آزمایشگاه
سخت افزار

گزارش پروژه

گروه ۷

محمدعلی خدابنده لو

۹۸۱۰۱۴۸۲

امیررضا میرزایی

۹۸۱۰۶۱۱۲

محمد ابول نژادیان

۹۸۱۰۳۸۶۷

بهار ۱۴۰۲

فهرست عناوین

مقدمه

در این پروژه، به ساخت خودرویی که با تشخیص حرکات دست حرکت می‌کند، می‌پردازیم. این پروژه در طی ۳ مرحله انجام شده است که مراحل و توضیحات کامل مربوط به هر مرحله را می‌توانید در بخش مراحل انجام پروژه، مشاهده کنید.

چالش اصلی این پروژه، تشخیص حرکات دست و ترجمه آن به حرکات ماشین بوده است.

مراحل انجام پروژه

تمرین مدل یادگیری‌ماشین

در طول مدت پروژه ۳ روش کلی در این بخش تست شدند که به صورت کوتاه آن‌ها را توضیح خواهیم داد.

1. استفاده از یک CNN برای تشخیص ژست و YOLO برای تشخیص دست

در ابتدا سعی کردیم YOLO را روی پیدا کردن خود دست آموزش بدهیم و سپس با استفاده از یک دسته‌ند با معماری CNN خروجی YOLO را تشخیص بدهیم. با توجه به داده‌ی کمی که در اختیار داشتیم دقت مدل CNN مناسب نبود. همچنین مشکلاتی مانند pad کردن خروجی YOLO نیز وجود داشت.

در این بخش استفاده از vision transformers ها هم تست شدند اما باز هم دقت خوبی بدست نیاوردیم. این موضوع احتمالا بدین خاطر بود که دیتاست ای که در این مرحله داشتیم تصاویر ورودی بسیار کوچکی داشت که با عکس واقعی نمی‌خواند.

2. استفاده از YOLO NAS

در این مرحله سعی کردیم خود فرایند تشخیص دست را نیز با YOLO انجام بدهیم با این روش از قدرت وزن‌های از پیش آموزش داده شده‌ی YOLO روی دسته‌بندی نیز استفاده می‌شود. در این بخش از دیتاست american sign language سایت roboflow استفاده کردیم.

به دلیل سرعت و دقت مدل YOLO NAS نسبت به دیگر ورژن‌های YOLO از آن استفاده شد.

در نهایت توانستیم دقت خوبی کسب کنیم اما متوجه شدیم YOLO NAS روی raspberry 3 قابل اجرا نیست.

3. استفاده از YOLOv8

در این بخش صرفا به جای YOLO NAS از YOLOv8 استفاده کردیم. دقت آن‌ها تقریبا برابر است اما زمانی که YOLOv8 برای محاسبه خروجی صرف می‌کند به طور مشهودی بیشتر است.

مدل YOLOv8 روی ۴ مدل مختلف آموزش داده شده است این مدل‌ها عبارتند از:

مدل YOLOv8

مدل YOLOv8 با ۴ وزن مختلف آموزش داده شده است.

