گزارش پروژه کارشناسی مکاترونیک

عنوان:

بازوی رباتیک فیلم بردار هوشمند با ردیابی سوژه

دانشجو:

مهدی حمزه

استاد راهنما:

دکتر سید علی هاشمی

دی ۱۴۰۴

فهرست مطالب

فهرست شکلها

چکیده

این پروژه به طراحی، ساخت و پیاده‌سازی یک بازوی رباتیک فیلم‌بردار هوشمند می‌پردازد که قادر است با ردیابی ژست‌های دست کاربر، حرکت دوربین و عملیات ضبط را به‌صورت خودکار کنترل نماید. هدف اصلی، ارائه یک راهکار تعاملی برای هدایت دوربین بدون نیاز به اپراتور انسانی است. در بخش سخت‌افزار، از یک بازوی رباتیک اپن سورس SO-101 با موتورهای سروو باس استفاده شده که سازه آن با پرینتر سه‌بعدی ساخته شده است. برای شبیه‌سازی و حل سینماتیک معکوس ربات از فریمورک PyBullet و برای پردازش تصویر و تشخیص ژست‌های دست از کتابخانه MediaPipe گوگل بهره گرفته شده است. یکی از چالش‌های کلیدی این پروژه، نگاشت موقعیت دوبعدی دست در تصویر دوربین به فضای کاری سه‌بعدی ربات بود که این مشکل با تعریف یک صفحه کاری مجازی و خمیده برای حرکت نرم-End Effector حل گردید. نتیجه نهایی، یک سیستم یکپارچه و کارآمد است که تعاملی نوین را برای کنترل دوربین در کاربردهای مختلف فیلم‌برداری فراهم می‌کند.

کلمات کلیدی: بازوی رباتیک، پردازش تصویر، سینماتیک معکوس، ربات فیلم بردار، تخمین ژست

فصل اول: مقدمه

۱-۱ بیان مسئله

این پروژه به طراحی و پیاده سازی یک بازوی رباتیک سبک و هوشمند با قابلیت ردیابی حرکات دست میپردازد. هدف اصلی فراهم کردن یک سیستم تعاملی برای کنترل موقعیت و زاویه دید دوربین است که بتواند بر اساس حرکات دست کاربر، دوربین را در فضا به جهات مختلف و مورد نظر هدایت کند و قابلیت ظبط و توقف فیلم برداری با استفاده از ژست های دست را داشته باشد.

۱-۲ پیشینه تحقیق

پروژه ای مشابه این پروژه که یک بازوی رباتیک به صورت خودکار و توسط دوربین سوژه را دنبال کند و از آن فیلم برداری کند تا کنون پیاده سازی نشده. اما پروژه هایی با اهداف مشابه ساخته شده اند که در دسته بازو های رباتیک فیلم بردار قرار گرفته میشوند، به طور مثال میتوان به بازوی PIXEL از شرکت camerabotics اشاره کرد که قابلیت اتصال دوربین فیلم برداری را در End effector خود دارد و توسط نرم افزار مسیر حرکت دوربین و زاویه و پارامتر های آن قابل برنامه ریزی و اجرا هستند. نمونه های دیگری هم مانند ربات evo از شرکت motorizedprecision وجود دارند که مانند ربات قبلی توسط نرم افزار قابل برنامه ریزی هستند.

فصل دوم: روش کار

۲-۱ قسمت مکانیکی بازو

برای بخش مکانیکی پروژه نیاز به طراحی یا استفاده از یک بازوی رباتیک آماده داشتیم. با بررسی های انجام شده بهترین راه حل استفاده از یک بازوی رباتیک آماده و ویرایش آن در صورت لزوم، بر اساس نیاز خود بود، به این دلیل که ساختار کلی بازوی مورد نیاز یک ساختار متداول است که با نام بازوی رباتیک 4R شناخته میشود. پس از بررسی های مختلف بازوی رباتیک SO-101 به عنوان یک بازوی دارای ۶ درجه آزادی، بهترین گزینه برای پیاده سازی پروژه در نظر گرفته شد. بدنه این ربات جهت پرینت با پرینتر سه بعدی FDM طراحی شده است و میتوان از متریال PLA برای پرینت بدنه ربات استفاده کرد.

شکل (۲-۱) : تصویری از بازوی رباتیک SO-101

بازوی SO-101 جهت حرکت از سروو موتور های باس مدل STS3215 استفاده میکند. این سروو موتور ها به دلیل ویژگی های خاص خود کاملا مناسب پروژه های تحقیقاتی آموزشی و یا حتی برخی پروژه های صنعتی میباشد. از جمله ویژگی این موتور ها میتوان به موارد زیر اشاره کرد:

- کنترلر PID داخلی قابل برنامه ریزی
- اتصال تا ۲۵۳ سروو به یکدیگر و کنترل همه با ۱ پین دیتا
- گزارش لحظه ای زاویه فعلی، بار، ولتاژ، حالت کاری، دما و...

شکل (۲-۲) : تصویری از سروو موتور های STS3215

برای فرمان دادن به این سروو موتور ها کافیست که موقعیت سروو موتور مورد نظر را به همراه شناسه آن سـروو توسـط پروتکل UART برای سـروو ارسـال کنیم. با توجه به این موضـوع ما میتوانیم این دیتای سریال را توسط کامپیوتر و با کمک یک مبدل ارسال کنیم.

در این پروژه ما از درایور سـروو سـریال باس قابل اسـتفاده برای سـروو های ST و SC اسـتفاده کردیم که یک ورودی تغذیه دارد که جهت تغذیه موتور ها اسـتفاده میشـود و میتوان توسـط یک آداپتور V۱۲ را به آن متصـل کرد، همچنین دارای یک ورودی USB اسـت که به کامپیوتر متصـل میشـود و دیتا های مورد نیاز برای موتور ها را دریافت و ارسال میکند. همچنین ما دو خروجی ۳ پین داریم که توسط یک کابل به موتور ها متصل میشود و تغذیه و دیتای موتور ها را تامین میکند.

شکل (۲-۳) : تصویری از برد درایور سروو موتور های باس

۲-۲ شبیه سازی حرکت بازو

با توجه به اینکه پروژه ما دارای یک بازوی رباتیک است که قابل جابجایی در محیط و اعمال نیرو را دارد، لازم است که ابتدا قسمت نرم افزار و کنترل ربات در شبیه ساز تست شود و پس از اطمینان خاطر از عملکرد قسمت کنترلی، کد بر روی سخت افزار واقعی اجرا شده و در صورت نیاز اصلاحاتی انجام شده و مشکلاتی که در شبیه سازی قابل بررسی نبودند، بررسی و اصلاح شود.

ما در این پروژه نیاز داریم تا موقعیت End Effector بازو را در فضا مشخص و جابجا کنیم، به همین دلیل نیاز است تا سینماتیک معکوس بازو پیاده سازی شود و با کمک حل معادلات موقعیت موتور ها مشخص شود.

