«به نام خدا»

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

مبانی مهندسی مکاترونیک

گزارش پروژه پایانی

عنوان:

مرتب و جداسازی بلوک‌های رنگی در خط تولید با استفاده از ربات UR10 بر مبنای پردازش تصویر

استاد: دکتر مهدی طالع ماسوله

نام و نام خانوادگی اعضای گروه:

آرمان برقی، محمدرضا تیموریان فرد، محمد مهدی رحیمی، امیرحسین دبیری اقدم، سیاوش شمس

مرداد ماه 1400

فهرست مطالب

عنوان **شماره صفحه**

چکیده

در این پروژه قصد داریم به وسیله ربات 6 درجه آزادی UR10 در محیط شبیه‌سازی CoppeliaSim عملیات Pick and Place مکعب‌های رنگی را انجام دهیم، در شبیه‌سازی مکعب‌ها با رنگ‌های تصادفی روی نوار نقاله قرار می‌گیرند و ربات، مکعب‌ها را بر اساس رنگ آن‌ها در جعبه مخصوص هر رنگ می‌اندازد. در ابتدای کار فضای کاری ربات در محیطی که در آن قرار دارد را به کمک نرم افزار CoppeliaSim و MATLAB بدست می‌آوریم. برای برداشتن مکعب‌های رنگی نیاز داریم تا مختصات مرکز آن‌ها را بدانیم، این کار به وسیله پردازش تصویر روی تصاویر بدست آمده از دوربین محیط شبیه‌سازی به کمک کتابخانه OpenCV در پایتون انجام می‌شود. برای کنترل ربات نیاز به حل معادلات سینماتیک معکوس داریم؛ برای این منظور معادلات مربوط به سینماتیک مستقیم را با توجه به پارامترهای DH ربات بدست می‌آوریم، با توجه به پیچیدگی روابط سینماتیک معکوس، محاسبات را به صورت عددی با استفاده از معادلات سینماتیک مستقیم انجام می‌دهیم. در نرم افزار MATLAB، کد پایتون مربوط به پردازش تصویر فراخوانده می‌شود و مختصات مکعب‌ها به عنوان خروجی کد پایتون در محیط MATALB بدست می‌آید، سپس با توجه به روابطی که بدست آوردیم و حل عددی با استفاده از الگوریتم Quasi-Newton، زاویه هر مفصل ربات جهت رفتن به موقعیت مورد نظر بدست می‌آید. در ادامه این زوایا به محیط CoppeliaSim ارسال شده و با استفاده از کنترل کننده PID، مجری نهایی[1] ربات به موقعیت مورد نظر می‌رود و عملیات Pick and Place انجام می‌شود.

کلمات کلیدی

ربات Pick & Place، ربات UR10، سینماتیک معکوس و مستقیم، پردازش تصویر با OpenCV، CoppeliaSim

[1] End Effector

۱ مقدمه

امروزه استفاده از ربات‌های برداشتن و گذاشتن[۲] در محیط کارخانه‌ها و تولیدی‌ها بسیار گسترش یافته است؛ استفاده از این ربات‌ها پروسه برداشتن و گذاشتن قطعات در مکانی دیگر را بسیار سرعت می‌بخشد و به این ترتیب باعث افزایش سرعت تولید می‌شود؛ به بیان دیگر این ربات‌ها اعمال تکراری که بعضاً برای انسان‌ها خسته کننده و طاقت فرسا هستند را با سرعت و دقت بیشتر انجام می‌دهند و نیز باعث کاهش هزینه‌ها و افزایش بازدهی می‌شود. این ربات‌ها معمولا دارای سیستم پردازش تصویر پیشرفته هستند که با استفاده از می‌توانند مکان، رنگ و دیگر ویژگی‌های قطعات مختلف را تشخیص داده و در کاربردهایی نظیر؛

سر هم کردن [۳] قطعات مختلف،

بسته بندی[۴] قطعات مختلف در یک جعبه یا قرار دادن آن‌ها روی پالت[۵]،

بررسی[۶] قطعات تولیدی و شناسایی قطعات معیوب و جدا کردن آن‌ها از بقیه تولیدات.

