UBC Thunderbird Robotics

Snowbots Robot Racing Team

Design and Implementation of Blizzard, Autonomous Race Car

Prepared for the 2011 Robot Racing Challenge

University of British Columbia

July 23, 2011

Advisor

Dr. John Meech - Faculty of Applied Science: Professor of Mining Engineering

Team Leader:

Navid Fattahi - Faculty of Applied Science: 2nd year Computer Engineering

Design Team:

Edward Han - Faculty of Applied Science: 3rd year Computer Engineering

Eduardo Silva - Mechatronics Student

Table of Contents

1.0 Introduction

This is a design report for Blizzard, one of the vehicles submitted for the 2011 Robot Racing Challenge by Snowbots. Snowbots is a student-led robotics club from the University of British Columbia (UBC). Blizzard competed in the 2010 Robot Racing Challenge at the University of Windsor and was awarded 1st place overall [1]. This year, the main focus was on navigation, and no major hardware changes were made.

Design process is as follows:

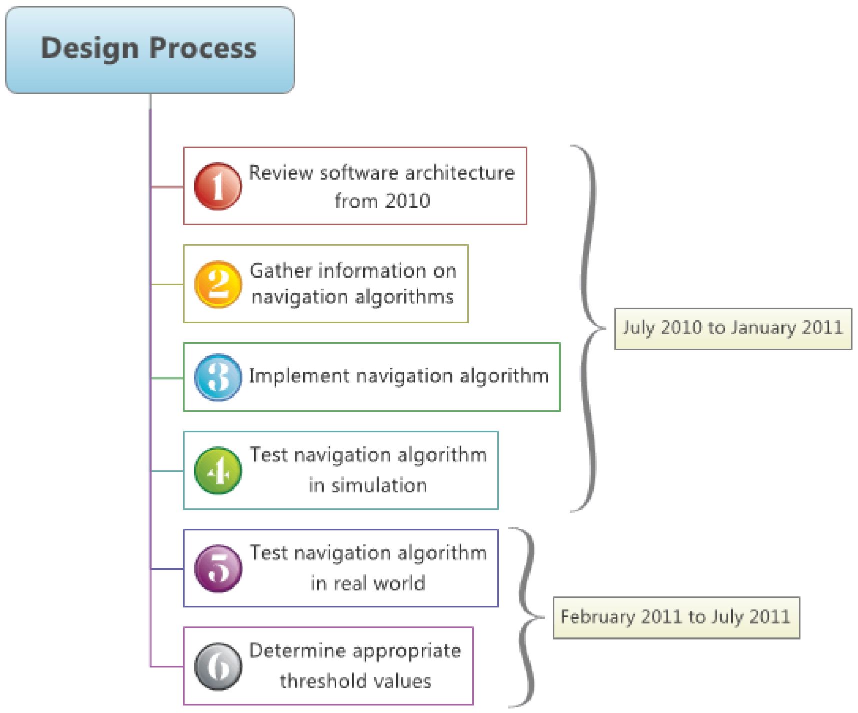
Design Process
1 Review software architecture from 2010
2 Gather information on navigation algorithms
3 Implement navigation algorithm
4 Test navigation algorithm in simulation
5 Test navigation algorithm in real world
6 Determine appropriate threshold values
July 2010 to January 2011
February 2011 to July 2011

Figure 1: Design Process

Section 2 of this report describes the hardware design of Blizzard, and section 3 describes the software architecture. Section 4 is an analysis of Blizzard, describing its strength, weakness, and safety.

2.0 Hardware

The hardware design has not been changed since last year's Robot Racing Challenge competition. Blizzard uses Traxxas-Emaxx Model 3903 chassis and is capable of reaching 48 km/h at maximum speed [2]. Two sensors are mounted on the chassis: LIDAR (Light Detecting and Ranging) by Hokuyo Automatic Co., LTD., and a Logitech camera. The data values from these two sensors are connected to the netbook via USB connection and are sent to the netbook for throttle-steering calculations and image processing. Finally, a throttle and a steering value are determined and sent to the microcontroller.