جهت پیاده سازی سینماتیک معکوس ربات از چند روش میتوان پیشرفت. روش کلاسیک تشکیل معادلات به صورت دستی و پیاده سازی آن در کد است. اما روش دیگر استفاده از فریمورک های آماده برای اینکار است که در ادامه آن را توضیح میدهیم.

فریمورک هایی مانند move it یا pybullet وجود دارند که میتوان به آنها فایل مدل شـده ربات با فرمت ufrd یا فرمت های مشـابه را ورودی داد و خود فریم ورک بر اسـاس فایل مدل شـده ربات معادلات آنرا بدسـت می آورد و آنها را حل میکند. فایل URDF یا Unified Robot Description Format در واقع یک فایل متنی (XML) اسـت که شـرح کامل هندسـه و ویژگی‌های مکانیکی ربات را به موتورهای شـبیه‌سـازی (PyBullet، Gazebo، MoveIt و ...) می‌دهد و شـامل لیسـت تمام لینک‌ها و مفصـل‌ها، مقیاس‌ها و مختصـات اجزای ربات، رنگ‌ها، متریال‌ها و ... اسـت. به دلیل اینکه ما از بازوی آماده SO-۱۰۱ اسـتفاده کردیم، فایل ufrd آن از قبل آماده بوده و ما از آن اسـتفاده کردیم. ما برای پیاده سـازی سینماتیک معکوس ربات و شبیه سازی آن از pybullet استفاده کردیم، چرا که نسبت به move it سبک تر، ساده تر و سریع تر است اما move it قابلیت های بسیار بیشتری دارد و در پروژه های صنعتی قابل استفاده است.

pybullet از روش حل عددی برای حل سـینماتیک معکوس ربات اسـتفاده میکند. الگوریتم اصـلی مورد استفاده، روش **Damped Least Squares (DLS)** بر پایه ماتریس ژاکوبین (Jacobian Matrix) است. این روش نسخه‌ای پایدارتر از روش نیوتن-رافسون است.

موقعیت نهایی (x) تابعی از زوایای مفاصل (θ) است:

$$x = f(\theta)$$

ما تغییرات کوچک در موقعیت (Δx) را به تغییرات کوچک در مفاصـل (Δθ) از طریق ماتریس ژاکوبین (J) مرتبط می‌کنیم:

$$\Delta x \approx J(\theta) \cdot \Delta\theta$$

در اینجا:

- Δx : بردار خطا (تفاوت بین موقعیت هدف و موقعیت فعلی).
- J(θ) : ماتریس ژاکوبین که مشتقات جزئی موقعیت نسبت به زوایاست.
- Δθ : تغییری که باید در مفاصل ایجاد کنیم.

مشکل ماتریس معکوس

برای پیدا کردن Δθ، به طور ایده آل باید معکوس J را محاسبه کنیم:

$$\Delta\theta = J^{-1}\Delta x$$

اما J اغلب مربعی نیست (تعداد مفاصل با درجات آزادی برابر نیست) و یا ممکن است "تکین" (Singular) باشد که معکوس‌پذیر نیست.

pybullet برای حل این مشکل و جلوگیری از پرش‌های ناگهانی در نزدیکی نقاط تکین (Singularities)، از فرمول DLS استفاده می‌کند. هدف مینیمم کردن تابع هزینه زیر است:

$$\| J\Delta\theta - \Delta x \|^2 + \lambda^2 \| \Delta\theta \|^2$$

پاسخ نهایی (فرمول محاسبه Δθ) به صورت زیر است:

$$\Delta\theta = J^T(JJ^T + \lambda^2 2I)^{-1}\vec{e}$$

که در این فرمول:

- Δθ : بردار تغییرات مورد نیاز برای مفاصل.
- J : ماتریس ژاکوبین در وضعیت فعلی.
- J^T : ترانهاده (Transpose) ماتریس ژاکوبین.
- λ : ضریب میرایی (Damping factor). این عدد غیرصفر باعث می‌شود در نقاط تکین، مخرج کسر صفر نشود و حرکت ربات نرم بماند (اما کمی دقت را در آن نقاط کاهش می‌دهد).
- I : ماتریس همانی (Identity Matrix).
- $\vec{e}$: بردار خطا ($\vec{x}_{target} - \vec{x}_{current}$)

۲-۳ پردازش تصویر

برای بخش دوربین ما نیاز به تشخیص ژست های حرکتی دست توسط دوربین و ضبط ویدیو با کمک دوربین داریم. برای بخش تشخیص ژست های حرکتی ما از کتابخانه mediapipe که توسط شرکت گوگل توسعه داده شده استفاده کردیم.

سیستم MediaPipe از چند pipeline یا «خط پردازش» تشکیل می‌شود که هر pipeline شامل تعدادی calculator است. هر calculator یک وظیفه خاص دارد (مانند تشخیص ناحیه دست، تخمین نقاط کلیدی و غیره).

برای تشخیص دست‌ها، pipeline معمولاً دو مرحله دارد:

۱. **Palm Detection** (تشخیص کف دست)
۲. **Hand Landmark Model** (استخراج مختصات دقیق مفاصل دست)

مرحله Palm Detection (مدل تشخیص کف):

مدل Palm Detection بر پایه شبکه عصبی BlazePalm کار می‌کند. هدف آن پیدا کردن مستطیل‌هایی است که ناحیه‌ی احتمالی کف دست را در تصویر مشخص می‌کنند. به‌جای اینکه مستقیماً کل دست را شناسایی کند، فقط کف دست را تشخیص می‌دهد چون کف از زاویه‌های مختلف بهتر قابل شناسایی است و نرخ خطا را پایین می‌آورد. خروجی این مرحله یک bounding box برای هر دست در تصویر است. این مدل روی ورودی تصویر RGB عمل کرده و معمولاً روی GPU یا CPU اجرا می‌شود.

مرحله Hand Landmark Model (مدل نقاط کلیدی):

در این بخش از مدل BlazeHand Landmark استفاده می‌شود که یک شبکه عصبی سبک است و بر اساس regression عمل می‌کند (مختصات دقیق نقاط را عددی تخمین می‌زند، نه طبقه‌بندی). ورودی این مدل تصویر بریده شده از ناحیه دست (از مرحله قبل) است و خروجی آن مختصات سه‌بعدی (x, y, z) برای ۲۱ نقطه اصلی دست شامل مچ، بندهای انگشتان و نوک انگشتان.

به طور مثال شناسه یکسری از نقاط دست به این صورت است:

- ۰ : مچ
- ۴ : نوک انگشت شست
- ۸ : نوک انگشت اشاره
- ۱۲ : نوک انگشت میانی
- ۱۶, ۲۰ : سایر انگشت‌ها

تخمین ژست (Gesture Recognition):

MediaPipe فقط نقاط را تحویل می‌دهد. اما برای تشخیص ژست باید با منطق یا مدل دیگری روی این نقاط کار کرد. دو رویکرد متداول برای تشخیص ژست وجود دارد، قوانین هندسی و استفاده از مدل یادگیری ماشین جداگانه.

