligence would make it a strong competitor for other robots!

7.0 References