و ...

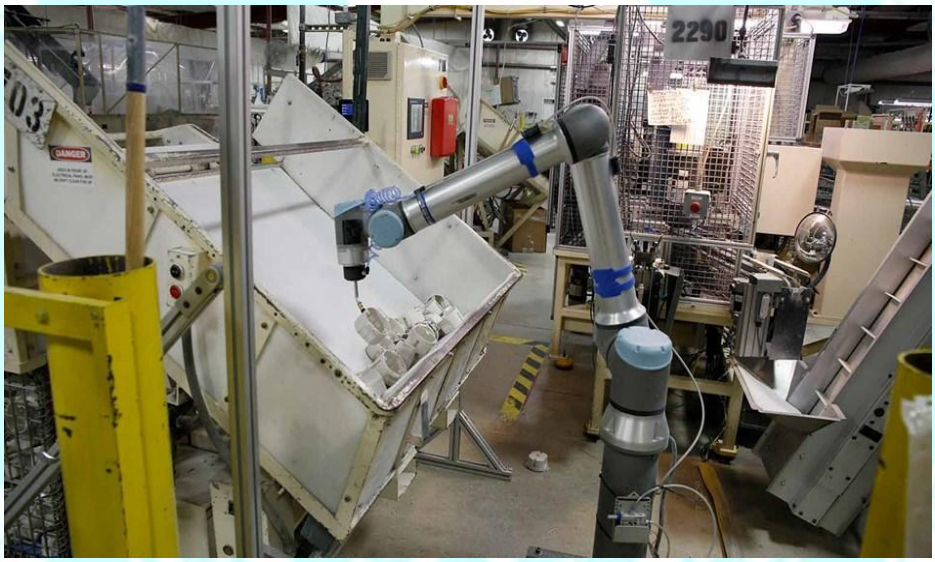
از آن‌ها استفاده کرد. انواع مختلف ربات‌ها برای کاربرد Pick and Place قابل استفاده هستند از جمله ربات‌هایی مثل: بازوی رباتیک[۷]، ربات دکارتی[۸]، ربات دلتا[۹]، ربات اسکارا[۱۰] که این ربات‌ها با توجه به کاربرد آن‌ها در اندازه‌های مختلف ساخته شده و مورد بهره برداری قرار می‌گیرند.

از میان ربات‌های مذکور، ربات‌های از نوع Robotic Arm به صورت گسترده در صنعت برای انجام عملیات-های مختلف از جمله عملیات Pick and Place استفاده می شود زیرا که یک ربات سری با 6 درجه آزادی است (3 درجه آزادی انتقالی و 3 درجه آزادی دورانی) که در عین سادگی کنترل آن، Payload قابل توجهی را نیز می‌تواند تحمل کند و همچنین فضای کمی را اشغال کرده ولی در عین حال فضای کاری بزرگی دارد، در مقابل ربات‌های موازی کنترل‌شان سخت‌تر و هزینه‌بر تر است و همچنین فضای بزرگ‌تری را اشغال می‌کنند و ... به همین دلایل نسبت به دیگر ربات‌ها، Robotic Armها در صنعت برای کاربرد مذکور متداول‌تر هستند.