قوانین هندسی:

با اندازه‌گیری زاویه بین انگشت‌ها، فاصله‌ی مفاصل و غیره، می‌توانید حالت‌هایی مثل " Thumbs“ ,”OK“ ,”Peace“ ,”Up" یا "Fist" را تشخیص دهید. به طور مثال اگر زاویه بین انگشت اشاره و شست کمتر از مقدار مشخصی بود علامت "OK" تشخیص داده میشود.

مدل یادگیری ماشین جداگانه:

مختصات (x, y, z) به عنوان ورودی به مدل (مثل SVM یا شبکه نورونی کوچک) داده می‌شود تا ژست را به‌صورت خودکار طبقه‌بندی کند. در این پروژه ما از این روش استفاده کرده ایم.

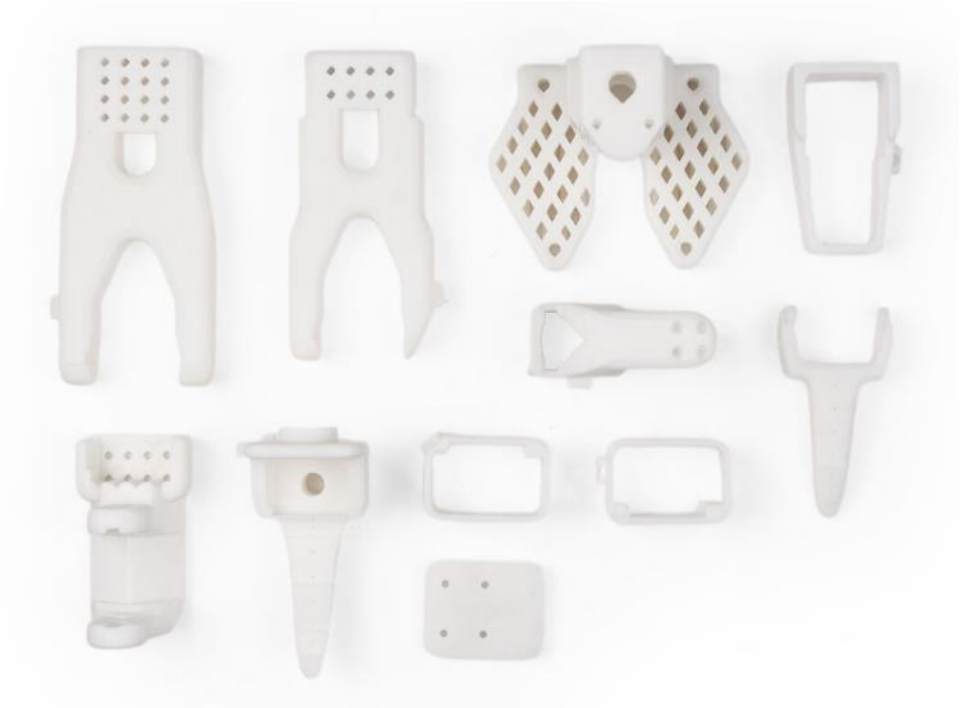
MediaPipe Hands خروجی زیر را به ما می‌دهد:

- multi_hand_landmarks: لیستی از ۲۱ نقطه برای هر دست
- multi_handedness: تشخیص چپ یا راست بودن دست
- multi_hand_world_landmarks : مختصات سه‌بعدی در فضای واقعی (نه فقط روی تصویر)

فصل سوم: ساخت دستگاه

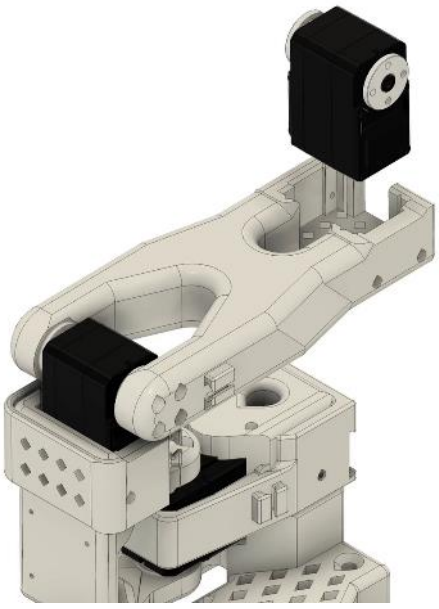
۳-۱ قسمت مکانیکی بازو

همانطور که در فصل گذشته اشاره شد، از بازوی رباتیک SO-101 جهت پیاده سازی مکانیک بازو استفاده شده، اما ۲ درجه آزادی از ربات جهت جابجایی اجسام استفاده میشد که به دلیل عدم کاربرد از ربات حذف شد. بخش های مختلف سازه ربات در سایت گیتهاب در دسترس هستند. با استفاده از این فایل ها بخش های مختلف سازه ربات را توسط پرینتر سه بعدی پرینت کردم.



شکل (۳-۱) : سازه پرینت شده بازو

مرحله بعدی مونتاژ بخش های مختلف و اتصال موتور ها بود که بر اساس راهنمای خود بازو انجام شد که در سایت هاگینگ فیس قابل دسترسی است.



شکل (۳-۲) : نمونه ای از راهنمای مونتاژ بازوی ربات SO-101

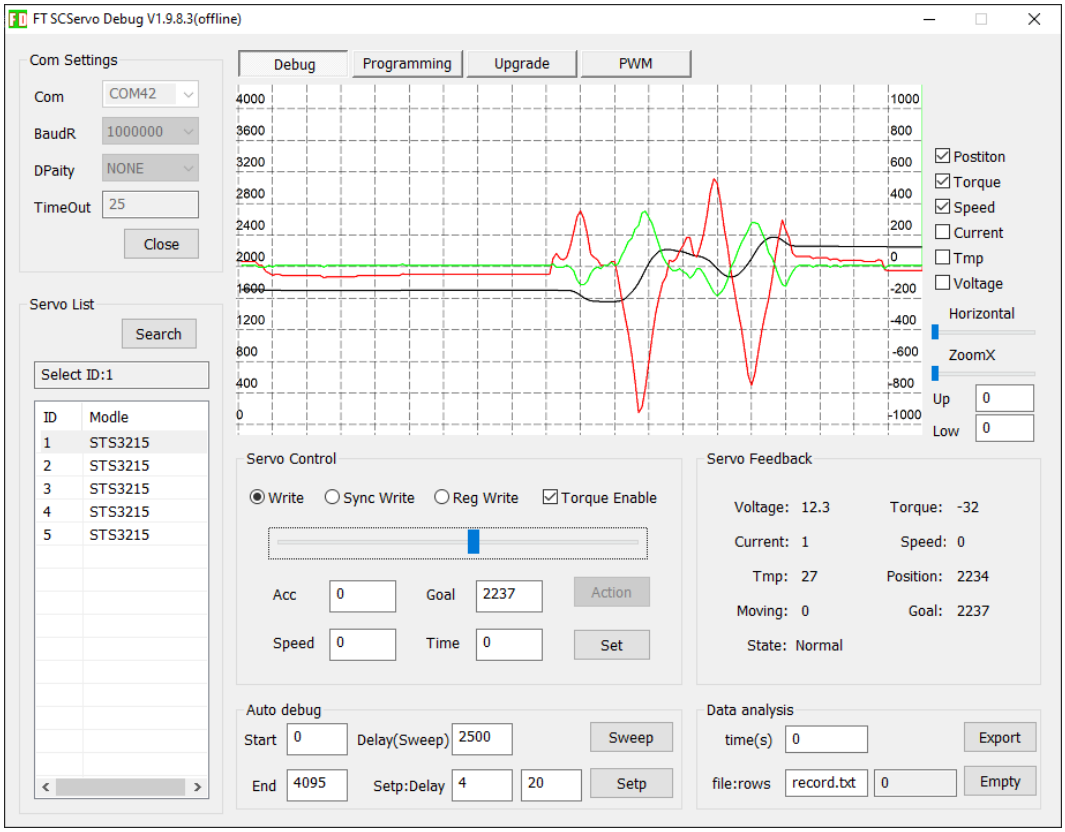
پس از مونتاژ موتور ها و تکمیل سازه مراحل از قبل آماده به پایان میرسد و بقیه مراحل بدون راهنمای مشخص و بر اساس پروژه نهایی انجام میشوند.

