ROBOTIC GRIPPER

A Robotic Gripper Operated by Gestures Learned Trough DeepLearning

This project allows a user to control a robotic gripper using gestures captured by a webcam.

1 - How does it works

The project is diveded in 3 main phases, in order to fulfill user requests:

```
- Phase 1: Images must be captured from the webcam to compound a labeled
 gestures dataset.
The dataset will feed trainning and testing datasets to be used in superv
ised learning.

- Phase 2: A deep learning model, basically a neural network, will be cre
ated and used to train the gestures recognition, using keras and tensorfl
ow.

- Phase 3: A program will be used to sequentially capture webcam images.
The images will be classifyed by the model trainned in Phase 2, and the r
esult will be used to operate the robotic gripper.
```

In [1]:

```
%load_ext autoreload
%autoreload 2
```

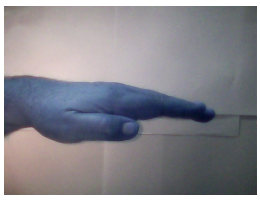
2 - Capturing labeled gestures images

Images will be captured from the webcam. A folder named **capture** will have several subfolders. The subfolders will have meaningful names, such as **left**, **right**, and so on. The subfolder named **left** will hold images of teh gesture that yields the command **turn to the left**. This is so that later the subfolders name will become the ground truth values of the datasets for the machine learning process.

For controlling the robotic gripper, we are going to use nine commands:

1. nothing
2. left
3. right
4. up
5. down
6. foward
7. back
8. grip
9. loose

Some examples of images are:

				
nothing	left	right	grip	loose
				
foward	back	up	down	

In [2]:

```
# imports

%pylab inline
import cv2
from IPython.display import clear_output
import time
from datetime import datetime
import os
import numpy as np
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [5]:

```
"""
    function  start_webcam_capture
```

```
    parameters:
    path - the path to save captured gesture images files
"""
def start_webcam_capture(path, number_of_captures=10):
    # variables to define play warning sound
    frequency = 100 # Hertz
    duration  = 50 # milliseconds
    #lets make sure the path exists!
    if not os.access(path, os.F_OK):
        os.makedirs(path)
    count_captures = 0
    #using webcam 0.
    #in some systems webcam may be under different numbers, i.e, 1 or 2 or 3 ...
    vid = cv2.VideoCapture(0)
    start_time = time.time()
    try:
        while(count_captures<number_of_captures):
            # Capture frame-by-frame
            ret, frame = vid.read()
            if not ret:
                # Release the Video Device if ret is false
                vid.release()
                # Message to be displayed after releasing the device
                print("Released Video Resource due to capture fail!")
                break
            # Convert the image from OpenCV BGR format to matplotlib RGB format
            # to display the image
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            # check if it is time to save frame to a file
            elapsed_time = time.time() - start_time
            if elapsed_time > 4:
                # make sound to indicate action
                os.system('play -n synth %s sin %s' % (duration/1000, frequency
))
                timestamp = datetime.utcnow().strftime('%Y_%m_%d_%H_%M_%S_%f')[:
-3]
                timestamp = timestamp + '.jpg'
                image_filename = os.path.join(path, timestamp)
                #print(image_filename)
                cv2.imwrite(image_filename, frame)
                #increment count_captures
                count_captures += 1
                #restart the timer
                start_time = time.time()
            # check for ESC
            key = np.int16(cv2.waitKey(1))
            if key == 27:
                print("Esc key interrupted!")
                break  # esc to quit
            # Turn off the axis
            axis('off')
            # Title of the window
            title("Robotic Gripper Gestures Capture")
            # Display the frame
            imshow(frame)
            show()
            # Display the frame until new frame is available
            clear_output(wait=True)
    except KeyboardInterrupt:
        # Message to be displayed after releasing the device
        print("keyboard interrupted!")
```

```
    # Release the Video Device
    vid.release()
    print("Released Video Resource")
    path, dirs, files = os.walk(path).__next__()
    file_count = len(files)
    print('There are now ', file_count, ' images in ', path)
```

Let's start by capturing the gesture for **nothing**. When you are done, select **Kernel** on jupyter notebook menu and then select **Interrupt** As the file names are bases on a complete and unique timestamp, if you wish, you can run the same code again to add more gestures images. You can even visually select and remove some files (in case of a mistake) using a external file manager from your operating system.

In [ ]:

```
path = 'capture/nothing'
#start capturing gesture images
start_webcam_capture(path)
```

Let's capture te gesture for **left**.

In [ ]:

```
path = 'capture/left'
#start capturing gesture images
start_webcam_capture(path)
```

Let's capture te gesture for **right**.

In [6]:

```
path = 'capture/right'
#start capturing gesture images
start_webcam_capture(path)
```

```
Released Video Resource
There are now  10  images in  capture/right
```

Let's capture te gesture for **up**.

In [7]:

```
path = 'capture/up'
#start capturing gesture images
start_webcam_capture(path)
```

```
Released Video Resource
There are now  10  images in  capture/up
```

Let's capture te gesture for **down**.

In [8]:

```
path = 'capture/down'
#start capturing gesture images
start_webcam_capture(path)
```

```
Released Video Resource
There are now  10  images in  capture/down
```

Let's capture te gesture for **foward**.

In [9]:

```
path = 'capture/foward'
#start capturing gesture images
start_webcam_capture(path)
```

```
Released Video Resource
There are now  10  images in  capture/foward
```

Let's capture te gesture for **back**.

In [10]:

```
path = 'capture/back'
#start capturing gesture images
start_webcam_capture(path)
```

```
Released Video Resource
There are now  10  images in  capture/back
```

Let's capture te gesture for **grip**.

In [11]:

```
path = 'capture/grip'
#start capturing gesture images
start_webcam_capture(path)
```

```
Released Video Resource
There are now  10  images in  capture/grip
```

Let's capture te gesture for **loose**.

In [12]:

```
path = 'capture/loose'
#start capturing gesture images
start_webcam_capture(path)
```

```
Released Video Resource
There are now  10  images in  capture/loose
```

3 - Build the Model and train it using the captured gestures from the first phase

We are going to build our deep learning (https://en.wikipedia.org/wiki/Deep_learning) robotic gripper gesture commands model using Keras (https://keras.io/) and TensorFlow (https://www.tensorflow.org/).

In [13]:

```
#imports

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.layers import Lambda, Conv2D, MaxPooling2D, Dropout, Dense, Flatten
from utils import INPUT_SHAPE, batch_generator
import argparse
import os
import cv2
import sys

np.random.seed(0)
```

```
Using TensorFlow backend.
```

The function **load_images_from_path** is auxiliary to the function **load_data**.

In [14]:

```
'''
    load_images_from_path
'''
def load_images_from_path(path, result, images, results):
    for filename in os.listdir(path):
      img = os.path.join(path,filename)
      if img is not None:
        images.append(img)
        results.append(result)
    return images, results
```

In [17]:

```
def load_data():
  images = []
  results =[]
  labels = ['nothing', 'left', 'right', 'grip', 'loose', 'foward', 'back', 'up',
 'down']

  #load a list of images and a corresponding list of results (images=640x480)
  images, results = load_images_from_path('capture/nothing/', 0, images, results
)
  images, results = load_images_from_path('capture/left/', 1, images, results)
  images, results = load_images_from_path('capture/right/', 2, images, results)
  images, results = load_images_from_path('capture/grip/', 3, images, results)
  images, results = load_images_from_path('capture/loose/', 4, images, results)
  images, results = load_images_from_path('capture/foward/', 5, images, results)
  images, results = load_images_from_path('capture/back/', 6, images, results)
  images, results = load_images_from_path('capture/up/', 7, images, results)
  images, results = load_images_from_path('capture/down/', 8, images, results)

  X_train, X_valid, y_train, y_valid = train_test_split(images, results, test_si
ze=0.2, shuffle = True, random_state=0)

  return X_train, X_valid, y_train, y_valid
```

In [18]:

```
X_train, X_valid, y_train, y_valid = load_data()

print("Train Images: ", len(X_train))
print("Valid Images: ", len(X_valid))
print("Train Results: ", len(y_train))
print("Valid Results: ", len(y_valid))

# if we wish to check some of the images, just change de index value
# note that the index can't be bigger than the number of images -1
#cv2.imshow('Capture', cv2.imread(X_train[80]))
#print(X_train[80])
#print(labels[results[80]])
#cv2.waitKey(0)
#cv2.destroyAllWindows()
#sys.exit(0)
```

```
Train Images:  79
Valid Images:  20
Train Results:  79
Valid Results:  20
```

In [23]:

```
def build_model(keep_prob):
    """
    Modified NVIDIA model
    """
    model = Sequential()
    model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
    model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(64, 3, 3, activation='elu'))
    model.add(Conv2D(64, 3, 3, activation='elu'))
    model.add(Dropout(keep_prob))
    model.add(Flatten())
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))
    model.summary()

    return model
```

Let's build the model.

In [24]:

```
keep_prob = 0.5
model = build_model(keep_prob)
```

```
____________________________________________________________________________
____________________________________
Layer (type)                 Output Shape              Param #
Connected to
============================================================================
==================================
lambda_2 (Lambda)            (None, 120, 160, 3)       0
lambda_input_2[0][0]

____________________________________________________________________________
____________________________________
convolution2d_6 (Convolution2D)  (None, 58, 78, 24)    1824
lambda_2[0][0]

____________________________________________________________________________
____________________________________
convolution2d_7 (Convolution2D)  (None, 27, 37, 36)    21636
convolution2d_6[0][0]

____________________________________________________________________________
____________________________________
convolution2d_8 (Convolution2D)  (None, 12, 17, 48)    43248
convolution2d_7[0][0]

____________________________________________________________________________
____________________________________
convolution2d_9 (Convolution2D)  (None, 10, 15, 64)    27712
convolution2d_8[0][0]

____________________________________________________________________________
____________________________________
convolution2d_10 (Convolution2D) (None, 8, 13, 64)     36928
convolution2d_9[0][0]

____________________________________________________________________________
____________________________________
dropout_1 (Dropout)          (None, 8, 13, 64)     0
convolution2d_10[0][0]

____________________________________________________________________________
____________________________________
flatten_1 (Flatten)          (None, 6656)          0
dropout_1[0][0]

____________________________________________________________________________
____________________________________
dense_1 (Dense)              (None, 100)           665700
flatten_1[0][0]

____________________________________________________________________________
____________________________________
dense_2 (Dense)              (None, 50)            5050
dense_1[0][0]

____________________________________________________________________________
____________________________________
dense_3 (Dense)              (None, 10)            510
dense_2[0][0]

____________________________________________________________________________
____________________________________
dense_4 (Dense)              (None, 1)             11
dense_3[0][0]
============================================================================
==================================
Total params: 802,619
Trainable params: 802,619
Non-trainable params: 0

____________________________________________________________________________
____________________________________
```

In [27]:

```
def train_model(model, psave_best_only, learning_rate, samples_per_epoch, nb_epo
ch, batch_size, X_train, X_valid, y_train, y_valid):
    """
    Train the model
    """
    checkpoint = ModelCheckpoint('model-{epoch:03d}.h5',
                                 monitor='val_loss',
                                 verbose=0,
                                 save_best_only=psave_best_only,
                                 mode='auto')

    model.compile(loss='mean_squared_error', optimizer=Adam(lr=learning_rate))

    model.fit_generator(batch_generator(X_train, y_train, batch_size, True),
                        samples_per_epoch,
                        nb_epoch,
                        max_q_size=1,
                        validation_data = batch_generator(X_valid, y_valid, batc
h_size, False),
                        nb_val_samples=len(X_valid),
                        callbacks=[checkpoint],
                        verbose=1)
```

Let's train the model.

In [28]:

```
psave_best_only = True
learning_rate = 1.0e-4
samples_per_epoch = 20000
nb_epoch = 10
batch_size = 40
train_model(model, psave_best_only, learning_rate, samples_per_epoch, nb_epoch,
batch_size,
            X_train, X_valid, y_train, y_valid)
```

```
Epoch 1/10
20000/20000 [==============================] - 454s - loss: 2.6744
- val_loss: 0.2206
Epoch 2/10
20000/20000 [==============================] - 454s - loss: 0.2870
- val_loss: 2.9278
Epoch 3/10
20000/20000 [==============================] - 453s - loss: 0.1437
- val_loss: 0.0942
Epoch 4/10
20000/20000 [==============================] - 453s - loss: 0.0824
- val_loss: 0.3310
Epoch 5/10
20000/20000 [==============================] - 454s - loss: 0.0637
- val_loss: 1.8243
Epoch 6/10
20000/20000 [==============================] - 448s - loss: 0.0480
- val_loss: 1.7545
Epoch 7/10
20000/20000 [==============================] - 447s - loss: 0.0377
- val_loss: 4.1550
Epoch 8/10
20000/20000 [==============================] - 444s - loss: 0.0280
- val_loss: 6.2238
Epoch 9/10
20000/20000 [==============================] - 449s - loss: 0.0233
- val_loss: 5.9612
Epoch 10/10
20000/20000 [==============================] - 446s - loss: 0.0196
- val_loss: 8.1218
```