IARRC 2018 Design Report

Véhicule Autonome Université Laval (VAUL)

***Abstract*— This report presents *Phantom*, the autonomous RC car conceived by the *Véhicule Autonome Université Laval* (VAUL) project. Its physical and software design are described. The key algorithms used to meet the IARRC competition objectives are explained. Experiments demonstrate that the car is safe and fully capable of competing in the drag race. We lay down the milestones left to achieve full competitiveness by next year.**

I. Introduction

Since its inception in the summer of 2017, the Véhicule Autonome Université Laval (VAUL) project has been hard at work to design and manufacture a fully capable autonomous RC car. The enthusiasm of both graduates and undergraduates from the Science and Engineering faculty, along with the support from professors François Pomerleau and Philippe Giguère from the Computer Science department, has allowed us to make significant progress to this end. Our efforts came to fruition, and we are proud to present our vehicle *Phantom* at IARRC 2018.

This report documents how we designed our vehicle for the drag race challenge, as well as some future plans to implement the more challenging circuit race. We hope this incremental approach will allow us to learn from our participation at IARRC 2018 and present a fully competitive vehicle next year.

We first review the physical design of our vehicle (section II), before discussing the software architecture we embedded on it (section III). We describe how we meet the IARRC competition challenges in section IV. Finally, section VI concludes the report and describes some of the challenges which VAUL will face further down the road.

II. Hardware design

The hardware design of *Phantom* is based of a Desert Buggy XL-E (see figure 2), which by itself is a highly capable RC car. We heavily modified the platform to embed our sensors and equipment on it. This section first describes the hardware we mounted on the vehicle, before describing our implementation of the security requirements of the competition. Then we describe our sensors in more details. Finally, we graduate to more high level hardware considerations such as vehicle status monitoring, the computer systems mounted on the vehicle, and how these devices communicate through networking.

A. In a nutshell

Phantom is equipped with the following items:

- Dynamite Fuze 1/5 6-pole brushless motor (800kV)
- (x2) 5000 mAh 22.2V LiPo batteries 6S
- S900S steering servo
- SRS4220 RF receptor 2.4GHz
- VESC electronic speed controller
- STM32F407 board data acquisition system
- Jetson TX2 on board computer
- (x2) DC Converter 5/12VDC 5A
- Archer C7 dual-band Wi-Fi router.
- RSX-UM7 Orientation sensor
- Leopard Imaging IMX185 camera

B. Security systems

1) Mechanical E-Stop: A mechanical E-Stop button is mounted on the top of the vehicle at a height of more than 30 cm. Figure 3 shows the button. When pressed, it cuts the motor's power supply. To re-activate it, one needs to twist the button counterclockwise. A red status LED indicates whether the motors are armed or not.

2) Wireless E-Stop: We re-purpose the RC transmitter shipped with the Desert Buggy XL-E as a dead man's switch. This is a low cost alternative to an off the shelf remote E-Stop button. When the throttle button is pressed, the vehicle can receive commands. When it is released, the vehicle immediately brakes. This provides additional security, since the vehicle now needs two operators: one that connects to the vehicle via Wi-Fi and sends commands, and another that monitors the vehicle behavior and holds the dead man's switch.

Please note that in autonomy scenarios, this RC transmitter cannot be used to control the vehicle and acts solely as a dead man's switch.

3) Maximum speed limitation: The maximum speed limit is guaranteed by several sub-systems with their own responsibility. First, the electronic speed controller is limited by a maximum RPM value for brushless motor (BLDC) drive. Using the gear ratios and the size of the wheels, we can use this RPM limitation to respect the 10 m/s requirement of the competition.

Another way to is the battery pack selection. We used a LiPo battery 6S to power brushless motor which limits the maximum speed. Finally, the embedded regulator (PID) on STM32F4 is limited at 10 m/s. If the regulator was to receive a higher speed, it would automatically bring it down to an acceptable command.

C. Sensors

This section describes the variety of sensors integrated into our vehicle.

