
دانشکده مهندسی کامپیوتر

آزمایشگاه سخت‌افزار

# گزارش پایانی پروژه

دکتر اجلالی

محمدرضا مفیضی - ۹۸۱۰۶۰۵۹
مهرشاد میرمحمدی - ۹۸۱۰۹۶۳۴
ارشان دلیلی - ۹۸۱۰۵۷۵۱

۱۲ تیر ۱۴۰۲

# فهرست مطالب

# ۱ چکیده

در این نوشته، سعی می‌کنیم توضیحاتی درباره نحوه پیاده‌سازی یک سیستم تشخیص حرکت دست ارائه دهیم.

سیستم تشخیص حرکت دست فناوری است که به کاربران اجازه می‌دهد با استفاده از حرکات دست به جای دستگاه‌های ورودی سنتی مانند صفحه کلید یا ماوس با دستگاه‌ها تعامل داشته باشند. در این پروژه، ما یک سیستم تشخیص ژست دست با استفاده از Raspberry Pi و یک ماژول دوربین توسعه خواهیم داد. این سیستم تصویری از ژست دست می‌گیرد، تصویر را پیش‌پردازش می‌کند و با استفاده از الگوریتم‌های یادگیری ماشین، ژست را تشخیص می‌دهد. این سیستم می‌تواند برای کاربردهای مختلفی مانند اتوماسیون خانگی، کنترل ربات‌ها و غیره استفاده شود. به طور خاص و با گسترش روز افزون واقعیت‌های مجازی و افزوده و مفهوم Spatial Computing، نیاز به دریافت فرمان و ورودی از روش‌های جدید مانند حرکت دست روز به روز بیش‌تر احساس می‌شود.

شکل ۱: واقعیت‌های مجازی و افزوده و مفهوم Spatial Computing

# ۲ مقدمه

برای پیاده‌سازی سیستم تشخیص حرکت دست به Raspberry Pi، ماژول دوربین، کارت Micro SD، منبع تغذیه و برد نیاز داریم. دوربین در کنار نمایش‌گر از حرکات دست فیلم‌برداری می‌کند و برد Rasp-berry Pi عملیات پیش‌پردازش که عمدتاً شامل تغییر رنگ و سایز عکس است و هم‌چنین پردازش اصلی تشخیص ژست که همان‌طور که در ادامه خواهیم دید، مبتنی بر روش‌های یادگیری ماشین است، را انجام می‌دهد. درنهایت نتایج پردازش با کابل به سیستم اصلی فرستاده می‌شود و تا دستورات کنترلی انجام شود.

برای پیش‌پردازش و پردازش تصویر از زبان برنامه‌نویسی پایتون و کتابخانه OpenCV استفاده می‌کنیم. بعد از انجام پیش‌پردازش و استخراج ویژگی‌ها از روی تصویر مدل یادگیری ماشین را روی آن‌ها آموزش می‌دهیم.

شکل ۲: شمای کلی از قطعه

## ۱.۲ مجموعه دادگان برای نقاط کلیدی دست

مجموعه دادگان بسیاری برای یافتن نقاط کلیدی دست، ارائه شده‌اند. در شکل ۳ تعداد از آنها را می‌توان مشاهده کرد.

| Dataset | Year | S/R | Mesh | Obj | #J | V | #S | #F | Paper |
|---|---|---|---|---|---|---|---|---|---|
| HIU-DMTL-Data | 2021 | R | ✗ | ✗ | 21 | 3rd/ego | 200 | 40,000 | ICCV 2021 [PDF] |
| InterHand2.6M | 2020 | R | ✗ | ✗ | 21 | 3rd | 27 | 2.6M | ECCV 2020 [PDF] |
| YouTube 3D Hands | 2020 | R | ✓ | ✓ | - | 3rd | - | 47,125/1525/1525 | CVPR 2020 [PDF] |
| OneHand10K | 2019 | R | - | ✗ | 21 | 3rd | 1 | 10k/1.3k | TCSVT 2019 [PDF] |
| FreiHAND | 2019 | R | - | ✓ | 21 | 3rd | - | 130k/3960 | ICCV 2019 [PDF] |
| GANerated Hands | 2018 | S | - | Both | 21 | ego | - | 330k | CVPR 2018 [PDF] |
| CMU Panoptic HandDB | 2017 | B | - | ✗ | 21 | 3rd | - | 14,817 | CVPR 2017 [PDF] |
| MHP | 2017 | R | - | ✗ | 21 | 3rd | 9 | 80k | IVC 2017 [PDF] |

