# **BÁO CÁO TỔNG KẾT CUỘC ĐUA SỐ 2019**

Vũ Tuấn Hải – Destiny Team

1/2019

Contents

[**1. Tổng quan** 3](#_Toc535738732)

[**1.1. Thể lệ cuộc thi** 3](#_Toc535738733)

[**1.2 Timeline** 4](#_Toc535738734)

[**2. Cài đặt môi trường để code, debug và chạy mô phỏng** 4](#_Toc535738735)

[**2.1 Ubuntu** 4](#_Toc535738736)

[2.1.1. Cài ubuntu 4](#_Toc535738737)

[2.1.2. Làm quen với Ubuntu 4](#_Toc535738738)

[2.1.3. Một số phần mềm hỗ trợ 5](#_Toc535738739)

[**2.2 ROS & Catkin** 5](#_Toc535738740)

[2.2.1. ROS là gì? 5](#_Toc535738741)

[2.2.2. Một số thành phần cơ bản của ROS 5](#_Toc535738742)

[2.2.3. Các lệnh về ROS 5](#_Toc535738743)

[2.2.4. Nguồn tự học 5](#_Toc535738744)

[**2.3. Cmake** 5](#_Toc535738745)

[**2.4. VS Code** 5](#_Toc535738746)

[2.4.1. VS Code là gì? 5](#_Toc535738747)

[2.4.2. Mẹo với những file json 5](#_Toc535738748)

[2.4.3. Extensions nên cài 6](#_Toc535738749)

[**2.5. Phần mềm mô phỏng** 6](#_Toc535738750)

[2.5.1. Tổng quan 6](#_Toc535738751)

[2.5.2. Đặc điểm về xe 6](#_Toc535738752)

[2.5.3. Đặc điểm về môi trường 6](#_Toc535738753)

[**2.6. Những điểm cần phải lưu ý** 6](#_Toc535738754)

[**2.7. Source code: toàn bộ source code được mình public trên github tại:** 6](#_Toc535738755)

[**3. Quá trình thực hiện** 7](#_Toc535738756)

[**3.1. Tuần 1** 7](#_Toc535738757)

[**3.2. Tuần 2** 11](#_Toc535738758)

[**3.3. Tuần 3** 13](#_Toc535738759)

[**3.4. Tuần 4** 13](#_Toc535738760)

[**3.5. Tháng 12** 14](#_Toc535738761)

[**4. Về source code** 14](#_Toc535738762)

[**4.1. Ảnh là gì?** 14](#_Toc535738763)

[**4.2. Nhận diện lane** 15](#_Toc535738764)

[**4.3 Điều khiển xe** 17](#_Toc535738765)

[**4.4. Nhận diện biển báo** 18](#_Toc535738766)

# **1. Tổng quan**

Cuộc Đua Số (hay FPT Car) là cuộc thi hàng năm được tập đoàn FPT tổ chức về chủ đề lập trình xe tự lái, 2018 - 2019 là năm thứ của 3 cuộc thi. Các anh chị trong khoa đều tham gia nên nhóm tụi mình cũng rất háo hức chờ đến lúc cuộc thi phát động.

Cuộc thi ra mắt trước khi thi giữa kì 1 tuần, cũng là lúc tụi mình gặp khá nhiều rắc rối khi phải làm quen với nhiều deadline, đồ án lớn nên cũng có hơi lưỡng lự không biết có nên đăng ký hay không. Ngay sau khi thi giữa kì xong thì mình bất ngờ nhận được tin nhắn từ thầy Tuấn Anh dạy LTTQ, thầy khuyến khích tụi mình đi để lấy kiến thức & kinh nghiệm, thầy còn miễn đồ án để tạo điều kiện cho tụi mình thi nữa. Do nhóm có 5 thành viên mà tối đa là 4 người/ đội thi nên chia làm 2 đội giống như nhóm đồ án luôn, nhóm mình tên Destiny bao gồm mình, Lịch và Đông.

## **1.1. Thể lệ cuộc thi**

Sau khi đăng ký, mình bắt đầu tìm hiểu về thể lệ thi vòng loại trường trước:

<https://cuocduaso.fpt.com.vn/en/the-le-0>

Vòng trường gồm 3 phần thi:

