به نام خدا

گزارش نهایی درس آز سخت افزار

نیم سال دوم سال تحصیلی ۱۴۰۱-۱۴۰۰

گروه ۶

آرمان زارعی

کیوان رضائی

سیدمحمد سیدجوادی

فهرست محتوا

۱. مقدمه

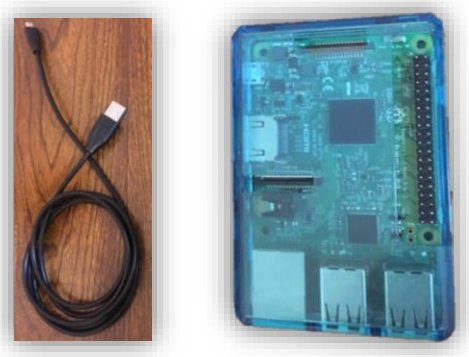
در این پروژه قرار است یک ماژول طراحی کنیم که حرکت چشم کاربر به چپ و راست را تشخیص دهد و برای کامپیوتری که به آن متصل است بفرستد. این ماژول قرار است به عنوان یک Gamepad در پروژه‌ی ما به کار گرفته شود و کابر بتواند به کمک این ماژول و حرکت دادن مردمک چشمش با نگاه کردن به جهات مختلف بازی کند. این بازی روی کامپیوتر کاربر بالا می‌آید و کاربر به کمک چشمش بازی می‌کند. برای دیدن کدها و دیگر اطلاعات پروژه به مخزن گیت‌هاب ما در لینک [۶] مراجعه کنید.

در ادامه این سند:

- در بخش قطعات، قطعات استفاده شده در این پروژه و اتصال آنها به یکدیگر مطرح شده است.
- سپس در بخش آماده‌سازی رزبری پای، کارهای انجام شده جهت کار با رزبری پای و نصب ابزارهای لازم روی آن آورده شده است.
- در ادامه، در بخش پیاده‌سازی ماژول تشخیص مردمک، شیوه تشخیص مردمک چشم و کدهای مربوط به آن توضیح داده شده است.
- پس از آن بازی‌ای که برای تست محصول پیاده‌سازی شد شرح داده شده است.
- سپس در بخش اتصال بازی و ماژول در مورد جزئیات کار با ماژول و گرفتن اطلاعات حرکتی مردمک از آن بیان شده است.
- نهایتاً در بخش آخر، تست سیستم انجام گرفته و فیلم‌های تست قرار داده شده است.

۲. قطعات استفاده شده

همانطور که در مقدمه گفته شد، برای ماژول تشخیص حرکت مردمک چشم ما از یک رزبری‌پای (Raspberry Pi 3 Model B) و دوربین مخصوص رزبری پای استفاده کردیم. در شکل ۱ می‌توانید رزبری‌پای تحویل گرفته شده به همراه قاب و کابل Power آن را مشاهده کنید. برای روشن کردن رزبری‌پای می‌توان این کابل را به لپتاپ یا هر منبع برق دیگری متصل کرد.



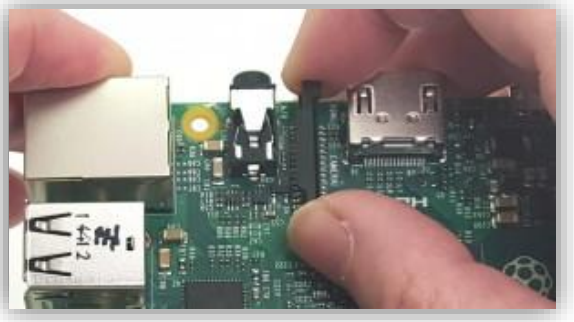
شکل ۱ رزبری پای تحویل گرفته شده به همراه قاب و کابل پاور آن

همچنین در شکل ۲ می‌توانید دوربین مخصوص رزبری‌پای را مشاهده کنید. بر روی دوربین یک اهرم وجود دارد که با آن می‌توان فوکوس دوربین را تنظیم کرد.

شکل ۲ دوربین مخصوص رزبری پای

نحوه‌ی اتصال دوربین به رزبری پای

پورت مخصوص دوربین در رزبری پای بین پورت صدا و پورت HDMI قرار دارد. برای اتصال دوربین به رزبری پای همانطور که در شکل ۳ می‌بینید، ابتدا پورت دوربین را باز می‌کنیم. سپس مانند شکل ۴، کابل را درون پورت می‌گذاریم به طوری که قسمت آبی رنگ کابل به سمت پورت صدا باشد. در نهایت، پورت دوربین را مانند شکل ۵ محکم می‌کنیم.