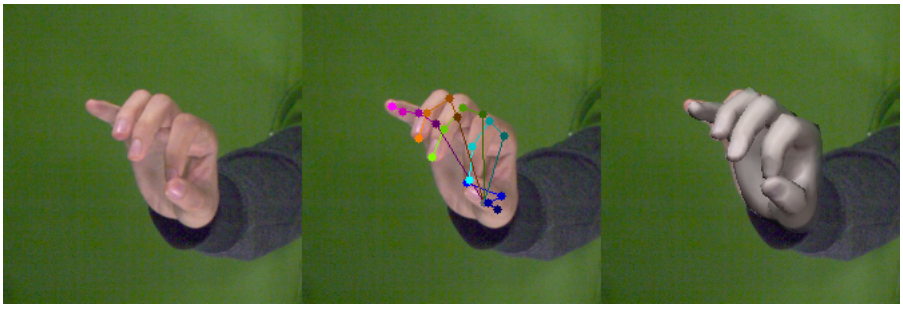
شکل ۳: دیتاست‌های موجود

از میان آنها ما دو مجموعه‌ی FreilHAND [۳] و OneHand10K [۲] را بررسی کردیم. محدود کننده‌ترین عامل در انتخاب‌امان، امکان دسترسی آزاد به آنها بود.

### ۱.۱.۲ مجموعه دادگان FreilHAND

- این مجموعه از هر دست در چهار جهت تصویر برداری کرده است.
- تصاویر دست‌ها در یک فضای آزمایشگاهی گرفته شده اند و in the wild نیستاند.
- تعداد تصاویر موجود در حدود ۱۳۰ هزار عدد است.

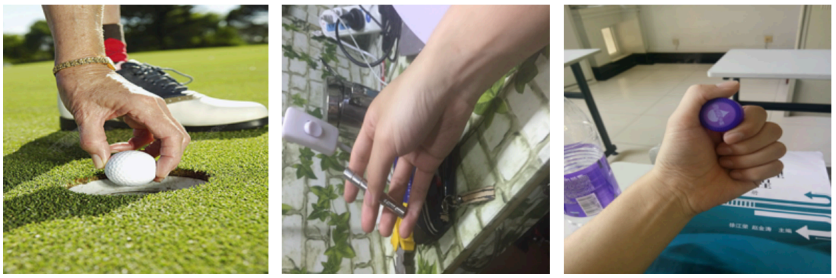
- در این میان حدود ۳۰ هزار تصویر پسپردازش شده و پیشزمینه‌ی آنها با یک تصویر از فضای in the whild عوض شده تا این تصاویر حالت فضای طبیعی‌تری داشته باشند.
- با این حال تصاویر پسپردازش شده از کیفیت خوبی برخوردار نیستاند و به راحتی مصنوعی بودن آنها قابل تشخیص است.
- همچنین اندازه و مکان دست بیش از حد ثابت است که این موضوع می‌تواند بر کیفیت خروجی شبکه‌ی ما بسیار تاثیرگذار باشد.



شکل ۴: نمونه داده‌های مجموعه دادگان FreilHAND

### ۲.۱.۲ مجموعه دادگان OneHand10K

- این مجموعه متشکل از تصاویر طبیعی (in the wild) است.
- تعداد تصاویر این مجموعه در حدود ۱۰ هزار عدد است.
- دست‌ها در این تصاویر از تنوع شکل و اندازه‌ی خیلی بیشتری برخوردار هستند.



