Self-Driving Car using Convolutional Neural Network

Chris Winsor

9/1/2018

The project applies a Convolutional Neural Network (CNN) to make a self-driving car. The car will be taught to follow a track which defined as a white piece of tape on a concrete floor. The CNN is taught by observing the left/right control signals of an operator as they drive the car, and images from a forward-mounted camera. The CNN is then taught to predict the left/right control signals from the camera image. The taught-CNN is then used to control the car without user input. The project demonstrates the application of CNN in using real-world (noisy/dirty) data.

The software library is Python/Scikit-Learn/TensorFlow. The hardware platform is Raspberry Pi/Raspbian.

The project is from the Metrowest Boston Developers Machine Learning Group.

Source code and data are available on Github at

This document includes:

- Overview of the car, track and general approach
- Hardware: interfacing the RaspberryPi to the RC controller and Pi Camera
- Software: CNN architecture, database structure, files, and approach to data collection

The Car

The car was the “RC Trucks” by New Bright ($10 Walmart). It includes a Handheld Controller with forward/backward and left/right (non-proportional) controls. The communication link between controller and car is 49mhz.

The Raspberry Pi

We used a Raspberry PI 2 Model B with optional Camera running the Raspbian (Ubuntu Linux).

The Track

The car will be taught to follow a “track” which is a piece of white tape on a cement floor. There are several interesting options on shape for the track - here are three:

- Oval
- Dog-bone
- Figure 8

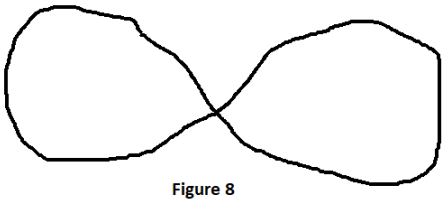
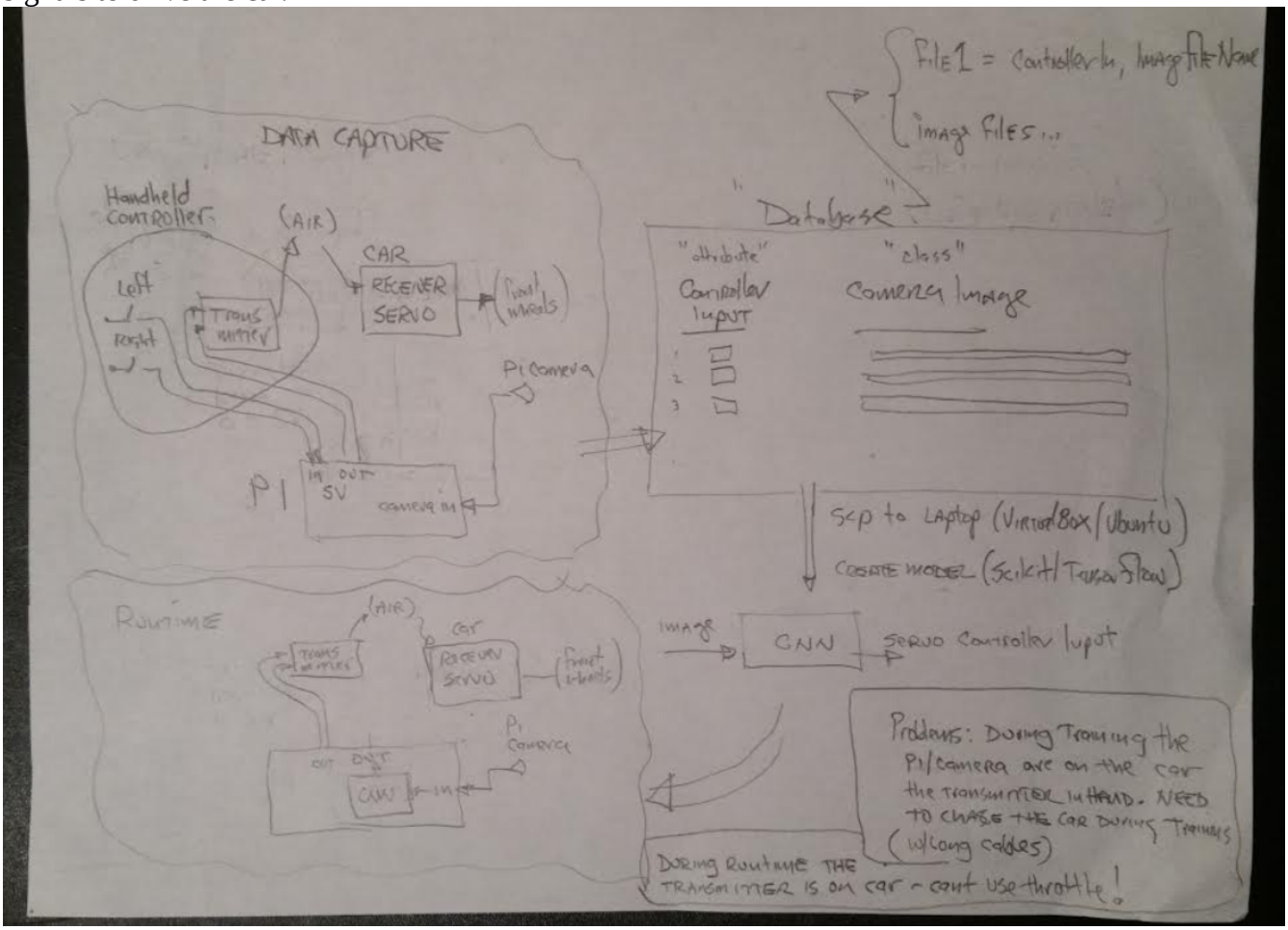