۳-۲ راه اندازی و تست موتور ها

موتور هایی که در این پروژه استفاده شده از نوع باس هستند و میتوان تا ۲۵۳ موتور را به یکدیگر متصل کرد و از طریق یک سیم دیتا تمامی آنها را کنترل کرد. برای اینکه بتوان موتور ها را به صورت جداگانه کنترل کرد، هر موتور قابلیت گرفتن یک شناسه (ID) به خصوص را دارد. برای تغییر شناسه موتور ها، هر موتور به صورت جداگانه توسط برد مبدل به کامپیوتر متصل شد و توسط نرم افزار FD که نرم افزار مخصوص شرکت *FEETECH* میباشد، شناسه آن تغییر داده شد. شماره گذاری موتور ها از ۱ تا ۶ انجام شد که به ترتیب از اولین موتور بعد از اتصال سازه به زمین شروع شده و تا آخرین موتور که برای End Effector استفاده میشود، شماره گذاری انجام شد.

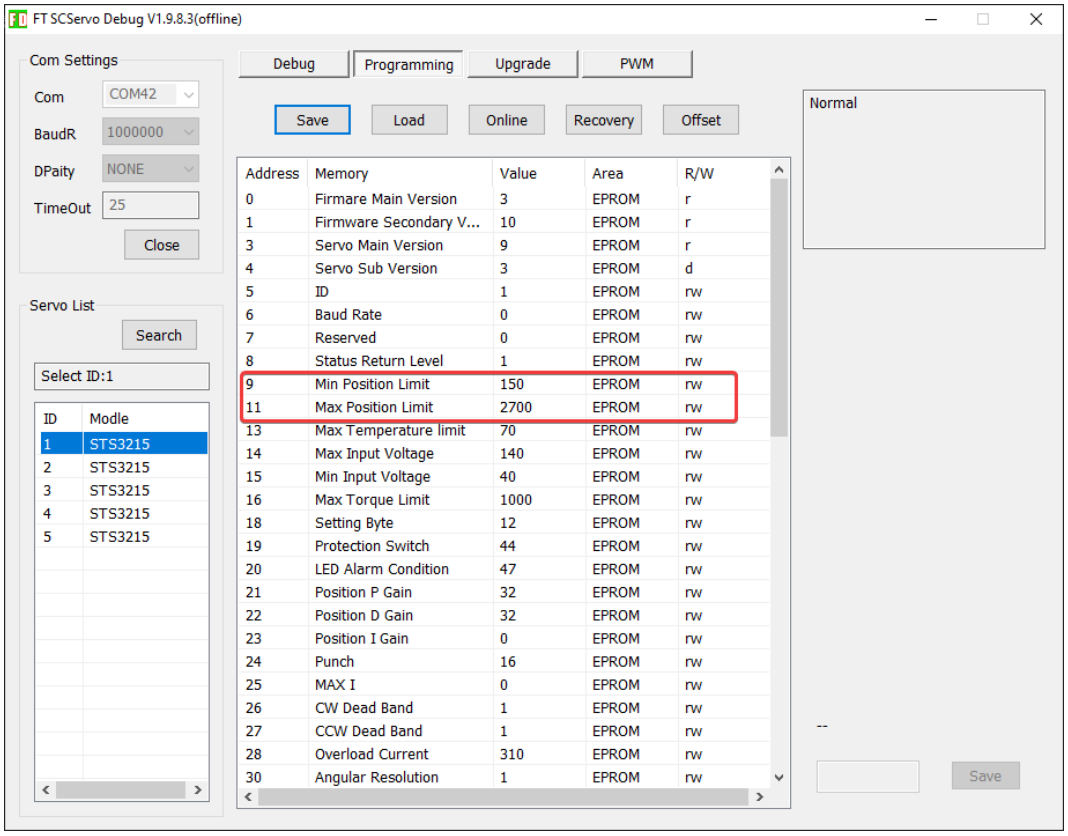
پس از آن موتور ها هر کدام جداگانه حرکت داده شد و از اتصالات سازه اطمینان حاصل شد. پس از کمی بررسی مشخص شد که به دلیل اینکه بازه حرکتی کامل موتور ها قابل دسترس نیست به خاطر محدودیت مکانیکی سازه اگر به موتور دستور داده شود که در آن محدوده حرکت کند (به اشتباه) فشار

زیادی به موتور ها وارد میشود و بهتر است که از این مشکل جلوگیری شود که در تست ها به موتور و سازه آسیبی نرسد.