شکل ۵: نمونه داده‌های مجموعه دادگان OneHand10K

در نهایت تصمیم بر استفاده از دیتاست OneHand10K گرفتیم چرا که تصاویر از کیفیت بهتر و تنوع بیشتری برخوردار بودند و تطابق بیش‌تری با دامنه هدف ما که تشخیص تشخیص نقاط کلیدی دست در محیط‌های طبیعی بود، داشتند.

### ۲.۲ مدل‌های یادگیری ماشین

انتخاب مدل مناسب برای دست‌یابی به دقت بالا و همچنین اجرای سریع بر روی سخت‌افزار موجود، از اهمیت بالایی برخوردار است.

از آن‌جایی که مدل‌های تشخیص bounding box روی دست پیاده‌سازی کارایی برای رزبری‌پای ندارند و اکثراً این سیستم‌ها سخت‌افزار لازم برای پردازش آن‌ها را ندارند، تصمیم گرفتیم تا به‌صورت مستقیم keypointهای دست را پیدا کنیم. همچنین برای بهینه کردن مدل، به‌جای پیدا کردن تمام keypointها فقط نوک انگشتان را پیدا می‌کنیم و با استفاده از آن‌ها ژست‌ها را تشخیص می‌دهیم.

## ۳ مدل‌های یافتن دست و تشخیص نقاط کلیدی آن

ما تصمیم گرفتیم تا برای سادگی بیشتر از خط‌لوله‌ی دو قسمتی که در قبل ارائه کرده بودیم، صرف نظر کرده و فقط یک مدل داشته باشیم که با گرفتن یک تصویر، مختصات سر پنج انگشت را تشخیص دهد. برای مدل ما از شبکه‌های عصبی پیچشی استفاده کردیم. این مدل با گرفتن یک تصویر، به ما تنسور ۴بعدی خروجی می‌دهد. این تنسور شکل $[7, 7, 5, 3]$ دارد. معنای این تنسور به این شکل است: اگر تصویر را به یک جدول ۷ در ۷ افراز کنیم، ما به ازای هر خانه از این جدول و به ازای هر سر انگشت، ۳ عدد پیدا می‌کنیم. این ۳ عدد بین ۰ تا ۱ هستند.

عدد اول نشان دهنده‌ی احتمال حضور آن سر انگشت در آن بخش از تصویر است. عدد دوم و سوم هم نشان دهنده‌ی جای دقیق انگشت در آن خانه را به ما می‌گویند، به این شکل که اگر فرض کنیم طول و عرض هر خانه ۱ واحد باشد، این دو عدد مختصات سر انگشت در این خانه خواهند بود. توجه شود عدد دوم و سوم فقط زمانی ارزشمند هستند که عدد اول به اندازه‌ی کافی بزرگ باشد، در غیر این صورت معنای خاصی ندارند.

### ۱.۳ در زمان اجرا (inference)

به ازای هر انگشت، خانه‌ای که احتمال پیشبینی شده‌ی بزرگتری دارد را پیدا می‌کنیم. اگر احتمال پیشبینی شده‌ی آن خانه از آستانه‌ای کمتر بود، در نظر می‌گیریم که آن انگشت در تصویر وجود نداشته، در غیر این صورت، با توجه به محل خود خانه و مختصات پیشبینی شده در خود خانه، محل دقیق سر انگشت را پیدا می‌کنیم.

