IT Carlow –

BSc.

Software Development

UAV using Convolutional Neural Networks

Technical Manual

Student Name: Josh Hudziak

Student Email: C00231846@itcarlow.ie

Date: 30/04/2021

Supervisor: Dr. Oisin Cawley

Supervisor Email: oisin.cawley@itcarlow.ie

GitHub

All files and documentation can be found on my GitHub. Here you will also see the file layout and how the dataset is stored.

Model Files

Due to technical difficulties the model was trained on a different laptop than the UAV is being run on. This is due to the laptop that trains the models is superior in computational power but cannot host Ubuntu 18.04 which is needed to run Olympe, the Parrot Anafi's controller.

The following sections code is run on Jupyter Notebooks which is running on Anaconda. The following paragraph explains how to install each component and links to desirable sites for troubleshooting and downloads.

- Anaconda: Download anaconda: https://www.anaconda.com/products/individual
    - Initiate anaconda shell and enter these commands to create a new environment called tensorflow
        1. conda install -y jupyter
        2. conda create --name tensorflow
        3. conda activate tensorflow
- CUDDN (for nvidia GPUs only): Follow Windows instructions on https://tensorflow.org/install/gpu **(Make sure to check software compatabilities)**
- Tensorflow:
    - While in the tensorflow virtual environment made earlier, run:
        1. pip install tensorflow
- Sklearn:
    - pip install sklearn
- Pickle:
    - pip install pickle
- Numpy:
    - pip install numpy
- matplotlib:
    - pip install matplotlib
- imutils:
    - pip install imutils
- wandb:
    - pip install wandb

Frame_Splitter.ipynb

Import Libraries

```
import cv2
import logging
import os
import random
import string
```

Get the length of a video

```
def video_length(video_path):
    cap = cv2.VideoCapture(video_path)
    #Built in function with cv2 to get total frames
    length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    return length
```

Implement a random name generator

```
def rand_string(length):
        rand_str = ''.join(random.choice(string.ascii_lowercase +
string.ascii_uppercase + string.digits) for i in range(length))
    return rand_str
```

Extract frames from a video

```
def extract_frames(video_path, save_path, skip_frames = 2):
    _, filename = os.path.split(video_path)
    filename_ext_drop = os.path.splitext(filename)[0]

    #check file
    length = video_length(video_path)
    if length ==0:
        print('No Video detected!')
        return 0

    #Create a video object
    cap = cv2.VideoCapture(video_path)

    #Count the frames
    count = 0
    random_string = rand_string(5)

    ret, frame = cap.read()
      test_path = os.path.join(save_path, filename_ext_drop[:6] +
"{}_{}.jpg".format(random_string, count))

    cv2.imwrite(test_path, frame)
    if os.path.isfile(test_path):
        print('Test sucessful, Continuing extraction')
        count =1
        while ret:
            ret, frame = cap.read()
            if ret and count % skip_frames == 0:
                                    cv2.imwrite(os.path.join(save_path,
filename_ext_drop[:6]+'{}_{}.jpg'.format('_frame_last', count)), frame)
                count += 1
                #print(count)
            else:
                count+=1
    else:
        print('Cannot save file')
        return 0

    cap.release()
    print('Finished!')

if __name__=="__main__":
    video = ["SomeVideo.MOV"]
    save_path = "InsertRawDataset"
    for this_video in video:
        print(this_video)
        extract_frames(this_video, save_path, skip_frames = 2)
```

Split_Dataset.ipynb

Purpose:

- Split the data set into 80% training and 20% test data.

```
import splitfolders
input_folder = r"InsertRawDataset"
output = r"Dataset"
splitfolders.ratio(input_folder, output, seed=42, ratio=(.9, .1))
help(splitfolders.ratio)
```

convolution_nueral_network.ipynb

```
Importing the libraries
##TensorFlow library, Implement Deep learning
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Activation,
Flatten, Add, ZeroPadding2D
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Input,
BatchNormalization, AveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.callbacks import ModelCheckpoint

import matplotlib.pyplot as plt
from PIL import Image
from matplotlib import cm
from mpl_toolkits.axes_grid1 import ImageGrid
import math
%matplotlib inline

import numpy as np
from skimage import io
from imutils import paths
from sklearn.preprocessing import LabelBinarizer
import pickle
from sklearn.metrics import classification_report


from tensorflow.compat.v1.keras.backend import set_session
config = tf.compat.v1.ConfigProto(allow_soft_placement=False)
config.gpu_options.allow_growth = True  # dynamically grow the memory
used on the GPU
config.log_device_placement = True  # to log device placement (on which
device the operation ran)
sess = tf.compat.v1.Session(config=config)
set_session(sess)

import wandb
from wandb.keras import WandbCallback
wandb.init(config={"hyper": "parameter"})
```

