ROS Vision Control Self-driving Car

Author Qifan He

Git Repo https://github.com/QifanHe/self_driving_rc **Email** heqf@bu.edu

BUID U78887372

Abstract:

This project discusses how to build a vision control self-driving car platform based on Robotics Operation System(ROS). The building process goes through all necessary components of ROS, hardware assembling, control and object detection.

1. Introduction

1.1 Background

Recent years, topics related to information technology became more and more popular among the public. Artificial Intelligence, including robotics, machine learning and computer vision, plays the most important role in the technology evolution in the 21st century. Unmanned Vehicle is the combination of robotics and machine learning. Some companies, such as google and Uber(Volvo), have been testing self-driving cars on roads since earlier 2016. While others such as Baidu and Amazon, are developing unmanned vehicles from self-driving cars to delivery quadrotors (Prime Air by Amazon). Also, there are always intermediate technologies like Driving Assistance System(Tesla), which can assist the driver to control the car in many scenarios while in emergency situations, the system would give the control back to the driver.

ROS has been widely used in variety of research field to fast design and deploy prototypes for robotics productions especially self-driving car. Once a ROS platform is established, adding / removing features is very easy with only few modifications.

1.2 Project Goal

For this project, the goal is to build a self-driving car based on ROS and vision control. The car is connected to a laptop through wireless network and send real-time camera video stream to the laptop. At the laptop end, some nodes would receive images and perform object detection then generate and send control messages to the car. Finally, the car could follow the road lane and stop at stop signs automatically.

2. **System Components**

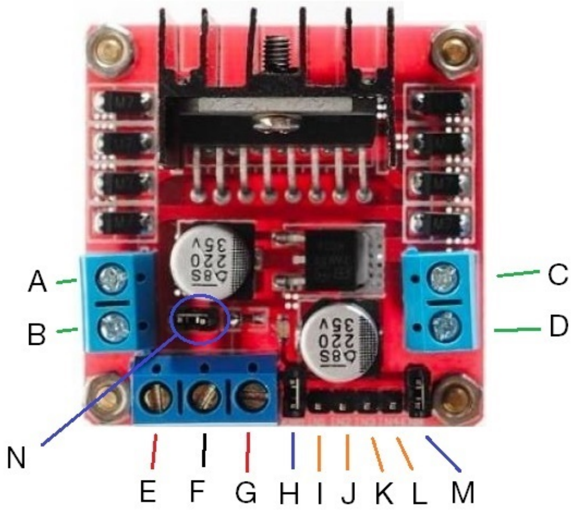
2.1 **Hardware**

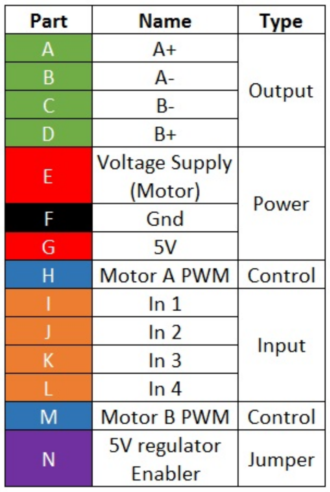
2.1.1 Components list

Name	Quantity
Raspberry Pi® 3 Model B	1
Laptop with Wifi	1
L298N Motor Driver	1
Emgreat® Motor Robot Car Chassis Kit	1
7.4v Li-ion Battery	1
Raspberry Pi® Camera	1
Dupont® Line(Female-Male)	10

2.1.2 Pin Layout

L298N [1]:

