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 ge
stures dataset.
The dataset will feed trainning and testing datasets to be used in supervis
ed learning.

- Phase 2: A deep learning model, basically a neural network, will be creat
ed and used to train the gestures recognition, using keras and tensorflow.

- 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 res
ult 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

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')[
                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:
```

```
60          # Message to be displayed after releasing the device

61          print("keyboard interrupted!")
62      # Release the Video Device
63      vid.release()
64      print("Released Video Resource")
65      path, dirs, files = os.walk(path).__next__()
66      file_count = len(files)
67      print('There are now ', file_count, ' images in ', path)
68
```

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 [ ]:

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

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

In [ ]:

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

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

In [6]:

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

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

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

In [7]:

```
1 path = 'capture/up'
2 #start capturing gesture images
3 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 [ ]:

```
#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)
```

In [ ]:

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

In [ ]:

```
def load_data(args):
  images = []
  results =[]
  labels = ['nothing', 'left', 'right', 'grip', 'loose']

  #load a list of images and a corresponding list of results (images=640x480)
  images, results = load_images_from_folder('capture/nothing01/', 0, images, re
  images, results = load_images_from_folder('capture/left01/', 1, images, resul
  images, results = load_images_from_folder('capture/right01/', 2, images, resu
  images, results = load_images_from_folder('capture/grip01/', 3, images, resul
  images, results = load_images_from_folder('capture/loose01/', 4, images, resu

  print("Images: ", len(images))
  print("Results: ", len(results))
  print("labels: ", len(labels), labels)

  # 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(images[80]))
  #print(images[80])
  #print(labels[results[80]])
  #cv2.waitKey(0)
  #X = np.asarray(images)
  #y = np.asarray(results)
  #X = X.reshape(len(images),1)
  #y = y.reshape(len(results),1)
  #print('X shape: ', X.shape)
  #print('y shape: ', y.shape)
  X_train, X_valid, y_train, y_valid = train_test_split(images, results, test_s

  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)

  return X_train, X_valid, y_train, y_valid
```

In [ ]:

```
def build_model(args):
    """
    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(args.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
```

In [ ]:

```
def train_model(model, args, 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=args.save_best_only,
                                 mode='auto')

    model.compile(loss='mean_squared_error', optimizer=Adam(lr=args.learning_ra

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

In [ ]:

```
def s2b(s):
    """
    Converts a string to boolean value
    """
    s = s.lower()
    return s == 'true' or s == 'yes' or s == 'y' or s == '1'
```

In [ ]:

```
def main():
    """
    Load train/validation data set and train the model
    """
    parser = argparse.ArgumentParser(description='Behavioral Cloning Training P
    parser.add_argument('-d', help='capture directory',        dest='capture_di
    parser.add_argument('-t', help='test size fraction',    dest='test_size',
    parser.add_argument('-k', help='drop out probability',  dest='keep_prob',
    parser.add_argument('-n', help='number of epochs',      dest='nb_epoch',
    parser.add_argument('-s', help='samples per epoch',     dest='samples_per_e
    parser.add_argument('-b', help='batch size',            dest='batch_size',
    parser.add_argument('-o', help='save best models only', dest='save_best_onl
    parser.add_argument('-l', help='learning rate',         dest='learning_rate
    args = parser.parse_args()

    print('-' * 30)
    print('Parameters')
    print('-' * 30)
    for key, value in vars(args).items():
        print('{:<20} := {}'.format(key, value))
    print('-' * 30)

    data = load_data(args)
    model = build_model(args)
    train_model(model, args, *data)
```

In [ ]:

```
main()
```