Data Preprocessing - Training set

```
##Image Augmentation to prevent overfitting, when train set outperforms
test set
# create generator to standardize images

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range = 0.2,
                                   width_shift_range = 0.2,
                                   height_shift_range=0.2
                                  ) ##transformation, preventing
overfitting

##Import Dataset
##flow_from_directory connects dataset to Augmentation
train_set = train_datagen.flow_from_directory(
    ##Path
    'Dataset/train',
    ##Image Size to be fed in
    target_size=(250, 250),
    ##How many images in Batch
    batch_size=128,
    color_mode='rgb',
    shuffle=True,
    ##Binary or Categorical
    class_mode='categorical')
```

Preprocessing the Test set

```
##Test Data should not be formatted, just resized as this is how it
will work in real situation
test_datagen = ImageDataGenerator(rescale=1./255)

##Import Test Dataset
##flow_from_directory connects dataset to Augmentation
##Keep testing parameters the same size as training
test_set = test_datagen.flow_from_directory(
    ##Path
    'Dataset/val',
    ##Image Size to be fed in
    target_size=(250, 250),
    color_mode='rgb',
    shuffle=False,
    ##How many images in Batch
    batch_size=128,
    ##Binary or Categorical
    class_mode='categorical')

# prepare an iterators to scale images
print('Batches train=%d, test=%d' % (len(train_set), len(test_set)))

x_train,y_train = train_set.next()
x_train.shape
x_test,y_test = test_set.next()
x_test.shape
y_test.shape
```

```
print('Train', x_train.min(), x_train.max(), x_train.mean(),
x_train.std())
print('Test', x_test.min(), x_test.max(), x_test.mean(), x_test.std())
```

Image array example

```
def
show_grid(image_list,nrows,ncols,label_list,show_labels=True,savename=N
one,figsize=(20,10),showaxis='on'):
    if type(image_list) is not list:
        if(image_list.shape[-1]==1):
                                image_list = [image_list[i,:,:,0] for i in
range(image_list.shape[0])]
        elif(image_list.shape[-1]==3):
                                image_list = [image_list[i,:,:,:] for i in
range(image_list.shape[0])]
    fig = plt.figure(None, figsize,frameon=False)
    grid = ImageGrid(fig, 111,  # similar to subplot(111)
                      nrows_ncols=(nrows, ncols),  # creates 2x2 grid of
axes
                     axes_pad=0.3,  # pad between axes in inch.
                     share_all=True,
                     )
    for i in range(nrows*ncols):
        ax = grid[i]
         ax.imshow(image_list[i],cmap='Greys_r')  # The AxesGrid object
work as a list of axes.
        ax.axis('off')
        if show_labels:
            ax.set_title(label_list[[i]])
    if savename != None:
        plt.savefig(savename,bbox_inches='tight')

show_grid(x_train,2,8,label_list=y_train,figsize=(20,10),savename='../.
./Images/image_grid_ALLeven.png')
```

