گزارش پروژه نهایی رباتیک

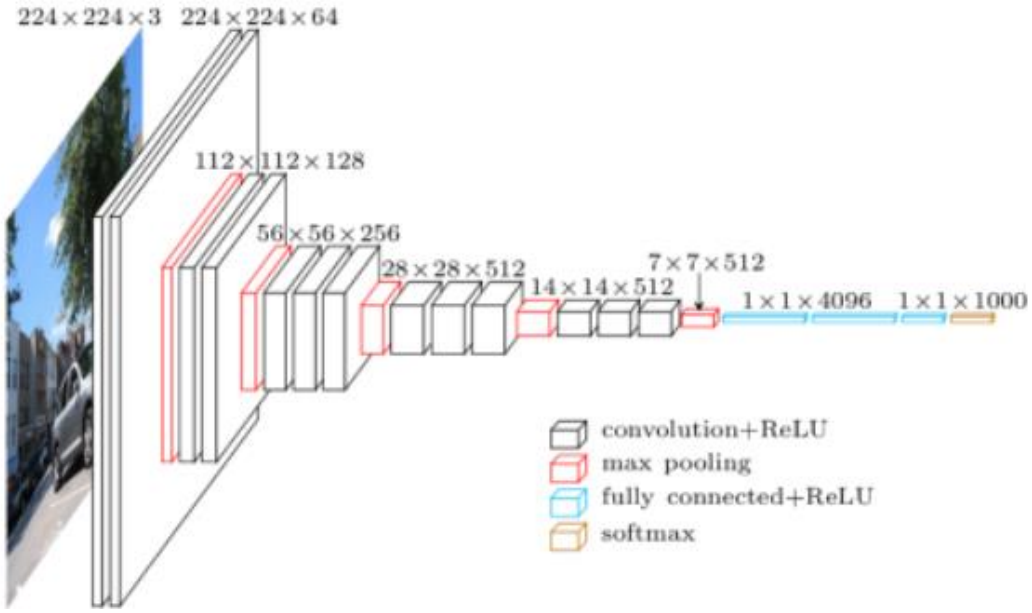
استاد درس: دکتر سلیمی

گروه صفر: ایلیا آخوندی، ریحانه سلجوقی، نرگس دهقان بنادکی

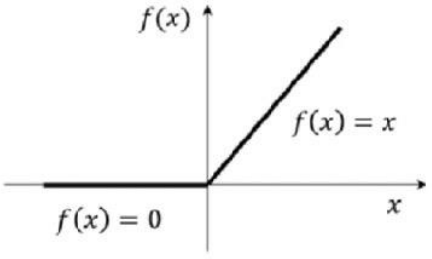
فهرست مطالب:

بخش اول، شبکه عصبی:

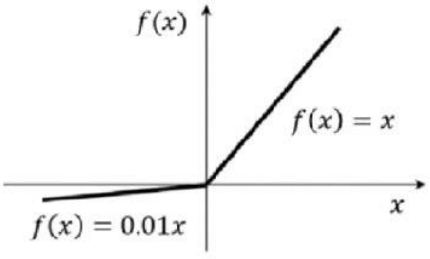
مدل ما برگرفته از VGG Net است که در آن کرنل ها کوچک تر شده اند (3×3) و تعداد فیلتر های لایه های کانوولوشنی توانی از 2 می باشد. و همچنین بعد از هر لایه کانولوشنی یک maxpooling داریم که برای کاهش ابعاد است×



تفاوتی که ما اعمال کردیم استفاده از leaky relu بجای relu بود چرا که از اورفیت جلوگیری میکند . فرق آنها هم در این است که در leaky relu بجای برگرداندن صفر برای مقادیر منفی، یک مقدار منفی کوچک بازگردانده میشود .همچنین تعداد لایه ها را نیز کمتر کردیم چرا که برای همچین مدل ساده ای باعث اورفیت میشد.



ReLU activation function



LeakyReLU activation function

ابتدا یک مدل sequential می سازیم که شامل دوتا لایه کانوولوشنی همراه با maxpooling ، یک لایه flattening و دوتا لایه dense است. لایه اخری که dense است از softmax استفاده میکند تا بتواند classification را انجام دهد.

ابتدا اولین لایه کانولوشنی که shape input آن عکس های 28*28 پیکسلی هستند، را تعریف میکنیم. به این صورت که Activation function آن ، leaky relu است. کرنل در این لایه3×3 است که 32 بار اعمال میشود.

سپس یک لایه مکس پولینگ اضافه میکنیم که سایز کرنل آن2×2 است. هدف از پولینگ کاهش ابعاد است .

یک مرحله دیگر این دولایه را تکرار میکنیم، با این تفاوت که به جای 32 ، 64 بار فیلتر اعمال می شود.