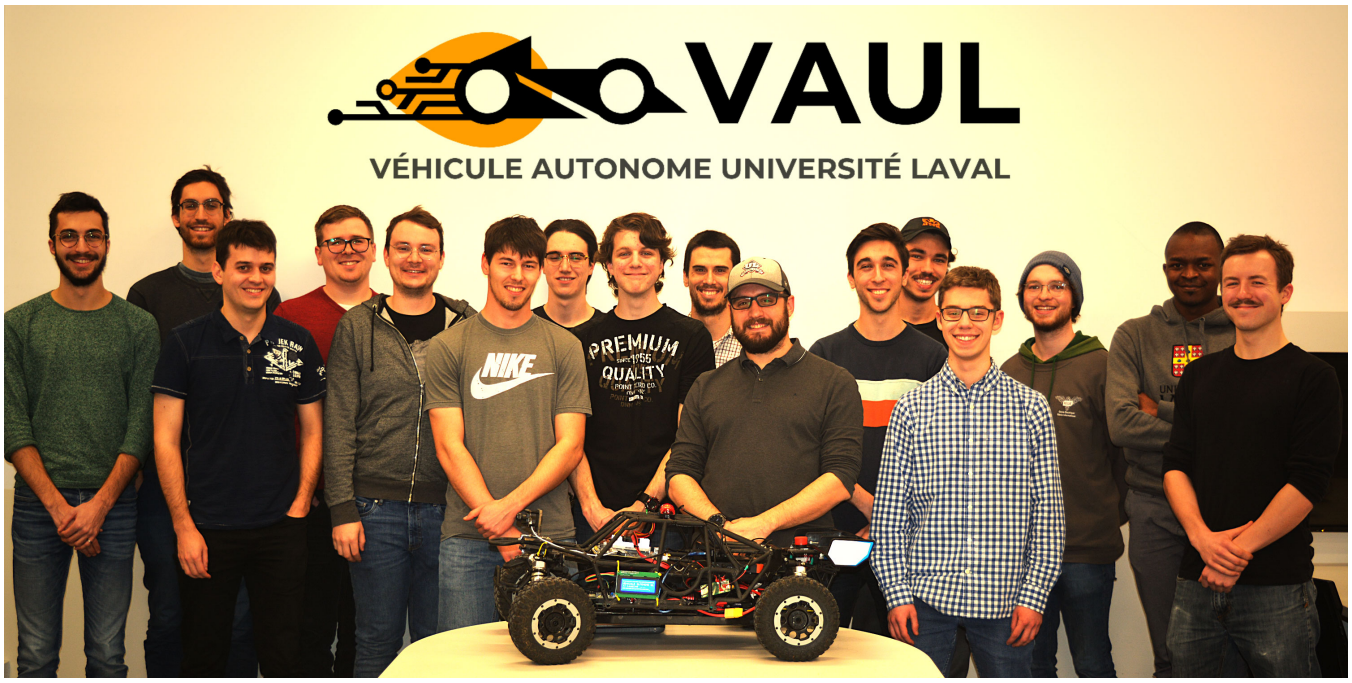

Fig. 1. Group photo!

Fig. 2. Desert Buggy XL-E

Fig. 3. Mechanical E-Stop. The motor power status is indicated by the red LED visible on the bottom of the figure.

1) Camera: We use the Leopard Imaging IMX185. It can capture 1080p videos at 60FPS. Since it is a solid-state camera, the is no rolling shutter effect in captured videos. In practice, we only use a 720p resolution because it is sufficient for both line detection and cone detection applications. It can be connected to the TX2 (see section II-E) for maximal performance.

The exposition time of the camera is adjusted automatically by its driver on the on-board computer.

2) Odometry: We mounted two quadrature encoders made with custom 3D printed parts on the rear wheels of *Phantom*. The parts consist of a press fit wheel gear, an encoder support and a second gear to drive encoder (ratio 3.6:1). The parts are seen in Figure 4. The modular incremental encoder has 2048 quadrature resolution and the quadrature signal increments a hardware counter is embedded on the data acquisition board. The latter has a real-time thread that computes the speed of the vehicle at a 100Hz frequency for each rear wheel.