Figure 8

Approach

The diagram below shows the approach we will use. During training (“Data Capture”) a user will drive the car around the track using the Handheld Controller. The PI will passively observe and collect the left/right control signals at the Handheld Controller and will capture images using its camera from the perspective of the car. The Pi will save data to a “Database” (actually a series of flatfiles) on its local usb drive. These will then be copied (“scp”) to a laptop for processing using the Scikit/TensorFlow to train the Convolutional Neural Network (“CNN”). The CNN will thus receive an image and output (predict) the left/right control signals. The trained CNN will then be ported to the Pi. During “Runtime” the Pi will now take in the image from the camera and drive (output) the left/right control signals to drive the car.



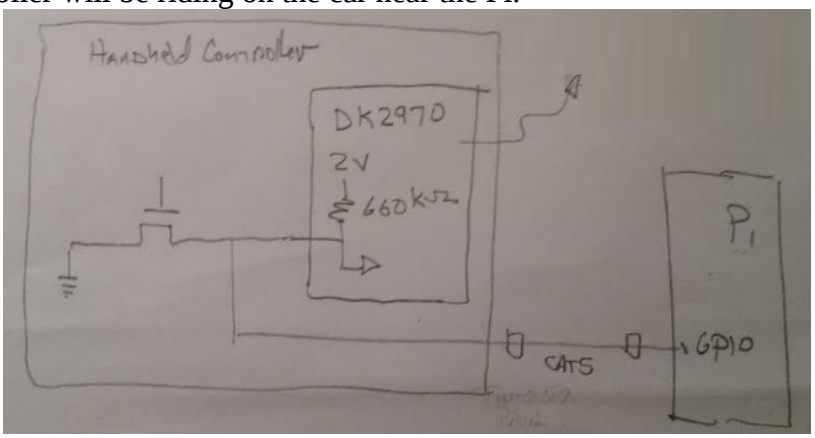
Some tactical considerations:
During training the Pi needs to be two places at the same time - on the car taking pictures, and near the Handheld Controller observing signals into the DK2970. To solve this we will use a cable so the Pi can be on the car, but can observe the signals going into the DK2970. There is no problem during Runtime since the Handheld Controller can be relocated to the car (no user is required) so it is co-resident with the Pi.