شکل ۳ باز کردن پورت دوربین

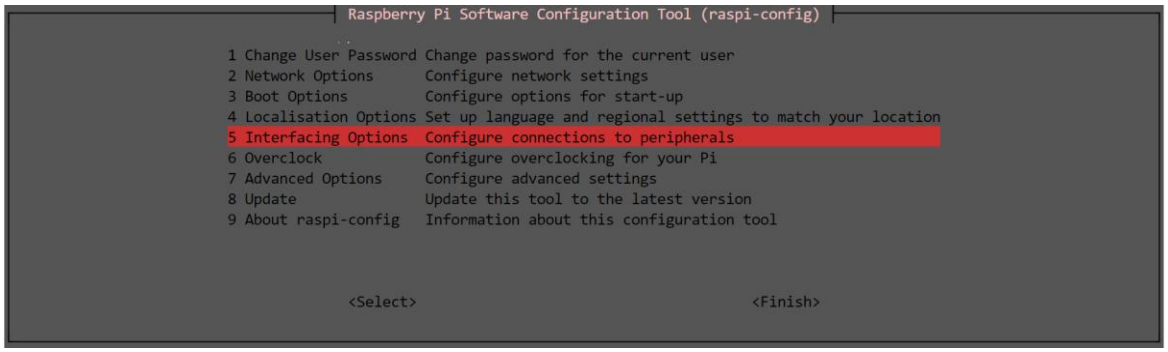
شکل ۴ قرار دادن کابل دوربین بر روی پورت

شکل ۵ بستن پورت دوربین

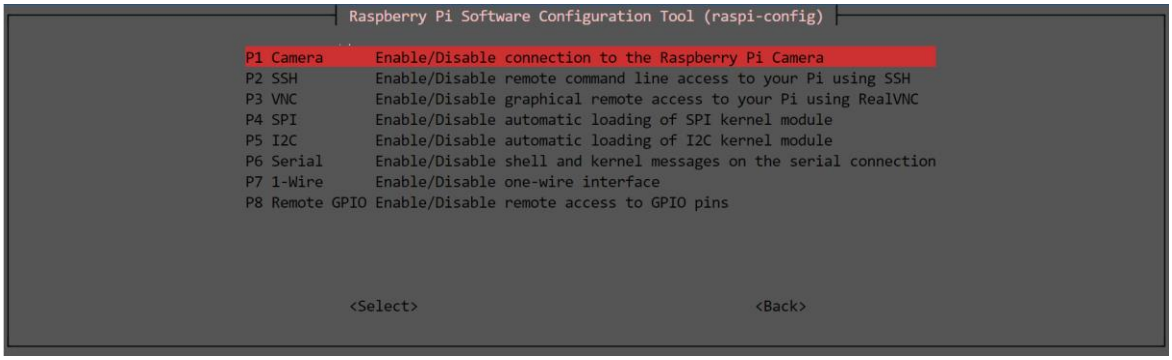
۰۳ نحوه کار با دوربین

بعد از اتصال دوربین به رزبری پای باید واسط دوربین را فعال کنیم. به این منظور، دستور raspi-config را اجرا می‌کنیم. بعد از اجرای دستور raspi-config که شکل ۶ مشاهده می‌شود.

سپس با انتخاب Interfacing Options به شکل ۶ می‌رسید. که در آنجا باید Camera را فعال کنید. (شکل ۷)



شکل ۶ raspi-config



شکل ۷ صفحه‌ی Interfacing Options

برای تست انواع مدهای مختلف دوربین و مناسب بودن فوکوس دوربین، از دستور raspistill استفاده کردیم که یک عکس با دوربین می‌گرفت. برای مثال دستور زیر یک عکس با تنظیمات پیشفرض می‌گیرد و در فایل output.jpg ذخیره می‌کند. [۲]

raspistill -o output.jpg

شرح تنظیمات کامل این دستور را می‌توانید با دستور زیر ببینید:

raspistill --help

برای فیلم‌برداری و استفاده از دوربین در میان کد پایتون، از کتابخانه‌ی Picamera استفاده کردیم. [۳] برای گرفتن عکس‌های متوالی از دستور زیر استفاده کردیم.

```
self._camera = PiCamera()
self._camera.resolution = (camera_width, camera_height)
self._camera.framerate = camera_framerate
self._raw_capture = PiRGBArray(self._camera, size=(camera_width, camera_height))
for frame in self._camera.capture_continuous(self._raw_capture, format="bgr", use_video_port=True)
```

۴. آماده‌سازی رزبری پای

اتصال به رزبری پای

در این بخش به شیوه اتصالمان به رزبری پای و نصب ابزارهایی که برای انجام پروژه به آنها نیاز بود، می‌پردازیم. نخست برای اتصال به رزبری پای، آن را به برق وصل کرده و با کابل شبکه به اینترنت متصل کردیم. در ادامه با لپتاپی که به همان شبکه متصل است، می‌توان به رزبری پای SSH زد و به آن وصل شد. ما از ابزار VSCode که محیط مناسبی برای Remote Exploring فراهم می‌کند برای ایجاد ارتباط SSH با رزبری پای استفاده کردیم. در شکل ۸ می‌توانید نتیجه‌ی دستور ساده‌ی ls را ببینید که بعد از SSH به درستی اجرا شده است.

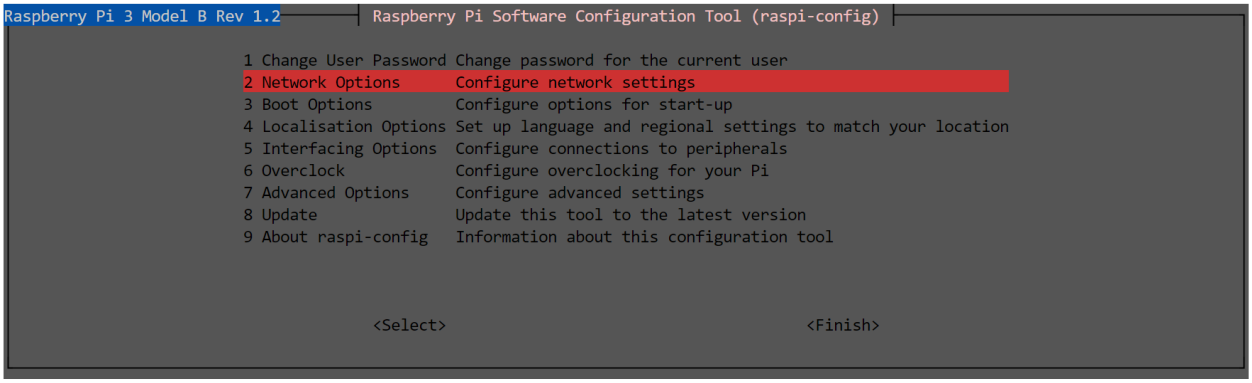
شکل ۸ اجرای دستور ls بعد از برقراری ssh به رزبری پای از طریق VSCode