سپس لایه flatten باید استفاده شود تا بتوانیم دوباره پیکسل هارا به لایه fulliconnected بدهیم.

اولین لایه دنس 128 پیکسل ورودی میگیرد و سپس خروجی را به لایه دنس بعدی میدهد که لایه بعدی از softmax استفاده می کند تا classification را انجام دهد.

سپس با استفاده از بهینه ساز adam و تابع loss که cross entropy است ، کامپایل میکنیم.در آخر نیز در 10 تا epoch ، training انجام میشود. تعداد batch هم مشخص کننده این است که پس از چه تعداد sample، گرادیان آپدیت شود.

برای اینکه دائما مدل train نشود ، یک بار مدل را train میدهیم و fit. را روی آن صدا میکنیم سپس وزن هارا در فایل تایپ h5. سیو میکنیم سپس در کد کنترلر فقط weight ها را دوباره لود و مدل را کامپایل میکنیم.

```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(5, activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=64)



import h5py
import numpy as np
f = h5py.File("E:\\Courses\\Term 7\\Robotics\\FinalProject\\model_weights.weights.h5", "w")
f.close()
model.save_weights("E:\\Courses\\Term 7\\Robotics\\FinalProject\\model_weights.weights.h5")
```

حالا در فایل اصلی کنترلر صرفا کامپایل میکنیم و وزن ها را لود میکنیم سپس آن را predict میکنیم بدین صورت که ابتدا یک ارایه درست میکنیم و لیبل و عکس flat شده را به عنوان یک عضو از آن اضافه میکنیم، سپس این آرایه را در یک دیتافریم ذخیره و دیتا فریم را در فایل اکسلی ذخیره میکنیم. سپس از روی همان فایل تست دیتا را میخوانیم و x_test را استخراج میکنیم و reshape میکنیم سپس x_test را به مدل می دهیم تا یک forward pass زده شود و سپس لیبل مربوط به عکس را پیش بینی میکنیم.

```
43
44 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
45 model.load_weights("E:\\Courses\\Term 7\\Robotics\\FinalProject\\model_weights.weights.h5")
46
```

```
image_data_list.append([labels[self.target_index]] + flattened_pixels.tolist())

columns = ['label'] + [f'pixel_{i}' for i in range(len(image_data_list[0]) - 1)]
df = pd.DataFrame(image_data_list, columns=columns)
df.to_csv('E:\\Courses\\Term 7\\Robotics\\FinalProject\\test.csv', index=False)

test_data = pd.read_csv('E:\\Courses\\Term 7\\Robotics\\FinalProject\\test.csv')

X_test_real = test_data.iloc[:, 1:].values
X_test_real = X_test_real.reshape(X_test_real.shape[0], 28, 28, 1).astype('float32') / 255.0

predictions = model.predict(X_test_real)
predicted_labels = np.argmax(predictions, axis=1)

print(predicted_labels)
```

بخش دوم، کنترلر و کد اصلی:

توضیح متغییر ها:

ابتدا باید چند متغییر توضیح داده شود:

k vertical trust سرعت اولیه مورد نیاز برای همه پره هاست تا بتواند بلند بشود، k_vertical_offset . برای تعیین ثبات ربات در پرواز است. K_vertical_p ضریب کنترلر p برای ارتفاع است و K_pitch_p ضریب کنترلر در جهت y است. برای yaw به کنترلر نیاز نیست و جلوتر توضیح داده میشود K_roll_p هم ضریب کنترلر در جهت x است. Max_yaw_disturbance بیشترین مقدار ممکنی است که ربات می تواند در محور yaw بچرخد. متغییر بعدی Target precision است که حداقل مقدار خطای ممکن برای فاصله اقلیدسی از هدف است.

```
class Mavic (Robot):

    K_VERTICAL_THRUST = 68.5
    K_VERTICAL_OFFSET = 0.6
    K_VERTICAL_P = 3.0
    K_ROLL_P = 50.0
    K_PITCH_P = 30.0
    MAX_YAW_DISTURBANCE = 0.4
    MAX_PITCH_DISTURBANCE = -1
    target_precision = 0.35
```