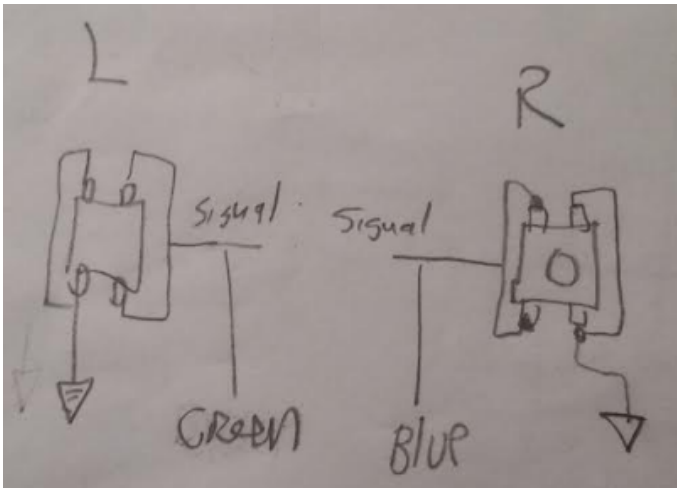
Hacking the Handheld Controller

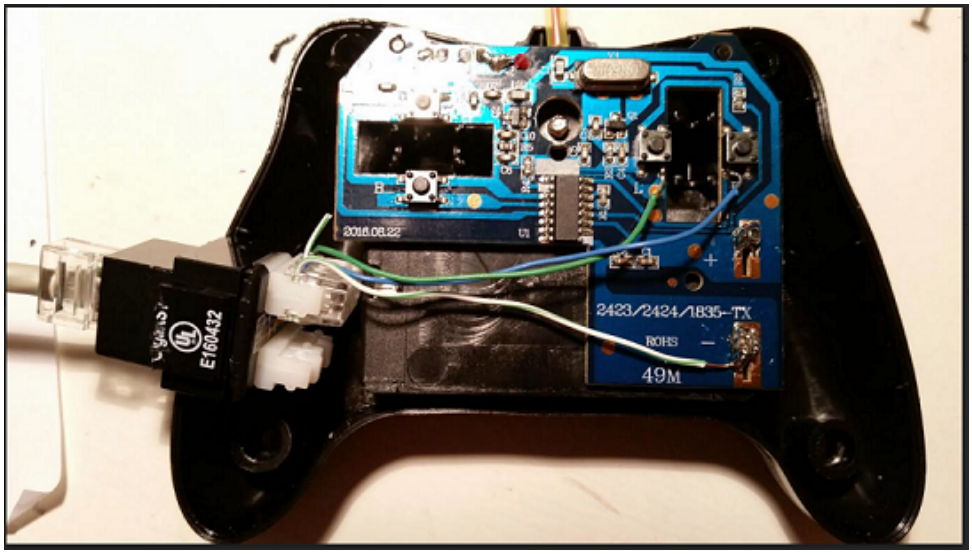
Inside the controller is a small circuit board with four button switches (left/right/forward/reverse), a transmitter chip (DK2970), and analog components to support the antenna. The chip handles everything - taking in the 4 control signals and outputting the 49mnz carrier. The control signal inputs are 2.1 volt “open drain” - a push of the button grounds the signal. The pullup is internal to the chip and measured at 660kohm. We are fortunate in that signal levels are compatible with the Pi GPIO.

During training the Pi wants to receive the left/right signals. During runtime the Pi will be driving them. The brute force approach (shown in the “Architecture Diagram”) involves breaking the signal between the button and the DK2970, having the Pi intercept, observe, and output the signals during training. In theory this is the correct approach, it would require 2 wires (a 'receive' and transmit') to/from the Pi for Left and 2 for Right (4 total).
A simpler approach can be pursued given the open-drain signal of this controller. During training the Pi simply observes the Handheld Controller signals - the push buttons on the Handheld Controller ground the wire to produce the signal. During runtime the handheld still provides the pullup, but the Pi will ground the wire to produce the signal. This means only 1 wire is needed between Handheld Controller and Pi for each of Left and Right (2 total).