سیستم‌عامل نصب شده روی رزبری پای همانطور که در شکل ۹ مشاهده می‌شود Raspbian OS است که مختص به رزبری پای می‌باشد.

```
pi@raspberrypi:~/Desktop $ lsb_release -a
No LSB modules are available.
Distributor ID: Raspbian
Description:    Raspbian GNU/Linux 9.13 (stretch)
Release:        9.13
Codename:       stretch
```

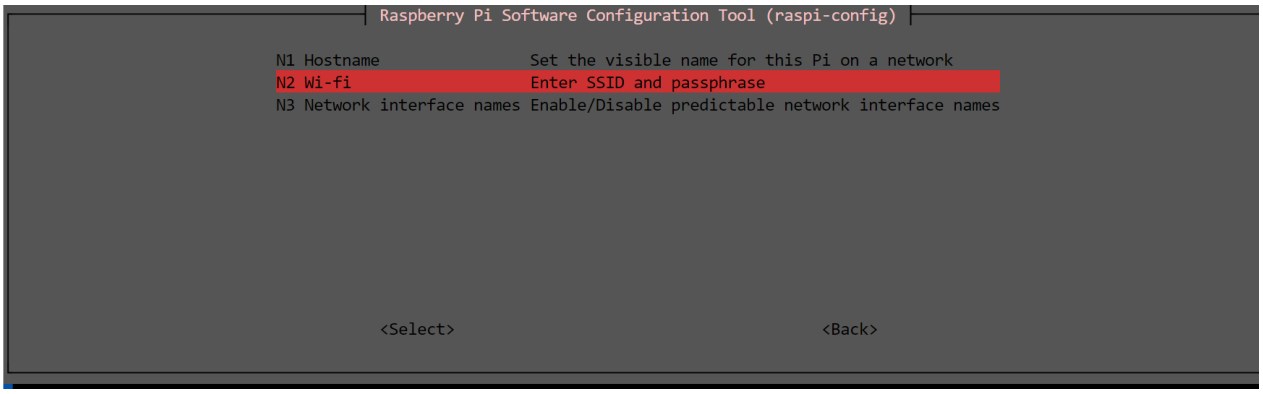
شکل ۹ مشخصات سیستم عامل رزبری پای

با توجه به اینکه رزبری پای ماژول WiFi دارد، برای اینکه در ادامه نیازی به اتصال دائمی رزبری پای به کابل شبکه نباشد، ما تنظیمات WiFi را روی آن بالا آوردیم و از طریق WiFi به شبکه وصل شدیم. به این منظور از دستور sudo raspi-config استفاده کردیم که تنظیمات سیستم را می‌آورد.



شکل ۱۰ نتیجه‌ی اجرای دستور raspi-config

سپس در بخش Network Options منوی زیر را می‌بینیم که در بخش WiFi می‌توان شبکه موردنظر و پسوورد اتصال به مودم را وارد کرد و به WiFi وصل شد.



شکل ۱۱ بخش Network Options در raspi-config

نصب کتابخانه‌های لازم

پایتون ۳،۵ روی رزبری پای نصب شده بود، ما از همین پایتون برای اجرای کدهایمان استفاده کردیم. منتها لازم بود چند کتابخانه برای اجرای کدهایمان نصب شوند.

لیست کتابخانه‌ها به شرح زیر است:

- numpy: جهت انجام محاسبات ریاضی و ماتریسی روی تصاویر
- 2cv: کتابخانه‌ای که امکانات پردازش تصویر فراهم می‌کند.
- picamera: کتابخانه‌ای که به کمک آن می‌توان با دوربین ارتباط برقرار کرد.
- socketio: کتابخانه لازم جهت ایجاد سرور و دریافت و ارسال سوکت‌ها
- eventlet: کتابخانه لازم جهت ایجاد سرور و اتصال به کلاینت

فرایند نصب کتابخانه‌ها هم طبق روال همیشگی پایتون و به کمک pip انجام شد.

نکته مهمی که برای نصب کتابخانه‌های پایتون روی رزبری پای وجود دارد این است که باید کتابخانه‌ها به شکل pre-built دانلود شوند. در غیر اینصورت build کردن محتوای دانلود شده و نصب کتابخانه‌ها بسیار زمان‌بر خواهد بود. جزئیات بیشتر در مورد این کتابخانه‌های pre-built شده در سایت [۵] موجود است. ما نیز از همین سایت استفاده کردیم.

به منظور دانلود کتابخانه‌ها از این سایت لازم است تغییرات زیر در config مربوط به pip عوض شود. به این منظور لازم است خط زیر در فایل /etc/pip.conf افزوده شود.

```
[global]
extra-index-url=https://www.piwheels.org/simple
```

جزئیات بیشتر در خصوص استفاده از این کتابخانه‌های نیمه آماده در این سایت [۵] قابل مشاهده است.

اتصال به Github و ریپوی پروژه

گام آخر اتصال به git بود تا بتوانیم کدها را روی رزبری پای بیاوریم. به این منظور git نصب شد و پروژه clone شد. محتوای فایل‌ها به صورت زیر است.