شکل (۳-۳) : بررسی پارامتر های موتور در نرم افزار FD

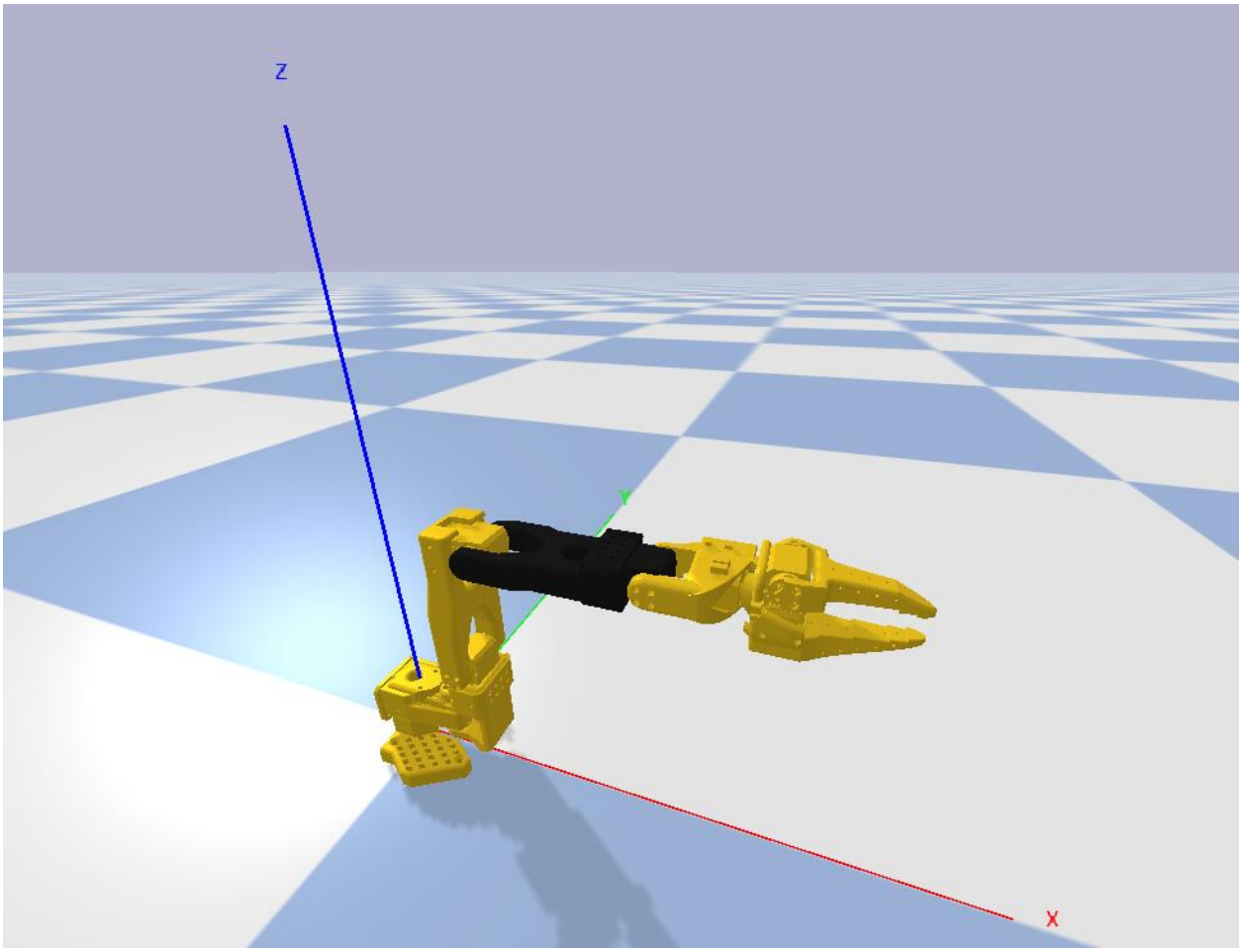
سروو موتور های STS3215 قابلیت این را دارا هستند که بازه حرکتی آنها به یک حداقل و حداکثر محدود شود تا در صورت فرمان اشتباه به سازه یا موتور آسیبی نرسد، به همین دلیل تمامی موتور ها به صورت جداگانه حرکت داده شدند و بازه حداقل و حداکثر اونها توسط نرم افزار FD بررسی شد و داخل پارامتر های هر موتور به صورت جداگانه وارد شد.



شکل (۳-۴) : تنظیم پارامتر های سروو موتور در نرم افزار FD

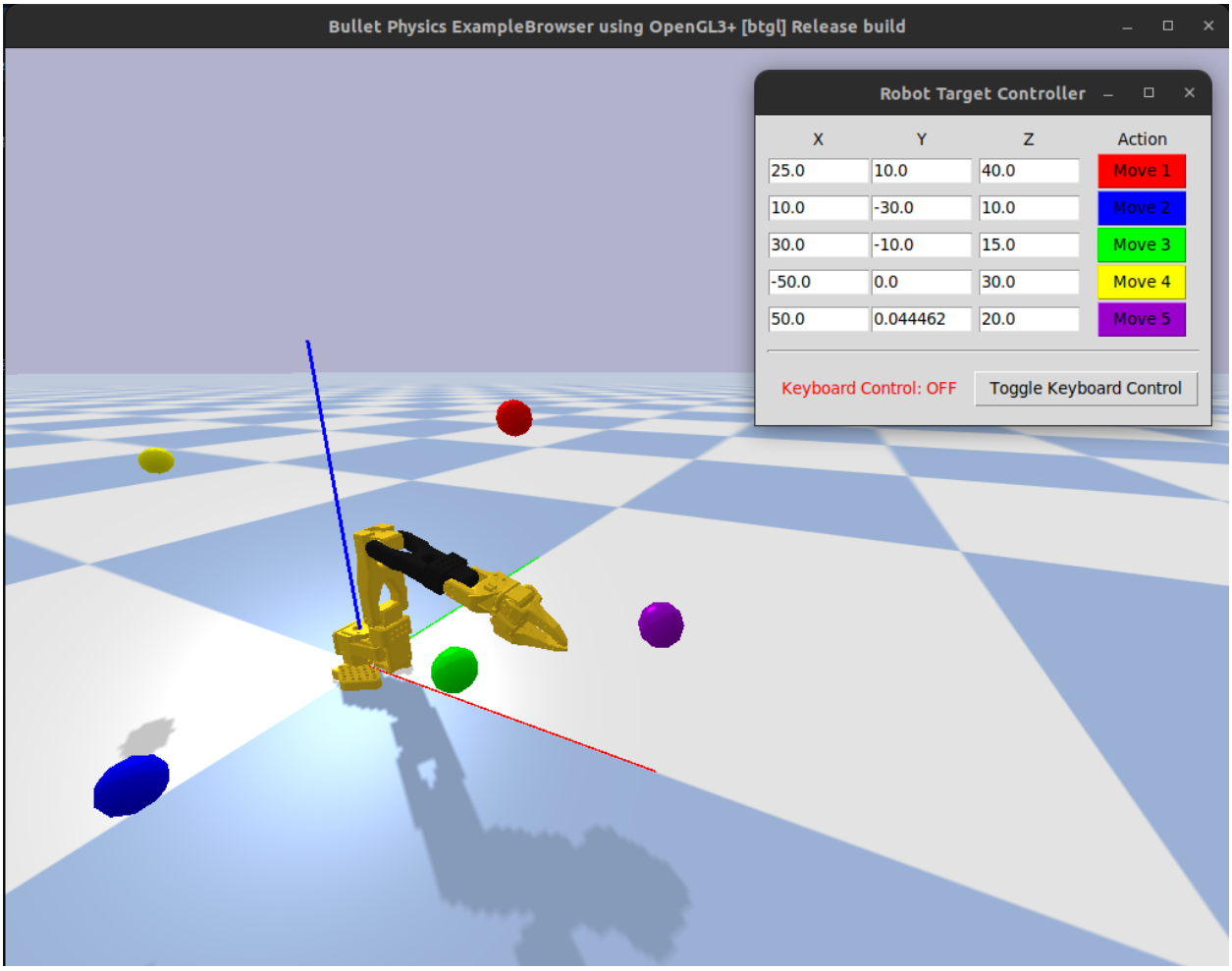
۳-۳ شبیه سازی بازو

پس از راه اندازی موتور ها و تست کامل حرکت سازه، نوبت به توسعه قسمت نرم افزاری پروژه میرسد. ابتدا آماده سازی و نصب pybullet در سیستم عامل لینوکس انجام شد و پس از آن بارگزاری مدل ufrd ربات و حرکت آن به صورت دستی در شبیه ساز و تست آن انجام شد.



