گزارش پروژه درس رباتیک

اعضای گروه

سینا طاهری بهروز

یزدان جاهدی

حسین پیشگاهی

استاد

دکتر سلیمی

بهمن ۱۴۰۲

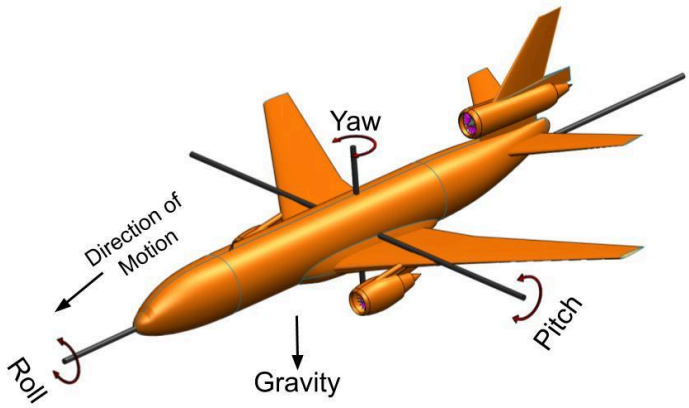
کنترلر ربات

برای کنترل کردن ربات از کنترلر PID استفاده شده است. برای ایجاد این کنترلر، ابتدا فایل پایه‌ای که برای کنترل کردن ربات های Mavic وجود دارد، بر روی ربات بارگذاری شد. لازم به ذکر است این کنترلر ضرایب مربوط به I و D را ندارد. در ادامه برای افزایش دقت آن، تاثیر خطای قبلی به واسطه ضریب D و همینطور تاثیر مجموع خطا های قبلی به واسطه ضریب I افزوده شد. برای اینکه از Wind-Up شدن انتگرال در 2 سمت منفی و مثبت جلوگیری شود، این مقدار می‌تواند عددی بین -0.1 و همینطور 0.1 داشته باشد.

```
roll_input = self.K_ROLL_P * roll_error \
             + self.K_ROLL_I * self.roll_integral \
             + self.K_ROLL_D * (roll_error - self.prev_roll) \
             + roll_acceleration + roll_disturbance
pitch_input = self.K_PITCH_P * pitch_error \
              + self.K_PITCH_I * self.pitch_integral \
              + self.K_PITCH_D * (pitch_error - self.prev_pitch) \
              + 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)
```

بخش محاسبه ورودی ها به واسطه ضرایب PID

به طور کلی روشی که کنترلر پیشفرض استفاده کرده است به این صورت است که ابتدا ارتفاع ربات، به حدی که توسط کاربر مشخص شده است می‌رسد (مقدار مورد نظر ما 3 بود) سپس ربات در مجموعه‌ای از مسیر ها که به واسطه کاربر مشخص شده است حرکت می‌کند. در هنگام حرکت جهت سر ربات به واسطه yaw به سمت هدف برده می‌شود و سپس در حین حرکت به واسطه مقادیر Roll و Pitch تضمین می‌شود که اولا ربات به جلو حرکت کند (دقت کنید به واسطه yaw مطمئن هستیم ربات به هدف نگاه می‌کند) و ثانیا ربات در هیچ یک از جهات به صورت ناگهانی نمی‌چرخد.



3 جهت چرخش برای ربات

برای اینکه عکس ها به طور مستقیم از موانع ثبت شوند، 5 نقطه ثانویه علاوه بر نقاطی که در صورت پروژه مشخص شده بود اضافه شده است. ربات قبل از رفتن به هر یک از نقاط اصلی، ابتدا به نقطه ثانویه مربوطه حرکت می‌کند سپس با زاویه ای مناسب به سمت نقطه اصلی حرکت میکند و هنگامی که به یک بازه مناسب رسید از مانع عکسبرداری می‌کند.

نمونه عکس ثبت شده توسط ربات

برای فرود آوردن ربات، بعد از اینکه در بخش شبکه عصبی تصویر با بیشترین نمره مشخص شد، ربات به سمت هدف حرکت می‌کند و در حین حرکت هر 3 ثانیه ارتفاع خود را به اندازه 0.2 واحد کم می‌کند. در نهایت وقتی ارتفاع ربات از 0.4 کمتر شد (این یعنی ربات تقریبا هم سطح زمین است) موتور ها خاموش می‌شوند.