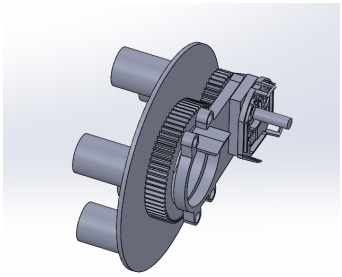

Fig. 4. 3D printed Wheel encoder support for quadratic encoder

3) Sonar range sensor: We began the design of a low range obstacle detection system (2-3m) around our vehicle with 8 low cost sonars. This feature is necessary for the circuit challenge to detect other vehicles and have an avoidance strategy. We selected the *Paralax 28015* sensor with 20° horizontal and 30° vertical angles of detection.

4) IMU: The UM7 Orientation Sensor is an Attitude and Heading Reference System (AHRS) that contains a three-axis accelerometer, rate gyro and magnetometer which communicate through a USB interface with the onboard computer.

It combines this data using an Extended Kalman Filter (EKF) to produce better estimates. A ROS node handles EKF parameters and publishes the estimate data.

D. Monitoring

Figure 5 represents three three LCD displays which we use for monitoring. On the larger display, the embedded system shows information such as current vehicle operation mode, speed, battery status and encoders status. The two other displays give real-time current and voltage information about the voltage converters. This debugging information is a crucial help during development.

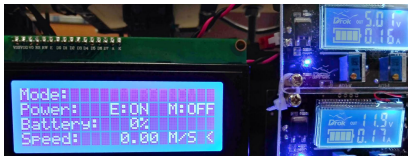

Fig. 5. LCD displays on the side of the vehicle

E. Computer systems

There are three embedded computers on *Phantom*.

The first one is a low-level controller programmed on a STM32F4 micro-controller as Master/Slave model with the high level computer (master). It is responsible for the acquisition of the data from the encoders, sonar sensors and RF receiver. It displays on-board information to the LCD and drives a red light to indicate vehicle status. It is also responsible for driving the second on-board computer, an Electronic Speed Controller (ESC) that exclusively deals with driving the brushless motor and the steering servo.

The last computer is an NVIDIA Jetson TX2 prototyping board. It is responsible for the high-level operation of the vehicle, such as planning and vision. It runs *Robot Operating System* (ROS) automatically at boot. The overall software architecture that we programmed on it is discussed in section III. The Jetson is interfaced with the low-level controller using a serial connection. It is also connected directly to the camera. Its embedded GPU allows use to do quick image processing, for tasks such as line or cone detection. The GPU also allows us to use deep neural networks into our pipeline, as discussed in subsubsection IV-A.2. Since the camera outputs images directly into the GPU memory of the Jetson, we avoid the latency associated with data transfers to the GPU.

F. Networking

We mount a TPLINK AC1750 router directly on the vehicle. The router emits its own Wi-Fi, so we never have to worry about the vehicle getting out of the range of a base station. SSH access to the on-board computer is possible through this network. This configuration allows for an easy access to the software.

III. Software design

The software architecture of *Phantom* is articulated around ROS [6]. It helps us manage the different information coming from sensors or algorithms, and helps with the separation of concerns in software design. ROS provides libraries and tools that we used to develop the software solution for our car. The information in this framework is represented in messages that have a specified format and then sent to a topic, which other programs or algorithms can read and use. This lets us manage all the information that we need for the race car in an efficient way.

A. ROS Nodes

In ROS we designed two nodes that are worth mentioning here. Firstly, there is the `phantom_base` node that manages the interactions with the vehicle's on-board controller. This node is in charge of sending commands to the wheels and receiving data from the sensors that are not directly interfaced to the on-board computer. The second node is called `drag_race` and is basically a state machine that manages the high-level task. It subscribes to the pertinent information from our various sensors by subscribing to the messages from smaller nodes such as `traffic_light_detector` and `line_detector`. It ensures that the vehicle drives straight and safe until we reach the finish line.

One of the main advantages of using ROS is code reuse. In the future we will design a new node called `circuit_race` that will simply replace the `drag_race` node, while reusing all the other nodes underneath. We will have to replace only the high level behavior, mainly to incorporate collision avoidance and a more intelligent path planning.

B. Gazebo simulation

It is important to test algorithms before running them on the actual car. To this end, we used Gazebo simulation [1]. Gazebo is a simulation tool that uses ROS and that supports several features that are important for this project. For example, this simulator supports physics engines, sensors and noise and all kinds of robot models. The model that we use to simulate our car is the MIT race car model, which is freely available on the Internet. Figure 7 shows a simulated drag race.