The ResNet Model

```
def resnet8(img_width, img_height, img_channels, output_dim):
    """
    # Arguments
       img_width:  image width.
       img_height: image height.
       img_channels: image channels.
       output_dim: Dimension of model output.
    # Returns
       model: A Model instance.
```

```
"""

# Input
img_input = Input(shape=(img_height, img_width, img_channels))

x1 = Conv2D(16, (7, 7), strides=[3,3], padding='same')(img_input)
x1 = MaxPooling2D(pool_size=(3, 3), strides=[1,1])(x1)

# 1st res block
x2 = BatchNormalization()(x1)
x2 = Activation('relu')(x2)
x2 = Conv2D(16, (3, 3), strides=[3,3], padding='same',
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(1e-4))(x2)

x2 = BatchNormalization()(x2)
x2 = Activation('relu')(x2)
x2 = Conv2D(16, (3, 3), padding='same',
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(1e-4))(x2)

x1 = Conv2D(16, (1, 1), strides=[3,3], padding='same')(x1)
#Skip connection
x3 = Add()([x1, x2])

# 2nd res block
x4 = BatchNormalization()(x3)
x4 = Activation('relu')(x4)
x4 = Conv2D(32, (3, 3), strides=[3,3], padding='same',
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(1e-4))(x4)

x4 = BatchNormalization()(x4)
x4 = Activation('relu')(x4)
x4 = Conv2D(32, (3, 3), padding='same',
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(1e-4))(x4)

x3 = Conv2D(32, (1, 1), strides=[3,3], padding='same')(x3)
#Skip connection
x5 = Add()([x3, x4])

# 3rd res block
x6 = BatchNormalization()(x5)
x6 = Activation('relu')(x6)
x6 = Conv2D(64, (3, 3), strides=[3,3], padding='same',
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(1e-4))(x6)

x6 = BatchNormalization()(x6)
x6 = Activation('relu')(x6)
x6 = Conv2D(64, (3, 3), padding='same',
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(1e-4))(x6)

x5 = Conv2D(64, (1, 1), strides=[3,3], padding='same')(x5)
#Skip connection
x7 = Add()([x5, x6])
```

```
    # 4th res block
    x8 = BatchNormalization()(x7)
    x8 = Activation('relu')(x8)
    x8 = Conv2D(128, (3, 3), strides=[3,3], padding='same',
                kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(1e-4))(x8)

    x8 = BatchNormalization()(x8)
    x8 = Activation('relu')(x8)
    x8 = Conv2D(128, (3, 3), padding='same',
                kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(1e-4))(x8)

    x7 = Conv2D(128, (1, 1), strides=[3,3], padding='same')(x7)
    #Skip connection
    x9 = Add()([x7, x8])

    x = Flatten()(x9)
    x = Activation('relu')(x)
    x = Dropout(0.5)(x)

    # Output probability
    steer = Dense(output_dim, activation='softmax')(x)

    # Define the model
    model = Model(inputs=[img_input], outputs=steer) #add call
    #print(model.summary())

    return model
```

```
Training the Model
resNet = resnet8(250,250,3,3)
##Connect CNN to optimiser and loss function
##Accuraccy metrics to measure CNN
resNet.compile(optimizer='adam',loss='categorical_crossentropy',metrics
=['accuracy'])
#tf.keras.utils.plot_model(cnn,                   to_file='model_1.png',
show_shapes=True, show_layer_names=True)
```

Training the Model on the Training set and Evaluating it on the Test set

```
#Early Stopping
#es = EarlyStopping(monitor='val_loss', mode='min', baseline=0.4)
#Save the best Model
mc = ModelCheckpoint('MODELS/Gun.h5', monitor='val_accuracy',
mode='max', verbose=1, save_best_only=True)

#Fit
ResNet_H = resNet.fit(x = train_set, validation_data = test_set,
epochs=30, callbacks=[mc, WandbCallback()], workers=10)
```

Save the best model

```
#cnn.save('../../MODELS/ResNet_V7.h5')
model = tf.keras.models.load_model("MODELS/Gun.h5")
```

Evaluate overall performance

```
test_loss, test_acc = model.evaluate(test_set, verbose=1)
print('\nTest accuaracy: ', test_acc)

#Evaluate the results of the data using the testdatset.

    #Using matplotlib
#       **Redundant as using Weights and Biases Library to Evaluate
Results while processing.

lb = pickle.loads(open('../../MODELS/lb.pickle', "rb").read())
print(ResNet_H.history.keys())

# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(x=x_test.astype("float32"), batch_size=1)
print(classification_report(y_test.argmax(axis=1),
    predictions.argmax(axis=1), target_names=lb.classes_))
# plot the training loss and accuracy
N = 30
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), ResNet_H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), ResNet_H.history["val_loss"],
label="val_loss")
plt.plot(np.arange(0, N), ResNet_H.history["accuracy"],
label="train_acc")
plt.plot(np.arange(0, N), ResNet_H.history["val_accuracy"],
label="val_acc")
plt.title("Training Loss and Accuracy on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper left")
plt.savefig('Images/Latest.png')
```

```
Test on Single Predictions
labels = {0 : 'centre',
          1 : 'left',
          2 : 'right'}

def predict_img(path) :
                                                      img              =
tf.keras.preprocessing.image.load_img(path,target_size=(250,250))
    img = tf.keras.preprocessing.image.img_to_array(img)
    img = img / 255.0
    img = np.array([img])
    print(model.predict(img))
    pred = labels[np.argmax(model.predict(img))]
    plt.imshow(img.reshape(250,250,1))
    plt.title(pred)


predict_img('Dataset/val/left/right frame_35displace_5630.jpg')
```