پردازش تصاویر

در ابتدا، ربات در مکان های متفاوتی که به به صورت از پیش تعیین شده هستند، شروع به عکاسی میکند. از آنجایی که در این شبیه‌سازی، همه اجسام و محیط ثابت هستند، تنها تصویر برداری از اجسام در یک سری نقاط و لحظات از پیش تعیین شده میتواند که کافی و مفید باشد. این باعث حجم کمتر محاسباتی و عملیاتی بر روی ربات نیز خواهد شد. همانطور که گفته شد، ربات در زمان هایی که مکعب ها به طور کامل مشاهده می شوند، یک عکس نمونه خواهد گرفت. نمونه عکس های اولیه برای این ربات با استفاده از کنترل گفته شده به شکل زیر است:

حال نیاز است برای ورودی مرحله بعدی کار، یعنی شبکه عصبی، از این عکس گرفته شده، بخش مربوط به مربع سیاه رنگ را استخراج کنیم. به طور خلاصه در این مرحله از پروژه، با استفاده از تکنیک های پردازش تصویر و بینایی ماشین، باید قسمت مربع سیاه رنگ در هر عکس را استخراج کرده و در مرحله بعدی در CNN استفاده کنیم. حال، میدانیم که لحظات عکاسی شده در هر حالتی یکسان است. برای همین به صورت کلی، یک crop اولیه میتوان روی تصاویر داشت. به این گونه که با استفاده از تابع crop_image، کمی گوشه ها و اطراف همه عکس ها را برش میدهیم.

```
def crop_image(image, left=44, right=2, top=14, bottom=74):
    height, width, _ = image.shape
    cropped = image[top:height - bottom, left:width - right, :]
    return cropped
```

همانطور که مشخص است، یک سری مقادیر و پیش فرض برای برش اولیه تصاویر در نظر گرفته شده است. این مقادیر با تحلیل و مقایسه بین هر ۵ تصویر گرفته شده به دست آمده و یک برش اولیه برای کم حجم کردن عکس ها و از بین بردن داده های بلا استفاده است.

در مرحله بعدی می توان تفاوت های رنگ برای بهتر مشخص کردن محدوده مربع استفاده کرد. به این

شکل که زمین در تصاویر گرفته شده، به رنگ زرد است و میتوان از این مشخصه استفاده کرد. پیکسل هایی که رنگ مربوط به زرد دارند را در یک پردازش استاتیک، به رنگ سفید تبدیل میکنیم. سپس رنگ های خیلی سیاه را نیز سیاه تر میکنیم. حال بعد از این تغییرات، قسمت سایه در تصاویر کمرنگ تر شده و قسمت مربوط به مربع اصلی پررنگ تر و واضح تر خواهد بود. حال، با استفاده از contour در کتابخانه openCV میتوان محدوده کلی کانتور های را پیدا کرد و با استفاده از آن، محدوده ی مربع را پیدا کرد. (البته خیلی از اعداد و محاسبات استاتیک هستند. این به دلیل این است که محیط نیز یک محیط ایستا است و نیازی به محاسبات پیچیده و لحظه ای نداریم!)

خروجی بعد از پیدا کردن کلی شکل مانند زیر است:

```
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
image = cv2.threshold(gray, 70, 255, cv2.THRESH_OTSU)[1]
image = cv2.medianBlur(image, 11)
contours, h = cv2.findContours(image, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
```

کد بالا مربوط به قسمت کانتور هاست که با توجه به یک سری حد آستانه، کانتور ها مشخص شده است.

همانطور که مشخص است، محدوده مربع ها به خوبی تشخیص داده میشود اما همچنان خروجی مطلوب نیست. زیرا حاشیه عکس ها همچنان به رنگ زرد دیده میشود و همچنین رنگ پس زمینه دقیقا سیاه و مطلوب نیست. برای همین در ابتدای کار، یک بوردر با ضخامت ۱۰ پیکسل به شکل اضافه میکنیم. به اعمال این کار، گوشه و مرز های تصاویر کاملا سیاه خواهد شد و مشکلی ندارد.

```
def add_border(image, border_size=10):
    height, width = image.shape[:2]

    bordered_image = cv2.rectangle(
        image,
        (0, 0),
        (width - 1, height - 1),
        (0, 0, 0),
        border_size
    )

    return bordered_image
```

کد این قسمت هم مشخص است. یک بوردر با ضخامت ۱۰ داریم که در دور تا دور عکس اعمال میشود و باعث از بین رفتن حاشیه ها و قسمت های زرد رنگ خواهد شد. همچنین خروجی نهایی را واضح تر و کاراتر میکند.

بعد از این مرحله، حالا یک تابع برای threshold خواهیم داشت تا قسمت هایی که سیاه هستند (عدد کمتر از ۹۵) دارند را به ۰ و سیاه کامل تبدیل کند. و بعد از همه اینها باید اندازه تصویر به ۲۸ در ۲۸ تغییر پیدا کند. بعد از همه اینها چنین خروجی هایی از عکس های اولیه خواهیم داشت که کاملا مطابق با ورودی مرحله بعدی که همان CNN است خواهد بود.

```
def black_back(image, thresh=95):
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  gray[gray < thresh] = 0
  return gray
```

تابع بالا نیز برای سیاه کردن کامل بک گراند است. به این شکل که به دلیل تابش نور در محیط ممکن است که پس زمینه در عکاسی کاملا سیاه نباشد. برای همین ابتدائا عکس را از حالت RGB به grayscale تبدیل کرده و یک حدآستانه روی آن میگذاریم و پیکسل های کمتر از حد آستانه را سیاه خواهیم کرد. مرحله نهایی این قسمت هم کوچک کردن سایز نهایی عکس هاست که با استفاده از تابع resize در openCv انجام شد و تصاویر از سایز ۱۲۷ در ۱۲۷ به سایز ۲۸ در ۲۸ تغییر پیدا کرد تا ورودی های درست به شبکه عصبی داده شود.