در کانستراکتور ربات تمامی آرگومان های اولیه را ست میکنیم و موتور ها و دیوایس های لازم را روشن میکنیم

```
def __init__(self):
    Robot.__init__(self)

    self.time_step = int(self.getBasicTimeStep())


    self.camera = self.getDevice("camera")
    self.camera.enable(self.time_step)
    self.imu = self.getDevice("inertial unit")
    self.imu.enable(self.time_step)
    self.gps = self.getDevice("gps")
    self.gps.enable(self.time_step)
    self.gyro = self.getDevice("gyro")
    self.gyro.enable(self.time_step)
    self.compass = self.getDevice("compass")
    self.compass.enable(self.time_step)

    self.front_left_motor = self.getDevice("front left propeller")
    self.front_right_motor = self.getDevice("front right propeller")
    self.rear_left_motor = self.getDevice("rear left propeller")
    self.rear_right_motor = self.getDevice("rear right propeller")
    self.camera_pitch_motor = self.getDevice("camera pitch")
    self.r_led = self.getDevice("front right led")
    self.l_led = self.getDevice("front left led")
```

```
self.camera_pitch_motor.setPosition(1.55)
motors = [self.front_left_motor, self.front_right_motor,
          self.rear_left_motor, self.rear_right_motor]
for motor in motors:
    motor.setPosition(float('inf'))
    motor.setVelocity(1)
self.heading = 0
self.current_pose = 6 * [0]
self.target_position = [0, 0, 0]
self.target_index = 0
self.target_altitude = 0
```

توضیح توابع:

در کد از چند تابع مهم استفاده شده که در ادامه توضیح داده می شود:

تابع clamp :

این تابع اگر عددی در بازه مورد نظر بود، خودش را برمیگرداند . در صورتیکه از این بازه بیشتر بود، ماکسیمم را قرار میدهد و در صورتیکه که کمتر بود مینیمم.

تابع فاصله اقلیدسی هم تعریف شده است و توضیح خاصی ندارد.

```
def get_robot_heading(compass_value):
    return math.atan2(compass_value[0], compass_value[1])
def clamp(value, value_min, value_max):
    return min(max(value, value_min), value_max)
def calculate_euclidean_distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
```

تابع ectract_gray_roi:

ابتدا روی هر پیکسل ، میانگین مقادیر rgb را درمی آوریم و اگر از یک threshold ای کمتر باشد یعنی قسمت تیره عکس است که در اصل از طریق آن می توان باکس اصلی را شناسایی کرد.

حالا مختصات نقاطی که مشکی هستند را به دست می آوریم. باید مینیمم و ماکسیمم این مختصات را به دست بیاوریم تا بدانیم که محدوده جعبه کجاست، بعد مقدار ایمیج را در این بازه بدست می آوریم (انگار کراپ صورت گرفته است.)

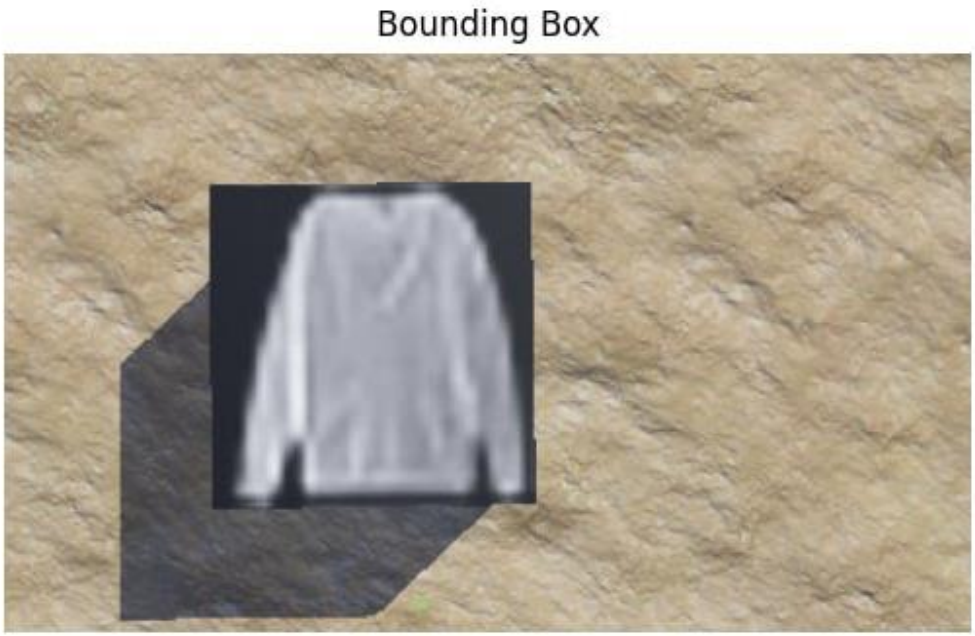
```
def extract_gray_roi(image, threshold=54):
    is_in_range = np.mean(image, axis=-1) < threshold
    non_color_pixels = np.where(is_in_range)

    return cv2.cvtColor(image[min(non_color_pixels[0]):max(non_color_pixels[0])
    , min(non_color_pixels[1]):max(non_color_pixels[1])], cv2.COLOR_BGR2GRAY)
```

حال outpot image رو resize میکنیم (28*28) و آن را برای اینکه در csv ذخیره کنیم flatten کرده و سپس برای نمایش دادن، نرمالایز میکنیم. سپس عکس را پردازش میکنیم که در بخش شکبه عصبی توضیح داده می شود.

