Steps1:

Install the anaconda

Step2:

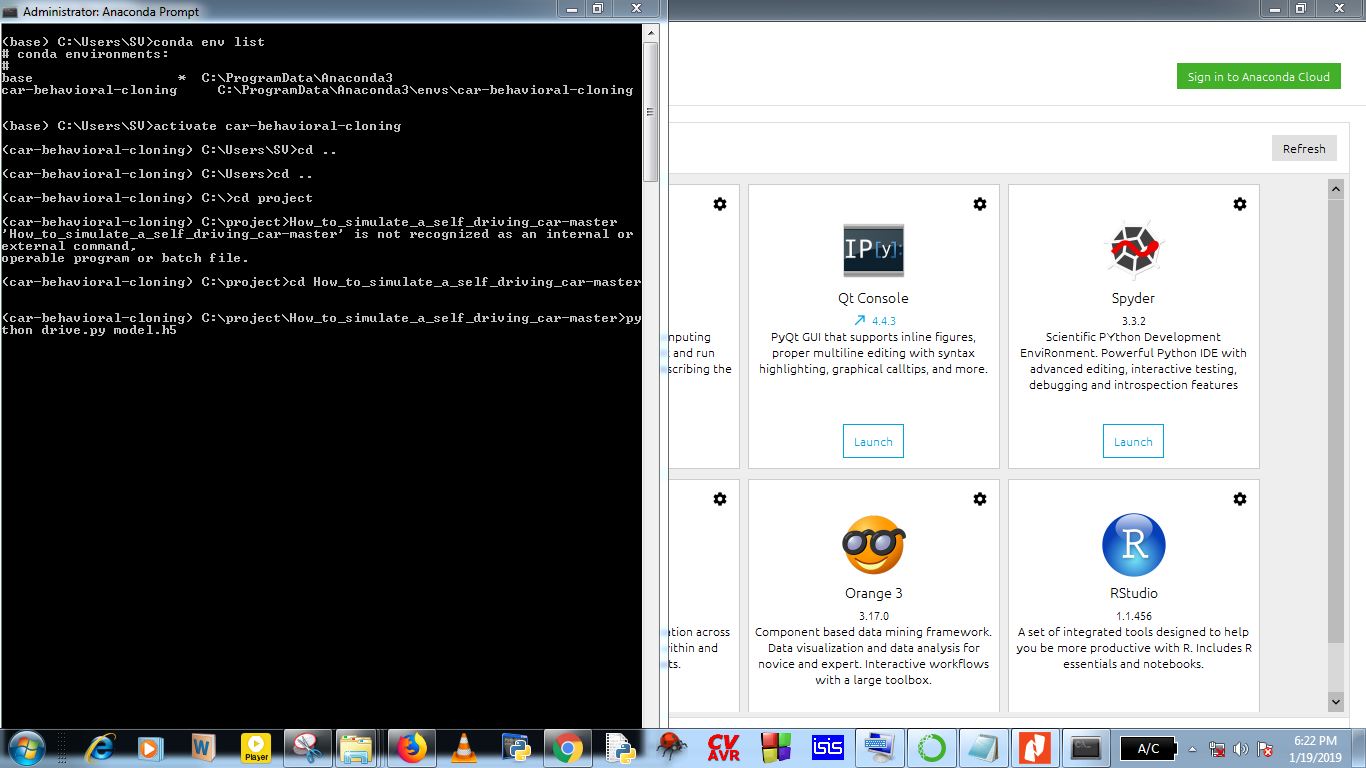
To set the environment of Anaconda, we have to run this command in Anaconda prompt

#TensorFlow without GPU

conda env create -f environments.yml

#Use TensorFlow with GPU

conda env create -f environments-gpu.yml



Step3:

Through simulator (training mode) we have to generate data set.

Use record button and select the directory where to generate data set (CSV file). Udacity simulator has two modes of operation.

Fig 1: Simulator preview, provided by Udacity

Download:-

https://github.com/udacity/self-driving-car-sim

step4:

In order to train our model, we are using Tensor flow machine learning libraries. Tensor flow library is based on the principle of CNN (Convolution Neural network):

