Olaf 3.0 Design Report

UBC Snowbots – University of British Columbia

Raad Khan ~ *Electrical Engineering*
Gareth Ellis ~ *Computer Science*
Marinah Zhao ~ *Computer Engineering*
Valerian Ratu ~ *Computer Engineering*
Vincent Yuan ~ *Electrical Engineering*

***Abstract*—This document details the design of UBC Snowbot's entry into the 2018 International Autonomous Robot Racing Challenge (IARRC). It will review the conceptual design of the robot along with its components, and how this design aims to address the main challenges of the competition.**

I. Introduction

UBC Snowbots is a student design team based at the University of British Columbia that focuses on autonomous robotics. We are an interdisciplinary team comprised of dedicated undergraduate students from various departments and faculties including Computer Science, Electrical Engineering, Computer Engineering, Mechanical Engineering, and Engineering Physics. Our team's goal is to allow members of all skill levels to learn and develop technical and leadership skills through solving challenges posed in autonomous robotics.

II. Mechanical

The mechanical component of our design is based on an off-the-shelf RC car. The drivetrain of the vehicle has a DC motor driving the two rear wheels, and a servo controlling the front wheels for steering. The drivetrain also includes a suspension system that is useful for mechanically stabilizing the camera and LIDAR images. A 16:90 gear ratio is used from the motor to the drive shaft to help ensure compliance with the maximum allowed vehicle speed.

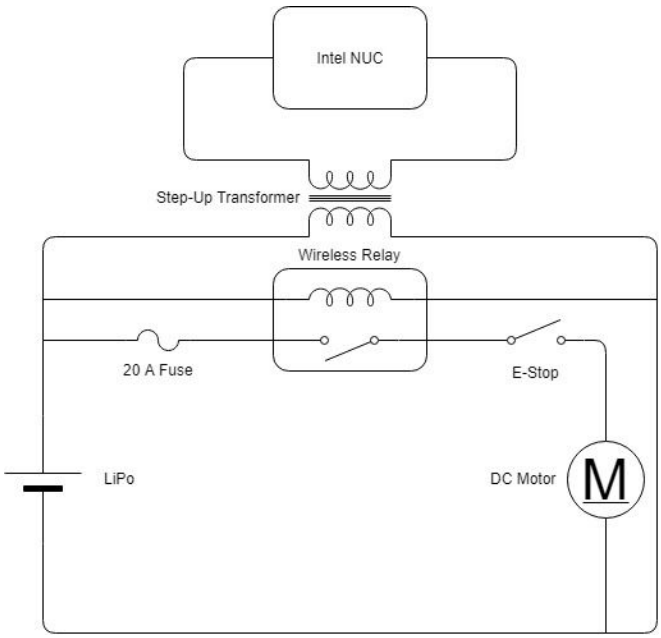
The original chassis of the car has been removed and the frame modified to fit our sensors and processing unit. Notable additions include a custom steel plate base to hold the main processing unit, an Intel NUC. Also mounted on this plate are two towers - one for our camera to provide it with a better field of view, and another for our mechanical e-stop to ensure it meets the safety requirements of being 30 cm above ground.

III. Electrical

A. Power System

All the components of the vehicle are powered by a single 2000mAh 7.4V lithium polymer (Li-Po) battery. Our main processing unit, an Intel NUC, is powered from the battery via a transformer, which converts the battery voltage to the requisite voltage for the NUC. The battery also powers the motor by way of an Electronic Speed Controller (ESC).

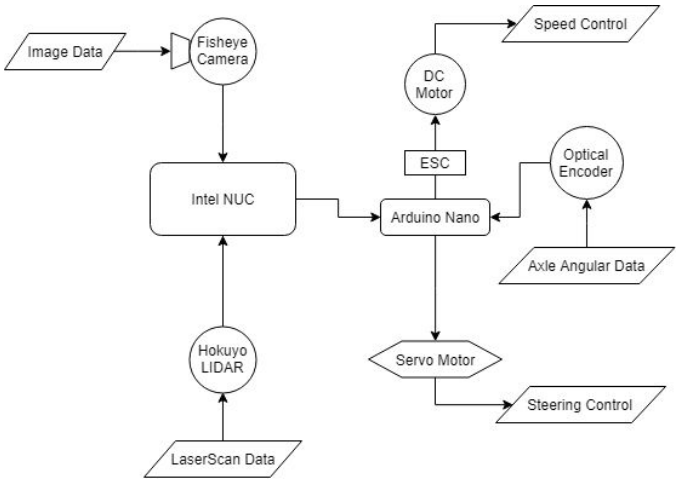
The power to the motors is directed through a wireless e-stop and a physical e-stop which are both normally open. If a failure is to occur in either of the e-stops or if either of them are switched on, power to the motor will be immediately cut.



