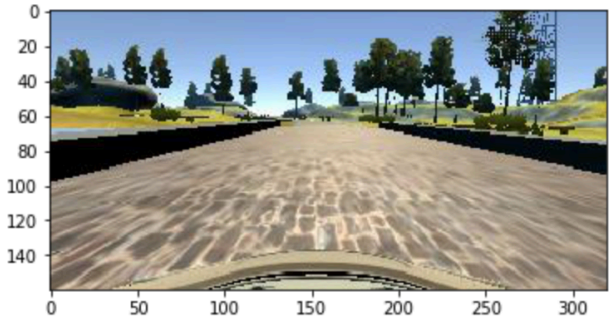
This is the second submit of my work.
As I did not pass the rubric: adding examples of images from your dataset and images resulted from your augmentation.

So here are the demo images created in jupyter notebook, to show what a flipped image looked like.

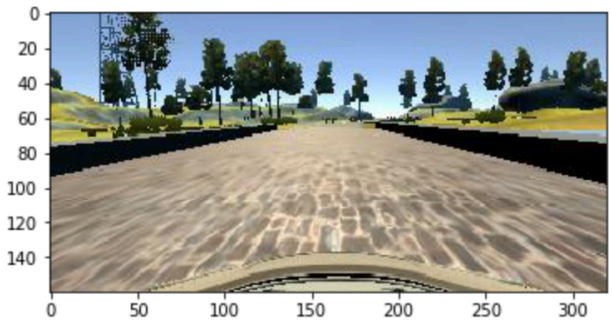
In [33]:
```
%matplotlib inline
image_path = "./data/IMG/center_2016_12_01_13_30_48_287.jpg"
# print('This image is:', type(image), 'with dimensions:', image.shape)
image = cv2.imread(image_path)
image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

Out[33]: <matplotlib.image.AxesImage at 0x12e9bc438>



In [4]:
```
image = cv2.flip(image, 1)
plt.imshow(image)
```

Out[4]: <matplotlib.image.AxesImage at 0x12bcd2be0>

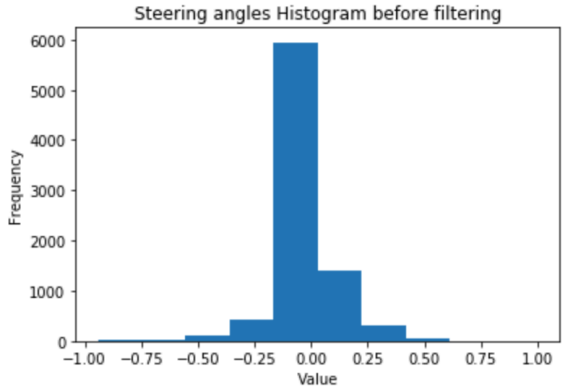


besides this correction, I also optimized my codes a lot as showed below:

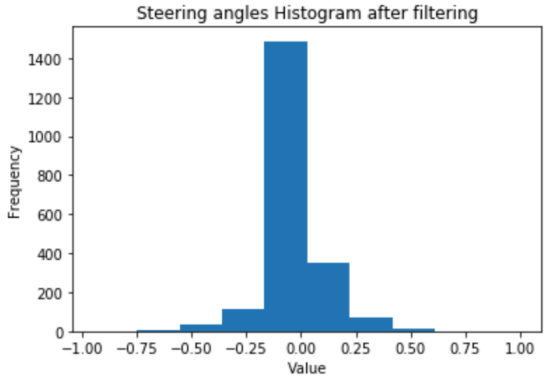
I printed out the histogram before / after the filtering(to drop out redundancy angles at 0deg)

In [31]:
```
angles = []
samples = []
center_counter = 0
for sample in lines:
    center_value = float(sample[3])
    angles.append(center_value)
    samples.append(sample)

plt.hist(angles)
plt.title("Steering angles Histogram before filtering")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.gcf()
plt.show()
```



In [27]:
```
plt.hist(angles)
plt.title("Steering angles Histogram after filtering")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.gcf()
plt.show()
```



similar distribution, but with reduced sample quantity.

I also improved the pipeline:

I have 3 functions in the first submission:

Cropping(outside the covnet) -> resizing(outside the convet) -> normalization(inside the covnet)

Def generator:

…
image = image[70:140,10:310]
image = cv2.resize(image,(24,24))