### ۲.۳ در زمان آموزش (train)

هر خانه از میان ۴۹ خانه را یک مسئله‌ی کلاس‌بندی دو کلاسه‌ی جدا در نظر می‌گیریم. حال به ازای هر انگشت، خانه‌ای که در آن قرار دارد را پیدا می‌کنیم. سپس از مدل می‌خواهیم تا کلاس آن خانه را ۱ پیشبینی کند و کلاس بقیه خانه‌ها را ۰ پیشبینی کند (کلاس یک خانه همان احتمال حضور انگشت در آن خانه و همان عدد اول پیشبینی شده است که در بالاتر توضیح دادیم.) سپس بر روی مختصات پیشبینی شده در آن خانه هم لاس $\mathrm{L1}$ اعمال می‌کنیم. برای مختصات بقیه‌ی خانه‌ها لاسی اعمال نمی‌کنیم. اگر انگشتی در تصویر حضور نداشت کلاس همه‌ی خانه‌ها را ۰ در نظر می‌گیریم و برای مختصات‌ها لاسی اعمال نمی‌کنیم. جمع وزن‌دار این لاس $\mathrm{L1}$ و آن لاس ۴۹ مسئله‌ی کلاس‌بندی، لاس کلی را تشکیل می‌دهند. در زیر کد لاس را می‌توانید ببینید:

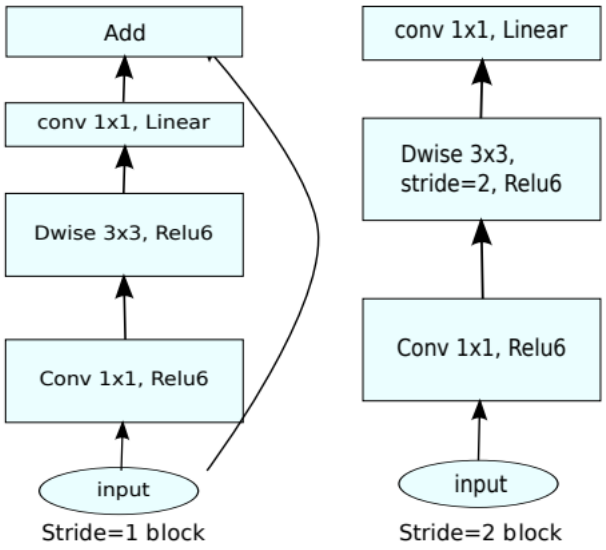
```
class CustomAccuracy(tf.keras.losses.Loss):
    def __init__(self):
        super().__init__()
        self.ce = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    def call(self, y_true, y_pred):
        alpha = 1.0
        num_fingers = 5
```

```
        p_true = y_true[:, :, :, :num_fingers]
        p_pred = y_pred[:, :, :, :num_fingers]
        p_loss = self.ce(p_true, p_pred)

        C = 7
        xy_true = tf.reshape(y_true[:, :, :, num_fingers:], (-1, C, C,
    2, num_fingers))
        xy_true = tf.transpose(xy_true, perm=(0, 1, 2, 4, 3))
        xy_pred = tf.reshape(y_pred[:, :, :, num_fingers:], (-1, C, C,
    2, num_fingers))
        xy_pred = tf.transpose(xy_pred, perm=(0, 1, 2, 4, 3))
        xy_pred = tf.math.sigmoid(xy_pred)

        l1_loss = tf.abs(xy_pred - xy_true)
        l1_loss = tf.reduce_sum(l1_loss, axis=-1)
        l1_loss = tf.where(tf.cast(p_true, bool), l1_loss, tf.
    zeros_like(l1_loss))
        l1_loss = tf.reduce_mean(l1_loss)

        return l1_loss + alpha * p_loss
```

## ۳.۳ معماری مدل

برای معماری مدل، ما از یادگیری انتقالی (Tranfer Learning) استفاده می‌کنیم. به صورت دقیق‌تر، ابتدا تصویر را به مدل از قبل آموزش دیده‌ی MobileNetV2 [۱] می‌دهیم و embedding با شکل $[7, 7, 1280]$ را به عنوان خروجی می‌گیریم. سپس با اعمال تعداد لایه‌ی پیچش دوبعدی بر روی آن، به شکل نهایی $[7, 7, 15]$ می‌رسیم. این پانزده عدد همان سه خروجی به ازای هر انگشت هستند.

```
IMG_SIZE = (224, 224)
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
include_top=False,
weights='imagenet')
base_model.trainable = False

inputs = tf.keras.Input(shape=(224, 224, 3))
x = base_model(inputs, training=False)
x = tf.keras.layers.Conv2D(32, 3, padding='same')(x)
x = tf.keras.activations.relu(x)
outputs = tf.keras.layers.Conv2D(15, 1)(x)
model = tf.keras.Model(inputs, outputs)
```




شکل ۶: معماری مدل سبک MobileNetV2

در نهایت تصمیم گرفتیم تا پیاده‌سازی‌ها را با استفاده از tensorflow انجام دهیم تا امکان تبدیل آن به tf-lite[1] راحت و با ریسک کمتری باشد.