Power Diagram

B. *Sensors*

Our vehicle uses a Hokuyo URG-04LX-UG01 LIDAR and fisheye camera to navigate. The fisheye camera is used for line and stoplight detection, while the the LIDAR is used to detect other vehicles and cones. There is also an optical encoder on the rear wheel axle which is used for PID control in the firmware.



Sensor Diagram

C. *Firmware*

The onboard NUC connects to an Arduino Nano via a USB serial interface. The high level control software on the NUC sends linear and angular velocity messages to the Nano, which translates them into PWM signals for both the servo motor to control the steering, and the ESC to control the driving motor. The Nano also implements a PID controller to guide the PWM signals.

IV. Software

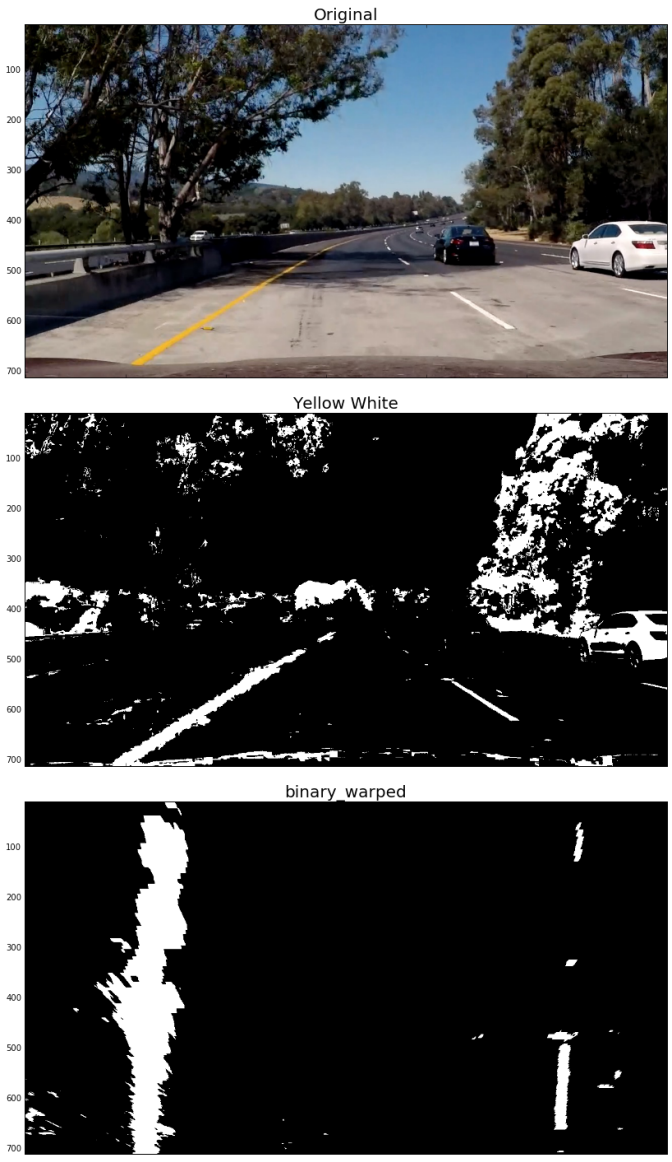
All computing is done on an Intel NUC [0] with a Celeron processor, 8GB of RAM, and a 128GB SSD. The two principal inputs to our system are the video stream from the camera, and the pointclouds from the LIDAR. Our software stack is built on the Robot Operating System (ROS) Framework [1] which allows multiple executables to communicate over a TCP/IP protocol. This decoupling between different components allows for independent development and validation.

V. Competition Objectives

A. *High Speed Vehicle Localization*

To localize the vehicle to the course, we first use the image data from the fisheye camera. The raw image is first rectified to account for distortion of the camera lense. Using the known angle of the camera, an Inverse Perspective Mapping (IPM) filter is then applied to translate this image to a birds eye view. The course boundaries are then filtered for by passing this translated image through Hue, Saturation, and Value (HSV) filters, which provide a more robust colorspace for filtering then the more common RGB colorspace. The result of this is a binary image outlining the lane boundaries relative to the vehicle.