2.1 Chassis

The Traxxas-Emaxx chassis has been modified to accommodate sensors, netbook, batteries, and an emergency stop button (e-stop). The chassis has two levels: Top level holds the netbook, while the bottom level holds majority of electronics, including Arduino microcontroller, LIDAR voltage regulator, and LIDAR battery. A mechanical e-stop is mounted on the tail of the chassis, while the two sensors (LIDAR and camera) are mounted on the nose of the chassis.

2.2 Sensors

The two sensors mounted on the nose of the chassis are LIDAR, and camera.

2.2.1 LIDAR

The LIDAR is mounted at the bottom of the nose of the chassis. It has a maximum and minimum viewing range of 5.2m and 0.25m respectively, and viewing range is restricted to -90 to 90 degrees, (reference point being the center of the LIDAR) . The angle increment between each laser pulse is 0.36 degrees [3].

2.2.2 Camera

The camera is attached to a pan tilt servo unit and is mounted on top of the nose of the chassis. It sends a snapshot to the netbook for image processing every 1/10 of a second.

2.2.3 Netbook

Netbook is where navigation and image processing algorithms are performed. It is connected to the LIDAR, camera and Arduino microcontroller via USB connection. After the calculations are done, it sends a throttle and a steering value to the microcontroller.

2.2.4 Arduino Microcontroller

Microcontroller generates PWM waves, based on the throttle and steering values, to the servo motors.

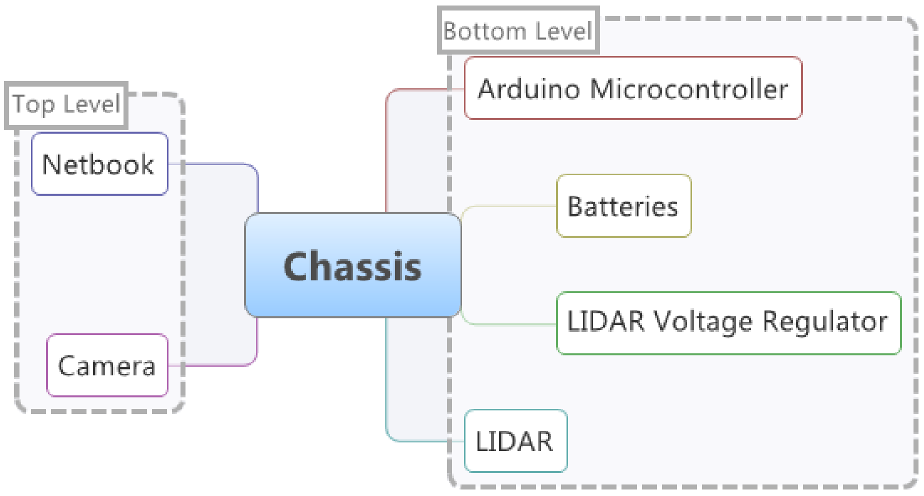

Figure 2: Hardware Design

3.0 Software

The software framework used for navigation, computer vision, and serial communication is called Robot Operating System (ROS) by Willow Garage [4]. The main advantage of ROS is its ability to run multiple processes, called *nodes,* simultaneously. For example, if Vision node shuts down, other nodes, such as LIDAR Navigation, do not get affected. ROS also allows nodes to communicate with other nodes via *messages*. The following describes Blizzard ROS architecture, navigation, and computer vision.

3.1 Blizzard ROS Architecture

There are 5 nodes running on Blizzard: Hokuyo LIDAR, LIDAR Navigation, Vision, Commander and Arduino Driver. Hokuyo node is developed by Brian P. Gerkey, Jeremy Leibs, and Blaise Gassend [5], and converts voltage values into distance values from the LIDAR sensor. LIDAR Navigation node is responsible for cone detection. It receives distances from the Hokuyo node, determines the positions of the cones relative to the vehicle and sends the data to Commander node. Vision node is responsible for stop sign and traffic light detection and also sends data to the Commander node. Commander node acts as a special node that prioritizes which data to process (Lidar, Vision, etc), calculates the throttle and steering values and publishes the data to Arduino Driver node. The Arduino Driver node handles serial communication, and acts as a bridge between netbook and microcontroller.