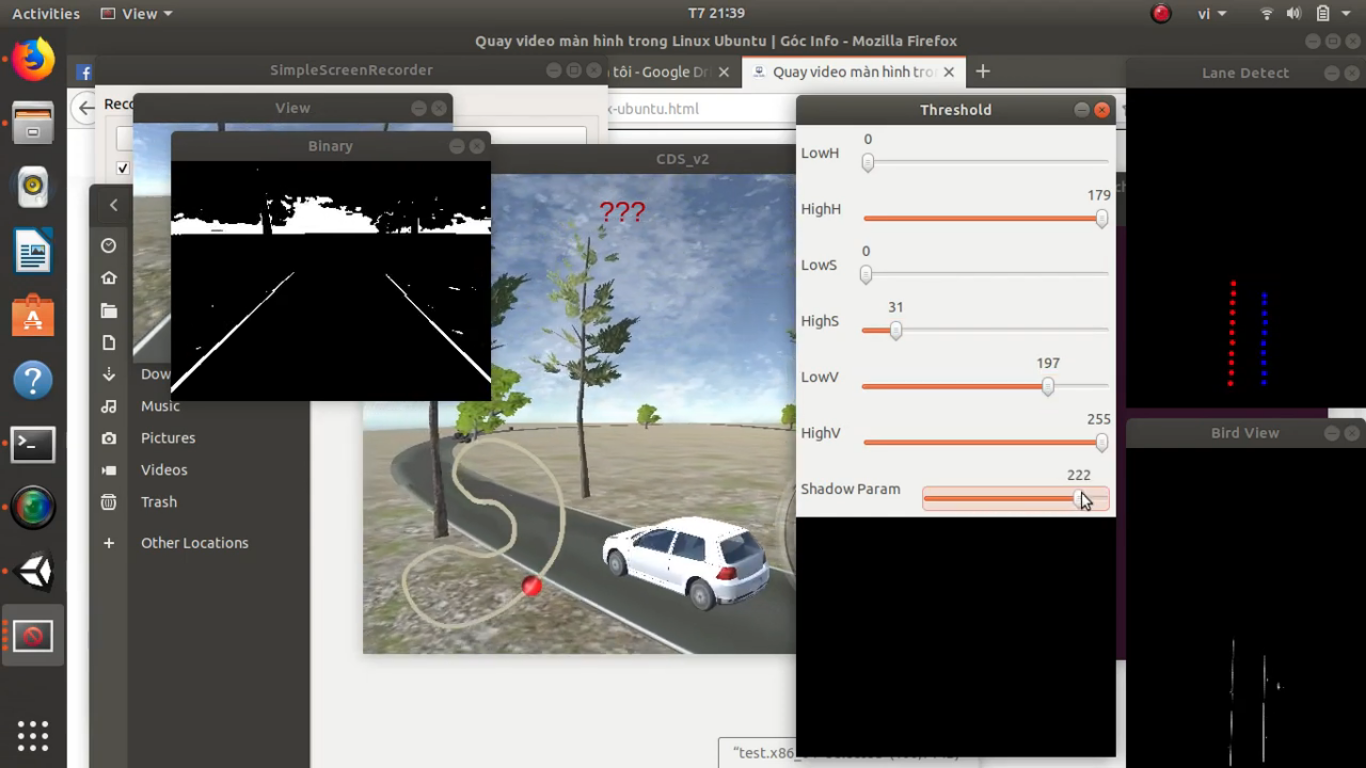
- Vòng 1 - năng lực xử lý ảnh: đội thi được cung cấp một phần mềm mô phỏng và nhiệm vụ là phải lập trình theo chuẩn ROS (Robotic operating system) để xe tự chạy trong môi trường giả lập, bao gồm (đi theo làn, tránh vật cản, nhận diện và tuân theo biển báo rẽ trái - phải, đi được trong điều kiện môi trường thay đổi).

- Vòng 2 - phản biển: trình bày, vấn đáp về thuật toán ở vòng 1.

- Vòng 3 - lập trình nhanh: giải các bài toán về xử lý ảnh.

Nhận xét về thể lệ năm nay: đề thi có độ khó tăng hơn so với năm ngoái (năm ngoái vòng trường chỉ có nhận diện biển báo).

Ban tổ chức chuẩn bị sẵn code mẫu để thí sinh có thể hình dung về cuộc thi tốt hơn, code mẫu chỉ có tác dụng tạo web socket, detect lane và điểu khiển xe cơ bản, còn lại chưa tối ưu. Ban tổ chức cũng cung cấp nhiều tài liệu đào tạo về ROS và ML nhưng bản thân mình cảm thấy không có nhiều hữu ích lắm.



## **1.2 Timeline**

Thời gian từ lúc đăng ký đến lúc dự thi khá gấp rút (từ 5/11 - cuối tháng 12) nên nhóm mình dành rất nhiều thời gian để kịp tiến độ. Rất may thì BTC cũng quyết định dời lịch thi sang 13/1/2019.

# **2. Cài đặt môi trường để code, debug và chạy mô phỏng**

Sẽ rất vất vả để làm quen những thứ liên quan đến cuộc thi nên Ban Tổ Chức có cung cấp một video hướng dẫn cài đặt ROS và OpenCV bằng CMake.

Link: https://www.facebook.com/cuocduaso/posts/1863561467012569

## **2.1 Ubuntu**

Mình phải lập trình theo chuẩn ROS node, mà ROS (ROS1) lại chỉ chạy trên Ubuntu nên mình cũng phải dual boot ubuntu và win cùng lúc. Nói sơ qua về ubuntu, đây là hệ điều hành mã nguồn mở có nhân linux và khá nhẹ. Ubuntu đáp ứng đầy đủ nhu cầu lướt web và code nên dùng càng lâu lại càng thấy thích.

### 2.1.1. Cài ubuntu

Vấn đề cài hệ điều hành là một điều tương đối khó với những ai chưa bao giờ làm (trong đó có mình (cười). Do đó bạn cần có một người đã làm qua để không mắc những sai lầm không đáng có. Những thứ cần chuẩn bị là usb 4 GB trở lên, Ubuntu 18.04 (hoặc thấp hơn) tải sẵn và rufus để boot Ubuntu vào usb.

Sau khi boot xong, chúng ta cắm USB vào máy và ấn F12 liên tục để vào BIOS. Đến đây thì bạn có thể xem hướng dẫn trên mạng để biết nên tích và điền những mục nào. Một kinh nghiệm khi cài đặt là các bạn không nên để vùng nhớ swap quá lớn vì nó ít khi được dùng. Nếu máy xài SSD thì nên chia khoảng 40 - 50 GB.

### 2.1.2. Làm quen với Ubuntu

Đối với những người trước giờ chỉ xài win thì cảm giác làm việc với Ubuntu sẽ giống như hồi mới biết mở máy tính vậy. Tuy nhiên cũng đừng lo lắng quá vì nó cũng không thực sự khó. Đây là những thứ mình nghĩ cần tiếp cận là làm quen trước:

- Terminal: là trình nhập/ xuất giống cmd bên window, thời gian làm việc trên terminal sẽ tương đối nhiều.

+ Phím tắt: ctrl + alt + t (Add to favorite cho tiện)

+ Một số lệnh cần nhớ: mkdir (make directory - tạo thư mục), cd (đi tới [đường dẫn]), cat (mở file), sudo apt-get install (cài chương trình), ...

+ Phím: tab (tự động hoàn thành câu lệnh), tab tab (gợi ý), up / down (xem câu lệnh đã gõ trước đó), shift + insert (paste).

+ Enter password: khi terminal yêu cầu nhập mật khẩu thì khi bạn gõ sẽ không thấy dấu \* như bên window nên cứ nhập đầy đủ rồi nhấn enter.

- File: tương tác với file cũng khá quan trọng trong Ubuntu.

+ Cấp quyền: mình thường dùng lệnh chmod 777 [tên file] cho đơn giản.