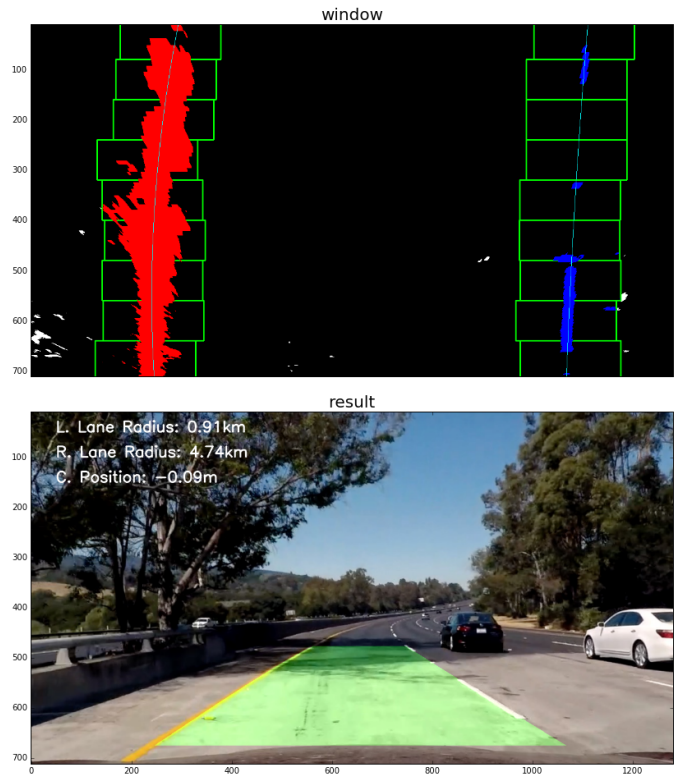
[0] – https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html
[1] – http://www.ros.org/
[2] - https://chatbotslife.com/self-driving-cars-advanced-computer-vision-with-opencv-finding-lane-lines-488a411b2c3d



[2]

B. *High Speed Vehicle Control*

To control the vehicle, we first abstract the lane boundaries as polynomial lines by applying the sliding window method to the binary image obtained in the previous section.



[2]

Using this abstracted interpretation of the lane boundaries, we are then able to find the intersection of polynomials to determine the "vanishing point" of the lane from the robot's perspective. Separate PID loops within the firmware are then used to control the linear and angular velocity outputs to send to the motors.

C. *Start/Stop Light Detection*

The start and stop lights are detected similarly as the lane boundaries are. The raw camera image is rectified to account for lense distortion, then the HSV filter is applied to the corrected image to find the colors of interest, red or green. The contours of this filtered image are then found, and a circle fitting algorithm is applied to check if a circle of the appropriate size is present, indicating that the specific light is present from the vehicle's view.

D. *Collision Avoidance*

Our LIDAR is the principle input for our collision avoidance algorithms. From the pointcloud produced by the

[0] – https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html
[1] – http://www.ros.org/
[2] - https://chatbotslife.com/self-driving-cars-advanced-computer-vision-with-opencv-finding-lane-lines-488a411b2c3d

LIDAR, obstacles are detected via clustering. If an obstacle of sufficient size is found to be within the projected path of the robot, we override other steering commands to take immediate evasive action.

VI. Software Validation

A rigorous and multi-level software validation strategy was taken to ensure correctness of the software outlined in previous sections. Firstly, all code was thoroughly unit tested at a per-function and per-class level. Secondly, the decoupled nature of ROS allowed us to write higher level integration tests to provide both individual components and groups of components with test input sensor data, and to validate the correctness of the resultant steering outputs. On top of this, physical testing was undertaken (as later described in this document), and all code was passed through a review process with several senior software members.

VII. Mechanical Validation

Mechanical changes were validated principally via direct vehicle control over courses similar to those of the competition. During these tests, both vehicle speed and stability were assessed and found to be satisfactory, despite additional weight added via the metal plate, tower, and electronics. There was a predicted reduction in top speed, but it remains well above the level required by our software system.

[0] – https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html
[1] – http://www.ros.org/
[2] - https://chatbotslife.com/self-driving-cars-advanced-computer-vision-with-opencv-finding-lane-lines-488a411b2c3d