Noise is a consideration especially considering the weak 660Kohm pullup and the length of the cable. We used CAT5 twisted pair to reduce noise exposure.
Finally - we added a CAT5 connector - this allows substituting a shorter cable during Runtime when the Handheld Controller will be riding on the car near the Pi.




L
R
Signal
Signal
GREEN
BLUE

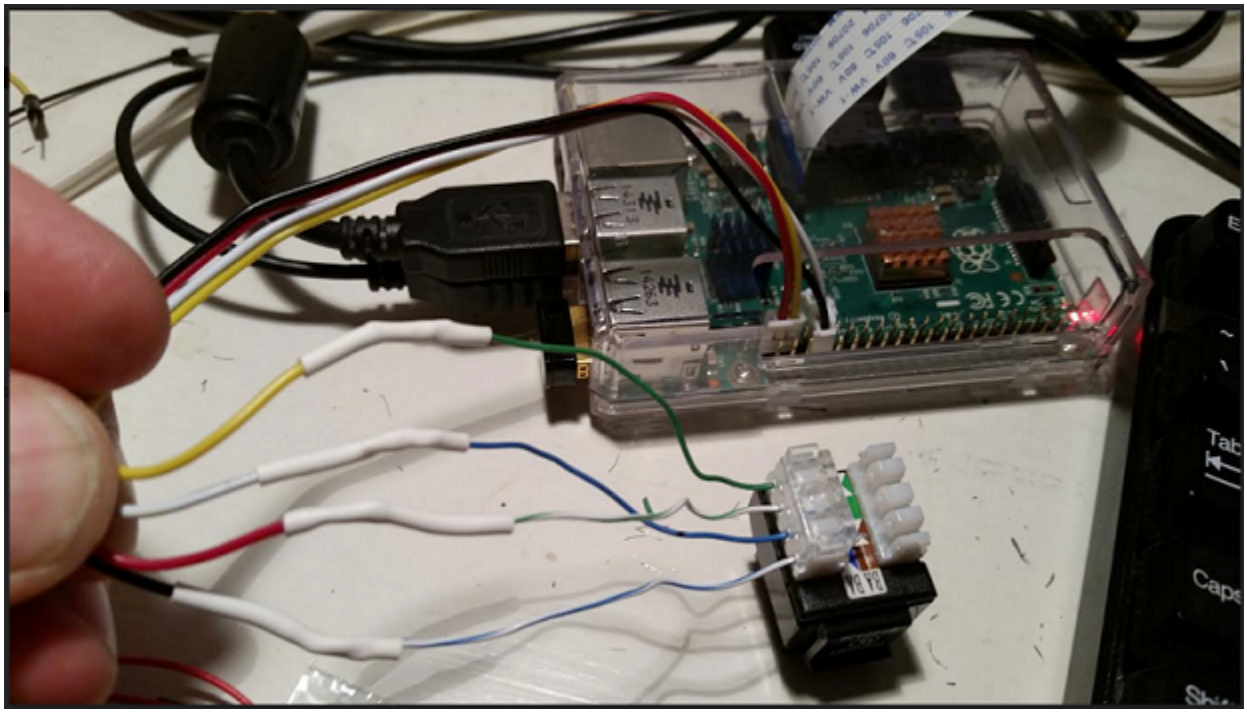

E160432
2016.08.22
2423/2424/1835-TX
ROHS
49M

GPIO connector and Software

For physical wiring we chose GPIOs 26 and 16 and again used a CAT5 connector. A reference for GPIO pinning on the Pi2-B (that is actually *correct*) is https://www.youtube.com/watch?v=6PuK9fh3aL8

Code to verify signaling between Pi GPIO and Handheld Controller is in Appendix X.
We use the “gpiozero' python library https://gpiozero.readthedocs.io/en/stable/index.html .