```
          …
model = Sequential()
row,col,ch = 24,24,3
model.add(Lambda(lambda x: x/255 - 0.5,input_shape=(row, col, ch)))
```

as suggested, this could not be the best structure, it is requested to put those 3 functions into the covnet, so I updated my codes:

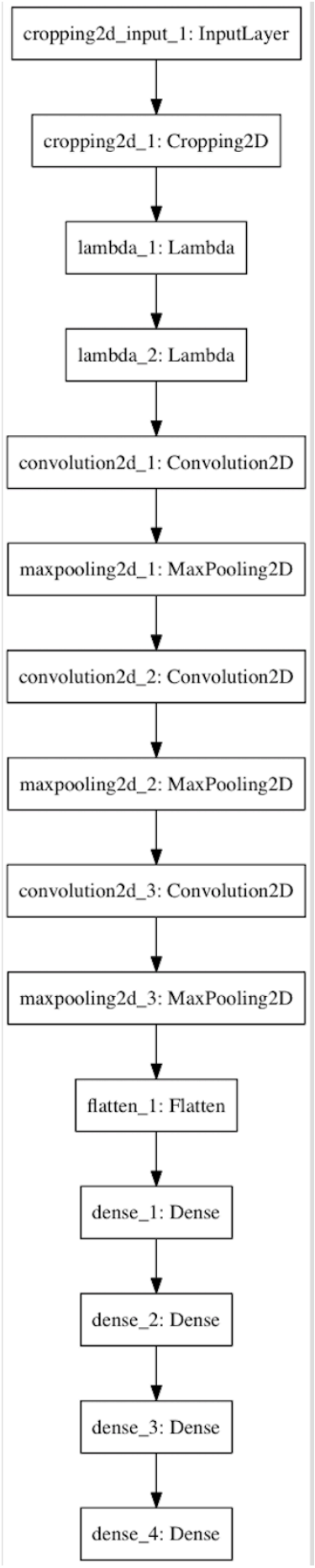
```
from keras.backend import tf as ktf
model = Sequential()
input_shape = (160,320,3)
model.add(Cropping2D(cropping=((70,20),(10,10)),input_shape = input_shape))
model.add(Lambda(lambda x: ktf.image.resize_images(x,(24,24))))
model.add(Lambda(lambda x: x/255 - 0.5))
```

in this case, there is no need to change the drive.py, so I delete those words former I added in my first attampt:

```
image_array = imgge_array[70:140,10:310]
image_array = cv2.resize(image_array,(24,24))
```

```
then I added the covnet visualization

from keras.utils.visualize_util import plot
plot(model, to_file='model.png')
```


cropping2d_input_1: InputLayer
cropping2d_1: Cropping2D
lambda_1: Lambda
lambda_2: Lambda
convolution2d_1: Convolution2D
maxpooling2d_1: MaxPooling2D
convolution2d_2: Convolution2D
maxpooling2d_2: MaxPooling2D
convolution2d_3: Convolution2D
maxpooling2d_3: MaxPooling2D
flatten_1: Flatten
dense_1: Dense
dense_2: Dense
dense_3: Dense
dense_4: Dense

then I introduced the modelcheckpoint usage:

checkpointer = ModelCheckpoint(filepath='model.h5', verbose=1, save_best_only=True)

with this call back method I achieved an early stopping automatically. I can get a good balance between underfitting and overfitting.

And here is the visualization of the MSE loss.

```
Epoch 00001: val_loss improved from 0.03119 to 0.03009, saving model to model.h5
Epoch 00002: val_loss improved from 0.03009 to 0.02921, saving model to model.h5
Epoch 00003: val_loss improved from 0.02921 to 0.02818, saving model to model.h5
Epoch 00004: val_loss improved from 0.02818 to 0.02799, saving model to model.h5
Epoch 00005: val_loss improved from 0.02799 to 0.02752, saving model to model.h5
Epoch 00006: val_loss did not improve
Epoch 00007: val_loss did not improve
Epoch 00008: val_loss did not improve
Epoch 00009: val_loss did not improve
```