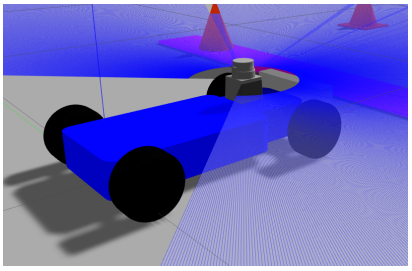

Fig. 6. Simulated race car

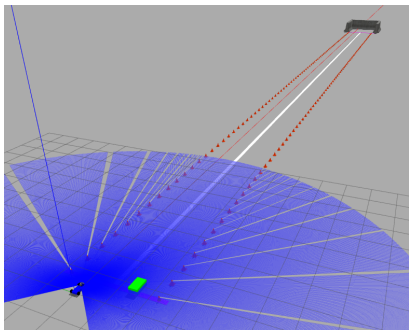

Fig. 7. Simulated drag race

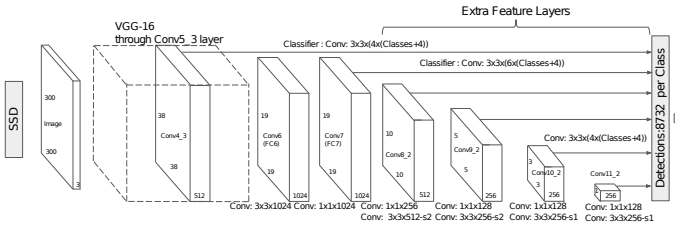

Fig. 8. "SSD: Single Shot MultiBox Detector". Figure from [5]l.

C. Remote control

A small node allows for the teleopration of the vehicle through our own software instead of the off-the-shelf remote. This proved useful to test the embedded systems in charge of controlling the vehicle's motors.

IV. Meeting the 5 IARRC challenges

Using this hardware and software design, we were able to make progress towards some of the five objectives of the competition. This section how our hardware and software innovations work together to meet them.

A. High-speed vehicle localization

There are three main components to our localization pipeline: cone detection using a deep neural network, cone measurement using classical algorithms and roadway line detection.

1) Line detection: Figure 9 presents the pipeline used to detect lines and position the vehicle relative to it. The first part of the pipeline, in gray, is the image filtering. It is done on GPU as it accelerates computations by a factor of 2 to 3 in our use case. Then, to map from pixel coordinates of the binary mask to world coordinates, we use a perspective transformed that has manually calibrated by measuring pixel coordinates of four points that have knows 2D coordinates on the XY plane. To eliminate outliers, we then use RANSAC to fit a line on the projected points. The tight coupling between our camera on-board computer, combined with the presence of a GPU on that computer, allows us to run line detection at around 20 Hz, which is essential to the stability of our controller.

2) Cone detection: Our cone detection method is a two-stage process. First, we use a neural network to have a rough estimation of the location of the cones in the image. Then, we use a classical computer vision technique to refine the bounding boxes of the found cones.

The neural network is based on "SSD: Single Shot Multi-Box Detector"[5] (see figure 8). It uses MobileNet [2] as a feature extractor because it provides good performance given a limited amount of computing power available on the TX2. We used Huang et al.'s Tensorflow implementation. For training, we fine-tune a model pre-trained on MSCOCO[4] on cone bounding boxes that we manually annotated in video frames. To deploy the neural network, we use a custom build of Tensorflow for the TX2 and load a frozen Tensorflow graph[1] in a ROS node.

3) Cone measurement: Even though the neural network gives good estimates of the cones' location, the bounding boxes are not precise enough to project from image coordinates to accurate 2D locations. The second stage of our cone detection algorithm takes the approximate bounding boxes of the cones as input and outputs their 2D location with respect to the camera. This is done in multiple steps by a classical computer vision algorithm. First, we extend each bounding box to make sure the cone is fully visible. Then, we extract an image patch for each one of them and smooth them using a Gaussian blur with a 3x3 kernel to remove some noise. The blurred patches are converted in the CIELAB color space for a better color segmentation. A thresholding operation is done using Otsu's method on the mean of the a and b components of the patches. The resulting binary image is then used for finding contours. The largest one is assumed to be the cones's contour. In order to remove false positives, the candidates are subject to a validation test which verifies the aspect ratio and the area ratio of the contour.