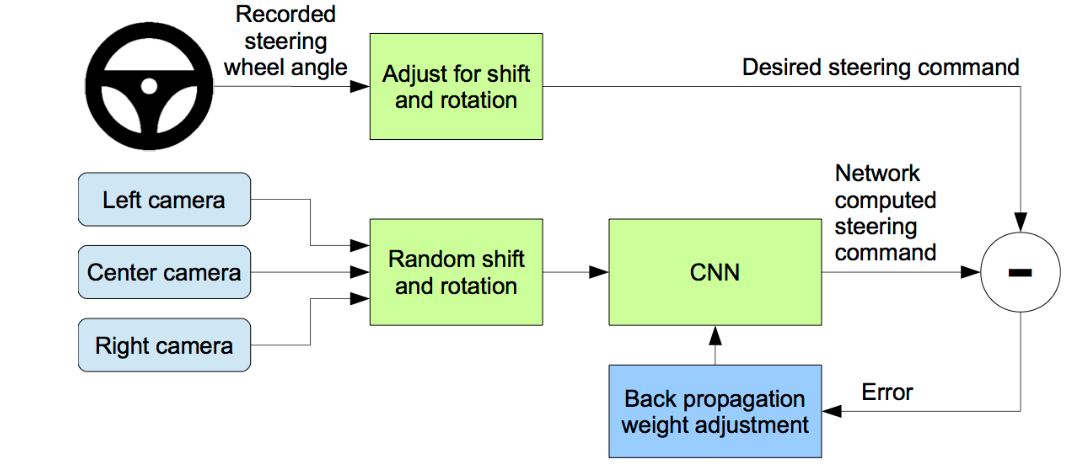


Fig 2: Block Diagram of Training software model.

In [deep learning](https://en.wikipedia.org/wiki/Deep_learning), a **convolutional neural network** (**CNN**) is a class of [deep neural networks](https://en.wikipedia.org/wiki/Deep_neural_network), most commonly applied to analyzing visual imagery.

CNNs use a variation of [multilayer perceptrons](https://en.wikipedia.org/wiki/Multilayer_perceptron) designed to require minimal [preprocessing](https://en.wikipedia.org/wiki/Data_pre-processing). They are also known as shift invariant, based on their shared-weights architecture and [translation invariance](https://en.wikipedia.org/wiki/Translation_invariance) characteristics.

Each neuron in a neural network computes an output value by applying some function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is specified by a vector of weights and a bias (typically real numbers). Learning in a neural network progresses by making incremental adjustments to the biases and weights.

In First time all neuron in a neural network in CNN assigns same weight randomly and convolve the internal network of neurons and give final output. After that it compare this final output with CSV file, which is generated by human drive input (steering). Onwards it uses to generate the error by difference. With the help of Back propagation feedback network. Every time it would reduce the error difference that would depend upon two factor .One factor is size of data set and another is Number of Epoch.

Step 3:

- Write Training script

Note: For training , the following cmd you have to type Anacoda Prompt:

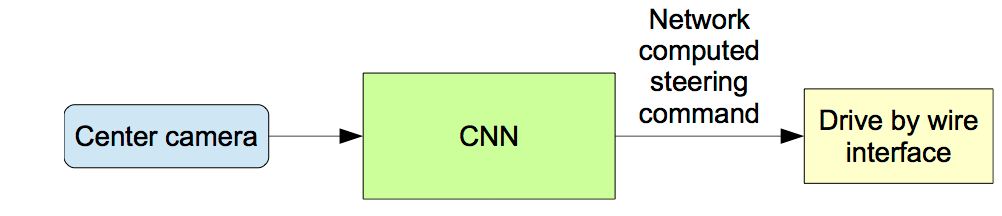
>Python model.py



This will generate model.h5 file.

Step 4:

Testing Model



Run the simulator in autonomous mode.

Run the client code on beaglebone.(socket programing)

Run the Program:

Environment set up

Go to Anaconda Promt

Activate car-behavioral-cloning

Go to your respective project folder, Run

Python drive.py model.h5



>BEAGLE BONE

## Testing the CAN driver

Let's create a virtual CAN device:

|  |
| --- |
| # modprobe vcan  # ip link add dev vcan0 type vcan  # ifconfig vcan0 up  # ip -details -statistics link show vcan0  5: vcan0: <noarp> mtu 16 qdisc noqueue state UNKNOWN mode DEFAULT group default  link/can promiscuity 0  vcan  RX: bytes packets errors dropped overrun mcast  0 0 0 0 0 0  TX: bytes packets errors dropped carrier collsns  0 0 0 0 0 0 |

Let's try sending/receiving data on that virtual device:

|  |
| --- |
| # candump vcan0 & |
| # cansend vcan0 442#DEADBABE |
| vcan0 442 [4] DE AD BA BE |

## **Using the real CAN**

## Plug your CAN transceiver the following way

|  |  |
| --- | --- |
| Pin | Description |
| 24 | CAN RX |
| 26 | CAN TX |

Make sure the dtbo is loaded in /boot/uEnv.txt:

|  |
| --- |
| ###Overide capes with eeprom uboot\_overlay\_addr0=/lib/firmware/BB-CAN1-00A0.dtbo |