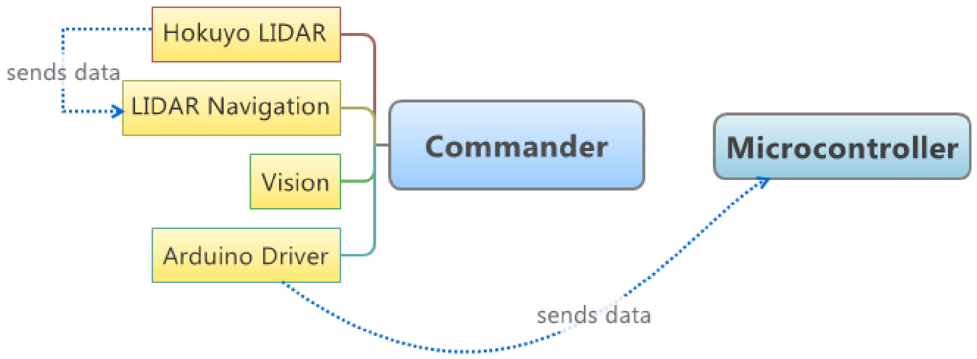

Figure 3: Blizzard ROS Architecture

3.2 Navigation

Navigation algorithm is split into two sections: Cone Detection, and Path Planning.

3.2.1 Cone Detection

In Cone Detection, an array of distances is received from the LIDAR. The first index of the array corresponds to the distance of the object at -90 degrees relative to the vehicle. The LIDAR has a angle increment of 0.36 degrees. Therefore, the next index corresponds to the distance at -89.64 degrees. Using the array, distances and their corresponding angles can be found. The Cone Detection algorithm is as follows:

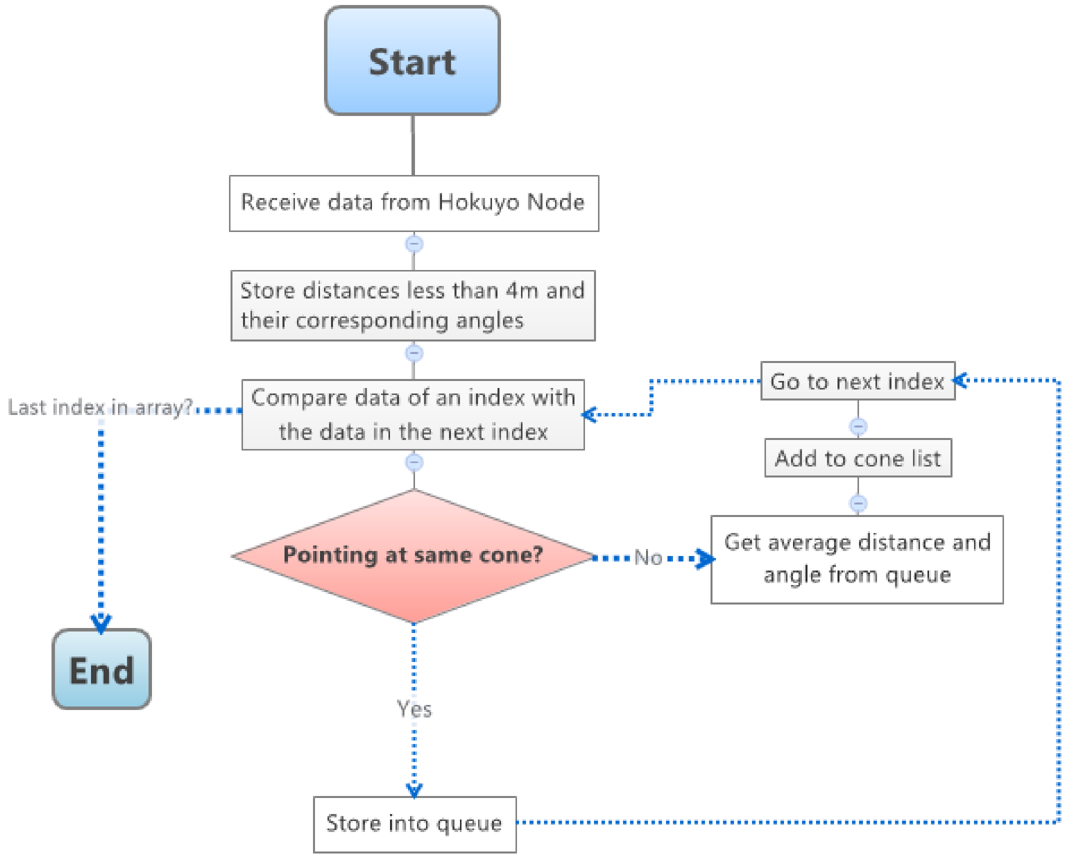
Start
Receive data from Hokuyo Node
Store distances less than 4m and their corresponding angles
Compare data of an index with the data in the next index
Last index in array?
End
Pointing at same cone?
No
Yes
Get average distance and angle from queue
Add to cone list
Go to next index
Store into queue