Assuming the camera has been previously calibrated and using the pinhole camera model, it is possible to compute the distance between the object(cone) and the camera using Thales' theorem(or similar triangles):

$$distance(m) = \frac{focallength(px) * coneheight(m)}{coneheight(px)}$$

The angle between the optical axis and the line joining the cone and the optical center can also be computed using basic trigonometry:

$$angle = arctan(\frac{conecenter(px) - imagecenter(px)}{focallength(px)}$$

In practice, the accuracy of the localization depends on the size of the cone in pixels. It varies from several centimeters to several meters. Therefore, we ignore cones that are less than 20 pixels high.

4) Odometry: We combine the signal from our quadrature encoders and the IMU to compute an estimate of the vehicle's displacement. This is used as an initial estimate for our other localization algorithms.

[1] See http://www.tensorflow.org/extend/tool/_developers/#freezing.

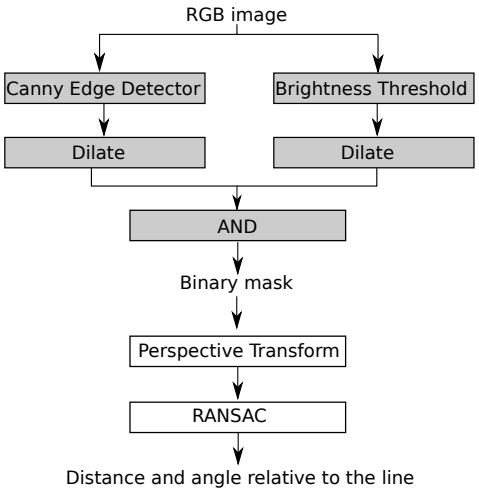

Fig. 9. Line detection pipeline

B. High-speed vehicle control

The vehicle control has two components. The low-level control lives on the embedded systems and is in charge of driving the wheels and steering. This is used by the high-level controller, which lives on the on-board computer. The latter is in charge of sending the correct commands to the low-level controller to complete the drag race and circuit race challenges.

1) Low-level control: The brushless motor (BLDC) is controlled by an ESC named VESC. This is an open source project initiated by Benjamin Vedder. The ESC allows us to control a brushless motor speed with Pulse Position Modulation (PPM) signal from micro-controller (STM32F4). The PPM signal to control speed may be switched with the PPM output from RF module. It's a simple way to separate power supplies for motor and servomotor and other electronic components. Other features like RPM limitation, command speed mapping, battery management and steering powering are implemented on this controller.

2) High-level control: Our high level controller is based on Stanley Method [8]. It is the method used by Stanford to win the 2005 DARPA Grand Challenge [7]. We choose this method because it is easy to implement and gives good results in both simulation and real world drag race context.

Figure 10 represents the geometry model used. Here is a definition of the terms used in the figure:

- path — Path to follow, we want the vehicle to be exactly on that line
- c_x, c_y — Closest point from the vehicle on the path
- e_{fa} — Signed distance between the vehicle's front wheel and the (c_x, c_y).
- θ_e — Signed heading error of the vehicle. Can be viewed as the angle between the camera's optical axis and the path's axis.
- v — Speed vector.
- δ — Steering command.

To determine the steering command δ at time t, there is a simple formula:

$$\delta(t) = k_1\theta_e(t) + tan^{-1}(\frac{k_2 e_{fa}(t)}{v_x(t)}) \quad (1)$$

where k_1, k_2 are adjustable gain parameters.

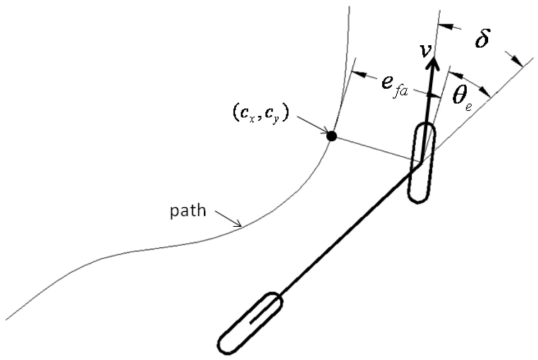

Fig. 10. Geometry model used for control. Figure from [8].