مدل شبکه عصبی

برای پیدا کردن کلاس هدف از بین تصاویر گرفته شده، از یک مدل شبکه عصبی استفاده می‌کینم تا با classification داده‌ها جعبه هدف را مشخص کنیم. برای آموزش این مدل مراحل زیر طی شده است.

ابتدا دیتاست‌های train و test خوانده شده و x و y آن‌ها جدا می‌شود. در این جا x به معنای داده‌های خود عکس‌ها و y به معنای کلاسی است که عکس مربوط به آن است. این کار برای هر دو دیتاست انجام می‌شود.

```
train_data = np.array(train_df, dtype = 'float32')
test_data = np.array(test_df, dtype='float32')

x_train = train_data[:,1:]/255

y_train = train_data[:,0]

x_test= test_data[:,1:]/255

y_test=test_data[:,0]
```

سپس باید داده‌های train را به دو دسته دیگر تقسیم کنیم، یکی داده‌هایی که آموزش اصلی روی آن‌ها انجام می‌شود، و دیگری داده‌های validation که مربوط به بهینه‌سازی هایپرپارامترهای مدل هستند.

```
x_train,x_validate,y_train,y_validate = train_test_split(
    x_train,y_train,test_size = 0.2,random_state = 42)
```

در ادامه ابتدا تعدادی از عکس‌های ورودی را به همراه لیبل آن‌ها رسم کرده و از صحت داده‌ها مطمئن می‌شویم.

سپس داده‌های ورودی را به داده‌هایی تبدیل می‌کنیم که قابلیت استفاده بعنوان ورودی شبکه عصبی را داشته باشند.

```
image_rows = 28
image_cols = 28
image_shape = (image_rows,image_cols,1)
```

```
x_train = x_train.reshape(x_train.shape[0],*image_shape)
x_test = x_test.reshape(x_test.shape[0],*image_shape)
x_validate = x_validate.reshape(x_validate.shape[0],*image_shape)
```

اکنون باید لایه‌های شبکه عصبی مشخص شوند. در مدل ما لایه‌ها به صورت زیر هستند. ابتدا یک کانولوشن ۲ بعدی، با فیلتر مشخص شده در کد زیر، و با تابع فعال‌سازی RELU داریم. سپس از MaxPooling استفاده می‌کنیم و ۰.۲ داده‌ها را به صورت رندوم از شبکه خارج می‌کنیم. (این کار برای جلوگیری از overfit شدن داده‌ها انجام می‌شود.) بعد از اعمال این فیلترها، لایه‌های اصلی شبکه عصبی را مشخص می‌کنیم. دو لایه در نظر گرفته شده که تابع فعال‌سازی یکی RELU و دیگری Softmax در نظر گرفته شده.

```
cnn_model = Sequential([
    Conv2D(filters=32,kernel_size=3,activation='relu',input_shape = image_shape),
    MaxPooling2D(pool_size=2) ,
    Dropout(0.2),
    Flatten(),
    Dense(32,activation='relu'),
    Dense(5,activation = 'softmax')
])
```

سپس مدل با مشخصات زیر کامپایل می‌شود.

```
cnn_model.compile(
    loss ='sparse_categorical_crossentropy',
      optimizer=Adam(learning_rate=0.001),
      metrics =['accuracy'])
```

و در ادامه این مدل را با تعدادی epoch (برای بالا رفتن دقت مدل و همچنین جلوگیری از overfit شدن داده‌ها) و دیگر مشخصات که در کد قابل مشاهده هستند fit می‌کنیم.

```
history = cnn_model.fit(
    x_train,
    y_train,
    batch_size=4096,
    epochs=75,
    verbose=1,
    validation_data=(x_validate,y_validate),
)
```

در ادامه cnn_model که fit کردیم را به عنوان یک مدل h5 ذخیره می‌کنیم تا در بخش پردازش تصویر ربات از آن استفاده شود. دقت مدل آموزش دیده به صورت زیر است.

	precision	recall	f1-score	support
Class 0	0.95	0.97	0.96	1000
Class 1	0.99	0.99	0.99	1000
Class 2	0.97	0.95	0.96	1000
Class 3	0.99	1.00	0.99	1000
Class 4	0.98	0.98	0.98	1000
accuracy			0.98	5000
macro avg	0.98	0.98	0.98	5000
weighted avg	0.98	0.98	0.98	5000

لینک ویدیو