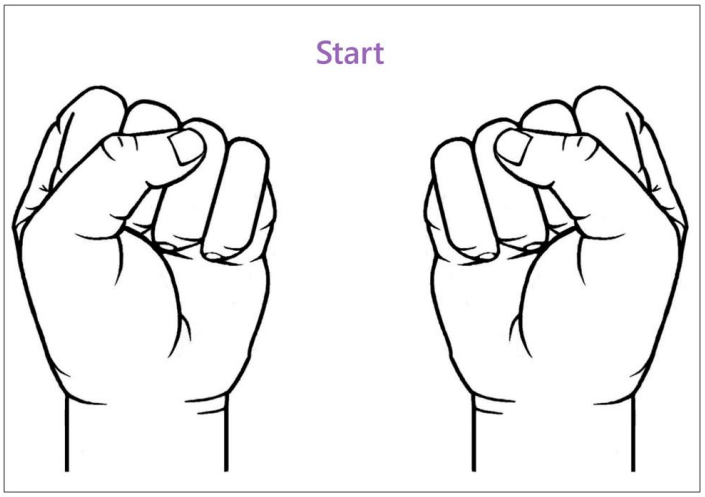
```
pi@raspberrypi:~/Desktop/project/project-team-6 $ ls
Code  Datasheet  Document  Miscellaneous
```

شکل ۱۲ همانطور که در شکل می‌بینید پروژه به درستی clone شده است.

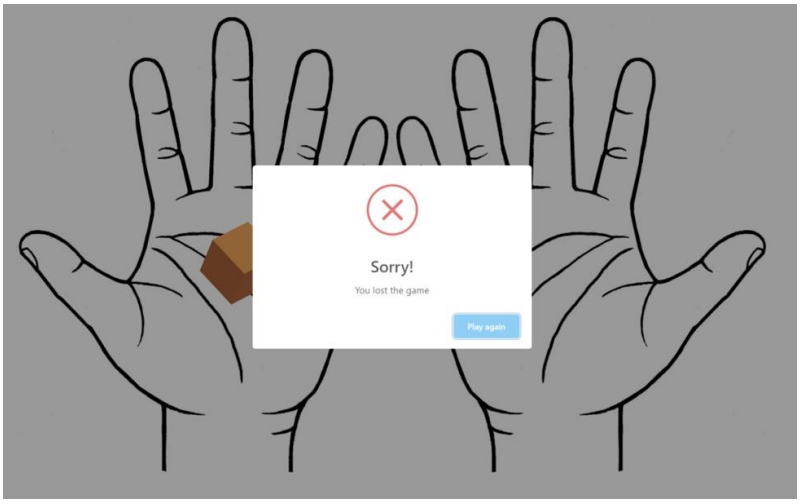
۰۵ پیاده‌سازی بازی گل یا پوچ

با توجه به اینکه بازی ماشینی که در نظر داشتیم با چشم به راحتی قابل بازی کردن نبود (جذاب نبودن آن و تجربه بازی نه چندان جالب)، بازی را به بازی گل یا پوچ تغییر دادیم. همانطور که میدانید در این بازی، ۲ دست بسته در ابتدا به شما نشان داده می‌شود که در یکی از آن‌ها مهره ای قرار دارد. ما باید حدس بزنیم که در کدام دست آن مهره قرار دارد. بعد از گفتن حدسمان به حریف، دست هایش را رو می‌کند و اگر دست درستی را انتخاب کرده باشیم، برنده بازی می‌شویم و در غیر اینصورت بازنده خواهیم شد.

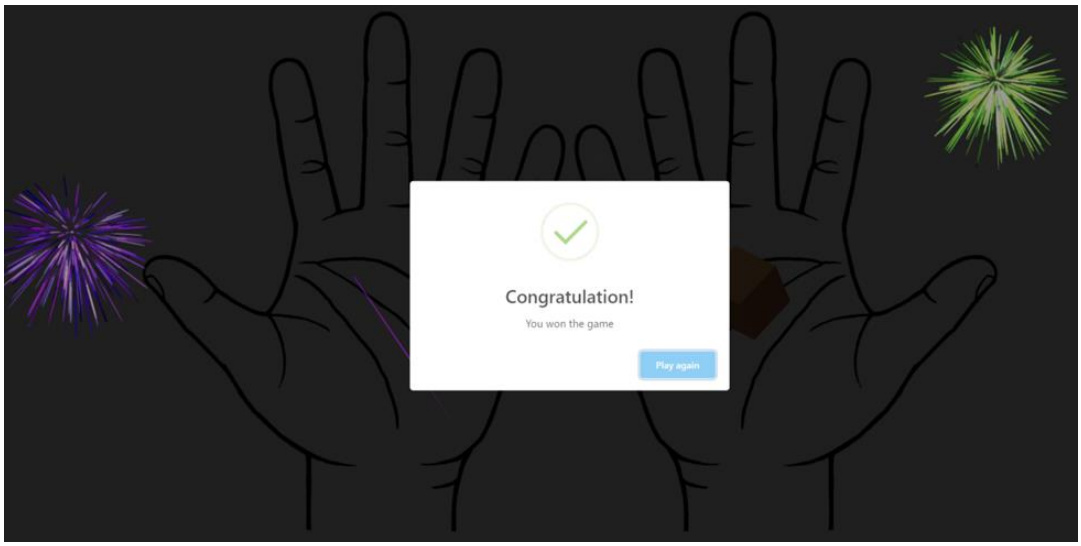
این بازی با برخی زبان ها به صورت متن باز در اینترنت موجود است ولی ما به دلیل اینکه این بازی را کمی جذاب تر کنیم، تصمیم گرفتیم که خودمان بازی را پیاده‌سازی کنیم. بازی را با HTML و CSS و JavaScript و به کمک برخی کتابخانه‌ها توسعه دادیم و می‌توانید آن را بر روی مرورگرتان اجرا کنید. برخی تصاویر از محیط بازی را می‌توانید در شکل‌های زیر مشاهده کنید:



شکل ۱۳ شروع بازی و انتخاب یکی از دو دست بسته



شکل ۱۴ انتخاب دست راست و عدم وجود مهره در آن که موجب باخت ما در این دست بازی شده است



شکل ۱۵ انتخاب دست راست و وجود مهره در آن که موجب به برد ما در این دست بازی شده است

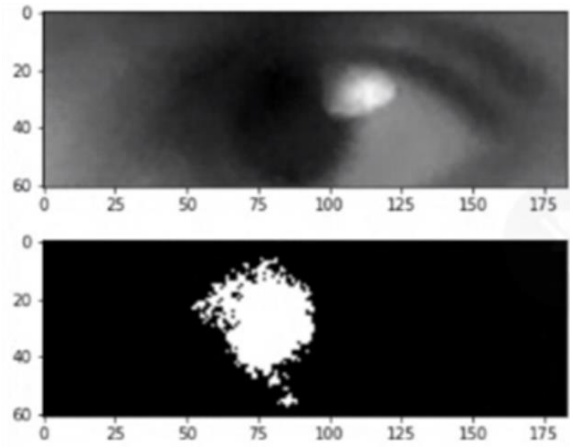
۶. پیاده‌سازی ماژول تشخیص مردمک

در ابتدا از تصمیم بر استفاده از ماژول های نوشته شده و آماده بود ولی بعد از تست کردن برخی از آن‌ها و کارکرد نامطلوبشان، تصمیم گرفتیم که این ماژول را خودمان پیاده‌سازی کنیم. ماژول‌های آماده متن-باز در برخی جهت‌ها و برخی تصاویر دچار مشکلاتی می‌شدند که ماژول ما به آن ها دچار نمی‌شود و با دقت بسیار بالاتری تخمینی از جهت چشم را به ما می‌گوید.

همانطور که در فایل gaze_tracking.py مشاهده می‌کنید دو کلاس برای این ماژول نوشته شده است. کلاس GazeDetector و کلاس GazeTracking. کلاس GazeDetector وظیفه تشخیص چشم و جهت نگاه را بر عهده دارد و کلاس GazeTracking وظیفه اتصال به دوربین و ضبط تصویر کاربر و تشخیص جهت چشم به صورت پیوسته (به کمک کلاس GazeDetector) را بر عهده دارد.

الگوریتم پیاده‌سازی شده

در ابتدا به کمک کتابخانه OpenCV و کلاس CascadeClassifier مربع‌های اطراف چشم را بدست می‌آوریم. حال با داشتن این مربع ها نمی‌توانیم تشخیص دهیم که چشم ها به کدام طرف نگاه می‌کنند. ناحیه ⅓ ام میانی عمودی عکس را برش می‌دهیم تا مستطیلی بدست بیاید که نسبت به مربع قبلی مساحت کمتری داشته باشد ولی همچنان چشم را شامل شود. حال عکس رنگی را به عکس سیاه سفید تبدیل می‌کنید. در این مرحله، هر کدام از پیکسل‌های عکس دارای یک شدت رنگی (intensity) می‌باشند. بر اساس شدت رنگ آن‌ها را سورت می‌کنیم و ۱۰ درصد کمترین را به عنوان رنگ سیاه و بقیه را به عنوان رنگ سفید در نظر می‌گیریم. بدین ترتیب، مردمک چشم تشخیص داده می‌شود. نمونه‌ای از نتیجه این فرایند را می‌توانید در عکس زیر مشاهده کنید (نواحی سفید مردمک تشخیص داده شده و سیاه بقیه عکس می‌باشد).



شکل ۱۶ تشخیص مردمک

در نهایت هم عرض (x) نقاط مردمک را در نظر گرفته و مقدار میانه (median) آن را با نصف طول مستطیل (خط وسط مستطیل) مقایسه می‌کنید. اگر کمتر از آن بود به معنای نگاه به سمت چپ، و در غیر اینصورت، نگاه به سمت راست تشخیص داده می‌شود. کد تابع اصلی این کلاس را می‌توانید در زیر مشاهده کنید.

```python
def detect_eye_and_direction(self, face: np.ndarray):
    # Detecting eyes using cv2.CascadeClassifier and haarcascade_eye.xml
    eyes = self._eye_cascade_classifier.detectMultiScale(face, scaleFactor=self.scale_factor, minNeighbors=self.min_neighbors)

    face_gray = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY) # Convert RGB image to Black-and-White image

    sum_of_dists = 0
    for idx, (eye_x, eye_y, eye_w, eye_h) in enumerate(eyes):
        if idx == 2: # We don't have more than 2 eyes!
            break;
        eye_rect = face_gray[eye_y+int(eye_h/3):eye_y+int(2*eye_h/3), eye_x:eye_x+eye_w] # Fit the rectangle to the eyes vertically
        pupil_mask = (eye_rect < np.sort(eye_rect.flatten())[int(self.pupil_blackness_threshold*eye_rect.size)]).astype(int) # 0/1 mask
        _, y_arr_pupil = np.where(pupil_mask) # Coordinates of the pixels of the eye
        y_median_pupil = np.median(y_arr_pupil) # Median of the pixels
        sum_of_dists += y_median_pupil - eye_rect.shape[1]//2 # Distance of the pixels from median

    return self.Directions.LEFT if sum_of_dists > 0 else self.Directions.RIGHT # Decide the direction of the eye based on the distances
```