---

[2] Pick and Place
[3] Assembly
[4] Packaging
[5] Palletizing
[6] Inspection
[7] Robotic Arm
[8] Cartesian Robot
[9] Delta Robot
[10] SCARA - Selective Compliance Articulated Robot Arm

شکل 1 - ربات **SCARA** در حال **Packaging** و **Palletizing** در خط تولید **IC**



شکل 2 - یک ربات **UR10** از شرکت **Universal Robots A/S** در حال انجام عملیات **Pick & Place**

در این پروژه نیز ما از یک ربات UR10 که از نوع Robotic Arm است استفاده کردیم و با استفاده از نرم‌افزار CoppeliaSim[11] نسخه Edu محیط خط تولید یک کارخانه را شبیه سازی کردیم که در این خط تولید قطعات مکعبی (با ابعاد کمی بزرگتر از مکعب روبیک استاندارد 3×3) با 3 رنگ مختلف (قرمز، آبی و سبز) روی یک نوار نقاله حرکت می‌کنند و ربات با پردازش تصویر، مکان و رنگ هر مکعب را تشخیص داده و با توجه به آن در جعبه مربوط به هر مکعب انداخته می‌شود؛ برای بخش پردازش تصویر از کتابخانه پرقدرت OpenCV نسخه Python استفاده کردیم و همچنین برای کنترل ربات و محاسبات مربوط به سینماتیک معکوس آن از نرم افزار MATLAB و به طور خاص از Robotics Toolbox توسعه داده شده توسط آقای Peter Croke [5] استفاده کردیم. رویه کلی کار به این ترتیب بود که MATLAB به صورت یک واسطه بین

[11] نرم افزار V-REP در نسخه‌های جدید با این نام عرضه شده است.

CoppeliaSim (محیط شبیه سازی) و Python (OpenCV) و جعبه ابزار[۱۲] Robotics (ابزارهای کنترل ربات) بود. همه این ابزارها و نرم افزارها در محیط ویندوز مورد بهره برداری قرار گرفته اند، البته به دلیل چند سکویی[۱۳] بودن به سادگی در سیستم عامل‌های دیگر نیز قابل استفاده و بهره برداری می‌باشند.

برای رسیدن به هدفمان در این پروژه نیاز بود که الگوریتم‌های مختلف پردازش تصویر که به تفضیل در بخش بعدی توضیحات آن آمده است را برای شناسایی رنگ، موقعیت مکانی و نیز دورانی مکعب‌ها روی نوار نقاله استفاده کنیم؛ همچنین برای کنترل حرکت ربات و مسیریابی[۱۴] نیاز به معادلات سینماتیک مستقیم و معکوس بود که با استفاده از پارامترهای DH[15] ربات UR10 بود. برای کنترل مکان (زاویه) مفاصل[۱۶] ربات از کنترل کننده PID استفاده کردیم و به طور خاص به دلیل نبود نویز و ایده‌آل بودن محیط شبیه سازی، در این پروژه استفاده از یک کنترل کننده P کافی بود؛ توجه داریم که اکثر تکنیک‌ها و الگوریتم‌های مورد استفاده در این پروژه را در کلاس در طول ترم آموخته بودیم و در این پروژه کاربرد عملی آموخته‌های تئوری مان را مشاهده کردیم.

در ادامه جزئیات مربوط به آماده سازی محیط شبیه سازی، یافتن فضای کاری ربات، پردازش تصویر و یافتن موقعیت و دوران مکعب‌ها و نیز مدل سینماتیکی ربات و کنترل آن در بخش بعدی آمده است.

---

[12] Toolbox
[13] Cross-Platform
[14] Path and motion planning
[15] Denavit–Hartenberg parameters
[16] Joints

۲ شرح و توصیف گزارش

2.1 محیط شبیه سازی:

یکی از بخش‌های اصلی پروژه محیط شبیه سازی می‌باشد و برای شروع کار ابتدا باید محیط شبیه سازی آماده شود. این محیط از اجزای مختلفی تشکیل شده است که ابتدا اجزای آن را نام برده سپس هرکدام را به صورت مختصر توضیح می‌دهیم؛ محیط نهایی طراحی شده به شکل زیر است.