If not, edit the file and reboot the board. You can check that the board is aware of its CAN interface:

|  |
| --- |
| #modprobe can |
| #modprobe can-raw |
| #modprobe can-dev |
| # cat /sys/kernel/debug/pinctrl/44e10800.pinmux/pinmux-pins | grep -i can  pin 96 (44e10980.0): P9\_26\_pinmux.53 (GPIO UNCLAIMED) function pinmux\_P9\_26\_can\_pin group pinmux\_P9\_26\_can\_pin  pin 97 (44e10984.0): P9\_24\_pinmux.51 (GPIO UNCLAIMED) function pinmux\_P9\_24\_can\_pin group pinmux\_P9\_24\_can\_pin |

Stop the driver, configure the link and start the interface again:

|  |
| --- |
| # ifconfig can0 down  # ip link set can0 up type can bitrate 125000 loopback off  # if config can0 up |

If everything goes right, the link should be working and you can query information about it:

|  |
| --- |
| # ip -details -statistics link show can0 |
| #candump can0 & |
| #cangen can0 -I i -L i -n 20 |

Initializing can module on system startup:  
ssh into beaglebone black through the following command on your terminal :  
 $ ssh debian@192.168.7.2  
$ sudo vim /etc/network/interfaces   
In that add these lines:  
      auto can0  
      iface can0 inet manual  
      pre-up /sbin/ip link set $IFACE type can bitrate 125000  
      up /sbin/ifconfig $IFACE up  
      down /sbin/ifconfig  
  
$ sudo vim /etc/modules  
  
In that add these lines:  
     modprobe can  
     modprobe can-raw  
     modprobe can-dev  
  
$ sudo reboot

NOTE: these all steps can also be found on :-

<http://www.embeddedsystemdesigning.com/p/blog-page_21.html>

Now install python-can on your BeagleBone (if not) using following steps:

From Python >3.3 the CAN Bus can be directly used via socketcan. For Python 2.7 the python-can-lib is used to communicate to the socketcan via the ctypes interface. The repository of the library can be found at https://bitbucket.org/hardbyte/python-can . The installation of the python-can lib. (version 1.5 from the 02.08.2016) can be done with the following lines:

$mkdir python-can

$cd python-can

$wget [https://bitbucket.org/hardbyte/python-can/get/6ab1d0eea5ff.zip --no-check-certificate](https://bitbucket.org/hardbyte/python-can/get/6ab1d0eea5ff.zip)

$unzip 6ab1d0eea5ff.zip

$cd hardbyte-python-can-6ab1d0eea5ff

$python setup.py install

NOW YOU ARE GOOD TO GO

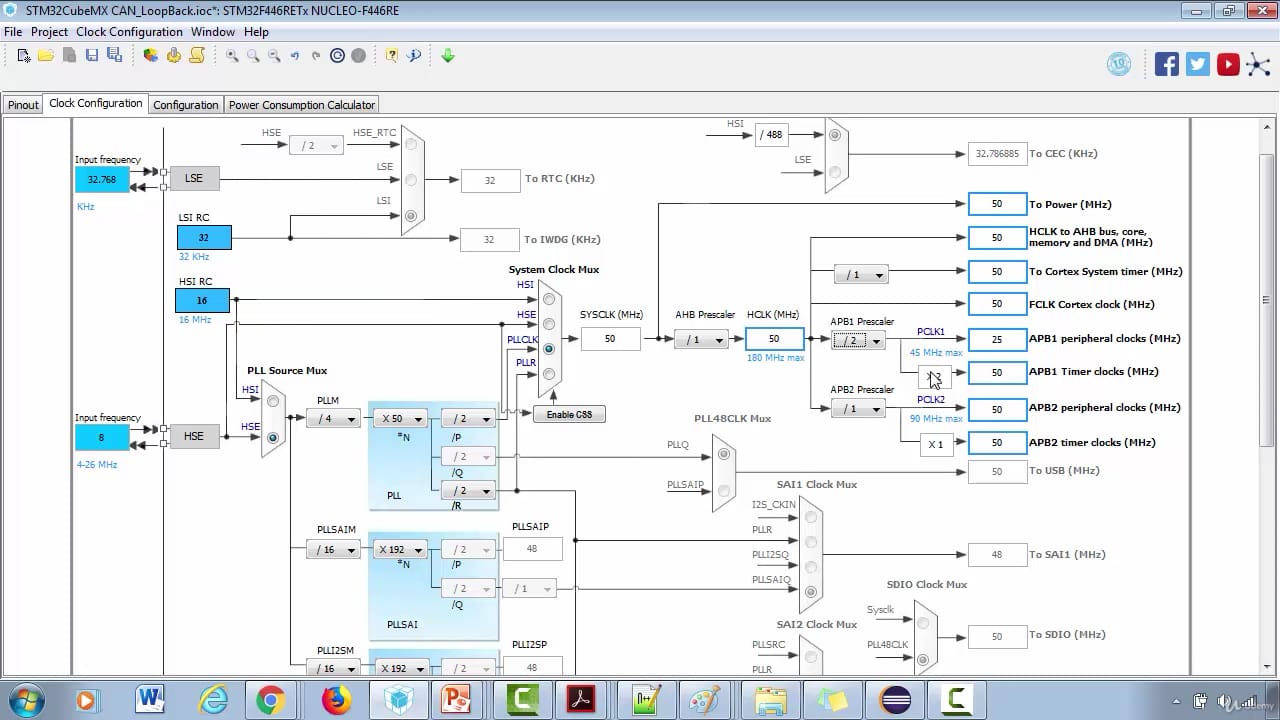
Now type send\_can.py or receive\_can.py(or test using “*$candump can0*” for receiveing CAN frames ) for CAN testing(sending and receiving) on BeaglBone Black

