Cavalier Robotics — Camera-Limited Autonomous Race Car

*Author of correspondence

Sophia Mourcos
Cavalier Robotics Member
ESC Père-René-de-Galinée
Cambridge, Canada

Jacob Hergott
Cavalier Robotics Member
ESC Père-René-de-Galinée
Cambridge, Canada

Joel Zhang
Cavalier Robotics Member
ESC Père-René-de-Galinée
Cambridge, Canada

Levi Kusaula
Cavalier Robotics Member
ESC Père-René-de-Galinée
Cambridge, Canada

*Andrew Mourcos
Cavalier Robotics Leader
ESC Père-René-de-Galinée
Cambridge, Canada
amourcos@gmail.com

***Abstract*—This paper describes the design and construction of an autonomous race-car. The mechanical design, starting from an R/C chassis, was modified and fitted to include required components. It has an interchangeable computer vision system which can utilize either a smartphone OR a microprocessor to compute the data obtained from a single front-facing camera. All software used is open-source. The simplicity and modularity of the project means that it can promote further interest for younger students in the domain of robotics and computer vision.**

Keywords—Open Source, Autonomous, Self-Driving Race Car, Android, Computer Vision, NDK, IARRC, OpenCV, Reverse Perspective, Arduino, PCB.

I. Introduction

The potential of autonomous technology is ever-growing in this 21st century. As a result, our contribution to this field will be to make it accessible to kids our age to promote further interest. The future of autonomy lies in the future of humanity: our youth. Following this goal, our car must be simple and cheap to build.

This document is the final report of the Cavalier Robotic's project to build a self-driving race car that will participate in the 2018 International Autonomous Robot Racing Challenge (IARRC) at the University of Waterloo.

We appropriately named our car *The Kinematic Autonomous Car Having Obvious Worth*, or K.A.C.H.O.W for short, referencing a catchphrase from the 2006 film, *Cars*.

The purpose of this competition is to promote further research into vehicle autonomy. The car in question was created to participate in 3 main events — design, circuit race and drag-race — while adhering to the main objectives listed

1) High speed vehicle localization;
2) High speed vehicle control on different surfaces;
3) Stop light and roadway detection;
4) Collision avoidance with static and dynamic obstacles.

This report presents a detailed outline of the entire project, explaining the mechanical design, electrical layout and software aspects of the car.

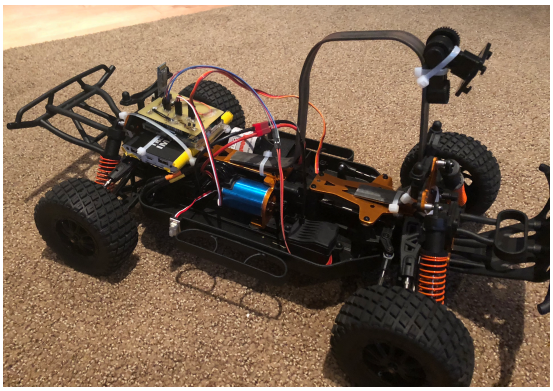

Figure 1 — the chassis

II. Mechanical Design

A. Chassis

The starting point for the car is a 1/10th scale DHK HUNTER hobby r/c chassis to which certain modifications were made. Firstly, the Electronic Speed Controller (ESC), and the DC motor were replaced with a stronger brushless motor system. Secondly, using a piece of iron, a n-shaped structure (see figure 1) was bent by hand. This structure was secured to the spine of the car. It works as an elevation point to (1) hold the camera for the computer vision, (2) house the emergency stop button, (3) protect the on-board electronics in the event of a roll and (4) act as a handle in case the car needs to be caught.

Thirdly, the shocks of the car were replaced with thicker springs to account for the added weight of the electronics.

This chassis has many structural advantages:

- Large base allows for enhanced stability;
- Flat chassis makes mounting components easy and increases usable surface area;
- Adjustable shocks allow for easy changes in case of variations in car weight;
- Front and rear bumpers for shock absorption upon accidental collision with pylons;
- Velocity of up to 35km/h;
- Four-wheel drive to avoid slippage (high speed vehicle control on various surfaces);
- Modularity — even though the car is complete, it allows for several future additions.

Next, a smartphone holder was secured onto the iron ‘n’, which will be used for the autonomy. Additionally, several aesthetic modifications were made to make the car resemble Lightning McQueen (a fictional movie character). In fact, a shell for the car was made using paper-mâché and painted to conform to the character’s design (see figure 2).

Figure 2 — K.A.C.H.O.W with decorative shell

III. HARDWARE

A. Processing

Prioritizing accessibility of materials, we used a readily-available smartphone device as our main processing unit. It is propped up on a support that angles the camera towards the road.