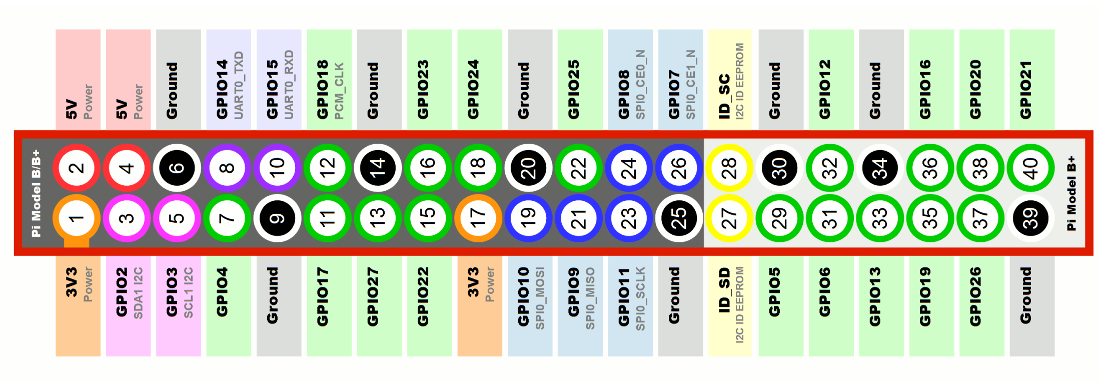




Part	Name	Type
A	A+	Output
B	A-	
C	B-	
D	B+	
E	Voltage Supply (Motor)	Power
F	Gnd	
G	5V	
H	Motor A PWM	Control
I	In 1	Input
J	In 2	
K	In 3	
L	In 4	
M	Motor B PWM	Control
N	5V regulator Enabler	Jumper

[1] http://cyaninfinite.com/tutorials/making-a-differential-robot-with-the-l298n-dual-motor-controller/

Raspberry Pi 3 Model B[2]:



[2] http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/#prettyPhoto/0/

2.1.3 Pin connection

Raspberry Pi	L298N	7.4V Battery	MOTOR A	MOTOR B
GPIO17	IN1			
GPIO18	IN2			
GPIO23	MOTOR A PWM			
GPIO27	IN3			
GPIO22	IN4			
GPIO24	MOTOR B PWM			
Ground	Gnd	"-"		
5V	5V			
	Voltage Supply	"+"		
	A+, A-		"+", "-"	
	B+, B-			"+", "-"

2.1.4 Recommended requirement for laptop

CPU: Equal or higher than Intel Core i7 3520 (Apple MacBook Pro Model 2012 or later).

2.2 Software

2.2.1 Laptop Requirements

System: Ubuntu 14.04 or higher (16.04 recommended)

ROS: Indigo or higher (Kinetic with fully installation recommended)

Python2.7 with OpenCV3 Installed

2.2.2 Raspberry Pi Requirements

System: Ubuntu Mate 16.04

ROS: Must be the same version as the laptop

2.2.3 Useful Links

Github repository of all my codes for this project:

https://github.com/QifanHe/self_driving_rc

Ubuntu Mate download:

https://ubuntu-mate.org/download/

ROS Installation guide:

http://wiki.ros.org/kinetic/Installation/Ubuntu

3. **Implementation details**

3.1 ROS framework

3.1.1 Nodes, Messages, Topics, Master

Each component in ROS acts as a node. A node could be a motor driver script, or an object detection program. Nodes can publish messages to topics or subscribe messages from topics. All topics are connected to one ROS master. A ROS master could be at the local computer or at a remote computer as long as two computers have two-way SSH connection. In other words, ROS is a messages driven non-realtime framework.