Figure 4: Cone Detection Algorithm

Once this process is done, a vector of cones is generated. Each cone has two properties: distance and angle. This vector is then passed to Path Planning.

3.2.2 Path Planning

Using the vector of cones received from Cone Detection, Blizzard is able to identify where the empty regions are and throttle and steering values needed to go to one of the empty regions are calculated. Path planning algorithm is as follows:

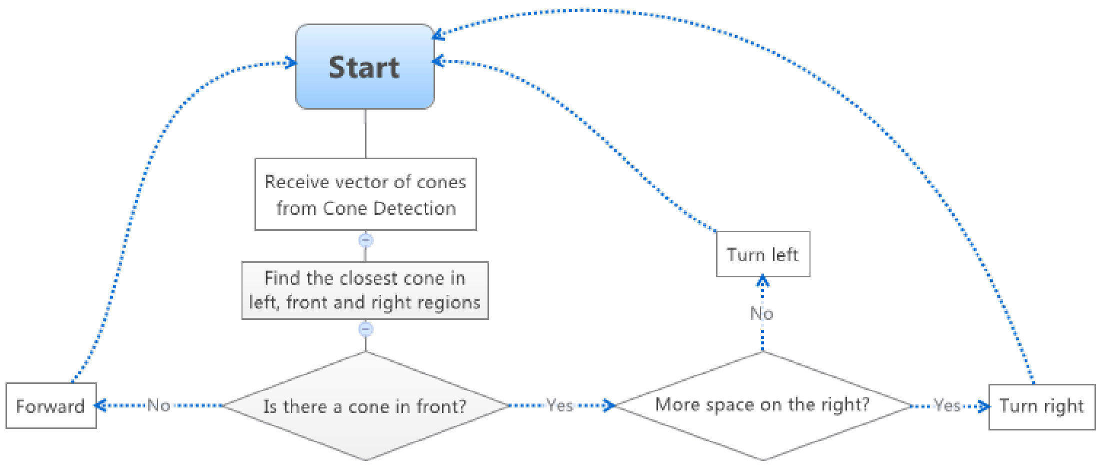
Start
Receive vector of cones from Cone Detection
Find the closest cone in left, front and right regions
Is there a cone in front?
No
Forward
Yes
More space on the right?
No
Turn left
Yes
Turn right

Figure 5: Path Planning Algorithm

After this process, the state (forward, turn left, turn right) is sent to the Commander node. The Commander node decides whether to use the information from LIDAR Navigation node or from Vision node, but Vision node usually takes priority. A special state, which tells Blizzard to move backward when it encounters obstacles such as dead-end, is triggered when cones are visible in all regions and within 1m range.

3.3 Vision

Blizzard's vision is composed of two sections: traffic light recognition and stop sign detection.