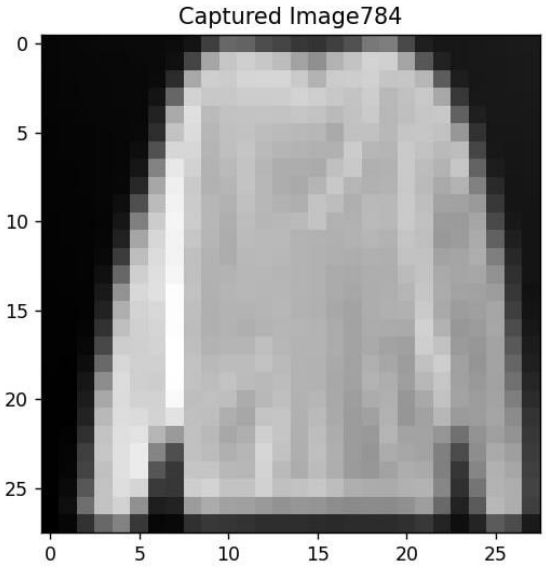
```
output_image = extract_gray_roi(image)
gray_image = np.array(output_image)

gray_image = resize(gray_image, (28, 28), preserve_range=True, anti_aliasing=True)

flattened_pixels = gray_image.flatten()
gray_image = gray_image / gray_image.max()
cv2.imwrite('C:\\Users\\USER\\Desktop\\output5.jpg', gray_image)

plt.imshow(gray_image, cmap="gray")
plt.title("Captured Image" + str(len(flattened_pixels)))
plt.show()
```

مثالی از مراحل دادن عکس به cnn :

عکس اولیه:



مرحله کراپ کردن ایمیج :



تابع move to target :

زمانی که به ارتفاع مد نظر برای شروع حرکت ربات به سمت یک هدف رسیدیم، این تابع صدا زده میشود و در شروع اگر تارگت پوزیشن ، 0 و 0 بود یعنی در مبدا هستیم پس 0 way points را به عنوان تارگت پوزیشن ست میکنیم. Way point آرایه ای است که در آن مختصات باکس ها وجود دارد.

برای رسیدن به هر کدام از باکس ها دو شرط مهم داریم:

1) اینکه فاصله اقلیدسی از حداکثر خطا کمتر باشد.

2) heading ربات هم به اندازه p 2 باشد تا عکس را صاف بگیرد.

اگر شرط برقرار بود، ارایه دیتای تست را آماده میکنیم که قبلا توضیح داده شده. سپس عکس را میگیریم.

```
def move_to_target(self, waypoints, labels):

    if self.target_position[0:2] == [0, 0]:
        self.target_position[0:2] = waypoints[0]

        print("First target: ", self.target_position[0:2])


    if(calculate_euclidean_distance(self.target_position[0], self.target_position[1], self.current_pose[0]
    ,  self.current_pose[1]) < self.target_precision
     and (3.13 < abs(self.heading) < 3.15)) :


        image_data_list = []

        image = self.camera.getImage()
```

اگر عکس با موفقت گرفته شد، ارایه ایمیج را میسازیم که آرایه ای 4 بعدی و متشکل از RGBA است. سپس ایمیج را از روی این آرایه میسازیم و به تابع extract gray roi میدهیم :

```
if image:

    width, height = self.camera.getWidth(), self.camera.getHeight()

    image_array = np.frombuffer(image, dtype=np.uint8).reshape((height, width, 4))
    image = Image.fromarray(image_array)
    image.save("E:\\Courses\\Term 7\\Robotics\\FinalProject\\image.png")
    image = cv2.imread("E:\Courses\Term 7\Robotics\FinalProject\\image.png")

    output_image = extract_gray_roi(image)
```

حال چک میکنیم که اگر لیبل یافته شده معادل همان لیبل هدف بود، باید LED ها را روشن کنیم و فرود بیاییم. (وقتی بالای باکس هستیم باید اول یکم به راست حرکت کنیم تا از مختصات جعبه یکم فاصله بگیریم و سپس ارتفاع ربات را کم میکنیم تا فرود بیایم.) بدین صورت:

چک میکنیم متغیر ایکس ربات در ناحیه جعبه نباشد و تا زمانی که بود این کار ها را انجام میدهیم. تمامی متغیر هارا از دیوایس ها دریافت میکنیم و rol disturbance را برابر منفی 0.1 می گذاریم تا به راست حرکت کنیم و تمام ورودی های ربات را ست میکنیم که بعدا توضیح داده خواهد شد.