1. YOLOv8n
2. YOLOv8s
3. YOLOv8m
4. YOLOv8l

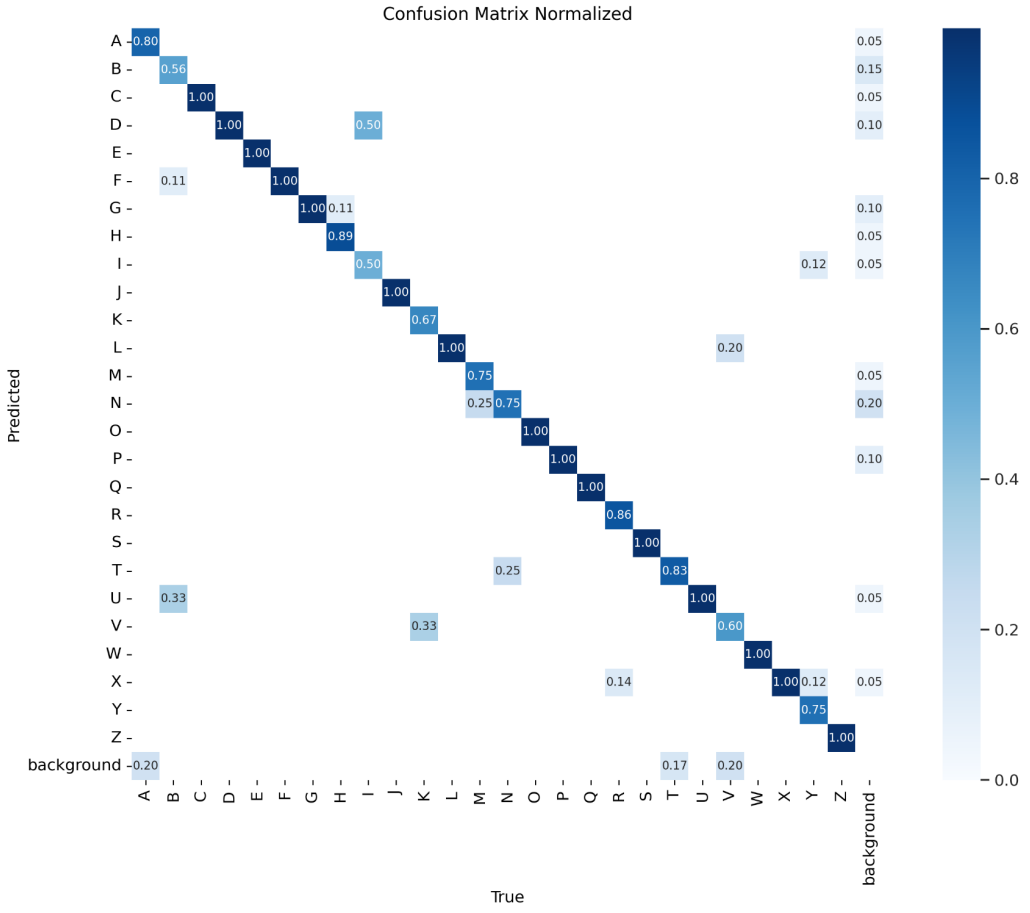
شبکه‌ی ۱ کم‌ترین سایز را از نظر وزن دارد و به ترتیب شبکه‌ی چهارم بیشترین وزن را دارد. دقت map0.5 برای تمامی ۴ مدل به مقدار بالای ۰.۹۵ می‌رسد اما در تست‌هایی که خودمان انجام دادیم دقت مدل سوم مناسب‌تر بود. همچنین لازم است که مقدار زمان پیش‌بینی هر مدل را بررسی کنیم. اطلاعات مربوط به دقت و زمان پیش‌بینی در جدول زیر موجودند:

	mAP50	**Inference time on raspberry**	**Inference time on laptop server**	**Weight size in mb**
YOLOv8n	0.95688	10.9 second	51.9 ms	6.2
YOLOv8s	0.96515	12.93 second	91.3 ms	22.5

