Instituto Tecnológico de Estudios Superiores de Monterrey.

Tecnológico de Monterrey Campus Querétaro.

Visión para Robots

Dr. Rick Swenson.

Proyecto final
Control de un dron a través de reconocimiento de señas por medio de redes neuronales de convolución

Equipo

José Ramón Romero Chávez - A01700318
Jorge Adán Campos Amador - A01272415
Gloria Isabel García Mariño - A01204957

Miércoles 15 de mayo del 2019

Índice

Índice **2**

Introducción **3**

Objetivos **3**

Desarrollo **3**

Inicialización del programa 4

Recuperación y procesamiento de imágen 4

Movimiento del dron 5

Fin del programa 6

Resultados **6**

Conclusiones **6**

Anexos **7**

Anexo 1: Programa completo 7

Anexo 2: Red neuronal de convolución 11

Referencias **12**

Introducción

El procesamiento de imágenes en tiempo real es una de las muchas herramientas que han cobrado fuerza en los últimos años. Obtener información de imágenes y ejecutar acciones de manera inmediata es una de las muchas aplicaciones que tiene el procesamiento de imágenes en el día a día. Otra de las aplicaciones que puede tener es el control de objetos a través de señales. A lo largo de este proyecto se estará trabajando con recuperación y procesamiento de imágenes a través de redes neuronales para lograr el control de movimientos de un dron.

Objetivos

El objetivo general del proyecto es lograr controlar distintos movimientos de un dron a través de un lenguaje de señas. Este objetivo general se dividió en objetivos específicos. Estos fueron los siguientes:

- Recuperar imágenes de una cámara.
- Determinar la seña que se realizaba con la mano.
- Mandar las instrucciones de movimiento al dron.
- Integrar las partes anteriores y validar el comportamiento del dron.

Desarrollo

Antes de iniciar el desarrollo del programa de control que pudiera recuperar y procesar imágenes para, posteriormente, controlar al dron, se realizó un diagrama del programa y la estructura que debería de llevar.

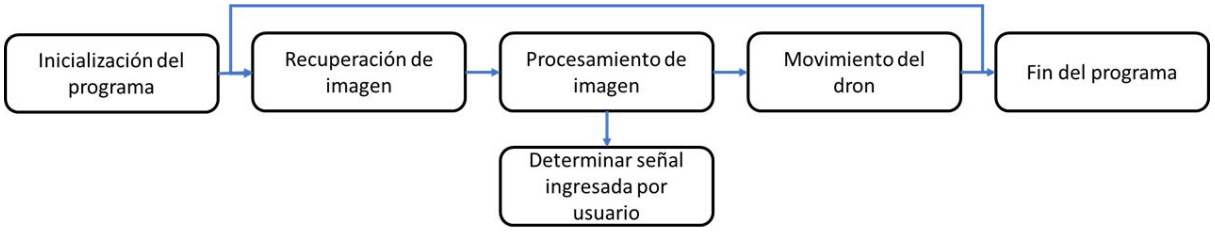

Fig. 1 Diagrama general del proyecto

Una vez que se tuvo la estructura, un plan de trabajo se desarrolló. En un inicio se trabajó en paralelo en las distintas etapas y, posteriormente, se realizó la integración de las mismas. A lo largo de esta sección se estará presentando una descripción de las etapas del programa y la integración de las mismas. El código utilizado puede encontrarse en los anexos al final de este reporte.

Inicialización del programa

En la inicialización del programa se importaron las herramientas y librerías necesarias para la ejecución del programa. Entre algunas de las herramientas y librerías importadas se encuentran Open CV, numpy, matplotlib y bebop, entre otras. Además de importar dichos elementos, en la inicialización del programa se llevaba a cabo la conexión entre la computadora y el dron que se estaría controlando. Para la parte de procesamiento de imagen se utilizó una red neuronal, de la cual se estará hablando más adelante. En la parte de la inicialización del programa se cargaba dicha red neuronal que permitía determinar la señal ingresada por el usuario, además de iniciar comunicación con la cámara. Una vez que la configuración inicial del programa se había llevado a cabo, se podía pasar a la siguiente etapa del programa.

Recuperación y procesamiento de imágen

Para la recuperación de la imagen se iniciaba una captura de video de la cámara de la computadora en la que se corría el programa. En la imagen recuperada se determinó un cuadrado, el cual sería considerada nuestra área de interés. Sería esta área de interés en la que el usuario haría la señal que determinaría el movimiento del dron. Esta área de interés se mostraría al usuario en una escala binaria, mientras que el resto de la imagen se mostraría en escala RGB.