برای فرود آمدن یک وایل دیگر داریم، ابتدا ارتفاع تارگت رو 0.01 می گذاریم (همان 0) و تا زمانی که ارتفاع به نزدیک تارگت نرسیده است ، تمامی متغیر های disturbance مربوطه را صفر میگذاریم تا ربات فقط در جهت عمودی حرکت کند.

```
self.target_index += 1
self.target_position[0:2] = waypoints[self.target_index % len(waypoints)]
print("Target reached! New target: ",
        self.target_position[0:2])
if self.target_index > len(waypoints) - 1:
    self.target_index = 0
if(predicted_labels[0] == label):

    self.r_led.set(1);
    self.l_led.set(1);
    while waypoints[self.target_index - 1][0] - 0.4 < self.current_pose[0] < waypoints[self.target_index - 1][0] + 0.4:
```

```
                roll, pitch, yaw = self.imu.getRollPitchYaw()
                x_pos, y_pos, altitude = self.gps.getValues()
                roll_acceleration, pitch_acceleration, _ = self.gyro.getValues()
                self.set_position([x_pos, y_pos, altitude, roll, pitch, yaw])
                cx, cy, _ = self.compass.getValues()
                self.heading = get_robot_heading((cx, cy))
                roll_disturbance = -0.1
                pitch_disturbance = 0
                yaw_disturbance = 0
                roll_input = self.K_ROLL_P * clamp(roll, -1, 1) + roll_acceleration + roll_disturbance
                pitch_input = self.K_PITCH_P * clamp(pitch, -1, 1) + pitch_acceleration + pitch_disturbance
                yaw_input = yaw_disturbance
                clamped_difference_altitude = clamp(self.target_altitude - altitude + self.K_VERTICAL_OFFSET, -1, 1)
                vertical_input = self.K_VERTICAL_P * pow(clamped_difference_altitude, 3.0)

                front_left_motor_input = self.K_VERTICAL_THRUST + vertical_input - yaw_input + pitch_input - roll_input
                front_right_motor_input = self.K_VERTICAL_THRUST + vertical_input + yaw_input + pitch_input + roll_input
                rear_left_motor_input = self.K_VERTICAL_THRUST + vertical_input + yaw_input - pitch_input - roll_input
                rear_right_motor_input = self.K_VERTICAL_THRUST + vertical_input - yaw_input - pitch_input + roll_input

                self.front_left_motor.setVelocity(front_left_motor_input)
                self.front_right_motor.setVelocity(-front_right_motor_input)
                self.rear_left_motor.setVelocity(-rear_left_motor_input)
                self.rear_right_motor.setVelocity(rear_right_motor_input)
                self.step(self.time_step)
            self.target_altitude = 0.01
            while altitude > 0.2:
```

```
    roll, pitch, yaw = self.imu.getRollPitchYaw()
    x_pos, y_pos, altitude = self.gps.getValues()
    roll_acceleration, pitch_acceleration, _ = self.gyro.getValues()
    self.set_position([x_pos, y_pos, altitude, roll, pitch, yaw])
    cx, cy, _ = self.compass.getValues()
    self.heading = get_robot_heading((cx, cy))
    roll_disturbance = 0
    pitch_disturbance = 0
    yaw_disturbance = 0
    roll_input = self.K_ROLL_P * clamp(roll, -1, 1) + roll_acceleration + roll_disturbance
    pitch_input = self.K_PITCH_P * clamp(pitch, -1, 1) + pitch_acceleration + pitch_disturbance
    yaw_input = yaw_disturbance
    clamped_difference_altitude = clamp(self.target_altitude - altitude + self.K_VERTICAL_OFFSET, -1, 1)
    vertical_input = self.K_VERTICAL_P * pow(clamped_difference_altitude, 3.0)

    front_left_motor_input = self.K_VERTICAL_THRUST + vertical_input - yaw_input + pitch_input - roll_input
    front_right_motor_input = self.K_VERTICAL_THRUST + vertical_input + yaw_input + pitch_input + roll_input
    rear_left_motor_input = self.K_VERTICAL_THRUST + vertical_input + yaw_input - pitch_input - roll_input
    rear_right_motor_input = self.K_VERTICAL_THRUST + vertical_input - yaw_input - pitch_input + roll_input

    self.front_left_motor.setVelocity(front_left_motor_input)
    self.front_right_motor.setVelocity(-front_right_motor_input)
    self.rear_left_motor.setVelocity(-rear_left_motor_input)
    self.rear_right_motor.setVelocity(rear_right_motor_input)
    self.step(self.time_step)

while self.step(self.time_step) != -1:
    self.front_left_motor.setVelocity(0)
    self.front_right_motor.setVelocity(0)
    self.rear_left_motor.setVelocity(0)
    self.rear_right_motor.setVelocity(0)
    pass
```

در این مرحله motion control توضیح داده خواهد شد :

ابتدا angle_left را در میاوریم که زاویه بین هدینگ ربات و مقصد است . سپس roll_disturbance را بدست می آوریم که در صورتی صفر نمیشود که از موقعیت مقصد فاصله داشته باشیم. به این دلیل که ربات همگام با جلو رفتن اگر مقصدش چپ بود به سمت جلو چپ حرکت کند. سپس yaw disturbance را بدست می اوریم که بر حسب angle_left نرمالایز شده بدست می آید .