This simple model achieves good results in simulation. However, in real world scenarios, we found that the latency on the computer vision pipeline made control unstable and caused oscillations around the desired path. We are able to reproduce this issue in simulation by introducing an artificial delay on the simulated localization. To fix this problem, we use predictive control in conjunction with the Stanley Method. Instead of directly using the last observation of θ_e and e_{fa}, we can use a locomotion model to estimate the true position at time $t + \Delta t$, where Δt is the time passed since the observation, and use this position in the controller.

We have a state vector

$$u_t = \begin{bmatrix} V_t \\ \omega_t \end{bmatrix} \quad (2)$$

where V_t is the linear speed of the vehicle and ω_t is its angular speed at time t. Knowing the distance between the front and back wheel L and the steering angle α, we can compute the turn radius R and ω_t using

$$R = \frac{L}{tan\alpha} \quad (3)$$

$$\omega_t = \frac{L}{tan\alpha} \quad (4)$$

Fixing the coordinate frame at the front of the vehicle (meaning that x_t, y_t, θ_t are all 0), we can compute u_{t+1}:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} \frac{V_t}{\omega_t} \sin \omega_t \Delta t \\ \frac{V_t}{\omega_t} - \frac{V_t}{\omega_t} \cos \omega_t \Delta t \\ \omega_t \Delta t \end{bmatrix} \quad (5)$$

Given this position, we can find the new angle from the line using $\theta_e - \theta_{t+1}$. We can also compute the updated e_{fa} by re-projecting x_{t+1}, y_{t+1} the tangential line to the path at (c_x, c_y).

In simulation, this predictive method did not improve the stability of the control. We have two main hypotheses to explain it:

1) We currently have only a rough estimate of the steering angle α at a given time. It takes some time to the servo to steer from an angle to another. This makes the computation of ω inaccurate. In the future, we will use an IMU to have a better measure of ω.
2) The Stanley Method is very sensitive to small oscillations of e_{fa}. In the future, we would like to experiment the Pure Pursuit method [8] in conjunction with the predictive control described above.

C. Stop light and roadway detection

Since we implemented only the drag race for this year, there was no need for an explicit roadway detection. This is one of our future objectives. However, we are proud to report that our traffic light detection is operational.

1) Traffic light detection: To detect the start signal of the traffic light, we begin by converting the current frame into grayscale. We then apply a binary threshold operation on the pixel intensities to highlight the brightest regions of the image. A closing morphology operation is used in order to close the remaining small holes inside these regions. We can now find the contours using the resulting binary image. At this stage, the list of detected contours should contain the red (or green) light, but also possibly several other contours. To reduce the number of potential candidates, we filter the contours using multiple constraints (hierarchy, area, roundness) to keep only those that look like a traffic light.

To actually detect the start signal, namely the transition from the red light to the green light, we need to keep track of the filtered contours at each frame. As soon as one of them is lost, we search in the consecutive frames for a new one located below the old one. If that is the case, we send a start signal to the system.

The algorithm suffers from the brittleness of classical vision algorithms. However, it still produces great results once the different thresholds have been correctly determined.

2) Finish line detection: We expect that our line detection algorithm from section IV-A.1 will work just as well for the purpose of finish line detection. The main challenge is to apply the thresholding in some colorspace instead of grayscale.

V. Experiments

This section describes how we empirically validated our vehicles behavior.

A. Simulation

We use our simulator to have a (partial) validation of our controller. We register the control error during a simulated drag race and validate that the controller is stable, at least in simulation. An example drag race run can be visualized in Figure 11. In this example, we skew the orientation of the vehicle at the start and validate that the controller is able to compensate. Since we merge the robot frame and the line frame in that example, the pose y and θ poses of the robot are equivalent to the errors in those axes.