شکل (۳-۵) : مدل سه بعدی بازو در شبیه ساز

برای بررسی عملکرد صحیح بخش سینماتیک معکوس، یک برنامه گرافیکی ساده طراحی شد که در آن بتوان چند نقطه را در فضا مشخص کرد و با انتخاب هر کدام، معادلات سینماتیک معکوس ربات حل شده و End Effector جابجا شود.



شکل (۳-۶) : تست حرکت ربات در فضا با کمک نقاط

پس از تســـت های اولیه نیاز به دو تغییر در ســـاختار ربات و کد بود که بازو با پروژه دوربین همخوانی داشته باشد، ابتدا نیاز به حذف ۲ درجه آزادی از انتهای ربات بود، به این دلیل که نیازی به گریپر نیست و نیازی به چرخش دوربین نیز در حال حاضـر نداریم، پس از آن هم نیاز بود تا بازو در هر زاویه ای نقطه ای که قرار میگیرد، End Effector همیشـه عمود بر سـطح زمین باقی بماند تا دوربین متصـل به آن دید خود را از دسـت ندهد. این کار با ورودی دادن یک آرایه که شـامل سـه زاویه x,y,z میشـود به تابع حل سینماتیک معکوس قابل انجام است.

در اینجا نیاز است که زاویه End Effector بر اساس کواترنیون (Quaternion) داده شود به جای زوایای اویلر.

زوایای اویلر (چرخش حول محورهای X، Y و Z) برای انسـان قابل فهم هسـتند اما یک مشـکل بزرگ به نام قفل گیمبال (Gimbal Lock) دارند. در این حالت، دو محور از ســـه محور دوران روی هم منطبق

می‌شـــوند و ربات یک درجه از آزادی حرکتی خود را از دســـت می‌دهد که باعث حرکات ناگهانی و غیرمنتظره می‌شـــود. کواترنیون‌ها این مشـــکل را ندارند و برای درون‌یابی (interpolation) بین دو جهت‌گیری بسیار نرم‌تر و پایدارتر عمل می‌کنند. برای فهم دقیق تر این موضوع ابتدا به بررسی کواترنیون میپردازیم.

یک کواترنیون یک عدد ۴ بعدی اسـت که به صـورت [x, y, z, w] نمایش داده می‌شـود و یک دوران در فضـای سـه‌بعدی را تعریف می‌کند. [x, y, z] بخش برداری است و محور دوران را نشـان می‌دهد و w نیز بخش اسکالر است و میزان دوران حول آن محور را نشان می‌دهد.

ما ســـه زاویه اویلر داریم که شـــامل Roll (چرخش حول محور X) ، Pitch (چرخش حول محور Y) و Yaw (چرخش حول محور Z) میشـــود. اکنون می‌خواهیم چهار مؤلفه کواترنیون(q=[w,x,y,z]) را پیدا کنیم. مراحل محاسبه به صورت زیر است:

ابتدا کسینوس و سینوسِ نصفِ هر یک از زوایا را محاسبه می‌کنیم:

$$c_r = cos(\phi/2) \qquad s_r = sin(\phi/2)$$
$$c_p = cos(\theta/2) \qquad s_p = sin(\theta/2)$$
$$c_y = cos(\psi/2) \qquad s_y = sin(\psi/2)$$

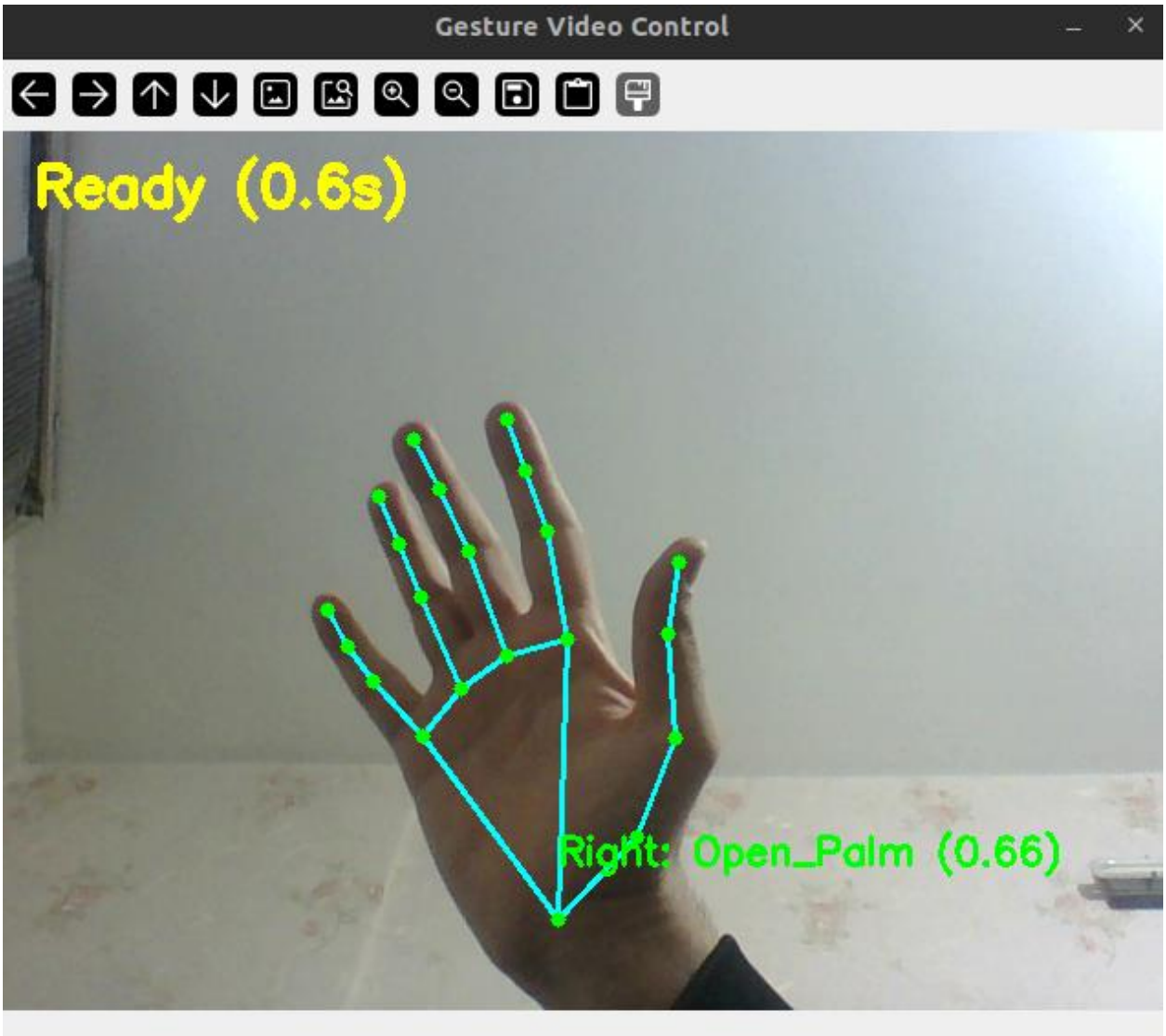
سپس این مقادیر را برای محاسبه مؤلفه‌های کواترنیون ترکیب می‌کنیم:

$$w = c_r \cdot c_p \cdot c_y + s_r \cdot s_p \cdot s_y$$
$$x = s_r \cdot c_p \cdot c_y - c_r \cdot s_p \cdot s_y$$
$$y = c_r \cdot s_p \cdot c_y + s_r \cdot c_p \cdot s_y$$
$$z = c_r \cdot c_p \cdot s_y - s_r \cdot s_p \cdot c_y$$

این فرمول‌ها نتیجه ضرب سه کواترنیون پایه (یکی برای هر محور دوران) هستند و یک دوران ترکیبی را در قالب یک کواترنیون واحد ارائه می‌دهند. این فرمول ها به کد پایتون تبدیل شـــده و در کد اســـتفاده شدند.

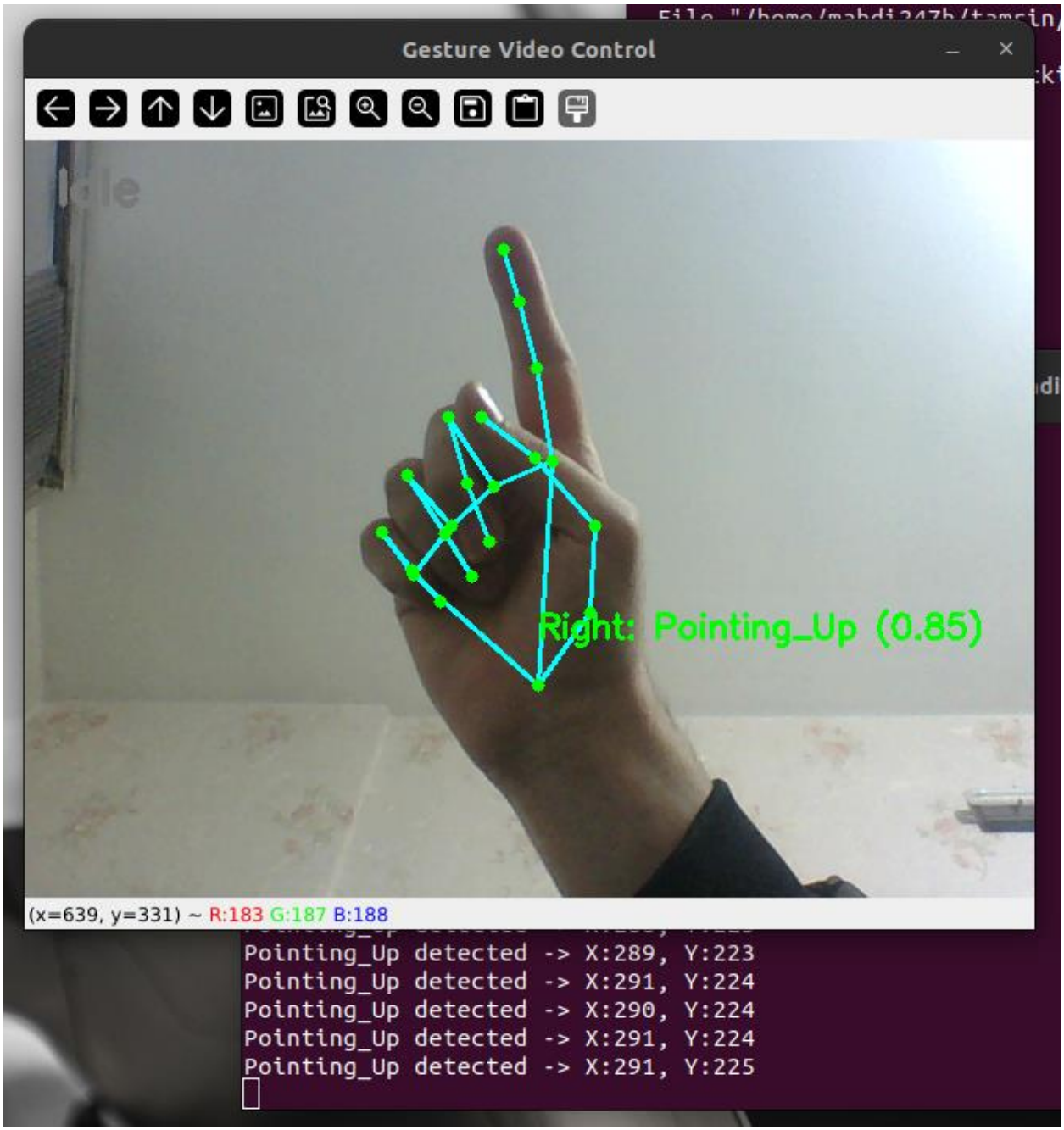
۳-۴ راه اندازی دوربین

پس از تشـخیص ژسـت های حرکتی نیاز به اجرای یک فیدبک پس از نمایان شـدن هر ژسـت داریم. در اینجا ما از ژست حرکتی دست باز و بسته برای شروع و توقف ضبط فیلم استفاده کردیم. نحوه کار به این صورت است که زمانی که دست بسته را تشخیص داد، یک تایمر شروع شده و بعد از ۱ ثانیه منتظر دست باز میماند و اگر دسـت باز تشـخیص داده شـد، در صـورتی که در حال ضـبط بود آن را متوقف میکند و در غیر این صـورت شـروع به ضـبط میکند. با کمک این روش از شـروع و توقف ناخواسـته ضـبط جلوگیری میکنیم.