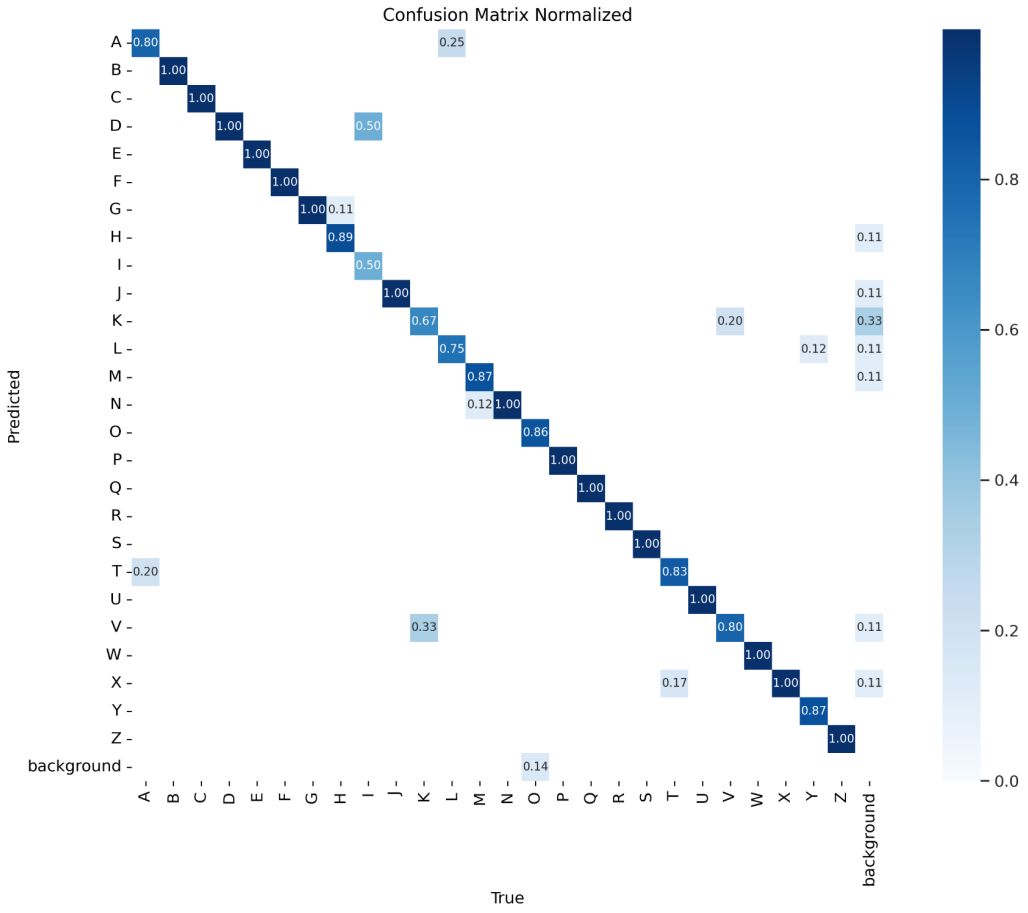
YOLOv8m	0.96973	-	183.2 ms	52
YOLOv8l	0.96216	-	318.9 ms	87.7

مدل سوم و چهارم روی raspberry اجرا نمی‌شوند.

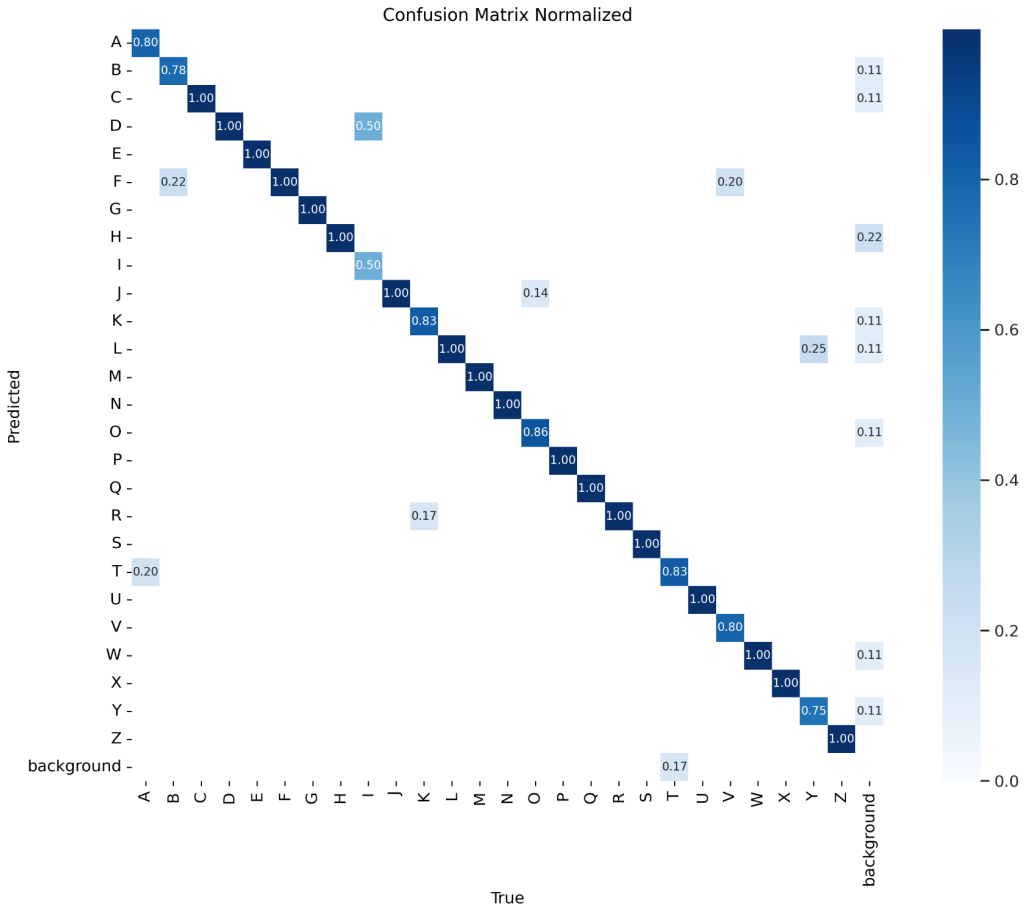
همچنین confusion matrix برای هر کدام از مدل‌ها به شرح زیر می‌باشند:



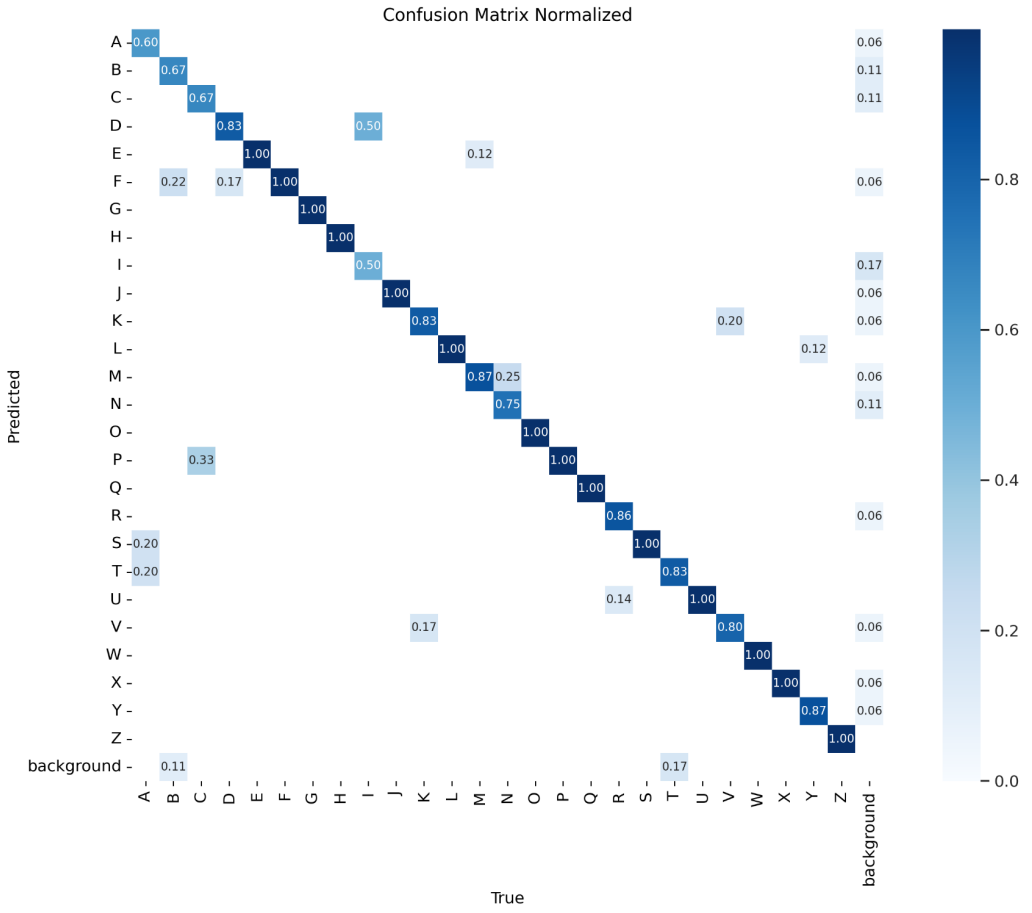
YOLOv8n


Confusion Matrix Normalized
Predicted
True
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
background
0.0
0.2
0.4
0.6
0.8

YOLOv8s


Confusion Matrix Normalized
Predicted
True
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
background
0.0
0.2
0.4
0.6
0.8

YOLOv8m



YOLOv8l

همانطور که مشاهده می‌شود، با بیشتر شدن وزن‌ها در مدل و سنگین‌تر شدن آن، دقت آن نیز افزایش پیدا می‌کند. فایل‌های مربوط به confusion matrix و دقت در epoch های مختلف در فولدر **training** **report** نیز موجود هستند.