شکل ۱۷ کد تابع اصلی کلاس GazeDetector

تشخیص جهت چشم در مدت زمان معین

همانطور که گفتیم، کلاس GazeTracking که کد تابع اصلی این کلاس را می‌توانید در زیر مشاهده کنید، وظیفه مشاهده دوربین کاربر و تشخیص جهت چشم در مدت زمان مشخصی را بر عهده دارد. در این کلاس در تابع get_eye_direction به کمک کتابخانه Picamera با دوربینی که به رزبری پای وصل شده است ارتباط برقرار می‌کند و از روی آن فریم‌هایی بدست می‌آورد. (خط ۱۹) سپس هر فریم را به کلاس GazeDetector می‌دهد (خط ۲۱) تا جهت چشم را بدست بیاورد. این کار را در فریم‌های مختلف انجام می‌دهد تا زمان مشخص شده به اتمام برسد.

```
15  def get_eye_direction(self, time_to_capture: int):
16      start_time = time.time()
17      directions = {GazeDetector.Directions.LEFT: 0, GazeDetector.Directions.RIGHT: 0} # Dict for counting estimations
18
19      for frame in self._camera.capture_continuous(self._raw_capture, format="bgr", use_video_port=True):
20          image = frame.array
21          direction = self._gaze_detector.detect_eye_and_direction(image) # Detect direction using GazeDetector
22          directions[direction] = directions[direction] + 1 # Update dict of estimations
23
24          if self._print_logs: # When Logging is enabled
25            print("[GazeTracking LOG] Eye Direction:", direction)
26
27          self._raw_capture.truncate(0)
28
29          curr_time = time.time()
30          if curr_time - start_time > time_to_capture: # Stop if exec. time is greater than time_to_capture
31              break
32
33      # Majority vote
34      answer = GazeDetector.Directions.LEFT
35      if directions[GazeDetector.Directions.LEFT] < directions[GazeDetector.Directions.RIGHT]:
36        answer = GazeDetector.Directions.RIGHT
37
38      return answer, directions
```

شکل ۱۸ کد تابع اصلی کلاس GazeTracking

سرور بازی

در فایل server.py که قرار است روی raspberry pi اجرا شود، روی یک پورت منتظر می‌ماند تا کد مربوط به بازی به آن وصل شود و سرور برای تشخیص مردمک چشم از فایل gaze_tracking.py استفاده می‌کند و تصمیم نهایی کاربر را برای بازی می‌فرستد. این ارتباط به کمک کتابخانه Socketio که یک سوکت می‌سازد صورت می‌گیرد. می‌توانید کد خلاصه‌ی آن را در زیر مشاهده کنید. شامل یکسری کد استاندارد است و خط‌های ۲۰ تا ۲۳ نیز سرور را بالا می‌آورند.

```python
@sio.event
def connect(sid, environ): # New Client connection
    ...

@sio.event
def handshake(sid, data): # Handshake between client and the server
    ...

@sio.event
def disconnect(sid): # Client disconnects
    ...

if __name__ == '__main__':
    print("Server is starting ...")
    gaze_tracking = GazeTracking(GazeDetector(), print_logs=True)
    eventlet.wsgi.server(eventlet.listen(('', 4343)), app)
```

شکل ۱۹ کد مربوط به server.py

۰۷ اتصال بازی و ماژول

به صورت کلی، ماژول پس از اجرا یک سرور سوکت به کمک کتابخانه socketio می‌سازد که آماده پذیرش درخواست‌های کلاینت‌ها (مثلاً بازی) می‌باشد. هر کلاینت که بخواهد از ماژول اطلاعات بگیرد، لازم است به سرور متصل شود و درخواست خودش مبنی بر گرفتن اطلاعات مردمک چشم را به سرور ارسال کند. در حال حاضر با توجه به نیازمان، این سیستم به این شکل پیاده‌سازی شده است که کلاینت یک مدت زمان t به سرور می‌دهد و ماژول به مدت t چشم کاربر را رصد می‌کند و جهتی (چپ یا راست) که کاربر در آن مدت به آن نگاه می‌کرده را برای کلاینت می‌فرستد.

به منظور اجرای سرور، لازم است فایل server.py اجرا شود. با اجرای این فایل سرور ساخته می‌شود و منتظر اتصال کلاینت است. سرور روی پورت 4343 منتظر اتصال است.

شکل ۲۰ اجرای سرور

سپس بازی اجرا می‌شود، این بازی به پورت 4343 و آدرس ip رزبری پای موجود در شبکه کانکشن می‌زند. هر بار که کاربر می‌خواهد بازی کند، یک درخواست از سمت بازی به سرور رزبری پای می‌رسد و به مدت ۵ ثانیه چشم کاربر رصد می‌شود. پس از این ۵ ثانیه، سرور جهت را به کلاینت اعلام می‌کند و بازی انجام می‌شود.