The specific model is a Samsung Galaxy S5 which encompasses a suite of features and sensors:

- 2GB of RAM;
- 2.5GHz quad-core Snapdragon processor;
- 16MP camera;
- Bluetooth 4.0;
- Accelerometer, Gyroscope, Proximity and Compass sensors (for inertial measurement);
- Ambient light sensor (for autofocus).

A phone was also used instead of a conventional processor since it provided the features mentioned above, thus removing the need to have an Inertial Measurement Unit (IMU) and a camera system separately.

An android application monitors the road and the inertial sensor readings (for vehicle localization), then decides what course of action to take. The final decision is sent over bluetooth via a HC-04 module to a microcontroller (MCU). The specific MCU used is an Arduino because of it's commercial availability. It uses an ATmega328P chip with 32KB of flash memory. The MCU in turn, is connected to the car's servo motor for steering, the ESC for throttle control and a radio receiver. The radio receiver listens for a stop command issued when someone pulls the trigger on the wireless emergency stop.

C. Circuit

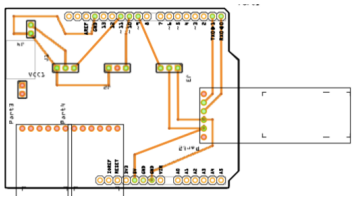

Figure 3 — CAD model of PCB

As part of the design, our team created a printed circuit board (PCB) to connect all individual components (bluetooth, radio, servo & ESC) to the Arduino in an organized manner. The circuit in question was designed to be a “shield" for the

Arduino (see figure 4), meaning that it has header pins that slide directly on top of the Arduino, like a hat. The first rendition of the design was created using EagleCAD, but was later remade using Fritzing (see figure 3). The design was printed on glossy paper, then ironed onto a single-sided copper board. As an experiment, two boards were dipped in two different solutions.

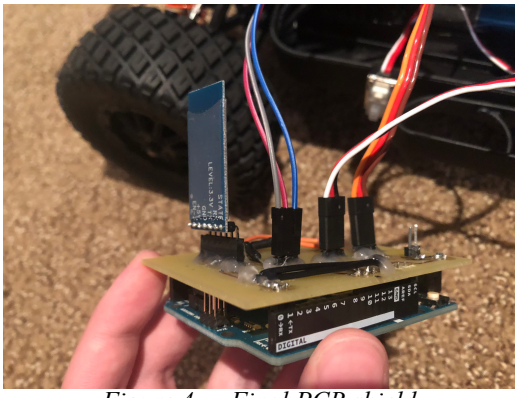

Figure 4 — Final PCB shield

The first experimental solution was made with common household chemicals: hydrogen peroxide (H_2O_2) and diluted acetic acid (CH_3COOH, also known as vinegar). For our theory, we came up with the chemical equation:

$$2CH_3COOH + H_2O_2 + Cu \rightarrow 2CH_3COO^- + Cu^{2+} + 2H_2O \quad (1)$$

Note that in (1), there is a formation of copper ions and we noticed that the reaction would not last long enough to finalize the etching. This was probably due to the copper ions saturating the reaction. We were able to bypass this problem by adding table salt (NaCl) periodically, which dissociated into it's ions Na+ and Cl-, which bonded with the loose copper ions. According to Le Chatelier's Principle, this pushed the equilibrium point forward, allowing the reaction to continue. This method was surprisingly functional and managed to etch the board to an extent. After several hours in the solution, we concluded that it works, but was not worth the wait, so we passed to the next solution.

The second solution was Ferric Chloride-based, which is a much more conventional acid for circuit etching. This time, the process took less than 40 minutes.

D. Power

In order to have a reliable system, we need reliable power. Firstly, the smartphone relies on it's internal battery which should last several hours in the created application with 2800mAh. Next, the Arduino and it's peripherals are powered via USB connection to an 8000mAh power bank. Finally, the servo motor has it's own separate battery pack.

IV. SOFTWARE

The app that runs on the phone is made superficially in Java, but was set up to use Android's native development tool (NDK). This allowed us to use native code — C++ — for the the backend and simply communicate to the frontend using JNI. As this was our first venture in app development, Java and C++ for computer vision, it proved to be a big challenge to develop.

To have an efficient algorithm, we implemented certain functions from the OpenCV library for C++ into our app.

A. Traffic light detection

To detect the traffic light state, we use the camera feed from the smartphone. From the frames obtained, we convert the image to a hue-saturation-value (HSV) format. This allows us to easily select a colour range for the red traffic light. Next, we apply a mask that deletes everything in the image that doesn't comply with the HSV colour range defined for the red traffic light. Then, we use simple blob detection to find the light in the mask. The method used was a variation of Laplacian of Gaussian (LoG). Blobs were then filtered by area and circularity. Then, once a blob-counting function detects zero blobs, it means the red traffic light turned off and the race has begun. This signals to our main program to begin the redline detection.