Video_Classification.ipynb

Purpose:

- Test a model that has been trained on the dataset against real world footage of new locations

Import Libraries

```
from tensorflow.keras.models import load_model
from sklearn.preprocessing import LabelBinarizer
from imutils import paths
from collections import deque
import numpy as np
import argparse
import pickle
import cv2
import os
# load the trained model and label binarizer from disk
print("[INFO] loading model and label binarizer...")
model = load_model('MODELS/Spear.h5')
lb = pickle.loads(open('MODELS/lb.pickle', "rb").read())

# initialize the image mean for mean subtraction along with the
# predictions queue
Q = deque(maxlen = 2)

# initialize the video stream, pointer to output video file, and
# frame dimensions
vs = cv2.VideoCapture('_data/_HOUSE/centre.mov')
writer = None
(W, H) = (None, None)

# loop over frames from the video file stream
while True:
    # read the next frame from the file
    (grabbed, frame) = vs.read()

    # if the frame was not grabbed, then we have reached the end
    # of the stream
    if not grabbed:
        break
    # if the frame dimensions are empty, grab them
    if W is None or H is None:
        (H, W) = frame.shape[:2]

    # clone the output frame, then convert it from BGR to RGB
    # ordering, resize the frame to a fixed 250x250, and then
    # perform mean subtraction
    output = frame.copy()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame = cv2.resize(frame, (250, 250)).astype("float32")
    # make predictions on the frame and then update the predictions
    # queue
    preds = model.predict(np.expand_dims(frame, axis=0))[0]
    Q.append(preds)

    # perform prediction averaging over the current history of
    # previous predictions
    results = np.array(Q).mean(axis=0)
    i = np.argmax(results)

    label = lb.classes_[i]
    #print(label)

    # draw the activity on the output frame
    text = "Operation: {}".format(label)
    cv2.putText(output, text, (35, 50), cv2.FONT_HERSHEY_SIMPLEX,
                1.25, (0, 255, 0), 5)

    # check if the video writer is None
    if writer is None:
        # initialize our video writer
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter('Video_With_Classification', fourcc,
60,
            (W, H), True)

    # write the output frame to disk
    writer.write(output)
    # show the output image
    cv2.imshow("Output", output)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# release the file pointers
print("[INFO] cleaning up...")
writer.release()
vs.release()
```

UAV Control Files

The following files are run wherever you have installed Ubuntu 18.04. The following instructions must be followed in order to gain Olympe controllers functionality.

Installation: Ubuntu 18.04 ONLT!

OLYMPE:

1. cd $HOME
2. mkdir code/parrot-groundsdk
3. repo init -u https://github.com/Parrot-Developers/groundsdk-manifest.git
4. repo sync

(If some libraries do not install when activating olympe env use pip install to get them)

CUDDN (for nvidia GPUs only):

Follow Ubuntu 18.04 instructions on

https://tensorflow.org/install/gpu

Tensorflow:

pip install tensorflow *Remeber to be in the Olympe enviroment *version >2.0 will be GPU capable