تمام کدهای مربوط به تمرین این مدل‌ها در دو Jupyter Notebook زیر قابل دسترس هستند.

نوتبوک مربوط به تمرین مدل YOLO NAS

https://colab.research.google.com/drive/1fdf1H-GKpsIh23f16u0ocgEgsb-V7smq?usp=sharing

نوتبوک مربوط به تمرین مدل YOLOv8

https://colab.research.google.com/drive/1y4v-xkycE23vJqwX9GjuTMRNQkj8gCZT?usp=sharing

ترجمه به حرکات ماشین

حرکات دست تشخیص داد در طی دو مرحله به حرکت ماشین ترجمه می‌شوند؛ ابتدا با تابع map_letter_to movement، حرف تشخیص داده شده توسط مدل، به یکی از ۵ حرکت جلو، عقب، چپ، راست و توقف مپ می‌شود. در ادامه نیز در تابع map_car_movement_to_motor_movement، این حرکت ماشین به حرکت موتور ترجمه می‌شود؛ برای مثال حرکت سمت راست به این صورت به حرکت موتور ترجمه می‌شود که فقط موتور سمت چپ حرکت کند و در نتیجه به سمت راست حرکت انجام می‌شود.

پس در نتیجه می‌توان گفت از ۲۶ کلاس دیتاست فقط به ۵ کلاس نیاز داریم. اما ما برای افزایش دقت در عمل چند کلاس را با هم ترکیب می‌کنیم و آن‌ها را به یک حرکت موتور مطابق می‌کنیم. این عمل در جهت به منظور افزایش دقت بیشتر تشخیص حرکت جهت مدنظر کاربر است و الگوریتم اکتشافی است. برای مثال ژست‌هایی که دارای مشت هستند را به توقف موتور مطابق می‌کنیم. این تطابق در جدول زیر آمده است.

مشخصه اصلی تمامی حروف	ژست دست در american sign language	حرکت موتور
بسته بودن دست	a,e,m,n,o,s,t	stop
نشان دادن سمت راست با انگشت	g,h	right
نشان دادن حرف L	l	left
نشان دادن انگشت‌ها به صورت علامت v	u,v,w,k	backward
باز بودن و نشان دادن کف دست	b,c	forward

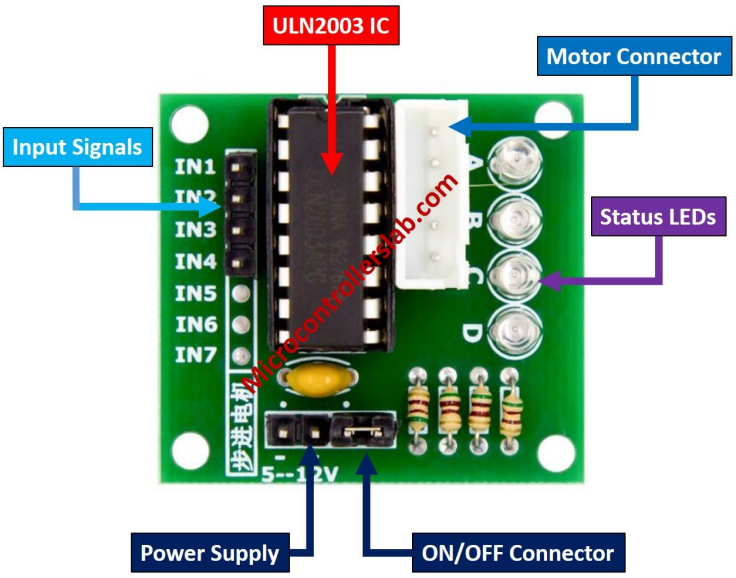
این که چندین حرف همزمان به یک حرکت مپ شده‌اند، دلیلش این است که این حروف gesture نزدیک به هم در دیتاست داشته‌اند. برای هر کدام از نیز یک مشخصه اصلی که در اصل همان دلیل مپ کردن همه حروف به یک حرکت بوده است نشان داده شده است. می‌توانید از روی تصویر زیر، دید بهتری نسبت به دلیل مپ کردن این حرکات مشابه پیدا کنید.