El procesamiento del área de interés se daba en un punto intermedio entre su captura y el despliegue en la pantalla. Los pasos que se realizaban para el procesamiento de la imagen se muestran a continuación

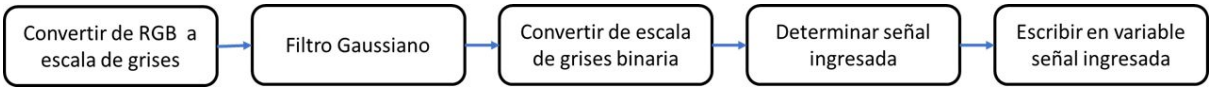

Fig. 2 Estructura de procesamiento de área de interés

Los primeros tres pasos se realizaron para acondicionar la imagen recuperada para realizar el reconocimiento de la señal. Inicialmente se realizó una conversión a escala de grises y se aplicó el filtro “Gaussian Blur” para determinar el contorno de la figura. Después de esto, se realizó una conversión a imagen binaria.

Una vez que la imagen había sido acondicionada, se determinó la señal ingresada por el usuario. Esto se logró a través de una red neuronal de convolución entrenada para identificar seis señales distintas, las cuales se muestran en la figura 3. Más detalles al respecto de la red neuronal utilizada se pueden encontrar en el Anexo 2.

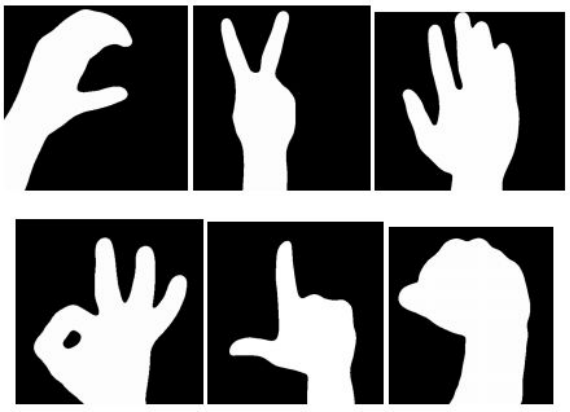

Fig 3. Señales Identificadas por la red neuronal de convolución

Con aproximadamente 3000 imágenes de cada una de las señales, la red neuronal aprendió a distinguir las señales correspondientes a “C”, “peace”, “palm”, “okay”, “L” y “fist”. Cabe mencionar que la red neuronal está entrenada para detectar dichas señas con distintos grados de rotación y traslación, por mencionar algunas de las modificaciones que puede tener la imagen.

Una vez identificada la señal ingresada por el usuario, se guardaba el nombre de la señal identificada en una variable. Esta variable después sería usada en el movimiento del dron. Cabe mencionar que el programa no procedía al paso del movimiento del dron hasta que el usuario presionara la tecla “s”. Una vez que esta tecla se presionaba, se detenía la captura de imágenes de la cámara y se procedía a mandara comandos al dron.

Movimiento del dron

Para lograr mover el dron se utilizaba el último valor guardado en la variable de que tenía el nombre de la última señal identificada. Cada señal correspondía a un movimiento, como se muestra en la tabla 1. Este movimiento era realizado hasta que se leía otra señal distinta.

Tabla 1. Señales con su movimiento correspondiente

Señal	**Movimiento**
C	Adelante
L	Atrás
fist	Aterrizar
okay	Despegar
palm	Mover hacia la derecha
peace	Rotar

Fin del programa

Se marcaba el final del programa cuando el usuario presionaba la tecla ”Esc”. En ese momento se interrumpe la conexión con el dron y con la cámara de la computadora. Posteriormente cerraban todas las ventanas generadas y se daba detenía el programa por completo.

Resultados

Después de la implementación, se obtuvieron los resultados esperados. La identificación de las señales ingresadas por el usuario se dio de manera exitosa y el dron reaccionaba de manera adecuada a los comandos que se le daban. Un video con el resumen de los resultados obtenidos se muestra en la siguiente liga:

https://youtu.be/kGY23pjs3hM

Conclusiones

Al final del proyecto se alcanzaron los objetivos que se plantearon en un inicio. La recuperación de imágenes por medio de la cámara web de la computadora y analizar las imágenes obtenidas para determinar la señal que el usuario estaba realizando. Esta parte se logró a través del uso de una red neuronal de convolución, entrenada para reconocer seis señales diferentes, lo que se logró de forma exitosa. Controlar el movimiento del dron a través de instrucciones que se le mandaban desde la computadora también se logró de manera exitosa.