### ۴.۳ آماده‌سازی مجموعه دادگان برای آموزش

نیاز است تا مجموعه‌ی دادگان به شکلی آماده‌سازی شود که tensoflow بتواند از آن استفاده کند. برای این کار ما ابتدا تمام تصاویر را پد کردیم تا مربعی شوند، سپس ابعاد آنها را به ۲۲۴ در ۲۲۴ تبدیل کردیم و آنها را در یک آرایه‌ی numpy ذخیره کردیم. سپس به ازای هر تصویر، مخصات سر انگشتان را بدست آوردیم و بر اساس آن، خانه‌ای که در آن قرار دارد و مختصات‌اش در آن خانه را محاسبه کرده و اینها را هم در آرایه‌ی numpy ذخیره کردیم. در نهایت با دادن این دو آرایه به مدل، آن را آموزش می‌دهیم.

## ۴ پیاده‌سازی

در ادامه به توضیحاتی درباره جزئیات کدهای پیاده شده می‌پردازیم:

### ۱.۴ اسکریپت `Manager`

این اسکریپت با استفاده از کتابخانه OpenCV تصاویر را از دوربین می‌گیرد و سپس با دادن آن‌ها به `Detector` و تشخیص محل انگشت‌ها، از `Handler` برای تشخیص ژست و فرستادن دستورات به کلاینت استفاده می‌کند.

[1] TensorFlow Lite

## ۲.۴ اسکریپت Detector

این اسکریپت وظیفه تشخیص keypointهای انگشت‌های دست از روی تصویر ورودی را دارد. با لود کردن مدل tflite و دادن ورودی می‌توان خروجی را از آن دریافت کرد. جزئیات نحوه تبدیل خروجی مدل به keypointهای انگشتان در بخش توضیحات مدل آمده است.

## ۳.۴ اسکریپت Handler

در این اسکریپت توابع لازم برای هندل کردن ارتباط رزبری با سیستم اصلی و هم‌چنین پردازش حرکات مشاهده شده و ارسال اطلاعات متناسب با آن به سیستم اصلی را تعریف می‌کنیم. برای بهینه بودن، فرض کردیم که ارتباط بین رزبری و سیستم اصلی از طریق شبکه محلی برقرار می‌شود و به همین دلیل از سوکت برای اتصال این دو دستگاه به یک‌دیگر استفاده می‌کنیم.

برای پردازش حرکت نیز به این صورت عمل می‌کنیم که مطابق مشاهده انجام شده از دست‌ها (این که کدام انگشت‌ها بالا هستند و اطلاعات دیگر) داده‌های لازم از قبیل نوع حرکت، جهت آن و بسته به نوع حرکت اطلاعات اضافی (از قبیل مختصات) را در قالب یک JSON برای سرور که همان دستگاه اصلی است، ارسال می‌کنیم.

همچنین در تابع `handle_gesture` که بخشی از آن در ادامه آمده است، با گرفتن وضعیت انگشت‌ها دستورات مورد نظر را به سرور ارسال می‌کند.

```
def handle_gesture(self, gesture, co1=(0, 0)):
  # move mouse
  if gesture == [0, 1, 0, 0, 0]:
      data = {'type': Command.MOVE, 'x': co1[0], 'y': co1[1]}
      self.client.send(json.dumps(data).encode('utf-8'))
      print("Move mouse case")
  # click
  elif gesture == [0, 1, 1, 0, 0]:
      ...
  # double click
  elif gesture == [1, 1, 0, 0, 0]:
      ...
```

## ۴.۴ اسکریپت Client

این اسکریپت با دریافت دستورات از رزبری‌پای آن‌ها را با استفاده از کتابخانه pyautogui بر روی سیستم اجرا می‌کند.