3.3.1 Traffic Light Recognition

A snapshot of the image seen by camera is sent to the Vision node. If there exists circle(s) within a rectangle, the rectangle is considered as a traffic light. Then, regions that contain red pixels within the circle(s) are identified. If a region within the circle has sufficient brightness, a stop command is sent to the Commander node.

3.3.2 Stop Sign Detection.

An image is considered as a stop sign if it satisfies the following two conditions: The image has sufficient red pixels, and it is contained within an octagonal shape. First, red pixels are identified and isolated from the image. Then, using the HoughLine function [6], the shape that contains red pixels can be determined. If the two conditions match, a stop command is sent to the Commander node.

4.0 Analysis

Although there was no significant hardware changes to Blizzard, the navigation algorithm proved to be extremely effective and promising during its early stages of testing. A simplistic path planning algorithm was implemented for the purpose of learning how to use ROS simulation, and it turned out with great success. With fine tuning, we believed Blizzard can defend its title in this year's competition.

4.1 Strength

The algorithm for cone detection is Blizzard's greatest strength. Development began in August 2010 under software lead, Nick Adams. However, due to some problems on path planning algorithm in late November, the algorithm for cone detection was split into two directions. Edward, on one hand, started developing the cone detection algorithm suited for a simplistic path planning algorithm for Blizzard. In late December 2010, the cone detection algorithm along with a simple path planning algorithm were completed and the vehicle was able to navigate an oval path in simulation.

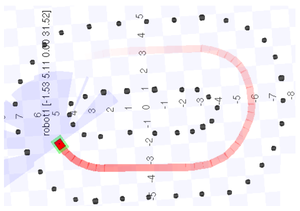

Figure 6: Test in simulation

4.2 Weakness

One of the overlooked flaws of Blizzard is that simulation does not guarantee the same result in the real world. The problem with Blizzard was that although navigation algorithm worked well in simulation, it did not work nearly as good as it did in the real world. Starting from January 2011 to April 2011, team members tried to use cone detection algorithm on Blizzard. The result was a lot of exceptions being thrown on the command line. It is later determined to be buffer overflow. From May 2011 to July 2011, path planning was put to the test. Although no exceptions were thrown this time, Blizzard was unable to make sharp turns. It is later found out that in simulation, when the vehicle is turning, it does not move forward. This was a huge overlooked discrepancy between simulation and real world. As a result, Blizzard is unable to make smooth 180-degree-turns.

4.3 Safety

A mechanical emergency stop button is mounted on the tail of Blizzard. Its purpose is to stop the vehicle by cutting off power from the batteries to servo motors. There are also bumpers added to the front and back of Blizzard, protecting chassis, LIDAR, and other important and expensive components. In addition, on the software side, a safety state is added. If no objects are detected in front, left, and right regions of the the vehicle for more than 20 seconds, the speed decreases to 0 over 10 seconds.

5.0 Conclusion

Combined with accurate sensors, simple navigation algorithm, and various testings, we are proud of what we have achieved on Blizzard. Although the learning curve was steep, it was a great learning process. We have overcome obstacles through team work, hard work, and dedication throughout the year. We hope to achieve high results in this year's Robot Racing Challenge at University of British Columbia.

5.1 Expenses

Item	**Cost ($)**
Netbook	500
LIDAR	2500
Traxxis E-maxx Chasis	400
Arduino Microcontroller	30
Electronics	100
Total	**3530**

5.2 Estimated Total Hours

Edward Han – 5 hrs / week for 40 weeks = 200 hours

- Cone Detection
- Path Planning

Eduardo Silva – 5 hrs / week for 40 weeks = 200 hours

- Traffic Light Recognition
- Stop Sign Detection

Total Hours = 200 + 200 = 400 hours.

6.0 References

1. http://robotracing.wordpress.com/awards/
2. http://traxxas.com/products/models/electric/3903emaxx
3. http://www.robotshop.com/ca/hokuyo-urg-04lx-laser-rangefinder-1.html
4. http://www.ros.org/wiki/About%20ros.org
5. http://www.ros.org/wiki/hokuyo_node
6. http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_ImageProcessing.htm#decl_cvHoughLines