سرهم بندی قطعات ماشین

اولین قدم برای انجام بخش سخت‌افزاری پروژه، اتصال stepper motor ها به raspberry pi و کنترل کردن آنها بود. نکته‌ی قابل توجه این است که اتصال موتورها به raspberry pi، به خودی خود امکان پذیر نیست و نیاز به استفاده از ماژول درایور وجود دارد. با توجه به اینکه نوع stepper motor که در اختیار ما بود، از مدل 28BYJ-48 بود، بعد از انجام بررسی‌های لازم، مشخص شد درایور مناسب برای این موتور، از نوع ULN2003 است. با توجه به اینکه این درایورها در دانشکده موجود نبودند، اقدام به خرید آنها صورت گرفت.

در ادامه می‌توانید تصویری از این نوع درایور مشاهده کنید. این درایور به صورت کلی دارای پنج بخش مختلف است:

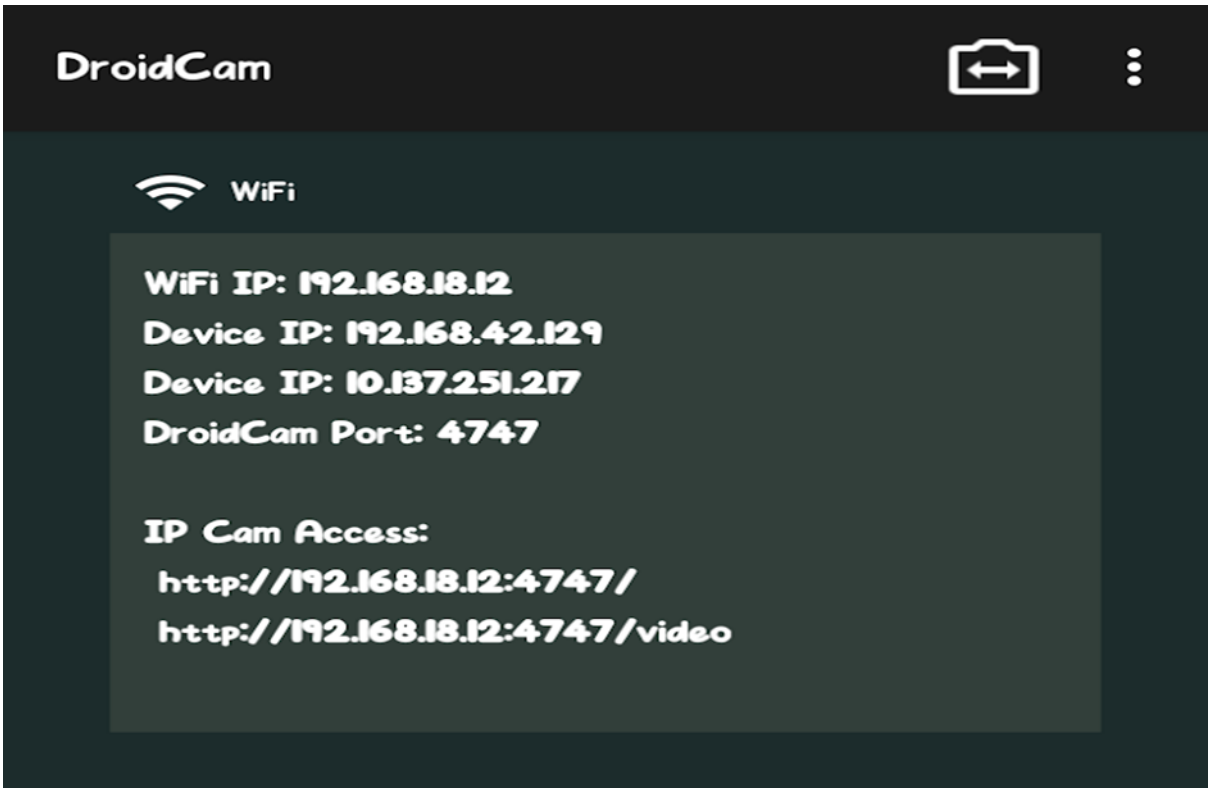
1. IC: بخش اصلی این درایور که در واقع مدار کنترل‌کننده آن به حساب می‌آید.
2. Input Signals: سیگنال‌هایی که از raspberry pi گرفته می‌شوند، به عنوان ورودی به این قسمت داده می‌شوند.
3. Motor Connector: سیگنال‌های کنترلی مربوط به موتور که توسط IC تولید می‌شوند. این قسمت در ادامه توسط تعدادی سیم به صورت مستقیم به موتور متصل می‌شود.
4. Status LED: این LEDها نشان‌دهنده وضعیت درایور هستند که با توجه به مقدار سیگنال‌های ورودی خاموش و روش می‌شوند.
5. Power Supply - ON/OFF Connector: همانطور که مشخص است، تغذیه‌ی درایور از این قسمت انجام می‌گیرد.