El mayor reto que enfrentó el equipo fue al momento de integrar las partes anteriores y lograr que el dron respondiera de manera adecuada. Si bien de forma separada los programas se ejecutaban de forma adecuada, al momento de conjuntarlos no llegaban a trabajar de la forma en la que se esperaba. La conclusión a la que llegó el equipo fue que se intentaban correr al mismo tiempo y, por ende, ninguno de los dos programas corría de manera adecuada. La solución a este problema fue realizar una secuencia, en la que se recuperaba la imagen primero y, en cuanto el usuario lo decidiera, se detenía la cámara y se mandaban los comandos al dron. De esta forma se arregló el problema que se tenía. Distintos aspectos del proyecto pueden ser mejorados. Dos de los principales serían agregar un paro de emergencia y aumentar el dataset que se tenía para poder incluir más movimientos. Además de esto, una observación importante es poder monitorear el nivel de la batería mientras se está ejecutando el programa.

En este proyecto se trabajó con recuperación y procesamiento de imágenes, redes neuronales de convolución y drones. La integración de todas estas tecnologías, si bien no fue sencillo, llevó al equipo al resultado esperado y ayudó a reforzar cómo se pueden complementar e integrar distintas tecnologías para llegar a un resultado interesante y más “completo”.

Anexos

Anexo 1: Programa completo

```
import cv2
import numpy as np

import cv2
from matplotlib import pyplot as plt

from keras.models import load_model
import argparse
import pickle
import cv2

# construct the argument parser and parse the arguments
def predict2(filename, model, lb):

    img2 = np.zeros((filename.shape[0], filename.shape[0], 3))
    img2[:,:,0] = filename
    img2[:,:,1] = filename
    img2[:,:,2] = filename


    image = img2

    output = image.copy()
    image = cv2.resize(image, (64, 64))

    # scale the pixel values to [0, 1]
    image = image.astype("float") / 255.0

    # check to see if we should flatten the image and add a batch dimension

    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

    # load the model and label binarizer

    # make a prediction on the image
    preds = model.predict(image)

    # find the class label index with the largest corresponding
    # probability
    i = preds.argmax(axis=1)[0]
```

```
    label = lb.classes_[i]

    # draw the class label + probability on the output image
    print("{}: {:.2f}%".format(label, preds[0][i] * 100))
    return label


def sketch_transform(image, model, lb):
    image_grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # gray scale


    image_grayscale_blurred = cv2.GaussianBlur(image_grayscale, (7,7), 0) # gaussian blur
    _, mask = image_canny_inverted = cv2.threshold(image_grayscale, 50, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    img = mask
    cv2.bilateralFilter(img, 9, 90,16)
    bgModel = cv2.createBackgroundSubtractorMOG2(0, 50)

    learningRate = 0
    cap_region_x_begin = 0.5  # start point/total width
    cap_region_y_end = 0.8  # start point/total width
    threshold = 50  # binary threshold
    blurValue = 41  # GaussianBlur parameter
    bgSubThreshold = 40
    learningRate = 0

    # vari
    # variableslt
    isBgCaptured = 0  # bool, whether the background captured
    triggerSwitch = False  # if true, keyboard simulator works

    fgmask = bgModel.apply(img, learningRate=learningRate)
    kernel = np.ones((3, 3), np.uint8)
    fgmask = cv2.erode(fgmask, kernel, iterations=100)
    res = cv2.bitwise_and(img, img, mask=fgmask)


    ret, thresh = cv2.threshold(res, threshold, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    final = cv2.bitwise_not(thresh)


    label = predict2(final, model, lb)
```

```
    return final, label



from pyparrot.Bebop import Bebop

bebop = Bebop(drone_type="Bebop2")
print("CONNECTING...")
success = bebop.connect(10)
print(success)
if success:

    # No cambiar los valores de las siguientes 3 configuraciones !!!!!!
    bebop.set_max_altitude(2)          #Establece la altitud máxima en 3 metros
    bebop.set_max_tilt(10)             #Establece la velocidad máxima de movimientos
angulares en 5°
    bebop.set_max_vertical_speed(0.5)   #Establece la velocidad máxima vertical en 0.5 m/s


cam_capture = cv2.VideoCapture(0)
cv2.destroyAllWindows()
upper_left = (50, 50)
bottom_right = (300, 300)

print("[INFO] loading network and label binarizer...")

model = load_model("output/cnn.model")
lb = pickle.loads(open("output/cnn.pickle", "rb").read())
while True:
    explicit_label="XXX"
    while True:
        _, image_frame = cam_capture.read()

        #Rectangle marker
        r = cv2.rectangle(image_frame, upper_left, bottom_right, (100, 50, 200), 5)
        rect_img = image_frame[upper_left[1] : bottom_right[1], upper_left[0] : bottom_right[0]]

        sketcher_rect = rect_img
        sketcher_rect, label = sketch_transform(sketcher_rect, model, lb)

        #Conversion for 3 channels to put back on original image (streaming)
        sketcher_rect_rgb = cv2.cvtColor(sketcher_rect, cv2.COLOR_GRAY2RGB)

        #Replacing the sketched image on Region of Interest
```

```
        image_frame[upper_left[1] : bottom_right[1], upper_left[0] : bottom_right[0]] =
sketcher_rect_rgb
        cv2.imshow("Sketcher ROI", image_frame)
        print(label)


        if cv2.waitKey(1)& 0xFF == ord('s'):
            explicit_label=label
            break


    if explicit_label=="C":
        print('forward')
        bebop.fly_direct(0, 10, 0, 0, 0.1)
    if explicit_label=="L":
        print('backward')
        bebop.fly_direct(0, -10, 0, 0, 0.1)

    ####

    if explicit_label=="fist":
        print('aterrizar')
        bebop.safe_land(10)
        break
    if explicit_label=="okay":
        print('despegar')
        bebop.safe_takeoff(10)

    ####

    if explicit_label=="palm":
        print('right')
        bebop.fly_direct(10, 0, 0, 0, 0.1)
    if explicit_label=="peace":
        print('yo')
        bebop.fly_direct(0, 0, 10,0, 0.1)

bebop.disconnect()
cam_capture.release()
cv2.destroyAllWindows()
```