move_test_video.py

Purpose

Thisfile is used to test the automation process and the evaluation of the model when it is running on a video that is loaded. The predictions made upon the video will then control the drone.

```
"""
Import Libraries
"""
import olympe
from olympe.messages.ardrone3.Piloting import TakeOff, Landing
from olympe.messages.ardrone3.Piloting import moveBy
from olympe.messages.ardrone3.PilotingState import FlyingStateChanged
from olympe.messages.ardrone3.PilotingSettings import MaxTilt
from olympe.messages.ardrone3.GPSSettingsState import
GPSFixStateChanged

from flight_algorithms import *
from tensorflow.keras.models import load_model
from sklearn.preprocessing import LabelBinarizer
from imutils import paths
from collections import deque
import numpy as np
import pickle
import cv2
import os
import time
from time import sleep

"""
video_classification()
        Test a model on a video. Let the drone make actions based on
model predictions that is being run on a video.
        Load in a model, the labels and a video directed by paths.
        Use the Queue function to make a pool of predictions.
        Initialize a writer. This will be set to the shape of incomming
frames.
        Loop over each frame, make a copy of the original frame and
then resize it to fit into our model
        Make prediction on this frame and add it to the queue. Average
these predictions and get the largest probability.
        Pass this probability into the LabelBinarizer to sort it's
class. use this label for a screen overlay on frames going out.
        Use the label to control the drone via moveby functions.
        Make sure the writer is available, if not create it and show it
using OpenCV.
        Use 'q' to break the loop and land the drone
"""
def video_classification(drone):

    # load the trained model and label binarizer from disk
    #print("[INFO] loading model and label binarizer...")
    model = load_model("MODELS/Spear.h5")
    lb = pickle.loads(open("MODELS/lb.pickle", "rb").read())
    # initialize the predictions queue
    Q = deque(maxlen = 2)

    # initialize the video stream, pointer to output video file, and
    # frame dimensions
    vs = cv2.VideoCapture("VIDEO/skip-right-1.MOV")
    writer = None
    (W, H) = (None, None)

    # loop over frames from the video file stream
    while True:
        # read the next frame from the file
        (grabbed, frame) = vs.read()

        # if the frame was not grabbed, then we have reached the end
        # of the stream
        if not grabbed:
            break
        # if the frame dimensions are empty, grab them
        if W is None or H is None:
            (H, W) = frame.shape[:2]

        # clone the output frame, then convert it from BGR to RGB
        # ordering, resize the frame to a fixed 250x250
        output = frame.copy()
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame = cv2.resize(frame, (250, 250)).astype("float32")

        # make predictions on the frame and then update the predictions
        # queue
        preds = model.predict(np.expand_dims(frame, axis=0))[0]
        Q.append(preds)

        # perform prediction averaging over the current history of
        # previous predictions
        results = np.array(Q).mean(axis=0)
        i = np.argmax(results)

        label = lb.classes_[i]
        # draw the activity on the output frame
        text = "Operation: {}".format(label)
        cv2.putText(output, text, (35, 50), cv2.FONT_HERSHEY_SIMPLEX,
                    1.25, (0, 255, 0), 5)

        if label == "left":
            print("-----LEFT------")
            left()
        elif label == "right":
            print("-----RIGHT------")
            right()
        else:
            print("-----CENTRE------")
            centre()

        # check if the video writer is None
        if writer is None:
            # initialize our video writer
            fourcc = cv2.VideoWriter_fourcc(*"MJPG")
            writer = cv2.VideoWriter('Video_With_Classification',
fourcc, 30,
                (W, H), True)

        # write the output frame to disk
        writer.write(output)
        # show the output image
        cv2.imshow("Output", output)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break

    # release the file pointers
    print("[INFO] cleaning up...")
    writer.release()
    vs.release()

"""
left()
    Command the drone to rorate Left.
"""
def left():
    drone(moveBy(0, 0, 0, -1))

"""
right()
    Command the drone to rorate Right
"""
def right():
    drone(moveBy(0, 0, 0, 1))

"""
centre()
    Command the drone to stay centered. Add variable to dx to move
forward.
    For testing purposes this is set to zero.
"""
def centre():
    drone(moveBy(0, 0, 0, 0))

"""
main()
    Start the drone using connection() and TakeOff(). Next start
video_classification.
    A window will pop up of video playback that is labeled with
predictions.
    Stop the drone with a Landing() and disconnection.
"""
def main(drone):
    """
    Taking Off
    """
    print("-----Taking Off-----")
    drone.connection()
    drone(TakeOff() >> FlyingStateChanged(state="hovering",
_timeout=5)).wait()
```

```
    """
    Video Processing
    """
    print("-----Video Processing-----")
    video_classification(drone)

    """
    Landing
    """
    write_stop_output()
    print("*****Landing*****")
    drone(Landing() >> FlyingStateChanged(state="landed",
_timeout=5)).wait()

    #print("**** Total Flight Time: " + flight_time() + "*****")
    drone.disconnection()
"""
__main__
    Initialise the drones IP to either the Skycontroller or the drone
itself. Pass this as drone to the main function.
"""
if __name__ == "__main__":

    SkyCntrl_IP = "192.168.53.1"
    Anafi_IP = "192.168.42.1"

    Anafi_URL = "http://{}/".format(Anafi_IP)

    with olympe.Drone(SkyCntrl_IP) as drone:
        main(drone)
```

flight_algorithms.py

Purpose:

In this file functions are used to write and read trough a text file what instructions the drone is to carry out. This file is needed due to a breakdown of asynchronous features when streaming from the drone.

```
import olympe
from olympe.messages.ardrone3.Piloting import TakeOff, moveBy, Landing
from olympe.messages.ardrone3.PilotingState import FlyingStateChanged
import math
from time import sleep
import cv2
import random
import time
import os
from threading import Thread

os.environ["OPENCV_FFMPEG_CAPTURE_OPTIONS"] = "rtsp_transport;udp"

"""
write_navigation_output(pred)
    Parameters: pred - prediction string that comes from yuv_show_frame
    Writes the prediction output of the model to navigation.txt.
    Allows the controller to simultaneously predict frames and issue
the UAV commands.
"""
def write_navigation_output(pred):

    f_handle = open("navigation.txt", "w")
    f_handle.write(str(pred))
    f_handle.close()

"""
read_navigation_output()
    Read the prediction output of the model in navigation.txt.
    Allows the controller to simultaneously predict frames and issue
the UAV commands.
"""
def read_navigation_output():

    f_handle = open("navigation.txt", "r")
    state = f_handle.readline()
    f_handle.close()
    if state == "left":
        return "left"
    elif state == "right":
        return "right"
    else:
        return "centre"

"""
write_stop_output()
    Writes the to navigation.txt file: 'stop'.
    Allow the flight algorithm to know when it's time to stop
controlling the drone.
"""
```

```
def write_stop_output():

    stop = "stop"
    f_handle = open("navigation.txt", "w")
    f_handle.write(str(stop))
    f_handle.close()


"""
def flight_time():
    *
    Time begins when the first navigation operation begins
    :return: totalTime
    *

    start_time = time.time()
    while read_navigation_output() != "stop":
        continue
    totalTime = time.time() - start_time
    return totalTime
"""
```