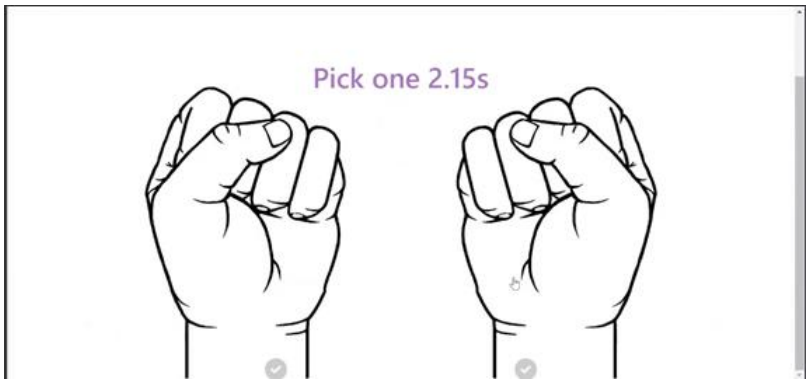
در شکل زیر لاگی که سمت سرور تولید می‌شود را می‌بینید. همانطور که می‌بینید ۱۰ بار در ۵ ثانیه عکس‌برداری شده و در هر عکس جهت مردمک چشم تشخیص داده شده است. در نهایت نیز سمتی که بیشترین بار در این ۱۰ بار تکرار شده به عنوان جواب برگردانده می‌شود.

```
(4628) accepted ('192.168.1.106', 57853)
New Connection: 9b481ad1b0724cc5973afdc544cff441
Connection disconnected: 9b481ad1b0724cc5973afdc544cff441
192.168.1.106 - - [29/May/2022 18:05:42] "GET /socket.io/?EIO=3&transport=websocket HTTP/1.1" 200
0 0.548729
(4628) accepted ('192.168.1.106', 57855)
New Connection: 5b1adf45bf194f4eb4c6b527b5f76dbd
Handshake {'duration': 5}
[GazeTracking LOG] Eye Direction: RIGHT
[GazeTracking LOG] Eye Direction: LEFT
[GazeTracking LOG] Eye Direction: LEFT
[GazeTracking LOG] Eye Direction: RIGHT
[GazeTracking LOG] Eye Direction: RIGHT
[GazeTracking LOG] Eye Direction: RIGHT
[GazeTracking LOG] Eye Direction: RIGHT
[GazeTracking LOG] Eye Direction: RIGHT
[GazeTracking LOG] Eye Direction: RIGHT
Result sent to game: RIGHT
```

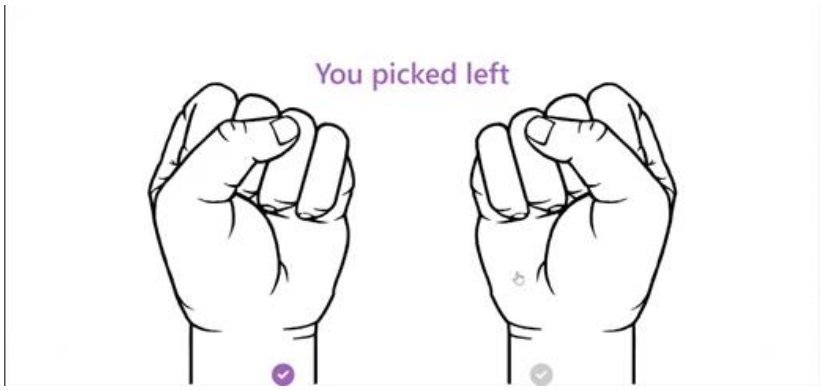
شکل ۲۱ نمونه‌ای از لاگ‌های سرور

در شکل های زیر مراحل بازی را بر روی مروگر مشاهده می‌کنید.

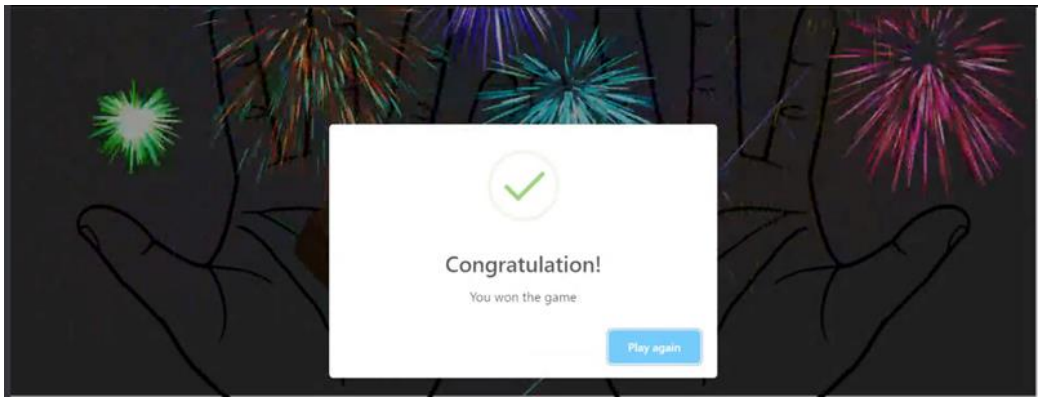
شکل ۲۲ شروع بازی



شکل ۲۳ زمان برای انتخاب جهت



شکل ۲۴ نمایش نتیجه انتخاب کاربر



شکل ۲۵ نمایش نتیجه بازی

۰۸ بسته‌بندی و طراحی فیزیکی

به دلیل موجود نبودن شرایط و اینکه باید به شکل اولیه رزبری پای و دوربین را در انتها تحویل می‌دادیم، با توجه به اجازه مدرس درس، قرار شد صرفا ایده‌هایی که برای بسته‌بندی داریم را ذکر کنیم. در ادامه به بررسی چند مورد از آن‌ها می‌پردازیم.

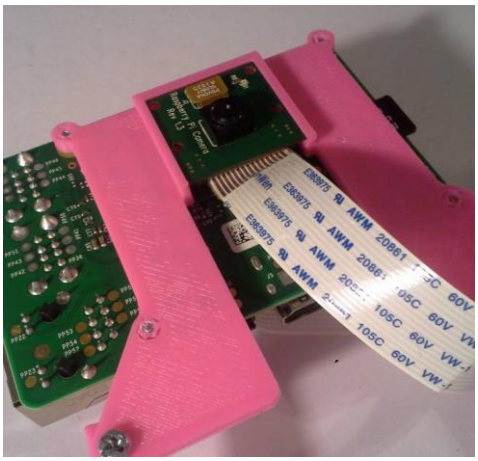
اولین ایده‌ی بسته‌بندی ما این است که پشت قاب رزبری پای یک گیره مثل شکل ۲۶ بچسبانیم در این صورت می‌توان رزبری‌پای را بر روی اکثر مکان‌ها به خصوص بالای لپ‌تاپ نصب کرد.