With the above wiring and software we are able to demonstrate the Pi can capture the necessary signal and camera training data, and can drive the left/right signals to turn the car. Now it is time to capture data!



Capturing Data

The Pi is zip-tied to the top of the car. A USB battery provides power for the Pi. The camera is rubber-banded to the front grill. The CAT5 provides the control signals to the Pi for observation during training.

We laid out an “Oval” track. We ran the car around the track in clockwise and counter-clockwise directions - a total of 11 loops. Each loop is about a minute and the capture rate is 2/second.

Raw data and Python code as is available on the Github.

The Dataset

The dataset is a collection of images and steering from a toy car as it is driven around a track.

The track is set up on a concrete floor. A piece of tape defines the centerline and the car will be trained to follow this line. During training the steering values applied by the driver to keep the car on the track are captured.

The dataset is split into training and test subsets of size 2582 and 646 respectively.

Images are square, size 64x64 using RGB encoding. This is kept as a 4-dimensional numpy array of numpy.uint8. The dimensions of this array are [N][64][64][3]. The first dimension is image number. The next two dimensions are height, width. The final dimension is color.

Steering values are categorical with 1=left, 3=straight 2=right. The data is kept as a 1-dimensional numpy array of numpy.uint8. The array is size [N] which is the direction the car is being steered at the time of the correspondingly-numbered image.

The structure of the dataset is:

- "DESCR" - an overview description of the dataset.
- "images_train" - training images
- "images_test" - testing image
- "steering_train" - training steering values
- "steering_test" - testing steering values
- "target_names" - a list of values of the steering class [1, 3, 2]

Appendix 1:
The following code was used when hacking the Handheld Contrlller and Pi to demonstrate operation of the physical wiring and connections between Handheld Controller and Pi GPIO via the CAT5 cable. This is a subset of the full code which is available on github.

The “train” switch sets the mode of operation. In “train” mode the Pi observes GPIOs and prints a left/right message when the Handheld Controller left/right buttons are pushed. In “non-train” (driving) mode the Pi will output left and right signals on a 1 second period. If the car is powered the front wheels will follow the Handheld Controller (training) or the Pi (driving). Camera images are also captured in “train” mode.

```
from gpiozero import InputDevice
from gpiozero import OutputDevice
from picamera import PiCamera
from time import sleep

train = True

pin_left  = 26
pin_right = 13

if train == True:

        pin_l = InputDevice(pin_left,True)
        pin_r = InputDevice(pin_right,True)

        camera = PiCamera()
        camera.start_preview()
        sleep(5)
        i = 0
        while True:
                if pin_l.value:
                        print("Left")
                if pin_r.value:
pi@raspberrypi2:~/tempdir$ more temp.py
from gpiozero import InputDevice
from gpiozero import OutputDevice
from picamera import PiCamera
from time import sleep

train = True

pin_left  = 26
pin_right = 13

if train == True:

        pin_l = InputDevice(pin_left,True)
        pin_r = InputDevice(pin_right,True)

        camera = PiCamera()
        camera.start_preview()
        sleep(5)
        i = 0
        while True:
                if pin_l.value:
                        print("Left")
                if pin_r.value:
                        print("Right")

                camera.capture('/home/pi/tempdir/pictures/image%s.jpg' % i)
                i = i + 1
                sleep(1)
        camera.stop_preview()


if train == False:
```

```
pin_l = OutputDevice(pin_left,True,True)
pin_r = OutputDevice(pin_right,True,True)

while True:
        print("center")
        pin_l.on()
        pin_r.on()
        sleep(1)

        print("right")
        pin_l.on()
        pin_r.off()
        sleep(1)

        print("center")
        pin_l.on()
        pin_r.on()
        sleep(1)

        print("left")
        pin_l.off()
        pin_r.on()
        sleep(1)
```