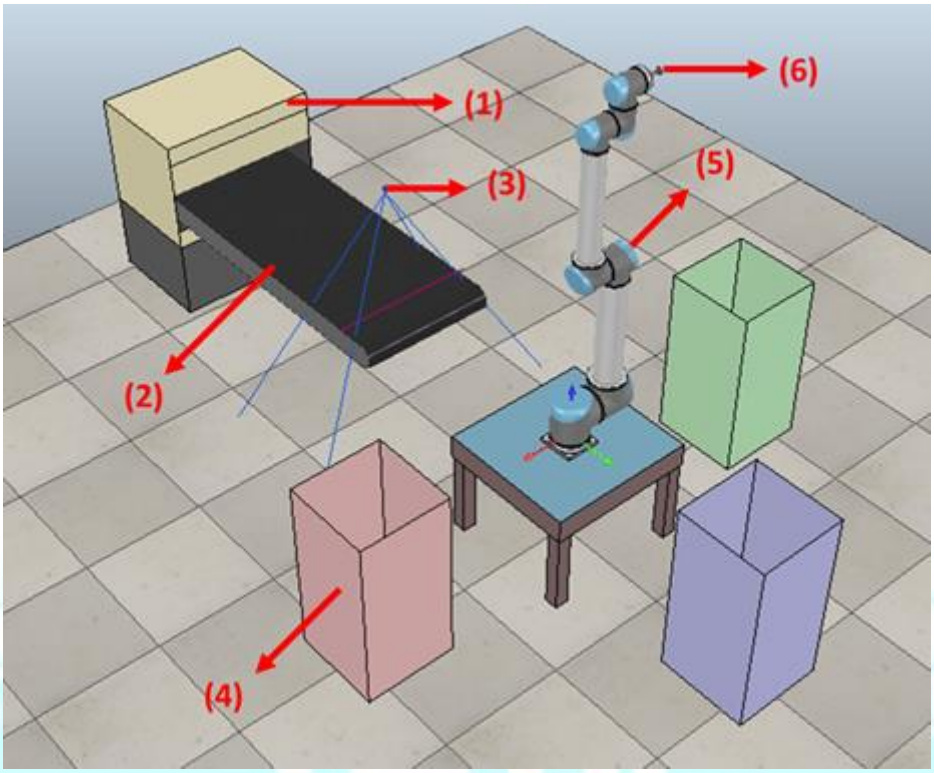
اجزای تشکیل دهنده محیط به شرح زیر است:

1. Part Producer: این قسمت از محیط وظیفه تولید مکعب‌ها را دارد.
2. Conveyor: این قسمت نوار نقاله‌ای می‌باشد که مکعب‌ها را به سمت ربات جا به جا می‌کند.
3. Vision Sensor: دوربینی می‌باشد که برای مشاهده مکعب‌ها و تشخیص رنگ آن‌ها استفاده می‌شود.
4. Deletion Box: جعبه‌هایی که قرار است مکعب‌ها به تفکیک رنگ در آن‌ها ریخته شود.
5. UR10: ربات استفاده شده در این پروژه برای جدا سازی مکعب‌ها.
6. Baxter Vacuum Cup: گریپری[۱۷] می‌باشد که برای نگه داشتن مکعب‌ها استفاده شده است.

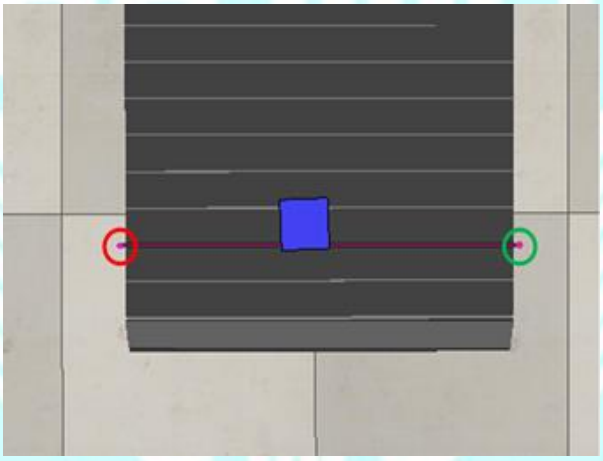
هر کدام از بخش های ذکر شده را به تفصیل بررسی می‌کنیم:

1. Part Producer: این قسمت با توجه به الگوریتمی که دارد مکعب‌هایی در ابعاد یکسان اما با سه رنگ مختلف قرمز، آبی و سبز تولید می‌کند و موقعیت‌های مختلف روی نوار نقاله و انتخاب رنگ و موقعیت به صورت تصادفی می‌باشد. با توجه به اینکه ممکن است در این فرایند 3 مکعب یا بیشتر دارای رنگ یکسان پشت سر هم تولید شود الگوریتم به شکلی است که حداکثر 2 رنگ تکراری پشت سر هم تولید شود و دلیل آن صرفا این است که بتوان عملکرد ربات برای همه رنگ‌ها را در فرایند جداسازی مشاهده و بررسی کرد. در جلوی نوار نقاله، نقطه ای که مکعب‌ها متوقف می‌شوند تا ربات آن‌ها را بردارد حسگری وجود دارد که زمانی که مکعبی در جلوی حسگر می‌باشد تولید مکعب‌ها متوقف شود. همچنین می‌توان سرعت تولید قطعات مکعب را از طریق تغییر متغیر shapeDropFrequency، عوض کرد.

[17] Gripper

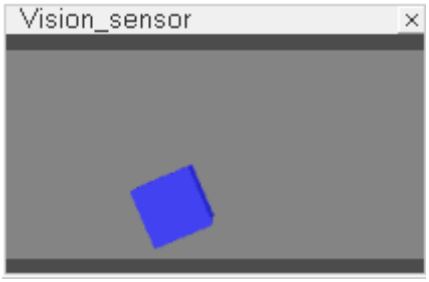


شکل 3 - محیط نهایی شبیه سازی



شکل 4 - سنسور های مربوط به نوار نقاله و **Part Producer**

2. Conveyor: نوار نقاله‌ای می‌باشد که مکعب‌های تولید شده روی آن قرار می‌گیرد و با یک سرعت مشخصی که قابل تغییر است به سمت ربات حرکت می‌کند. ابعاد آن نیز قابل تغییر می‌باشد. در سمتی از نوار نقاله که به ربات نزدیک است حسگری به موازات حسگر قسمت قبل وجود دارد که با قرار گرفتن یک مکعب در جلوی آن سرعت نوار نقاله را صفر می‌کند تا موقعیت مکعب ثابت بماند. در شکل زیر مکعبی جلوی حسگرها قرار گرفته است که در این حالت حسگرها در حالت چشمک زن قرار می‌گیرند و در شکل زیر حسگری که با رنگ قرمز نشان داده شده است مربوط به قسمت Part Producer و حسگری که با رنگ سبز نشان داده شده است مربوط به نوار نقاله می‌باشد.



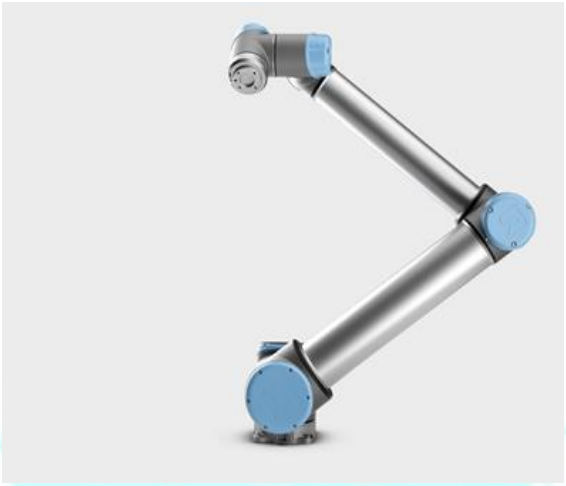
شکل 5 - نمونه تصویر مشاهده شده توسط **Vision Sensor**