## ۵.۴ دستورات

برای انتقال دستورات حاصل از تشخیص ژست‌های دست به سیستم و اجرای آن‌ها از کتابخانه pyautogui استفاده می‌کنیم که می‌تواند تغییرات ماوس و کیبورد را به رایانه منتقل کند.

دستورات طراحی‌شده عبارت‌اند از:

- حرکت ماوس
- کلیک ماوس
- کلیک‌راست ماوس
- دابل‌کلیک ماوس
- گرفتن اسکرین‌شات

### ۶.۴ نصب کتابخانه‌های لازم روی رزبری‌پای

کتابخانه‌های لازم از جمله OpenCV و TF Lite را با استفاده از اسکریپت زیر بر روی رزبری‌پای نصب کردیم.

```
#!/bin/bash

# Get packages required for OpenCV
sudo apt-get install build-essential cmake pkg-config libjpeg-dev
    libtiff5-dev -y
sudo apt-get install libjasper-dev libpng-dev libavcodec-dev
    libavformat-dev libswscale-dev -y
sudo apt-get install libv4l-dev libxvidcore-dev libx264-dev
    libfontconfig1-dev libcairo2-dev -y
sudo apt-get install libgdk-pixbuf2.0-dev libpango1.0-dev libgtk2.0-dev
     libgtk-3-dev -y
sudo apt-get install libatlas-base-dev gfortran libhdf5-dev libhdf5-
    serial-dev libhdf5-103 python3-pyqt5 python3-dev -y
pip install --default-timeout=100 opencv-python

# Get packages required for TensorFlow
pip install https://github.com/google-coral/pycoral/releases/download/
    v2.0.0/tflite_runtime-2.5.0.post1-cp39-cp39-linux_aarch64.whl
```

## ۵ جمع‌بندی

در این پروژه، ما یک دستگاه تشخیص حرکت دست طراحی کردیم. در ابتدا با بررسی نیازمندی‌های موجود، سخت‌افزار لازم را برای طراحی چنین سیستمی به دست آوردیم و در فاز بعد با استفاده از تعدادی از مجموعه دادگان تشخیص keypointهای دست، مدل‌هایی را بر مبنای یادگیری ماشین، آموزش دادیم. در گام بعد، از مدل MobileNetV2 برای تشخیص نقاط کلیدی دست استفاده کردیم و با دادن خروجی آن به قسمت Handler آن را بررسی می‌کنیم و ژست دست را تشخیص می‌دهیم. پس از آن نوع حرکت و ویژگی‌های آن را در یک قالب JSON برای سرور (سیستم اصلی) ارسال می‌کنیم و در آن‌جا دستور را بررسی کرده و آن را بر روی سیستم اعمال می‌کنیم. با توجه به اهمیت این تسک، می‌توان کارهای زیادی در زمینه توسعه مدل‌های بهینه‌تر برای اجرا بر روی دستگاه‌های mobile و edge انجام داد.

## ۶ مراجع

[1] Mark Sandler et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2019. arXiv: 1801.04381 [cs.CV].

[2] Cong Peng Yangang Wang and Yebin Liu. “Mask-pose Cascaded CNN for 2D Hand Pose Estimation from Single Color Images”. In: IEEE Transactions on Circuits and Systems for Video Technology 29.11 (2019), pp. 3258–3268. DOI: 10.1109/TCSVT.2018.2879980. URL: http://yangangwang.com/papers/WANG-MCC-2018-10.html.

[3] Christian Zimmermann et al. FreiHAND: A Dataset for Markerless Capture of Hand Pose and Shape from Single RGB Images. 2019. arXiv: 1909.04349 [cs.CV].