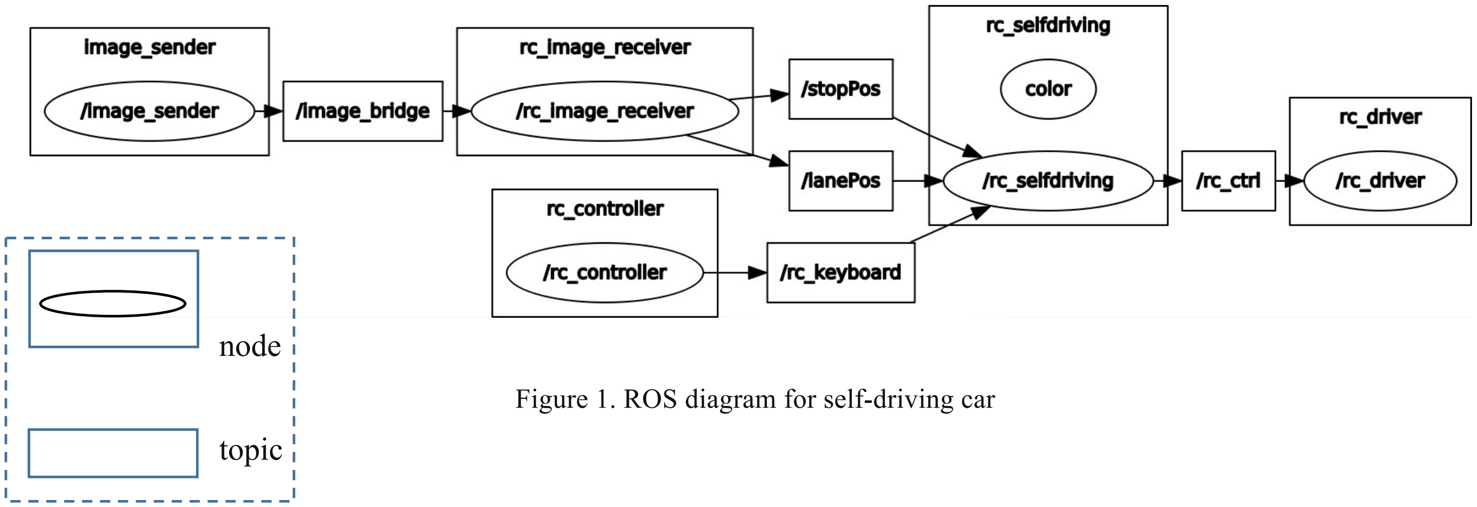


Figure 1. ROS diagram for self-driving car

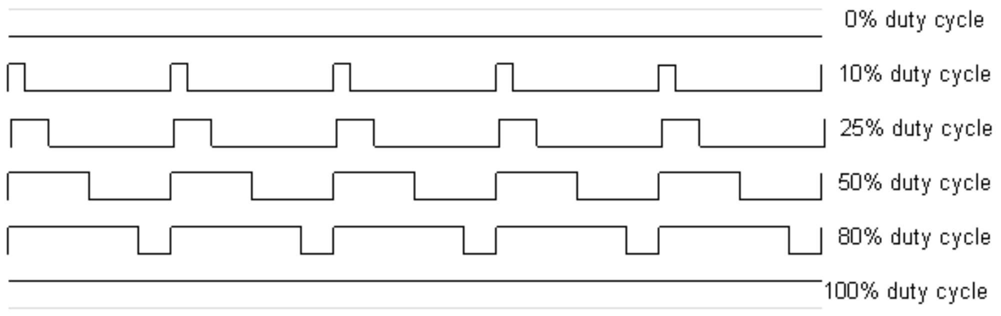
Figure 1 shows the ROS graph of the self-driving car of this project. In order to maximum the system performance and reduce latency, nodes /rc_driver and /image_sender are at Raspberry Pi end while all other nodes are at the laptop end. The ROS master is run on Raspberry Pi and ROS_master_URI = 10.42.0.1:11311. The laptop connects to ROS master where ROS_IP = 10.42.0.31.

3.1.2 Data Flow

From the data flow perspective, /image_sender node capture images from the camera and transfer it to CompressedImage.msg then publish to /image_bridge. /rc_image_receiver subscribes images from /image_bridge then do image processing and send some geometry messages to /rc_selfdriving. The later one subscribes control messages from /rc_controller and geometry messages from /rc_image_receiver and generate updated control messages then publish to /rc_ctrl topic, which is subscribed by /rc_driver on Raspberry Pi and used to drive the car.

3.2 Motor Driver*(rc_driver)*

The driver uses Pulse Width Modulation(PWM) mechanism to set variety speeds for two motors by changing duty cycle of each motor.



https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM

3.2 Controller*(rc_controller)*

The keyboard controller is implemented using TKinter library in Python. The controller GUI is timer triggered and refreshed 30 cycles per second. In each time refreshed, the GUI will check binding keys are pressed or released. The advantage is this mechanism supports multiple keys pressed at the same time, which is useful to control the car going forward while turning. The control message sent to /rc_driver is represented in data.linear.x and data.linear.y in geometry_msgs.msg to control duty cycles of two motors respectively.

3.3 Image Processing*(rc_image_receiver/sender)*

3.3.1 Video streaming

Raw video stream captured from camera is 640x480 @ 30fps, 26MBps which is much higher than the Wireless network band width. The video must be compressed on Raspberry Pi in order to broadcast to the laptop without latency. In /image_sender, raw images are compressed in CompressedImage.msg to reduce video bandwidth from 26MBps to 1-2 MBps.