ایده‌های با خرج بیشتر برای این قسمت نیز موجود است که می‌توان از پایه‌های نگهدارنده موبایل همانطور که در شکل ۲۷ بعضی از آن‌ها را مشاهده می‌کنید استفاده کرد. در این صورت می‌توان رزبری پای را در هر زاویه و موقعیتی قرار داد.

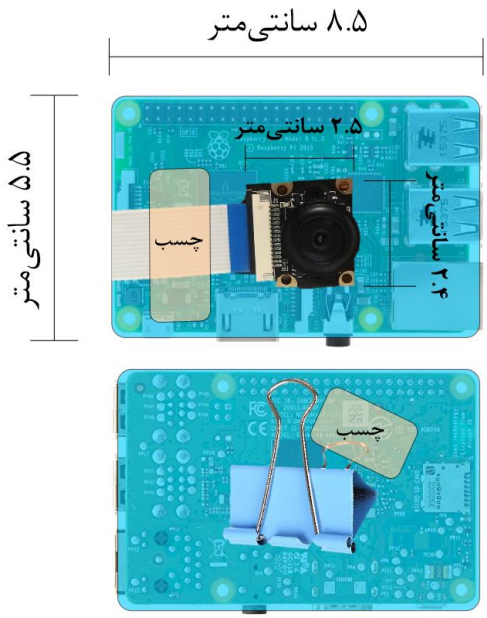
شکل ۲۶ گیره برای اتصال به قسمت جلوی رزبری‌پای

شکل ۲۷ انواع پایه‌های مناسب

همچنین برای فیکس کردن دوربین بر روی رزبری پای می‌توان با استفاده از پرینتر سه بعدی مانند شکل ۳۰ قطعه‌ای ایجاد کرد که مکانی مناسب برای قرارگیری دوربین بر روی رزبری پای فراهم می‌کند. برای طراحی ساده‌تر می‌توان از چسب نیز استفاده کرد. (از آنجایی که رزبری‌پای ما خود دارای قاب بود) یک نمونه از بسته‌بندی ساده را به همراه ابعاد دقیق می‌توانید در شکل ۲۹ ببینید.



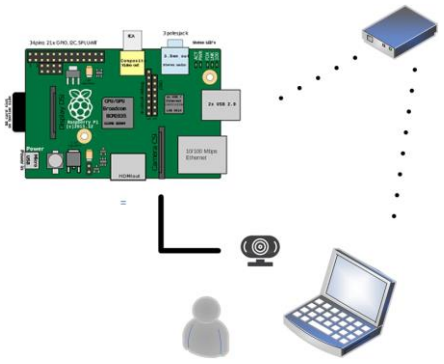
شکل ۲۸ فیکس کردن دوربین بر روی رزبری‌پای [۱]



شکل ۲۹ یک مدل بسته‌بندی ساده برای محصول

۰۹ تست و بررسی و مزیت رقابتی

مطابق با توضیحات اجرای بازی و سرور و اتصال آنها به هم، در فیلمی که در لینک [۴] قرار دارد، یک نمونه از اجرای بازی که عملکرد سیستم و صحت آن را نشان می‌دهد مشاهده می‌شود. همانطور که در شکل ۳۰ نیز می‌بینید کامپیوتر استفاده کننده از محصول و محصول باید به یک شبکه وصل باشند و تصویر کاربر توسط دوربین محصول ثبت می‌شود.



شکل ۳۰ دیاگرام بلوکی محصول

مزیت رقابتی

بازی‌های زیادی طراحی شده‌اند که توسط سنسورهای مختلف کنترل می‌شوند. سنسورهایی از قبیل: تسخیص صدا، تشخیص حرکت دست و غیره. ماژول‌هایی که به‌وسیله این سنسورها تخمین درستی از حرکت مورد نظر کاربر داشته باشند، ماژول‌های مطلوب کاربران هستند. در حوزه‌ی تشخیص مردمک چشم و جهت نگاه، کارهای بسیار زیادی انجام شده است و بازی‌هایی طراحی شده‌اند که با کمک آن‌ها کنترل می‌شوند. درصد خطا در اکثر این ماژول‌ها در مقایسه با ماژول نوشته شده در این پروژه بیشتر می‌باشد. در این پروژه سعی شده با بهبود مدل‌های کنونی و استفاده از روش‌های خلاقانه در حوزه‌ی پردازش تصویر، درصد خطا تا حد بسیار زیادی کم شود تا کاربر بتواند تجربه‌ی کاربری خیلی مطلوب‌تری را داشته

باشد و همچنین درصد رضایت بیشتری را از مشتریان کسب کنیم. ضمنا مدل موجود از نظر محاسباتی سبک است و نیازمند سخت‌افزاز قدرتمندی برای تشخیص نیست. همچنین، بازی نوشته شده در این پروژه با به کارگیری ماژول‌های گرافیکی پیشرفته، تجربه بازی بسیار دلپذیری را برای کاربران فراهم کرده است.

۱۰. منابع

۱. https://www.myminifactory.com/object/۳d-print-raspberry-pi-b-adjustable-camera-clamp-mount-۲۰۷۲۱

۲. https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/۰

۳. https://picamera.readthedocs.io/en/release-۱.۱۳/

۴. https://aparat.com/v/CvT۰g

۵. https://www.piwheels.org/faq.html

۶. https://github.com/Sharif-University-ESRLab/project-team-۶