As Beagle Bone Black is acting as client for drive.py script as well as CAN node for STM32.

Type send\_socket.py(on pyCharm on PC) and receive\_socket.py(on beaglebone black ) for testing socket programing between PC and beaglebone black

So, we have to embed socket programming (receive\_socket.py) and CAN coding(send\_can.py) into a single python script and just run the script

This will connect to the socket opened by the drive.py and will acquire the data, which will be in string form, but on CAN it can only send data in HEX from hence we are comparing the first element of string if it is ‘0’,’+’or ’-’and will just send 0, 1 or 2 respectively over CAN to the STM32 and will blink LEDs accordingly. And the final code is beaglebone.py

ARM CAN CLOCK CONFIGRATION 

Link for beaglebone black codes mentioned above:

<https://drive.google.com/drive/folders/1E59WfuYn9DNkILBiXYrZouZ4Uor6rY8e?usp=sharing>

Link for codes to be run on PC (simulator and CNN)mentioned above:

<https://drive.google.com/drive/folders/1cc8fGZWgrWGqyXXZzP44NNYE27_58wis?usp=sharing>

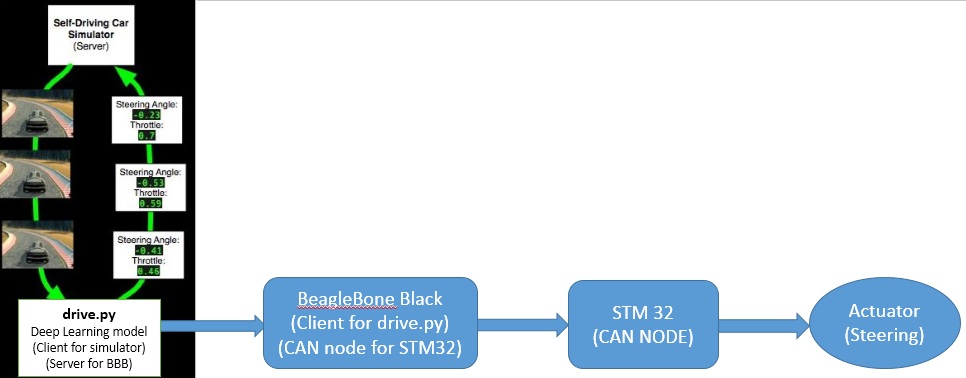
Link for STM32 codes mentioned above:

<https://drive.google.com/drive/folders/1Dau_R6e9YvsiUQI-ilWLb1lED8lLBQBU?usp=sharing>

ALL CODES:

<https://drive.google.com/drive/folders/1zusCykrPq7CSGck5XNjjYUgJ_C6By47m?usp=sharing>

BIG PICTURE



REFERENCES :

<https://in.udacity.com/course/self-driving-car-fundamentals-featuring-apollo--ud0419>

<https://www.youtube.com/watch?v=EaY5QiZwSP4&vl=en>

<https://github.com/llSourcell/How_to_simulate_a_self_driving_car>

<https://github.com/hardbyte/python-can>

<https://www.youtube.com/watch?v=r7-WPbx8VuY>

<https://www.youtube.com/watch?v=uIm3DMprk7M>

<https://www.youtube.com/watch?v=umGJ30-15_A&t=594s>

<https://www.youtube.com/watch?v=yX8KuPZCAMo&t=2617s>

<https://becominghuman.ai/behavioral-cloning-in-deep-learning-using-keras-3786f5914b72>

<https://medium.com/@maxdeutsch/how-to-build-a-self-driving-car-in-one-month-d52df48f5b07>

<https://shrikar.com/self-driving-car-lane-detection/>

<https://github.com/sarchak/self_driving_car/tree/master/CarND-LaneLines-P1>

<https://www.youtube.com/watch?v=ndUNTPlJnfM>

<https://github.com/maunesh/advanced-lane-detection-for-self-driving-cars>

<https://github.com/ndrplz/self-driving-car>

<https://github.com/SullyChen/driving-datasets>

<https://github.com/SullyChen/Autopilot-TensorFlow>

<https://towardsdatascience.com/understanding-convolutional-neural-networks-221930904a8e>

<https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>