توضیح روش بدست آوردن تغییرات pitch (حرکت رو به جلو و جهت y)هم بصورت زیر است:

pitch_disturbance = clamp(log10 |angle_left| , -1, 0.1)

angle_left

تحلیل:

۱- → منفی

خودش → کسری منفی

۰ → صفر

خودش یا ۱ → کسری +

واضح است که اگر تفاوت بین heading ربات و مقصد زیاد باشد ربات باید حرکت کند در جهت pitch و یا اصلا به سمت عقب حرکت می کند زیرا زاویه زیاد بنابراین است که نباید به جلو حرکت کنیم و وضعیت داشته باشد که yaw باشد کند.

اگر زاویه کم باشد پس او به جلو حرکت می کند.

```
self.target_position[2] = np.arctan2(
    self.target_position[1] - self.current_pose[1], self.target_position[0] - self.current_pose[0])

angle_left = self.target_position[2] - self.current_pose[5]

angle_left = (angle_left + 2 * np.pi) % (2 * np.pi)
if (angle_left > np.pi):
    angle_left -= 2 * np.pi


if abs(self.target_position[0]
 - self.current_pose[0]) > 0.2 and abs(self.target_position[1]
 - self.current_pose[1]) > 0.2 :
    if self.current_pose[0] < self.target_position[0]:
        roll_disturbance = -0.3
    else:
        roll_disturbance = 0.3

yaw_disturbance = self.MAX_YAW_DISTURBANCE * angle_left / (2 * np.pi)

pitch_disturbance = clamp(
    np.log10(abs(angle_left)), self.MAX_PITCH_DISTURBANCE, 0.1)


distance_left = np.sqrt(((self.target_position[0] - self.current_pose[0]) ** 2) + (
    (self.target_position[1] - self.current_pose[1]) ** 2))

return yaw_disturbance, pitch_disturbance, roll_disturbance
```

تابع run (اصلی):

ابتدا مختصات باکس ها را در آرایه waypoint و لیبل های باکس ها را هم در یک آرایه میریزیم. تارگت altitude را برای این ست میکنیم که ربات در شروع حرکت خود ابتدا به این ارتفاع برسد و بعد به سمت باکس ها حرکت کند. سپس لیبل مقصد را هم مینویسیم .

حالا در یک لوپ بینهایت roll و pitch و yaw را از طریق imu به دست می آوریم که زاویه ربات در هر ۳ جهت است سپس موقعیت کارتزین ربات را از طرق gps به دست می آوریم، سپس سرعت زاویه ای ربات در جهت roll و pitch را از ژیروسکوپ به دست می آوریم، بعد موقعیت ربات را در این ایتریشن ست میکنیم . هدینگ ربات را نیز از طریق قطب نما یا همان compass بدست می آوریم. چک میکنیم اگر ارتفاع ربات از حدی بیشتر شد، به سمت مقصد حرکت کند. خروجی این تابع ۳ متغیر است که جلوتر توضیح داده میشود.

```
def run(self):
    t1 = self.getTime()

    roll_disturbance = 0
    pitch_disturbance = 0
    yaw_disturbance = 0


    waypoints = [[-3, -2], [3, -3], [5, 0], [2, 5], [-5, 4]]
    labels =[2, 4, 3, 1, 0]

    self.target_altitude = 2.8
    label = 3
    while self.step(self.time_step) != -1:


        roll, pitch, yaw = self.imu.getRollPitchYaw()
        x_pos, y_pos, altitude = self.gps.getValues()
        roll_acceleration, pitch_acceleration, _ = self.gyro.getValues()
        self.set_position([x_pos, y_pos, altitude, roll, pitch, yaw])
        cx, cy, _ = self.compass.getValues()
        self.heading = get_robot_heading((cx, cy))
        if altitude > self.target_altitude - 1:

            if self.getTime() - t1 > 0.1:
                yaw_disturbance, pitch_disturbance, roll_disturbance = self.move_to_target(
                    waypoints, labels, label)
                t1 = self.getTime()
```

توضیح کنترلر p :

در اینجا سعی داریم که به ۴ موتور ربات دستور حرکتی بدهیم که این کار با منطق کنترلر P پیاده سازی شده است که هر دستور حرکتی بر اساس خطاهای در جهت roll و pitch و yaw و ارتفاع است. میخواهیم این ۴ خطا را کم کنیم، پس برای هر کدام به کنترلر P نیاز داریم.