قدم بعدی، برنامه‌ریز ها بود. کد این قسمت را می‌توانید در ریپوی GitHub ارائه شده مشاهده کنید.

استفاده از دوربین

یکی دیگر از منابعی که برای انجام پروژه به آن نیاز داشتیم، یک ماژول با قابلیت ثبت تصویر، برای تشخیص حرکات دست بود. برای فراهم کردن امکان تصویر برداری، از دوربین یک گوشی اندرویدی استفاده شد. برای اتصال به دوربین گوشی، از اپلیکیشن DroidCam استفاده شد. شیوه‌ی کار این اپلیکیشن به این صورت است که باید raspberry pi و گوشی اندرویدی، هر دو به یک شبکه متصل باشند و در این صورت، این امکان وجود دارد که با استفاده از آدرس IP مربوط به گوشی و اتصال به Portی که توسط اپلیکیشن DroidCam باز شده است، به دوربین گوشی دسترسی پیدا کرد.



LED

یکی دیگر از ماژول‌های مورد استفاده در این پروژه، LED است. کاربرد LED به این صورت است که زمانی که سیستم در حال تشخیص حالت دست می‌باشد، با استفاده از این LED، این نکته را به کاربر اطلاع می‌دهد. برای کار با LED نیز از خروجی‌های GPIO استفاده شد.

زمان‌بندی

در انتها نیز تسک‌هایی که اساین شده بودند نیاز بود تا بر اساس یک الگوریتم زمان‌بندی، به درستی مشخص شوند که به چه صورت اجرا شوند و تضمین شود که حرکت خودرو بدون مشکل انجام می‌شود. در ادامه می‌توانید یک لیست از حرکاتی که در اصل به صورت task استخراج می‌شود را مشاهده کنید:

- **FORWARD = 1**
- **BACKWARD = 2**
- **LEFT = 3**
- **RIGHT = 4**

- **STOP = 5**

این لیست به ترتیب اهمیتی که هر کدام از تسک‌ها برای ما دارند مرتب شده است. به این ترتیب که عدد بالاتر، نشان‌دهنده درجه اهمیت و اولویت بالاتر آن تسک است. منطق پشت این اولویت‌بندی نیز روشن است؛ به این صورت که متوقف شدن خودرو دارای بالاترین درجه اهمیت است و به جلو رفتن کمترین اهمیت را دارد (از دیدگاه امنیت مسافر)

برای زمان‌بندی این تسک‌ها، به دو فاکتور hard realtime بودن سیستم و همچنین aperiodic بودن تسک‌ها باید توجه شود. به همین دلیل یکی از بهترین الگوریتم‌های زمان‌بندی که برای این منظور می‌توانستیم استفاده کنیم، الگوریتم Earliest Deadline First بود که تسک با اولویت بیشتر، ددلاین نزدیک‌ترین برای آن متصور می‌شد.