finalAutonomy.py

```
"""
Import Libraries
"""
import csv
import cv2
import math
import os
import queue
import shlex
import subprocess
import tempfile
import threading
import traceback
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import LabelBinarizer
from imutils import paths
from collections import deque
import numpy as np
import pickle
from flight_algorithms import write_navigation_output,
write_stop_output, read_navigation_output

import olympe
from olympe.messages.ardrone3.Piloting import TakeOff, Landing
from olympe.messages.ardrone3.Piloting import moveBy
from olympe.messages.ardrone3.PilotingState import FlyingStateChanged
from olympe.messages.ardrone3.PilotingSettings import MaxTilt
from olympe.messages.ardrone3.GPSSettingsState import
GPSFixStateChanged


"""
Global Variables-
    DRONE_IP: connection ip for the drone. Using Sky controller IP for
range and control.
    model:    Call the ResNet model using Tensorflow library functions.
This is used for predictions
    lb:       Pickle file stores all label classes and is used to
associate the prediction with an understandable lable.
"""
olympe.log.update_config({"loggers": {"olympe": {"level": "WARNING"}}})

DRONE_IP = "192.168.53.1"

print("***** [INFO] loading model and label binarizer *****")
model = load_model("MODELS/Spear.h5")
lb = pickle.loads(open("MODELS/lb.pickle", "rb").read())

"""
Creating a class for a drone object creation provides threading that
makes asynchronous actions much easier and stops video feedback
stopping.
"""
```

```
class Navigation(threading.Thread):

    """
    Construct a drone object with a frame queue to implement threading
on video feedback
    """
    def __init__(self):
        # Create the olympe.Drone object from its IP address
        self.drone = olympe.Drone(DRONE_IP)
        self.tempd = tempfile.mkdtemp(prefix="olympe_streaming_test_")
        print("Olympe streaming example output dir:
{}".format(self.tempd))
        self.frame_queue = queue.Queue()
        self.flush_queue_lock = threading.Lock()
        super().__init__()
        super().start()

    """
    start()
        Initialises the drone object connecting it to Olympe
controller(laptop) via ip address. Callbacks for image processing are
initiated which
        provide video feedback with prediction labels.
    """
    def start(self):
        # Connect the the drone
        self.drone.connect()

        # Setup your callback functions to do some live video
processing
        self.drone.set_streaming_callbacks(
            raw_cb=self.yuv_frame_cb,
            start_cb=self.start_cb,
            end_cb=self.end_cb,
            flush_raw_cb=self.flush_cb,
        )
        # Start video streaming
        self.drone.start_video_streaming()

    """
    stop()
        Once the drone has finished call write to stop function from
flight_algorithms, next, The drone will initiate landing,
        then, stop the streaming output and finally disconnect
    """
    def stop(self):
        # Properly stop the video stream and disconnect
        write_stop_output()

        print("Landing...")
        self.drone(
            Landing()
            >> FlyingStateChanged(state="landed", _timeout=5)
        ).success()
        print("Landed\n")

        self.drone.stop_video_streaming()
        self.drone.disconnect()

    """
    yuv_frame_cb()
        parameter:
                yuv_frame:Take a raw yuv frame from the drone's video
stream and add it to the queue via reference.
        Add a yuv frame reference to the queue so that threading can
take place. This unables the drone to carry out asynchronous movements.
    """
    def yuv_frame_cb(self, yuv_frame):
        """
        This function will be called by Olympe for each decoded YUV
frame.

            :type yuv_frame: olympe.VideoFrame
        """
        yuv_frame.ref()
        self.frame_queue.put_nowait(yuv_frame)

    """
    flush_cb()
        Return drames from the queue while waiting or entering without
blocking past items on the queue
    """
    def flush_cb(self):
        with self.flush_queue_lock:
            while not self.frame_queue.empty():
                self.frame_queue.get_nowait().unref()
        return True

    def start_cb(self):
        pass

    def end_cb(self):
        pass


    """
    show_yuv_frame()
        parameters:
            window_name: Name of the window opened.
            yuv_frame: Yuv frame which will be outputted.

        The yuv frame is first converted using OpenCV.

        The new frame drom the queue is being passed to another queue Q
of frames.
        Here the frames are passed into the Tensorflow Navigation
model, This model predicts the probability
        the current frame should be labelled as a right turn left turn
or centre.

        These probabilities are grouped where a mean of the probability
is gotten for 10 frames. This probability
        is labelled using the LabelBinarizer from Sklearn. This label
will be left, right or centre.
```

```
        This Label is added to a text file Navigation.txt. From there
the drone can uses that file to make a movement.

        Next the label can also be added to the writer for OpenCV. This
will add text to the current frame when it is outputted.
        While the OpenCV window is open if the 'q' button is pressed
run stop() function to land drone and cut video streaming.
    """
    def show_yuv_frame(self, window_name, yuv_frame):
        info = yuv_frame.info()
        height, width = info["yuv"]["height"], info["yuv"]["width"]

        # convert pdraw YUV flag to OpenCV YUV flag
        cv2_cvt_color_flag = {
            olympe.PDRAW_YUV_FORMAT_I420: cv2.COLOR_YUV2BGR_I420,
            olympe.PDRAW_YUV_FORMAT_NV12: cv2.COLOR_YUV2BGR_NV12,
        }[info["yuv"]["format"]]

        # Use OpenCV to convert the yuv frame to RGB
        cv2frame = cv2.cvtColor(yuv_frame.as_ndarray(),
cv2_cvt_color_flag)
        Q = deque(maxlen = 10)

        writer = None
        (W, H) = (height, width)

        # if the frame dimensions are empty, grab them
        if W is None or H is None:
            (H, W) = cv2frame.shape[:2]