شکل (۳-۷) : تشخیص ژست حرکتی دست باز و شروع تایمر

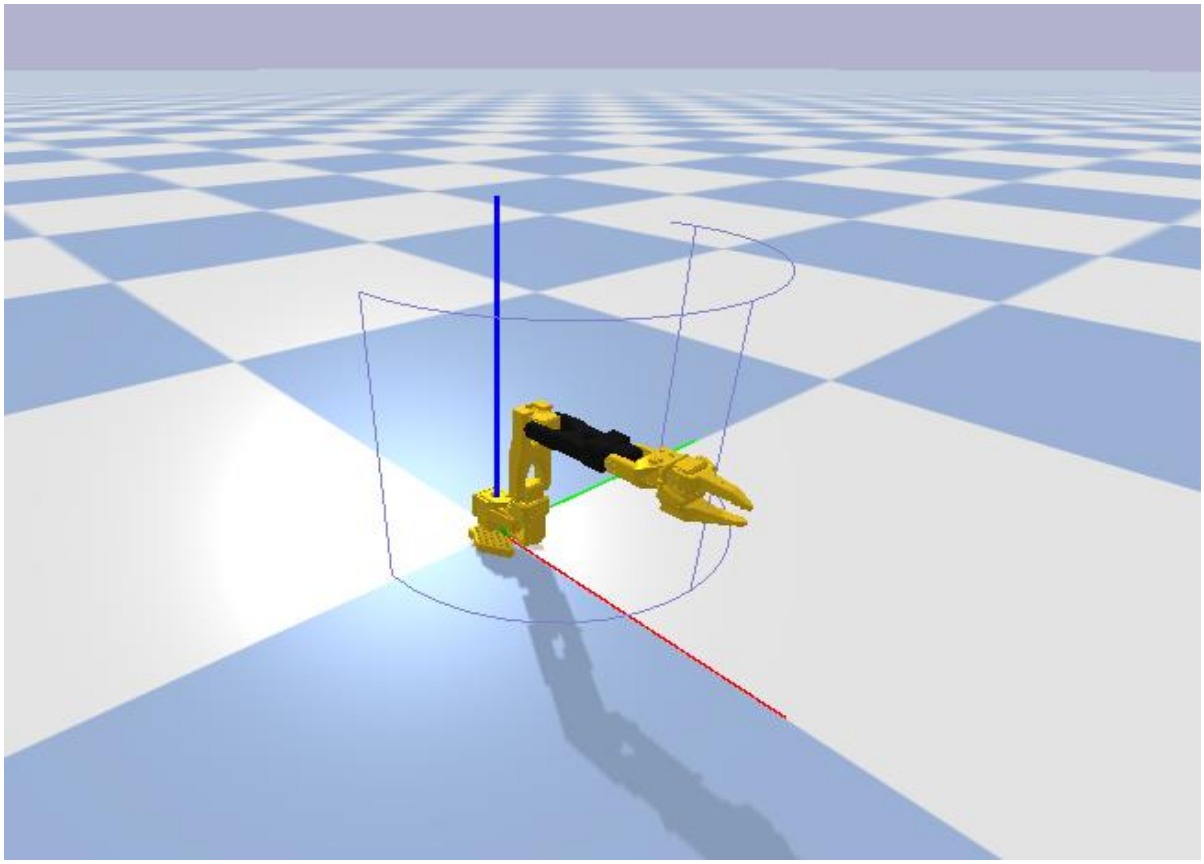
همچنین از ژست حرکتی "انگشت اشاره" برای حرکت دوربین استفاده می کنیم و با کمک کتابخانه MediaPipe زمانی که این ژست حرکتی تشخیص داده شد، موقعیت آن در صفحه را خروجی گرفته تا در مراحل بعدی بر اساس آن بازو را حرکت دهیم



شکل (۳-۸) : تشخیص موقعیت در زمانیکه ژست تشخیص داده شده

۳-۵ حرکت بازوی رباتیک بر اساس خروجی دوربین

برای حرکت بازو توسط دوربین ابتدا به صورت ساده موقعیت دست در خروجی دوربین به تابع سینماتیک معکوس بازو داده شد و بر اساس آن بازو حرکت داده شد، اما به دلیل اینکه دوربین محیط را به صورت ۲ بعدی بررسی میکند و موقعیت دست را به صورت ۲ بعدی تشخیص میدهد، زمانیکه ما این yوx را به برای حرکت در ۳ بعد خروجی میدادیم، بازه حرکت بازو بسیار محدود و معمولا اشتباه بود. پس از بررسی راه حل های مختلف به این نتیجه رسیدیم که باید برای بازو یک فضای کاری ۲ بعدی تعریف کنیم که End Effector فقط بر روی آن حرکت کند، همچنین برای اینکه بتوان از تمام درجات آزادی ربات استفاده کرد و بازه حرکتی ربات محدود نشود، این محدوده به صورت یک صفحه خمیده در جلوی ربات تعریف شد. پس از بررسی و تست مجدد، خروجی کاملا قابل قبول بود



شکل (۳-۹) : محدوده حرکتی ربات

فصل چهارم: جمع بندی

در این پروژه یک بازوی رباتیک هوشمند فیلم‌بردار با قابلیت ردیابی حرکات دست طراحی، ساخته و پیاده‌سازی شد. هدف اصلی، ایجاد یک سیستم تعاملی برای کنترل موقعیت و زاویه دید دوربین تنها از طریق حرکات دست بود؛ سیستمی که بتواند بدون نیاز به اپراتور، عملیات فیلم‌برداری را آغاز، متوقف و هدایت کند. برای دستیابی به این هدف، ابتدا بخش مکانیکی بازو با استفاده از مدل اپن‌سورس SO-101 ساخته و برای نیازهای پروژه بهینه شد. سپس بخش کنترلی شامل راه‌اندازی سروو موتورهای باس، تنظیم بازه‌های حرکتی و ایجاد ارتباط پایدار میان موتورها توسعه یافت.

در ادامه، بخش نرم‌افزاری پروژه با شبیه‌سازی کامل ربات در PyBullet پیش رفت و سینماتیک معکوس با روش‌های عددی مبتنی بر ماتریس ژاکوبین پیاده‌سازی شد تا بازو قادر به دنبال‌کردن نقاط هدف با دقت مناسب باشد. برای پردازش تصویر نیز از MediaPipe استفاده شد تا سیستم بتواند ژست‌های مختلف دست مانند دست باز، بسته و انگشت اشاره را تشخیص داده و بر اساس آنها فرمان‌های حرکتی لازم را تولید کند. یکی از چالش‌های اصلی، تبدیل مختصات دوبعدی دوربین به فضای سه‌بعدی ربات بود که با تعریف یک صفحه کاری خمیده در جلوی ربات و محدود کردن حرکت End Effector روی آن، به شکل پایدار و قابل اعتماد حل شد.

نتیجه نهایی پروژه، یک سیستم یکپارچه بود که می‌تواند موقعیت دوربین را بر اساس حرکت دست کنترل کند و فرآیند فیلم‌برداری را تنها با ژست‌های ساده مدیریت نماید.

منابع و مآخذ

[1] pixel robot arm, “camerabotics”, https://camerabotics.com/pixel

[2] feetech servo datasheet, “feetech”, https://www.feetechrc.com/525603.html

[3] Overview of Inverse Kinematics, “Luis Bermudez”, https://medium.com

[4] mediapipe , “joefernandez”, https://github.com/google-ai-edge/mediapipe