+ Mở file: dùng lệnh cat (nếu không ở trong thư mục chứa file) hoặc nhập tên file (nếu đang ở trên thư mục chứa file) trong terminal.

+ Code trên file: file được định dạng là code (.cpp, .py) có thể edit trực tiếp trên editor và biên dịch trên terminal, khá tiện lợi.

+ File .bash, .setup, ... : đây là những file log hoặc môi trường, nếu trong terminal có báo lỗi command not found thì hãy xem xét đến những file này (trong thư mục Home).

### 2.1.3. Một số phần mềm hỗ trợ

- ibus: là phần mềm hỗ trợ gõ tiếng việt trên ubuntu, ibus có khá nhiều bug và một trong những bug gây ức chế nhât là cướp hook, làm cho vscode không nhắc lệnh, làm mình khổ sở trong nhiều ngày liền. Lời khuyên là tạo thói quen tắt ibus mỗi lần code.

- simple screen recorder: chụp/ quay màn hình, rất cần thiết để quay tư liệu, source test hay viết báo cáo như mình làm hiện tại.

- anydesk: giống teamview

## **2.2 ROS & Catkin**

### 2.2.1. ROS là gì?

Link: <http://wiki.ros.org/Documentation> (tất tần tật về ROS)

Tại thời điểm viết báo cáo này thì mình sử dụng phiên bản Melodic for Ubuntu.

### 2.2.2. Một số thành phần cơ bản của ROS

https://drive.google.com/open?id=1n6QQnLfJaNPFqOBm4\_I7YsVvkXhSevip

### 2.2.3. Các lệnh về ROS

Các lệnh về ROS đều có tiền tố là tên class, ví dụ như rospack, rostopic, rosnode.

Theo sau là các từ khoá như list, info, echo, bw, ... và hậu tố là tên object. Một số lệnh mình thường dùng để chạy và debug như:

* roslaunch lane\_detect launch\_detect.launch
* rostopic echo /Destiny\_speed
* rostopic echo /Destiny\_steer\_angle

### 2.2.4. Nguồn tự học

Trong tài liệu đào tạo có một file hướng dẫn tiếng việt về ROS message/srv và một cuốn sách kinh điển bằng tiếng anh. Thành thật mà nói thì tài liệu tiếng việt chỉ nói chung chung còn tiếng anh thì liệu không có thời gian để cày hết.

Thời gian đầu mình có tìm hiểu trên theconstructor, sau đó trang này tính phí thì mình nhảy qua ros wiki để học. Học trên wiki phải tập trung và liên tục mới có thể hiểu và nhớ lệnh, thời gian học những thứ cơ bản mất khoảng 1 tuần. Đến giờ mình vẫn thấy tiếc nguồn học đầu tiên vì trong đó có phần mềm giả lập lẫn lộ trình học lên Master trong 7 ngày.

## **2.3. Cmake**

CMake là trình build đa dụng, ở đây mình dùng để quản lý opencv. Quá trình sử dụng hơi phức tạp nên mình chỉ sử dụng theo hướng dẫn của video và cũng không nghiên cứu nó nhiều lắm.

## **2.4. VS Code**

### 2.4.1. VS Code là gì?

VS code là editor gọn nhẹ lại phổ biến. VS code trên ubuntu khá khó xài đối với người mới bắt đầu như mình.

### 2.4.2. Mẹo với những file json

Để build được thì cần có trình biên dịch (như gcc cho C++) mà vs code chỉ là editor nên cần phải thông qua những file như setting.json, task.json để liên kết. Sau khi build thành công thì các bạn nên copy folder .vscode để dùng sau này. Mỗi khi tạo một project mới thì paste folder này vô. Nếu code python thì khá sướng bởi vì thông dịch luôn chứ không cần build.

### 2.4.3. Extensions nên cài

- C/C++

- C/C++ snippets

- C.C++ Intellsense

- CMake tools

- Include complete: nhắc mỗi khi include lib

- ROS snippets: khi ros master được chạy, ta có thể xem hoạt động của các node, service, … hiện tại.

### **2.5. Phần mềm mô phỏng**

### 2.5.1. Tổng quan

Phần mềm mô phỏng được thiết kế bằng Unity3D bao gồm một chiếc xe oto và môi trường xung quanh nó, ảnh từ camera phía trước oto sẽ được truyền về cho topic /Destiny\_image\_compress, ngược lại xe sẽ nhận tốc độ và góc lái từ topic /Destiny\_steer\_angle và /Destiny\_speed.

### 2.5.2. Đặc điểm về xe

- Nhận vào góc lái trong khoảng (-50, 50), với mốc 0 độ là xe đi thẳng.

- Nhận tốc độ trong khoảng (-120, 120), có thể truyền vào vận tốc âm để đi lùi.

- Camera cố định, không bị rung lắc.

### 2.5.3. Đặc điểm về môi trường

- Luôn đảm bảo 2 lane (đường liền), lane ở giữa (đường đứt) sẽ xuất hiện ngẫu nhiên.

- Có nhiều vật cản trên đường đi: thùng gỗ, hòn đá lớn, ... che kín một lane trong một đoạn.

- Nhiều đoạn có cây ven đường, bóng cây che mất làn đường.

- Điều kiện thời tiết: có ánh sáng.

- Biển báo đặt ở phía bên phải và không bị bóng cây che.

- Độ rộng làn đường có thể thay đổi.

- Có thể có vỉa hè.

## **2.6. Những điểm cần phải lưu ý**

- Cẩn thận làm từng bước giống như hướng dẫn nếu không muốn xảy ra lỗi.

- Nếu bạn muốn cài thêm thứ gì thì hãy tìm hiểu xem nó có xung đột gì với những thứ đã có hiện tại hay không? Ví dụ như xung đột OpenCV và Anoconda. May mắn là mình không có ý định code bằng python nên không dính tới trường hợp này.