        #clone the output frame, then convert it from BGR to RGB
        # ordering, resize the frame to a fixed 250x250, and then
        output = cv2frame.copy()
        cv2frame = cv2.cvtColor(cv2frame, cv2.COLOR_BGR2RGB)
        cv2frame = cv2.resize(cv2frame, (250, 250)).astype("float32")

        # make predictions on the frame and then update the predictions
queue
        preds = model.predict(np.expand_dims(cv2frame, axis=0))[0]

        Q.append(preds)

        # perform prediction averaging over the current history of
        # previous predictions
        results = np.array(Q).mean(axis=0)
        i = np.argmax(results)

        # extract the label for the maxmimum label probability
        label = lb.classes_[i]

        # Add the label to Navigation.txt using flight_algorithm
function write_navigation_output
        write_navigation_output(label)

        # draw the activity on the output frame
        text = "Operation: {}".format(label)
        cv2.putText(output, text, (35, 50), cv2.FONT_HERSHEY_SIMPLEX,
                    1.25, (0, 255, 0), 5)
        # check if the video writer is None
        if writer is None:
            # initialize our video writer
            fourcc = cv2.VideoWriter_fourcc(*"MJPG")
            writer = cv2.VideoWriter('UAV CNN Navigation', fourcc, 30,
                (W, H), True)

        # write the output frame
        writer.write(output)

        # show the output image
        cv2.imshow("Output", output)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            self.stop()

    """
    run()
        create a window to proccess threading of the frames. Here
frames can be passed to show_yuv_frame and
        be outputed in another window. Each frame can then be dropped
from the queue and resources can be restored.
    """
    def run(self):
        window_name = "UAV CNN Navigation"
        cv2.namedWindow(window_name, cv2.WINDOW_NORMAL)
        main_thread = next(
            filter(lambda t: t.name == "MainThread",
threading.enumerate())
        )
        while main_thread.is_alive():
            with self.flush_queue_lock:
                try:
                    yuv_frame = self.frame_queue.get(timeout=0.01)
                except queue.Empty:
                    continue
                try:
                    self.show_yuv_frame(window_name, yuv_frame)
                except Exception:
                    # We have to continue popping frame from the queue
even if
                    # we fail to show one frame
                    traceback.print_exc()
                finally:
                    # Don't forget to unref the yuv frame. We don't
want to
                    # starve the video buffer pool
                    yuv_frame.unref()
        cv2.destroyWindow(window_name)

    """
    fly()
        Allow the drone to takeoff.
        If the drone is not labelled as stop, allow a loop to control
the drone. The read_navigation_output function will take a string
```

```
        to be cross referenced through conditional operations. Judging
the string, if it is 'left', the drone will turn left and wait for the
next instruction.
        If it is 'right' the drone will turn right and wait for the
next instruction to come through. If the string is 'centre' the drone
will move forward.
        (For testing purposes the centre function is set to move 0 as
to examine the feature without damges occuring)
    """
    def fly(self):
        # Takeoff, fly, land, ...
        """
        Uncomment the code snippet below, enabiling the user to use the
Sky Remote on the drone
        while the drone can still returns Video feed with Operation
predictions on Screen.
        """
        """
        # Takeoff, fly, land, ...
        print("Takeoff if necessary...")
        self.drone(
            FlyingStateChanged(state="hovering", _policy="check")
            | FlyingStateChanged(state="flying", _policy="check")
            | (
                GPSFixStateChanged(fixed=1, _timeout=10,
_policy="check_wait")
                >> (
                    TakeOff(_no_expect=True)
                    & FlyingStateChanged(
                        state="hovering", _timeout=10,
_policy="check_wait")
                )
            )
        ).wait()
        """
        # This landing condition is used as a redundancy incase of
stop() failing.
        if read_navigation_output == 'stop':
            print("Landing...")
            self.drone(
                Landing()
                >> FlyingStateChanged(state="landed", _timeout=5)
            ).success()
            print("Landed\n")

        print("***** Takeoff if necessary *****")
        self.drone(TakeOff(_no_expect=True) >>
FlyingStateChanged(state="hovering", _timeout=10,
_policy="check_wait")).wait().success()

        # If the UAV has not stopped
        while read_navigation_output() != 'stop':

            print("**** Navigation has started ****")

            if read_navigation_output() == 'left':
                self.drone(moveBy(0, 0, 0, -0.2)).wait().success()
                print('LEFT')
            if read_navigation_output() == 'right':
                self.drone(moveBy(0, 0, 0, 0.2)).wait().success()
                print('RIGHT')

            if read_navigation_output() == 'centre':
                self.drone(moveBy(0, 0, 0, 0)).success()
                print('CENTRE')
"""
__main__
    create the Navigation instance of a drone. Activate the streaming
protocols and start the fly processes.
    If the CLI is interupted use write_stop_output to cease drone
movement and threads will move onto the stopping function.
"""
if __name__ == "__main__":

    flightMind = Navigation()
    # Start the video stream
    flightMind.start()
    print("******START COMPLETE*****")
    # Perform some live video processing while the drone is flying
    try:
        flightMind.fly()
    except KeyboardInterrupt:
        write_stop_output()
        flightMind.stop()
```

Plagiarism Declaration

Declaration

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offense.

Student Name: Josh Hudziak

Student Number: C00231846

Signature: Josh Hudziak

Date: 30-04-21