تاثیری که roll میگذارد : ضریب کنترلر p × میزان خطا + سرعت فعلی ربات + roll_disturbance

یعنی خطای دستور ورودی roll به این چندتا چیز بستگی دارد.

برای pitch هم به همین نحو است.

Yaw_input فقط به yaw_disturbance بستگی دارد.

حالا تاثیر vertical_input به صورت زیر است که:

ابتدا یک اختلاف بین تارگت و موقعیت فعلی در می آوریم و ارتفاع به ثبات رسیدن ربات را هم با آن جمع میکنیم که Clamped_diffrence_altitude است. بعد ضریب کنترلر ارتفاع را ضرب در اختلاف قبلی به توان ۳ میکنیم.

حالا میخواهیم که سرعت موتور ها را ست کنیم:

سرعت همه موتور ها k_vertical_thrust رو دارن چون حداقل سرعت مورد نیاز برای بلند شدن است. یه vertical_input داریم که بعنوان مثبت تاثیر داده شده است که یعنی اگر مثبت باشد ، همه موتور ها سرعتشون بیشتر میشه و اگر کمتر باشه برعکس. و در نتیجه بالا یا پایین می رویم.

حالا تاثیر yaw_input بصورت ضربدری است. چون مثلا اگر بخواهیم به راست حرکت کنیم موتور جلو راستی ما را به راست میکشاند و باید موتور عقب چپی هم بصورت عقربه های ساعت پشت ربات را بچرخاند تا در نهایت هر دو کمک کنند به راست بپیچیم.

تاثیر pitch :

Front_left و front_right را با آن جمع میکنیم به این دلیل که ماهیت pitch_input یک مقدار منفی است. اگر مثبت بزاریم در واقع یک چیزی کم میشود . پس در فرانت مثبت میزاریم تا سرش رو بیاریم پایین تا بتونیم به جلو حرکت کنیم (pitch منفی باشه سر ربات به سمت پایین است و به جلو حرکت میکنیم) پس یعنی دو چرخ جلو سرعتشون کمتر میشه و دو چرخ عقب مقدارشون بیشتر میشه.

تاثیر roll :

roll هم مثل پیچ هست تنها فرقی که دارد این است که در جهت راست یا چپ است.

```
roll_input = self.K_ROLL_P * clamp(roll, -1, 1) + roll_acceleration + roll_disturbance
pitch_input = self.K_PITCH_P * clamp(pitch, -1, 1) + pitch_acceleration + pitch_disturbance
yaw_input = yaw_disturbance
clamped_difference_altitude = clamp(self.target_altitude - altitude + self.K_VERTICAL_OFFSET, -1, 1)
vertical_input = self.K_VERTICAL_P * pow(clamped_difference_altitude, 3.0)

front_left_motor_input = self.K_VERTICAL_THRUST + vertical_input - yaw_input + pitch_input - roll_input
front_right_motor_input = self.K_VERTICAL_THRUST + vertical_input + yaw_input + pitch_input + roll_input
rear_left_motor_input = self.K_VERTICAL_THRUST + vertical_input + yaw_input - pitch_input - roll_input
rear_right_motor_input = self.K_VERTICAL_THRUST + vertical_input - yaw_input - pitch_input + roll_input

self.front_left_motor.setVelocity(front_left_motor_input)
self.front_right_motor.setVelocity(-front_right_motor_input)
self.rear_left_motor.setVelocity(-rear_left_motor_input)
self.rear_right_motor.setVelocity(rear_right_motor_input)
```

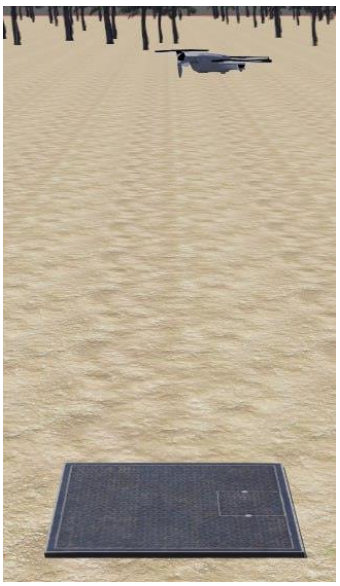
بخش سوم، چالش ها:

1) در ابتدا عکس هایی که گرفته میشد سایز بسیار بزرگی داشتند که عملا resize کردن آنها باعث کاهش کیفیت عکس میشد. برای هندل کردن این وضعیت کد بینایی کامپیوتری فراهم گردید که در آن ابتدا باکس مورد نظر پیدا میشود و سپس آن قسمت کراپ میشود و قسمت هایی از عکس که شامل زمین هستند، ریمو میشوند. کد آن قبلا به تفصیل توضیح داده شده است.
2) چالش دیگر که با آن مواجه شدیم ، کج بودن عکس ها بود ک شرایط را برای پیدا کردن باکس در عکس سخت میکرد. پس کاری که کردیم این بود که وقتی به مختصات مورد نظر رسید، ابتدا کاملا سر کوادکوپتر به سمت بالا قرار میگیرد تا بتواند عکس را صاف بگیرد .
3) چالش دیگر ست کردن مقادیر roll yaw pitch disturnace بود که باید مقادیر مختلفی را امتحان میکردیم تا به بهترین نتیجه برسیم.
4) چالش دیگر شبکه عصبی بود که دقت آن در ابتدا کم بود و بعضی مواقع اشتباه تشخیص میداد. با اضافه کردن تعداد فیلتر های کانوولوشنی و همچنین گذاشتن leaky relu دقت آن افزایش یافت.

بخش چهارم، خروجی ها و نتایج:

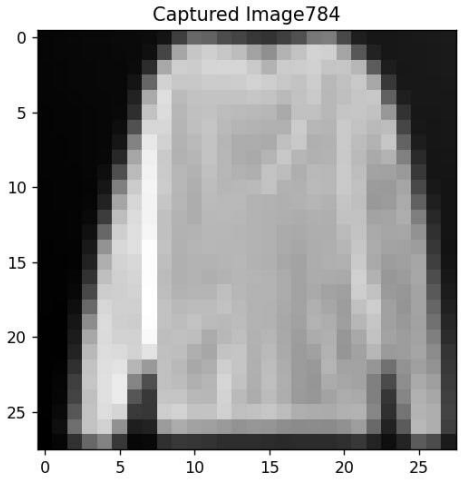
یک سناریو تصور می کنیم، فرض می کنیم به دنبال کیف هستیم پس لیبل ما ۴ است:

مرحله اول، بلند شدن از زمین و در ارتفاع خاصی قرار گرفتن:

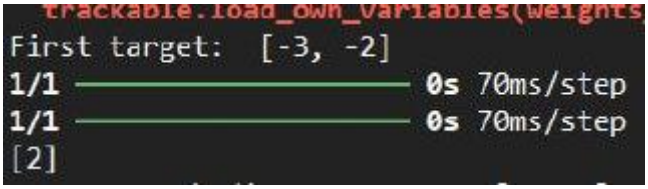


مرحله دوم :

هدف درست نیست و پلیور است.

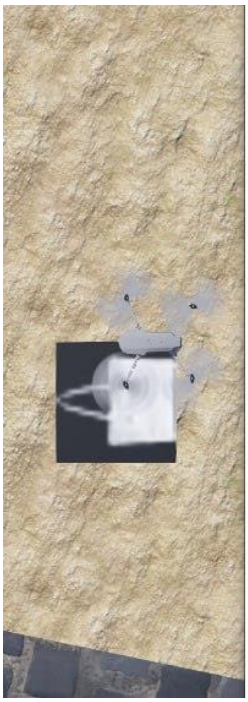


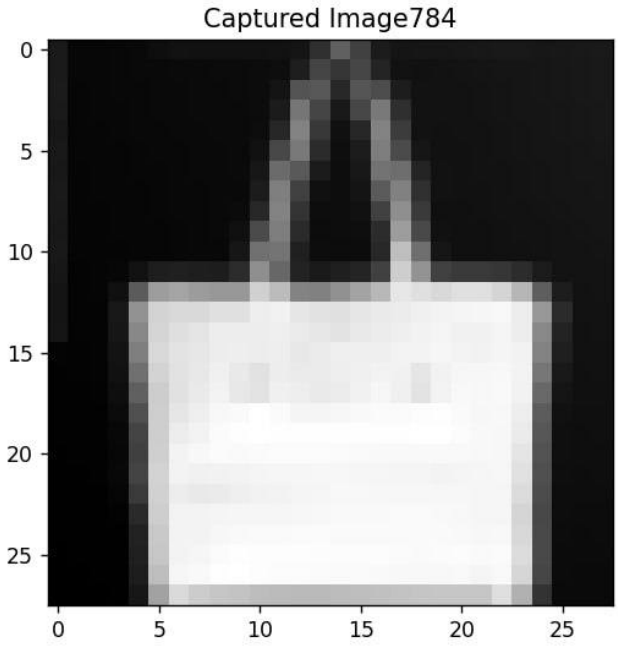
تشخیص شبکه عصبی هم لیبل دو بود که مربوط به پلیور است:



مرحله سوم :

به سراغ باکس های بعدی می رود و حالا به هدف رسید چون شبکه عصبی آن را تایید میکند و لیبلش ۴ است.





مرحله چهارم :

ربات LED هایش را روشن کرده است و در حال فرود آمدن است.

مرحله آخر :

نشستن ربات کنار هدف.

بخش پنجم، لینک آپارات:

https://aparat.com/v/dhAeG