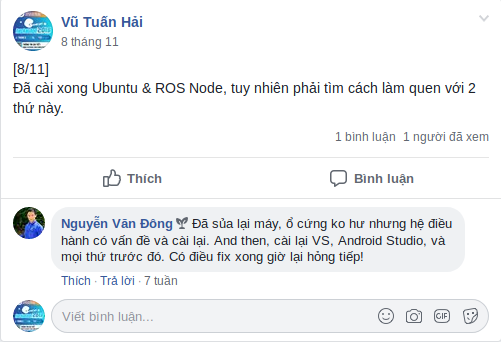
- Luôn phải test mô phỏng ở chế độ đồ họa cao nhất.

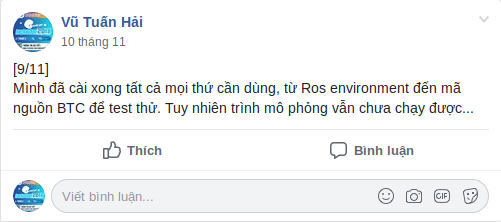
## **2.7. Source code: toàn bộ source code được mình public trên github tại:**

<https://github.com/haimeohung/Self_Drivring_Car>

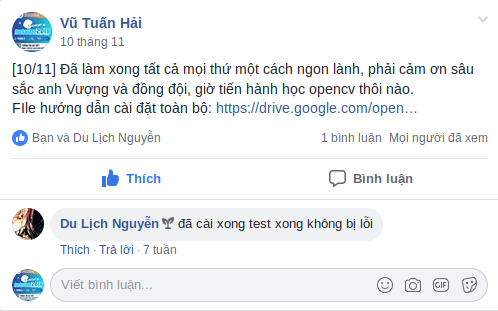
# **3. Quá trình thực hiện**

## **3.1. Tuần 1**

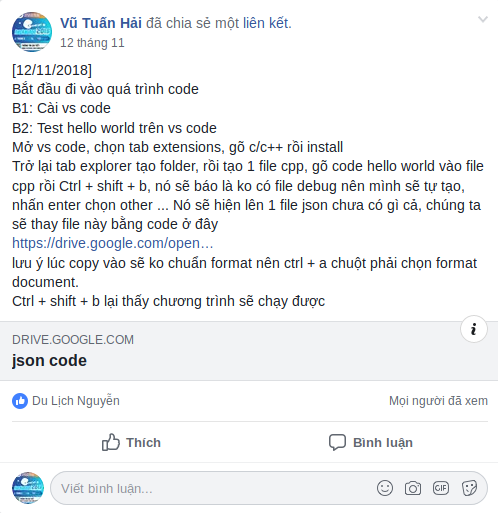




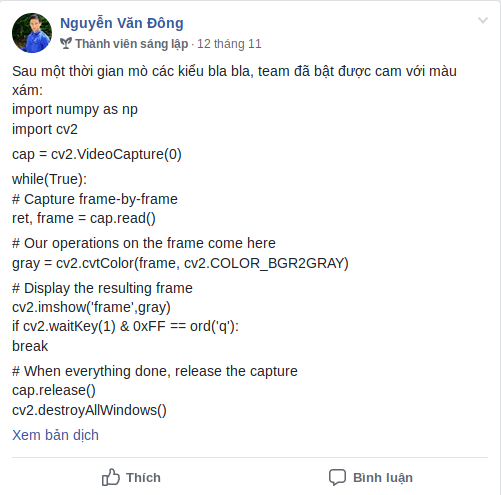
Sau khi cài đặt xong hết thì mình tiến hành chạy mô phỏng thử bản unity (không biển báo) đầu tiên của BTC. Lần chạy đầu khá thốn và phải nhờ đến anh khóa trên vì source code mẫu của BTC có bug.



Link chạy thử lần đầu tiên: <https://youtu.be/wNmtWmEfe3c>

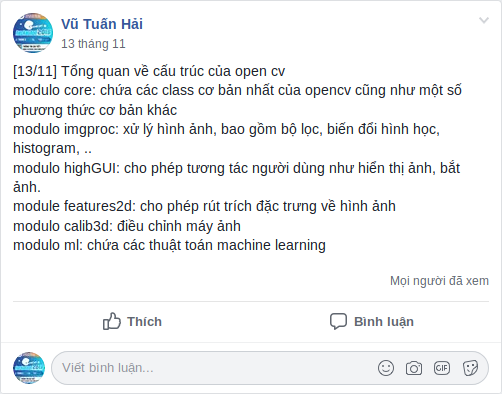


Những ngày tiếp theo mình chật vật với editor trên ubuntu, sau khi tham khảo thì vscode là phù hợp, vì ưu điểm gọn nhẹ và code cực sướng, tuy nhiên rất khó sử dụng đối với những tay gà mờ. Cả nhóm mất cả buổi để chạy được Hello world bằng vscode và cài đặt môi trường để hiển thị ảnh xám thu được từ cam. Sau đó nhiều ngày thì thêm được ROS. Mấu chốt là do mình chưa từng tiếp xúc với file dạng json bao giờ, mà để edit được thì phải tốn thời gian để đọc hiểu nữa. Code được trên vscode có lẽ là một trong những thành tựu mình tự hào nhất từ trước đến giờ (cười).



(Chương trình này mới test trên python, chứ còn qua C++ chắc còn nhiều lỗi hơn nữa)

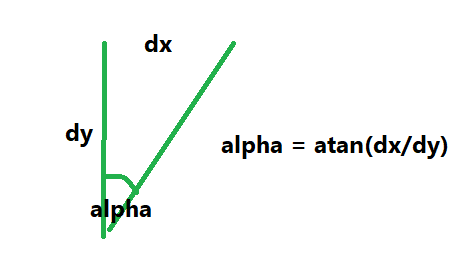
Ngày tiếp theo mình có vọc đến opencv và tìm hiểu nó là gì?



Đến thời điểm này thì nhóm chỉ sử dụng highgui và improc là đủ do chưa cần dính dáng tới ML. Nhiệm vụ code opencv được chuyển giao cho Lịch và 2 nguời còn lại thì cứ tiếp tục tìm hiểu xử lí ảnh.