TABLE I
Deceleration results

Speed (m/s)	Braking distance (m)			
	Trial 1	Trial 2	Trial 3	Average
2	0.6	0.7	0.7	0.67
5	2.0	2.1	2.0	2.05
10	4.3	4.0	4.2	4.16

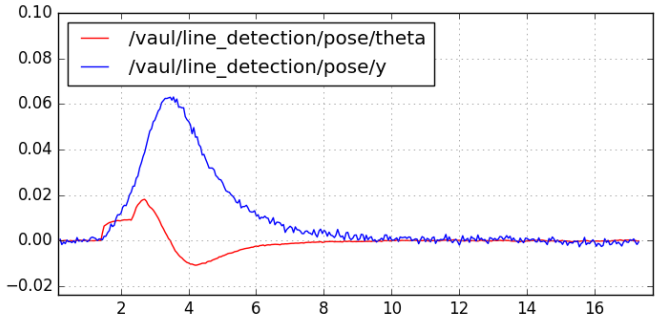

Fig. 11. Distance and orientation error from simulation. The controller was robust enough to bring the vehicle back on the correct path in this instance.

B. Deceleration

We did deceleration test using IARRC's safety qualification protocol (section *Autonomy*).

1) Drive at $x\,m/s$ for $10m$ in a straight line.
2) Release the dead man's switch (see section II-B.2) when the vehicle arrives at the $10m$ mark.
3) Wait for the vehicle to stop.
4) Measure the distance between the $10m$ mark and the front wheel of the vehicle.

We performed tests at $x = 2m/s$ (as stated in the IARRC rules), but also at speeds of $x = 5m/s$ and $x = 10m/s$. For each speed, we did three trials. Table I presents the results.

C. Drag race

At the time of writing, we did some preliminary tests with our drag race software. The results can be seen at https://www.youtube.com/watch?v=bUVCU9WQ97k. After this particular test we started investigating possible reasons for the unstable control, including line detection latency and inappropriate gain parameters.

D. Cone detection

To evaluate the cone detection network, we use the Intersection over Union (IoU) metric. It is computed as follows:

$$IoU = \frac{A \cap B}{A \cup B} \tag{6}$$

where A is the bounding box predicted and B is an annotated ground truth. We consider a prediction positive if the IoU metric is over 0.5.

For now, since we have very few annotated data, we evaluated the network on the training set only which contains 312 images. We obtain a precision of 88.5%. Note that we would not benefit from splitting this dataset into a training set and a validation set since all images are from the same environment.

We also run the network on the Jetson TX2. We achieve a detection rate of 15 FPS.

These results are only preliminary. We currently do not use the cone detection network for the drag race module as we found the line following method to be sufficient. For future work, we will integrate the cone detection to the circuit module. Until then, we need a bigger data set to improve generalization to unseen environment.

Figure 12 shows qualitative results of the cone detection network on a training image.

Fig. 12. Qualitative result of the cone detection network on a video.

VI. Conclusion and Future Work

This year was filled with challenges for VAUL. We had to create our vehicle from scratch as well as secure the position of our project with the help of sponsors and faculty representatives. Although we have not achieved as much as we wanted, because we are not able to compete in the circuit race, we are still very proud of what has been done so far. Our participating at IARRC this year will serve as a strong basis for the next steps. Using the experience from this year's competition, we intend to refine our vehicle design and implement new algorithms to make *Phantom* fully competitive in a circuit race. One of our priorities will be to implement a planning algorithm to navigate a circuit race using the cone detection software we have already developed. This planning algorithm will have to implement collision avoidance strategies using our sonar sensors. One of the requirements for the planning algorithm is to develop an explicit roadway detection. Finally, we also want to improve the physical design of *Phantom* to make it safer to drive and easier to experiment with. With the ever so strong enthusiasm of our student members, we intend to tackle these challenges head on. Stay tuned!

References

[1] *Gazebo simulation*. URL: http://gazebosim.org/.

[2] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[3] Jonathan Huang et al. "Speed/accuracy trade-offs for modern convolutional object detectors". In: *CoRR* abs/1611.10012 (2016). arXiv: 1611.10012. URL: http://arxiv.org/abs/1611.10012.

[4] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[5] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[6] *ROS Documentation*. URL: http://wiki.ros.org/.

[7] Thrun Sebastian et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of Field Robotics* 23.9 (), pp. 661–692. DOI: 10.1002/rob.20147. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20147. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20147.

[8] Jarrod M Snider et al. "Automatic steering methods for autonomous automobile path tracking". In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08* (2009).