B. Road line detection

From the camera feed of the smartphone, we extract road lines in several steps, the first being converting the input video to a matrix of grayscale pixel values.

1) A region of interest is selected, thereby cropping out unnecessary pixels.

2) A blurring function is applied. While a Gaussian blur is preferable, a different method was required to economize processing capabilities (see figure 5).

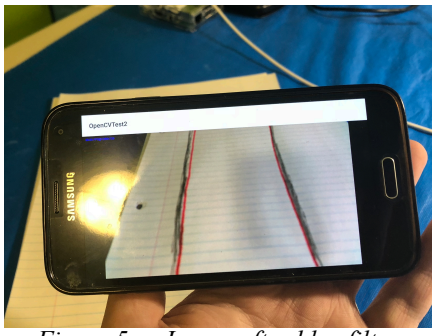

Figure 5 — Image after blur filter

3) Canny edge detection — finding intensity gradients, applying non-maximum suppression, applying double threshold, suppressing all insignificant edges (see figure 6).

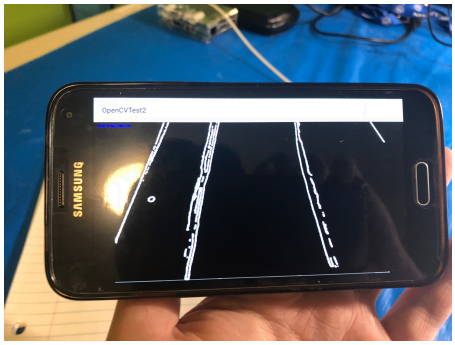

Figure 6— Image after edge detection

Then, we used a Hough Line Transform in order to detect the road lines (see figure 7).

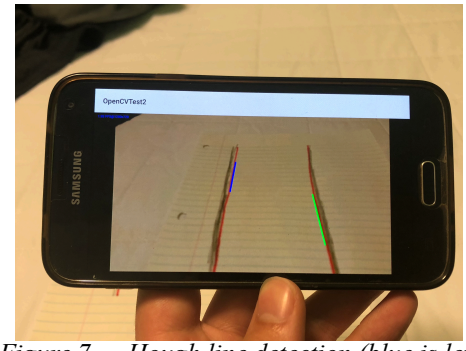

Figure 7 — Hough line detection (blue is left line, green is right line)

It uses Hesse normal form to describe equations of lines:

$\rho = x\cos\theta + y\sin\theta$.

Where ρ is the distance from the origin to the closest point on the line and θ is the angle made by ρ and the x-axis. This means that we can represent a line in this form using 2 parameters (ρ, θ). We then create an array/accumulator of these parameters and using a system of "votes", we determine the rho and theta describing the best road line.

After converting the (ρ, θ) coordinates to (x, y) pairs, the lines' slopes can easily be extrapolated with the following equation:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

For a mono-camera setup, two perfectly straight lines in a picture seem to be angled due to a camera's single-point perspective. Using this knowledge, the program can determine that a line is the right road line if it has a negative slope or that it is a left road line if it has a positive slope (see figure 8).

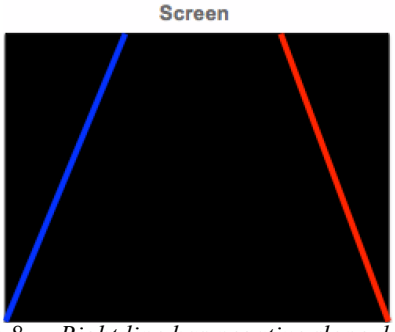

Figure 8 — Right line has negative slope, left line has positive slope

Now, to establish a course of action, the program needs to know what the lines look like in real life, not what they look like in this single-perspective. This requires a "perspective reversal algorithm". Our program uses the Ilyas method, which is described in our last report for IARRC-2017 (see works cited).

Summary of the method: the program feeds a pair of (x, y) coordinates, representing a pixel on one of the lines, into a 2-dimensional function that triangulates it's position in the real-world. It returns an (x, y) pair describing a "bird's eye view" of

the road line. This is a model representing the function in the Y direction

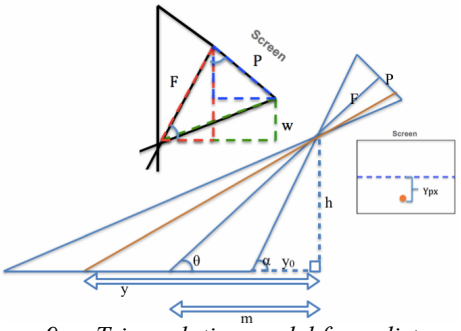

Figure 9 — Triangulation model for y-distance of a point in front of camera

This is a model representing the function in the X direction:

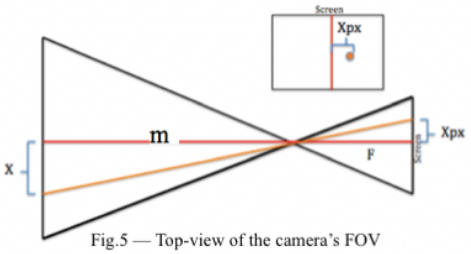

Figure 10 — Triangulation model for X-distance of a point in front of camera

If every pixel in the image has its perspective transformed (which is highly inefficient, this is only for example purposes), it would look like the following:

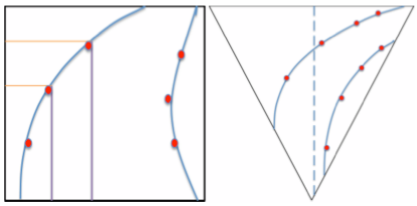

Figure 11 — Reversing the single-point perspective of a camera using the Ilyas method

After feeding two points from each line into the function, we are given two points in real life with which we can calculate their slopes. Using the arctangent on the slope will give us the angle of the road in reference to the car's direction. This value is then sent to the MCU over serial bluetooth connection.

C. *Obstacle avoidance*

The last two objectives for the IARRC is obstacle avoidance with static objects and other vehicles. To solve this challenge, we positioned 2 ultrasonic sensors (HC-SR04) on the front of our car. They each have a measuring angle of 15 degrees and can measure up to 4m ahead. These distance sensors send out "pings" that travel through the air until they hit an object and rebound back towards the sensor. Counting the time it takes for the "ping" to go and come back, we divide that time by 2 and use the following equation to determine how far away an obstacle is:

$$d = v*t$$

Where ***d*** is the distance, ***v*** is the speed of the "ping" (~344m/s) and ***t*** is the time it took.

We use the distance measurement to stop the car in the case of a frontal collision.

D. *Microcontroller*

The MCU, runs a C/C++ program. It waits for data to be obtained via serial communication with the radio device and the bluetooth module. The program parses the incoming bytes into 2 containers: motor speed and servo direction.

After receiving a coded message from the radio or a distance measurement from the ultrasonic sensors, it knows that it will either have to set the ESC to brakes or allow it to continue. Also, after receiving the value for the angle of the road line, it will set the servo motor to that angle to stay parallel with the road.

V. CONCLUSION

The mechanics of the car are simple, yet reliable and sturdy. The hardware used is inexpensive, however is capable of supporting our use and promotes accessibility amongst fellow high school students. The mechanical system has excelled in our crash tests, speed tests and emergency stop tests. The electronics have proven to be quite useful, especially with the design of the PCB, which keeps all components reliably organized. Finally, the software is unique in the sense that it is mobile, however it has certain drawbacks. Certain tradeoffs were made in order to have a functional system,

however performance could easily be increased with the use of a newer smartphone, or with a different micro computing device such as a raspberry pi. We currently have a raspberry pi on standby with android installed. If we see a major difference in performance between our app on a phone and the same app on the raspberry pi, certain modifications may be made.

In the creation of this car, our whole team benefitted from several engineering learning experiences. We definitely got to challenge ourselves in many ways by trying things for the first time. In fact, none of us have ever made mobile apps before, it's the first time we have programmed using Java or C++ for NDK, we learned how to design PCBs, etc.

Overall, our system is big step forward from last year and has brought us an array of new skills.

VI. Appendix

Table I : Cavalier Robotics Team Members for IARRC

Member Name	Role
Andrew Mourcos	Project lead & software lead
Jacob Hergott	Mechanical lead
Joel Zhang	Electronics lead
Sophia Mourcos	Design lead
Levi Kusaula	Research lead

Table II : Budget

Item	Real cost	Cost to team
Samsung Galaxy S5	$200	Personal donation
Arduino	$30	Personal donation
1/10th scale Hunter DHK + upgrades	$300	300
Bluetooth module	$5	Personal donation
Copper board + Acid	$30	30
Phone holder	$5	Personal donation
Raw materials	$10	School donation
Total:	$580	$330

Acknowledgments

We would like to thank the other members of Cavalier Robotics who worked on a different project with us: Gabriel Quintana, Guillaume Fernandes, Jacob Chaussé, Jennifer Rhett, Jonah Schuck, Lyne Baaj, Gabriella Kik, Pavly Fayek, Renée-Tipler Corpuz. Additionally, we would like to thank the IARRC team for creating such an event.

References

1. Mourcos, Andrew et al. *Cavalier Robotics: Simply Autonomous Driving*. report. June 2017