Công việc tiếp theo là bắt đầu tìm hiểu code mẫu. Code mẫu khá đơn giản, được viết kiểu OOP và chỉ có 2 class: class carcontrol để điều khiển xe và lanedetect để nhận diện lane. Vì thời điểm này chưa biết gì nhiều về opencv cũng như ROS nên mình vọc đoạn code dễ hiểu nhất là CarControl. CarControl nhận tham số là vị trí của 2 lane, sau đó tính ra vị trí giữa (vị trí cần đi tới) và tính góc lái bằng góc giữa quỹ đạo xe muốn đi tới và trục tung. Xử lí trường hợp mất làn thì giữ nguyên góc lái trước (mất 2 làn) hoặc bám theo làn còn lại error = errorAngle(left[i] + Point(laneWidth / 2, 0)) với laneWidth là hằng số (40 - 80). Về tốc độ thì được cố định là 35.



Vài lần chạy thử gặp vấn đề thì mình lần lượt cải tiến code như sau:

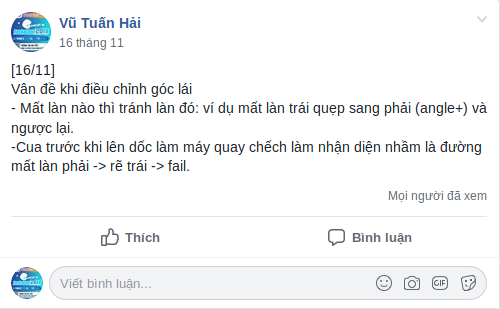
- Gặp vật cản: được xem như là mất làn, trong trường hợp tại vòng cua phải gặp vật cản bên trái thì xe đụng vật cản. Lý do là tại nhiều điểm thì laneWidth bé/lớn hơn bình thường, dẫn đến việc cua không được lực. Mình sửa lại bằng cách cho laneWidth động, laneWidth được cập nhật = (laneL + laneR) / 2 mỗi khi có đầy đủ 2 lane đồng thời chỉnh cho vị trí đi tới nằm xa hơn vật cản một xíu. Kết quả cuối cùng là xe đánh lái mượt.

- Chỉnh ngưỡng: mình chạy thử nhiều lần để tìm ngưỡng tối ưu nhất cho xe, có trường hợp buồn cười như thế này



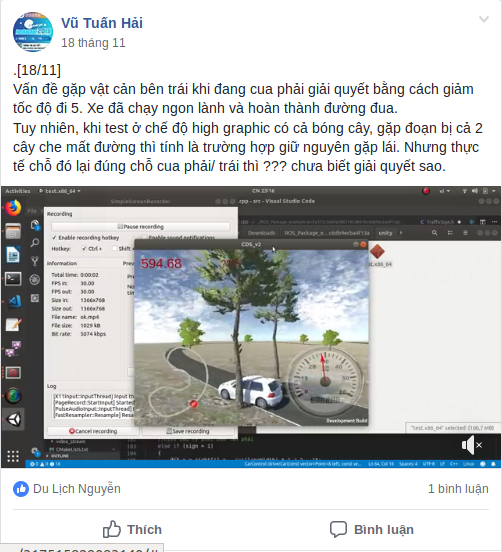
## **3.2. Tuần 2**

Một vấn đề mình vọc ra là trường hợp khi lên dốc / xuống dốc



Khi lên dốc thì camera bị chếch lên làm code nhận diện cả bầu trời là lane. Còn khi xuống dốc thì xe cần phải hãm tốc độ để xử lý trường hợp còn một cua tiếp theo. Mình xử lý vấn đề này bằng cách giới hạn lại khu vực nhận diện.

- Tốc độ quá chậm: mình tăng tốc độ max lên 50, đồng thời tính lại tốc độ động = 50 - góc lái \* 3.



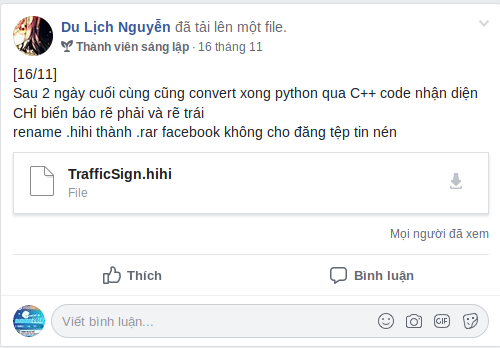
Mình chạy lại thì thấy xe hoàn thành đường đua khá nhẹ nhàng, tuy nhiên khi test ở chế độ high graphic thì lại gặp thêm vấn đề:

- Đường bị bóng cây che: bóng cây che trên đường làm cho trạng thái có/ mất lane thay phiên liên tục, làm xe lắc mạnh. Cách đầu tiên mình sử dụng là sử dụng các biến để lưu trạng thái trước đó, nhưng không phù hợp.

- Ngưỡng động có thể giải quyết vấn đề này: đến đoạn bị bóng cây che mình nhanh tay tăng kênh HighS lên để tăng phạm vi nhận diện màu trắng (màu xám xịt nhưng vẫn cho là màu trắng). Đi hết đoạn đường thì chỉnh lại như ban đầu để không nhận diện nhầm vật cản/ lòng đường với lane. Để ngưỡng có thể tùy biến như vậy thì phải sử dụng ngưỡng động.

Đây cũng là lúc mình nhận ra rằng phải xây dựng lại toàn bộ class lane\_detect.

## **3.3. Tuần 3**



Cũng trong khoảng thời gian này thì Lịch có viết xong cái class TrafficSign để nhận diện trái phải rồi cũng gặp nhiều vấn đề như sau:

- Nhiễu: code ban đầu chạy nhiễu do nhiều nguyên nhân, như hình dạng bị thay đổi do góc nhìn thay đổi (dùng HOG transform sẽ fix được lỗi này), do ánh sáng (ban đầu dùng colour space IHSV nhưng sau đó chuyển sang IHLS)

- Nhận diện nhầm: code counter tất cả những vật có màu xanh, như cả bầu trời nên mình đề xuất chỉ bắt lấy những vật có hình elip., tuy nhiên bầu trời là một mảng xanh rộng nên mình có ý định cắt bỏ luôn. Nhưng giới hạn khu vực nhận diện như vật sẽ không xử lý được trường hợp có biển báo khi đang lên dốc/ bên trái.