3.3.2 Lane tracking

In order to simplify lane tracking algorithms, this project uses lanes with unique color. Images are transferred from RGB to HSV color space first. Then after some filtering, lanes with unique color would be extracted from original images. Last, I select part of lanes to be the desired direction for the car to follow. Also, the program computes difference between the current direction and the desired direction and send this bias to next node.

3.3.3 Object Detection

In object detection, I use OpenCV LBP(Local Binary Pattern) Cascade classifier to train stop signs detector. Compared to Haar Cascade, LBP is faster though a little less accurate (around 10%).

	Epoch 10	Epoch 15	Epoch 20
LBP	7 min	29 min	83 min
Haar	41 min	105 min	-

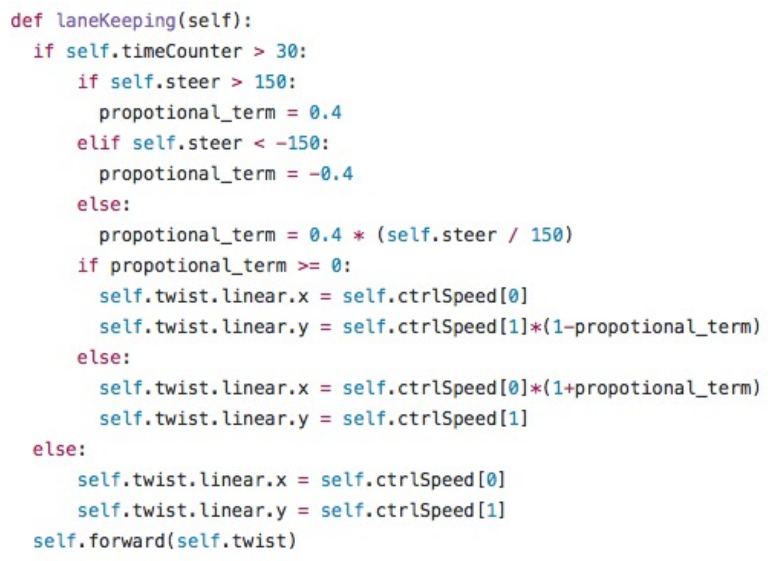
Table 1: Training time between LBP and Haar

In this project, I use 1500 positive samples for stop signs generated from 15 raw stop sign pictures and 3000 negative samples from variety of background pictures which don't include any stop sign. From Table 1, both LBP and Haar can detect stop sign properly after Epoch 15 while Haar takes 3 times longer. After Epoch 20, LBP can perfectly detect stop sign so I did not test Haar to Epoch 20.

3.4 Self driving*(rc_selfdrivng)*

3.4.1 Lane following

In this project, I use proportional control to implement lane following. The bias between the current direction and desired direction is transformed in a steering value.



```
def laneKeeping(self):
  if self.timeCounter > 30:
      if self.steer > 150:
        propotional_term = 0.4
      elif self.steer < -150:
        propotional_term = -0.4
      else:
        propotional_term = 0.4 * (self.steer / 150)
      if propotional_term >= 0:
        self.twist.linear.x = self.ctrlSpeed[0]
        self.twist.linear.y = self.ctrlSpeed[1]*(1-propotional_term)
      else:
        self.twist.linear.x = self.ctrlSpeed[0]*(1+propotional_term)
        self.twist.linear.y = self.ctrlSpeed[1]
  else:
      self.twist.linear.x = self.ctrlSpeed[0]
      self.twist.linear.y = self.ctrlSpeed[1]
  self.forward(self.twist)
```

Figure 2: Part of proportional control code in rc_selfdriving.py.

3.4.2 Auto stop

The distance between the car and stop sign is decided by bounding box size of the stop sign published by /rc_image_receiver. In rc_selfdriving, the threshold of determination of stop or not stop is flexible based on the current speed. When the car stopped, it will not move unless the user control it backward.

4. **Future works**

Since the ROS platform is established, I will never want to waste it. The next step is installing two motor encoders on the car and implement PID control on lane following. Also, due to the flexibility of ROS frame, I want to test real-time SLAM on a desktop with GPU.