شکل 6 - شمارنده ها

3. Vision Sensor: حسگری می‌باشد که در محلی قرار گرفته است که مکعب‌ها پس از متوقف شدن نوار نقاله در دید آن باشند و تصویر آن توسط MALTAB دخیره شده و سپس به وسیله OpenCV مختصات و رنگ آن را تشخیص می‌دهیم. کیفیت این حسگر قابل تغییر است و در این پروژه کیفیت 256×128 در نظر گرفتیم. نمونه‌ای از تصویری که این حسگر می‌بیند که می‌توان آن را در محیط نرم افزار مشاهده کرد در زیر آمده است.

4. Deletion Box: جعبه‌هایی هستند که برای جداسازی مکعب‌ها می‌باشند. ابتدا باید رنگ مکعب تشخیص داده شود و سپس ربات به سمت جعبه مربوطه هدایت می‌شود. سه جعبه برای رنگ‌های سبز و قرمز و آبی می‌باشد که رنگ هر جعبه نیز به همان رنگ می‌باشد. در هر جعبه یک حسگر مجاورت وجود دارد که با انداختن مکعب در آن یک عدد به شمارشگر اضافه می‌شود. در شکل زیر می‌توان نمونه‌ای از شمارشگر را مشاهده کرد که پس از مدتی از شبیه سازی و جداسازی مکعب‌ها اعدادی مانند شکل زیر نمایش داده می‌شود.

5. UR10: ربات استفاده شده در این پروژه می‌باشد که نمونه شبیه‌سازی آن در نرم افزار موجود می‌باشد. این ربات 6 محوره می‌باشد و گریپر آن نیز قابل تغییر می‌باشد. نمونه واقعی این ربات نیز موجود می باشد که وزن آن 33.5 کیلوگرم و payload آن نیز 10 کیلوگرم بوده و range آن 1300 میلی‌متر و دقت آن نیز 0.1 میلی‌متر می‌باشد که در شکل زیر تصویر این ربات در واقعیت دیده می‌شود.

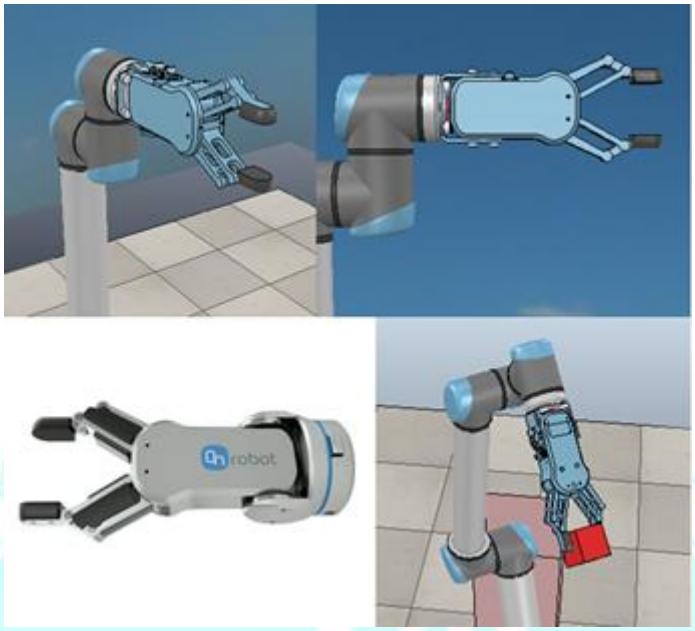


شکل 7 - تصویر ربات **UR10** در واقعیت

شکل 8 - تصویر گریپر **Baxter Vacuum Cup** و نمونه ای از این گریپر در واقعیت

6. Baxter Vacuum Cup: نوعی گریپر می‌باشد که با استفاده از مکش هوا می‌تواند جسم را نگه داشته تا ربات آن را بلند کند. مزیت استفاده از این گریپر این است که دوران جسم حول محور عمود بر سطح آن اهمیتی ندارد. اگر جسم مورد نظر سطحی مناسب داشته باشد با چسبیدن گریپر می‌توان جسم را بلند کرد. در تصویر زیر نیز گریپر استفاده شده و نمونه ای از آن در واقعیت مشاهده می‌شود.