## **3.4. Tuần 4**

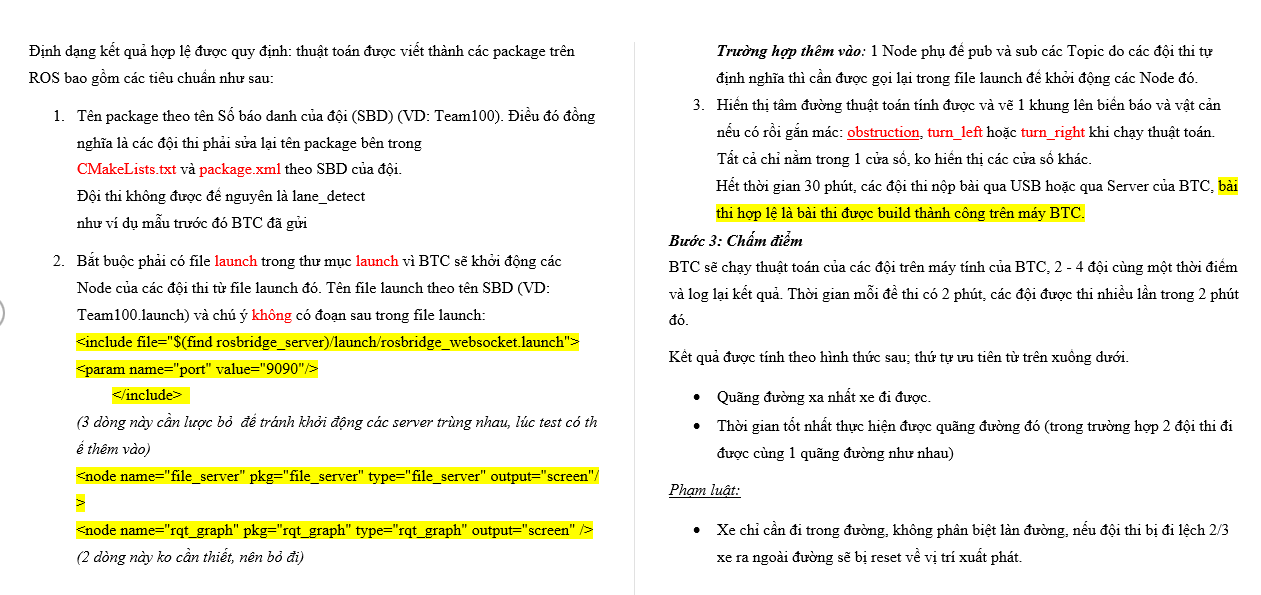
Sau nhiều cải tiến nữa thì mình thấy code khá ngon, tỉ lệ nhận diện đúng tương đối cao, tỉ lệ nhận diện đúng trên 90 %. Đến đầu tháng 12 thì mình và Lịch chia mỗi người một phần: một người code ROS và một người code lanedetect.

Thời gian mình học ROS tương đối nhanh vì xác định chỉ học những thứ căn bản như hiểu cấu trúc source của một ROS node, cách code file package.xml, CMakeList.txt, cách debug, ... Code về ROS cũng khá ít và cũng dễ hiểu nên mình cũng có tham gia phụ phần còn lại.

Để tiết kiệm thời gian thì bọn mình test trên một video thu ở chế độ manual. Bên Lịch thì code bằng Visual Studio.

## **3.5. Tháng 12**

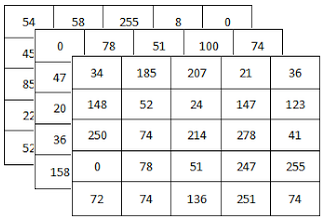
Khoảng thời gian hoàn thành và láp rắp code diễn ra sau giữa tháng 12. Giai đoạn cuối là chỉnh lại format sao cho chuẩn với yêu cầu của BTC, như tên node, tên topic phải theo số báo danh.



# **4. Về source code**

## **4.1. Ảnh là gì?**

Ảnh là tập hợp những ma trận lưu giá trị của màu, với số cột \* số dòng tương ứng với số pixel của ảnh. Số ma trận / ảnh và giá trị từng phần tử trong ma trận tùy thuộc vào không gian màu. Không gian màu được nhóm sử dụng là đen trắng, BRG và IHSL, lý do tại sao sẽ được giải thích sau.



Đối với BRG, đây là không gian màu của ảnh thu được từ camera trong phần mềm mô phỏng. Mỗi ảnh trong BRG sẽ được lưu duới dạng 3 ma trận kích thước 240 x 320 trong đó mỗi phần tử sẽ là 8 bit do không nén nên dung lượng mỗi ảnh sẽ khoảng 0,23 MB (nên lưu ý con số này vì nó liên quan đến tốc độ xử lý của chương trình, mỗi giây phần mềm mô phỏng truyền về hàng chục ảnh ~ thấp nhất phải là 23 fps).

Không gian màu đen trắng: là không gian cho ảnh thu được khi xử lý

Không gian màu IHSL: dùng để nhận diện biển báo.

## **4.2. Nhận diện lane**

Một vài tham số đặc biệt cho class này như:

static const float alpha: giá trị làm mềm size lane

static const int SkyLine; // =80

static const Size Box; // 5x5 - Box detetion lane

static double Fx\_a, static double Fx\_b: hàm f(x) = Fx\_a\*x+Fx\_b - trả về khoảng cách ignore from mid lane.

static const int TooFew: khi số lượng point trong vector<Lane> thấp hơn tooFew thì hiểu là mất lane.