برای مدل کردن بهتر پیاده سازی این الگوریتم، یک فایل به نام scheduling.py در اختیار قرار داده شده است که با اجرای کد این فایل، یک زمان‌بند EDF در حال اجرا روی همین تسک‌هاست. یک taskqueue قرار داده شده است که با یک minheap، هر کدام از تسک‌ها که ددلاین نزدیک‌تری داشتند، آن را در آن واحد زمانی انتخاب و اجرا می‌کند. برای مثال در ۴ تیک اول اجرای این کد خروجی زیر مشاهده می‌شود:

```
Task queue: ['Task 3: STOP', 'Task 1: BACKWARD', 'Task 2: LEFT',
'Task 4: BACKWARD']
Executing task: Task 3: STOP

Task queue: ['Task 5: RIGHT', 'Task 2: LEFT', 'Task 4: BACKWARD',
'Task 1: BACKWARD']
Executing task: Task 5: RIGHT

Task queue: ['Task 6: RIGHT', 'Task 2: LEFT', 'Task 4: BACKWARD',
'Task 1: BACKWARD']
Executing task: Task 6: RIGHT

Task queue: ['Task 2: LEFT', 'Task 1: BACKWARD', 'Task 4: BACKWARD']
Executing task: Task 2: LEFT
```


همانطور که مشاهده می‌شود، همواره در هر تیک، تسک با بیشترین اولویت (نزدیک‌ترین ددلاین) انتخاب می‌شود.

راه‌اندازی

به طور کلی، تمامی مراحل اجرای این پروژه را می‌توان به صورت یک خطلوله در نظر گرفت که از ابتدا با تشخیص حرکت دست (توسط خود رزبری یا سروری که در حال اجرای مدل است) آغاز می‌شود و در نهایت با ترجمه حرکت دست به حرکت ماشین و ارسال فرمان مناسب برای حرکت ماشین، ماشین به حرکت در می‌آید و دوباره این چرخه تکرار می‌شود.

برای این بخش نیاز به موارد زیر داریم:

- پکیج ماشین راه اندازی شده
- یک لپ تاپ به عنوان سرور و همچنین برای برقرار ارتباط ssh با رزبری
- یک موبایل به همراه برنامه IP Camera به عنوان دوربین

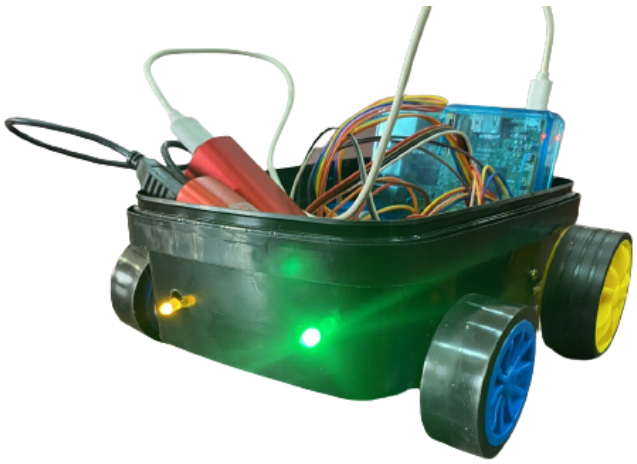
با اجرای برنامه، prompt زیر از کاربر می‌پرسد که تمایل دارد درخواست‌ها در سرور پردازش شوند یا روی خود رزبری:

```
team@team:~/Desktop/clean_project $ python main.py
Which way of processing do you want?
1. Local on Raspberry
2. On Laptop
```

و با وارد کردن هر کدام از گزینه‌ها، چرخه‌ای که جلوتر توضیح داده شد شروع می‌شود.

نتایج

نتیجه کلی بخش سخت‌افزاری به صورت packaging زیر قابل مشاهده است:



همچنین کدهای بخش نرم‌افزاری در پوشه **Codes** تحویل داده شده‌است. همچنین کدها را می‌توانید در مخزن Github زیر نیز مشاهده کنید:

مخزن گیت‌هاب برای کدها و مدل‌ها: https://github.com/mothegoat/Hardware-Lab-Project

تمامی نتایج مربوط به مدل‌ها نیز در بخش تمرین مدل‌یادگیری و همچنین در پوشه **training report** در اختیار قرار داده شده است.

با تشکر از جناب‌آقای یونسی و آقای بحرینی بابت زحمات و اهتمام ایشان به این پروژه.