Anexo 2: Red neuronal de convolución

La identificación del gesto realizado por el usuario se lograba a partir de una red neuronal. En este caso, se optó por usar una red neuronal de convolución en lugar de una red neuronal ordinaria por su capacidad y facilidad para trabajar con imágenes. La red neuronal utilizada contaba con una capa de entrada, una de salida, y un conjunto de capas intermedias. Las funciones de cada parte de la estructura de la red neuronal se muestra en la siguiente tabla.

Tabla A2.1 Estructura de la red neuronal de convolución

Capa	**Función**
Entrada	Recibe la imágen que se procesará
Procesamiento	Las capas interconectadas que aplican la máscara que determinará el contorno de las imágenes son conocidas como las **capas densas**. En estas capas se utiliza el algoritmo ReLU, el cual lleva a una respuesta lineal de acuerdo a la entrada de la capa. Además de las 20 capas encargadas de realizar la tarea antes mencionada, también se cuenta con capas que reducen la cantidad de datos introducidos a la red conforme se trabaja en niveles más profundo de la misma, y con algoritmos de optimización, que hace que el 25% de las neuronas de una capa de forma aleatoria. Esta acción lo que evita es que la red sobre-aprenda el modelo.
Salida	Las capas de salida clasifican las probabilidades de que sea cada etiqueta correspondiente a la imágen y las ordena de mayor a menor valor. Esto para, al final, regresar la etiqueta que tenga la mayor probabilidad de encontrarse en la imagen.

La red neuronal que fue usada por el equipo fue implementada en el proyecto a través de un ambiente de LINUX en windows. También es necesario la instalación de complementos con la ayuda del comando *pip install* en el prompt command o con el programa en python que realiza esta tarea.

Para hacer uso de este proyecto es necesario tener una interfaz de LINUX en la computadora. Una vez que esto se logra, se debe de crear y activar un ambiente virtual en donde se ejecutará la red neuronal. Una vez que se ha logrado esto, se descargan los datos de la red neuronal y se instalan los complementos necesarios para correrla. Todos los recursos necesarios para el proceso antes mencionados pueden encontrarse en el link https://github.com/RamonRomeroQro/Convolutional_Droning/blob/master/README.md . Cabe mencionar que para poder ejecutar todo de manera correcta, es necesario contar con *Python3*.

Referencias

Romero, R. (2019). Machine Learning:Convolutional Neural Network for hand sign recognition. *ITESM*.