static Mat draw (RGB): dùng để vẽ

void Setting(): cài đặt 2 giá trị blur (2n+1) và accuracy

void Detect(const Mat): dùng để gọi các hàm trong phạm vi lane\_detect

void DrawLane(): dùng để vẽ 2 vector LaneL và LaneR

void UpdateMidLane(): cập nhật lại lanesize dựa vào LaneL & LaneR

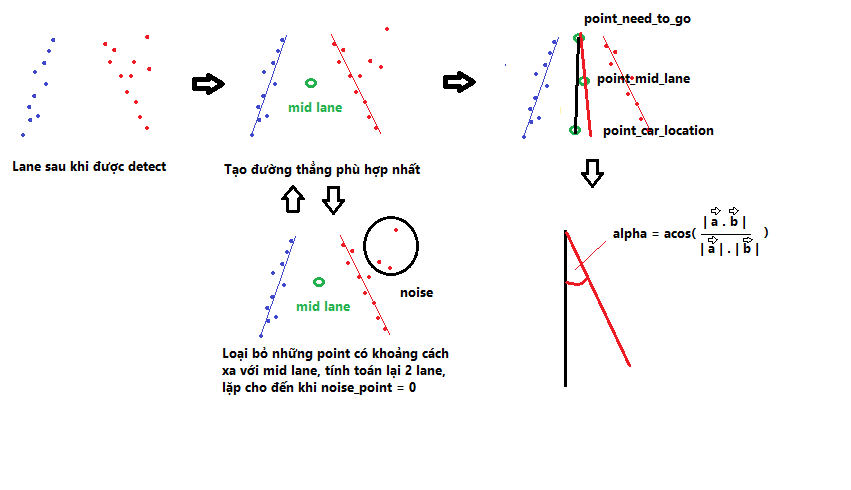
static void DrawVirtualLane(): vẽ lane ảo dùng khi cua

Mat ReduceNoise(const Mat &src): input và matrix RGB, output là matrix gray đã được giới hạn phạm vi xử lý

Mat CvtBinary(const Mat &src): input là matrix gray, output là matrix binary đã áp dụng giải thuật Laplacian

void FindLane(const Mat &Binary): input là binary matrix

Thứ tự gọi hàm:



Code detect\_lane của BTC vừa khó hiểu, vừa không tối ưu nên nhóm quyết định đập đi xây lại từ đầu với mục tiêu chính như sau:

- Nhận diện lane trắng trong điều kiện bình thường

- Nhận diện lane trắng trong điều kiện bóng râm

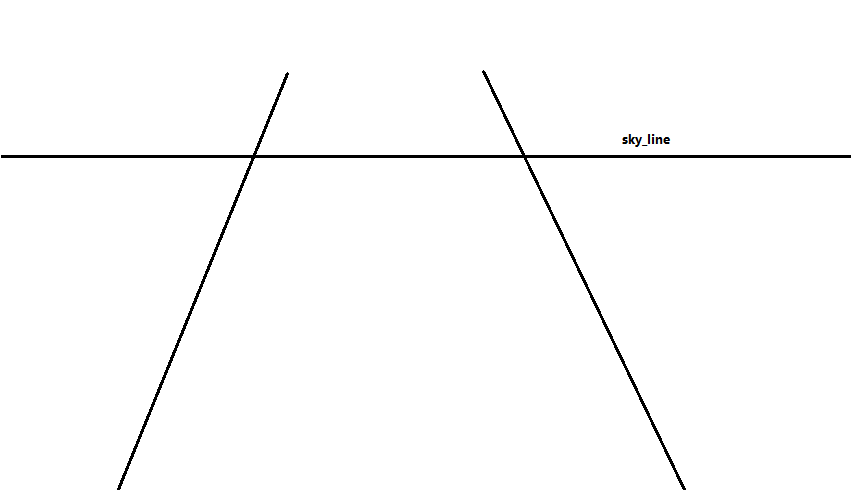
- Tạo được lane khi rẽ trái hoặc rẽ phái (vì khi rẽ sẽ xảy ra mất làn cục bộ, làm sao có thể điều khiển xe khi không biết thời gian rẽ là bao lâu? )

-> Được xử lý bằng cách sử dụng phép lọc Laplacian để lấy và ngưỡng viền để chuyển thành ảnh trắng đen. Sau đó push vào stack của mỗi Lane, ở đây sẽ là LaneL (trái) và LaneR (phải), LaneM (giữa) sẽ được tính bằng công thức:

MidLane = MidLane \* alpha + ((saveL + saveR) >> 1)\*(1 - alpha);

Công thức trên được nhóm lấy từ công thức tính thời gian timeout của gói tin bên mạng (với alpha tùy chỉnh, có thể là 0.1 hoặc 0.2). LaneM rất quan trọng tới quá trình điều khiển, vì đây là vị trí mà xe mong muốn đi tới. Để tăng tính chính xác cho LaneM hơn nữa thì mình có tạo thêm hàm UpdateMidLane().

Trong nhiều trường hợp khó như đi vào cua, hay không nhận diện lane được do có vật cản. Thuật toán rất dễ nhận diện nhầm làn phải và làn trái, để cải tiến thì chúng ta phải viết thêm hàm IgnoreFromMid, đại khái hàm này sẽ tính toán ra một giá trị, sau đó kiểm tra xem khoảng cách từ điểm vừa nhận diện đến MidLane có thỏa hay không, nếu thỏa thì mới thêm vào.



Một vấn đề khá quan ngại đó là về màu sắc, lưu ý rằng bầu trời và làn trong phần mềm giả lập đều màu trắng, nên nếu lấy toàn bộ hình để nhận diện thì chúng ta sẽ gặp phải trường hợp nhận diện nhầm lane khi lên dốc (camera bị chếch lên phía trên). Mình chia khung hình ra làm 2 và lấy phần phía dưới, độ rộng phần phía trên xác định bằng biến SkyLine (80 tức là 1/3 tổng khung hình). Việc lấy SkyLine lớn (ví dụ > 80) có thể giảm bớt khối lượng tính toán, nhưng lại không giải được trường hợp khi xe xuống dốc và làm xe không đủ thời gian phản ứng trước những đoạn cua.

Trong class này, mình viết thêm hàm DrawVirtualLane() dùng để hỗ trợ CarControl, như dã nói trước đó, khi rẽ chúng ta sẽ gặp hiện tượng mất làn cục bộ trong thời gian ~ 1s, nếu không có lane R/L cũng như lane M, làm thế nào để chúng ta có thể biết thời gian duy trì góc lái (+-30 - +- 45) ? Có 2 phương án:

* Đếm số frame hay sử dụng timer: khi gặp biển báo bắt đầu đếm ngược và xe bắt đầu cua, khi biến đếm trả về 0 khi xe trở về trạng thái bình thường. Ưu điểm: đơn giản. Nhược điểm: không thể biết chính xác số frame phù hợp, đối với mỗi máy thì số frame này khác nhau.
* Vẽ lane ảo: khi gặp biển báo cờ sẽ được bật, lane ảo được vẽ khi vào cua dựa trên việc mất lane và phương trình đường thẳng (đã được tính toán). Sau đó xe bám theo lane ảo cho tới khi nhận diện lại được lane. Ưu điểm: xe chỉ cua khi cần vào cua nên rất chính xác.

## **4.3 Điều khiển xe**

Trong quá trình điều khiển chúng ta luôn gặp 1 trong 3 trường hợp như sau:

* Mất 2 lane (khi bị bóng cây che kín đường hoặc trường hợp hiếm hơn là cả 2 bên lane đều có 2 vật cản ngang nhau): trả về góc trước đó.
* Mất 1 trong 2 lane:
  + Mất khi có biển báo: tiến hành vẽ lane ảo và bám theo lane ảo, lane ảo này là laneL hoặc laneR chứ không phải LaneM.
  + Mất do vật cản, nhiễu hoặc camera không nhìn thấy lane trong đoạn đường cong: bám theo lane còn lại.
* + Đủ 2 lane:
  + Tính toán tạo vector LaneM: L.UpdateMidLane();
  + Tính đường thẳng đi qua vector LaneM: res = F.Linear(LaneDetect::LaneM);
  + Nếu không có biển báo: cập nhật kích thước đường
  + Tính góc dựa theo vị trí xe hiện tại và điểm cắt giữa đường thẳng res và LineDetect

Khi đã có góc lái, chúng ta tính đến tốc độ cần truyền theo nguyên tắc: nếu góc lái càng lớn thì vận tốc càng nhỏ. đồng thời phải nằm giữa MinSpeed và MaxSpeed. Khi rẽ, MaxSpeed được gán bằng DefaultSpeed có giấ trị thấp hơn.

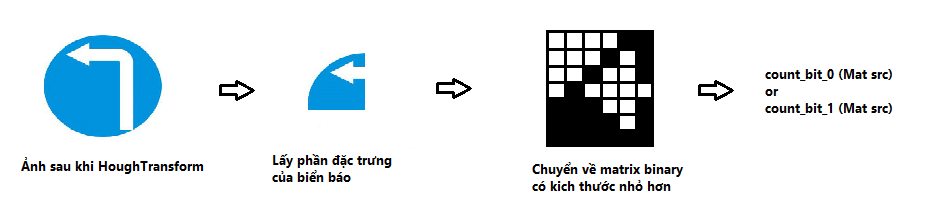
## **4.4. Nhận diện biển báo**

Nhóm sử dụng không gian màu IHLS vì thấy nó dễ đặt ngưỡng để nhận diện những vật thể đặc trưng tốt hơn so với không gian màu IHSV. Ví dụ như kênh L (Light) sẽ tùy biến được khả năng nhận diện trong môi trường ánh sáng thay đổi.

Thông thường đối với bài toán nhận diện biển báo thì người trong ngành sẽ nghĩ ngay đến việc dùng các mô hình như SVM, K – Mean, … để giải quyết classifition. Tuy nhiên , mình đã đề xuất giải quyết vấn đề này bằng thuần xử lý ảnh, có vài lý do như sau:

* Cả nhóm chưa biết gì về ML, nếu tìm hiểu thì sợ không kịp thời gian.
* Bài toán phân loại ở đây chỉ có 2 class (rẽ trái & rẽ phải) khá đơn giản.
* Tối ưu chương trình hơn.

Ý tưởng chính như sau: đầu tiên chúng ta sẽ giới hạn lại khu vực nhận diện (góc trên bên phải) bằng biến SkyLine và LeftLine. Sau đó dùng hàm cvtColor(des, des, COLOR\_BGR2HLS) để chuyển không gian màu theo thứ tự Blue, Green, Red sang Hue, Lightness, Saturation. Dùng thêm 2 hàm erode() và dilate() trong hàm PreFIx() để khử nhiễu, sau đó trả về matrix binary (ảnh đen trắng). Chúng ta tiến hành counter lại những khi vực có diện tích lớn hơn MinSquare và có hình elip. Cuối cùng là hàm Find() đếm số pixel 1 (0), nếu số pixel này lớn hơn giá trị nào đó (tính bằng thực nghiệm) thì ta có thể kết luận frame đó có biển báo rẽ trái/phải.



1 frame liệu có đủ để cờ rẽ bật lên không? Câu trả lời là không, vì gần đến đoạn cua thì biển báo không nằm trong ‘tầm nhìn’ của xe nữa (lưu ý là xe chỉ có duy nhất 1 camera soi tầm 60 độ). Giải pháp đặt ra là tạo thêm biến LockFlag và UnlockFlag, biến đếm sẽ được cộng dồn khi bắt được biển báo từ xa cho đến khi bằng lockFlag( = 10) thì cờ rẽ sẽ được bật. Sau khi biến rẽ được bật, biến đếm sẽ được đếm ngược từ UnlockFlag ( = 50 ~ 2s) cho đến 0 để cờ rẽ tắt. Biến Sign sẽ lưu giá trị sign hiện tại và OldSign lưu giá trị sign trước đó.

Một số mẹo khi làm về class này:

- Nên viết thêm test unit (Setting()) để độ ngưỡng phù hợp, đã có lúc mình tham khảo các giá trị của ngưỡng nhưng điều không phù hợp, lý do là môi trường thực tế luôn có sự khác biệt lớn với môi trường mô phỏng.

- Để không tốn thời gian thì nên test trên ảnh (hàm find) và video (hàm CheckSign) trước. Về phần source thì có thể capture trong chế độ lái Manual. Debug cũng dễ hơn nữa.