توجه: این گریپر استفاده شده قابل تغییر به گریپرهای دیگر نیز می‌باشد ولی باید تغییراتی در کد برنامه متناسب با گریپر مورد استفاده ایجاد کرد؛ با توجه به نیاز می‌توان از گریپرهای دیگر نیز استفاده کرد زیرا ممکن است در عمل جسمی که قرار است جابجا شود توسط مکنده قابل جابجایی نباشد مثلا دارای پرز باشد یا ... به همین دلیل برای مثال یک نسخه دیگری از پروژه به همراه گریپر دیگری به نام RG2 وجود دارد که تصویر آن به شکل زیر است.



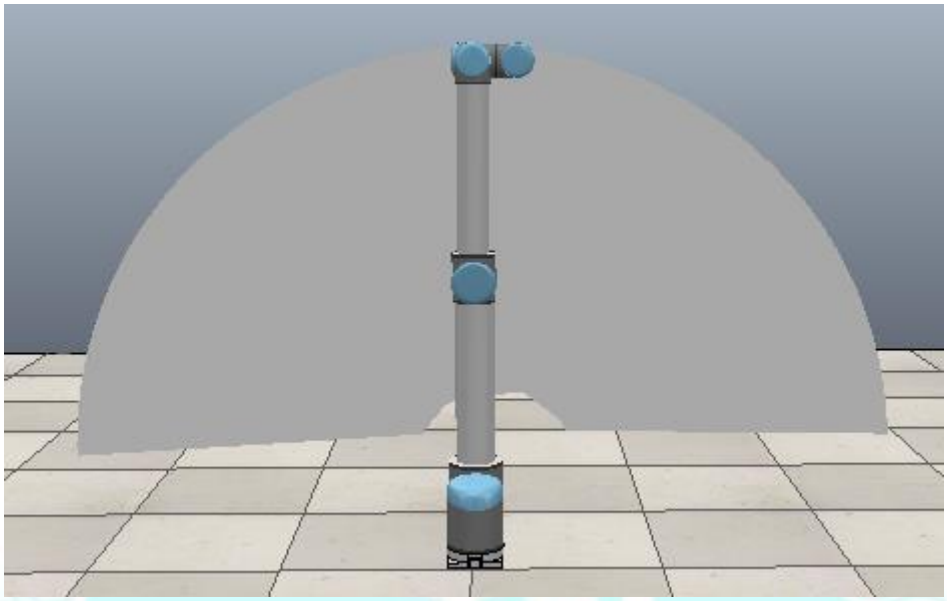
شکل 9 - تصویر گریپر **RG2** در محیط شبیه سازی و نمونه واقعی آن

برای استفاده از این گریپر چالشی که داریم باید بتوانیم میزان دوران جسم را تشخیص دهیم تا گریپر بتواند به شکل صحیح جسم را بردارد که سعی کردیم تا حدی این کار انجام شود اما بعضاً دقت کافی ندارد به همین دلیل این گریپر نسبت به گریپر مکنده قبلی دقت کمتری دارد؛ البته توجه داریم که اضافه کردن این قابلیت از اهداف این پروژه نبود و صرفا برای نشان دادن امکانپذیر بودن استفاده از ربات در کاربردهای مختلف و متناسب با نیاز با حداقل تغییر یک نسخه از پروژه با این گریپر نیز تهیه کردیم.

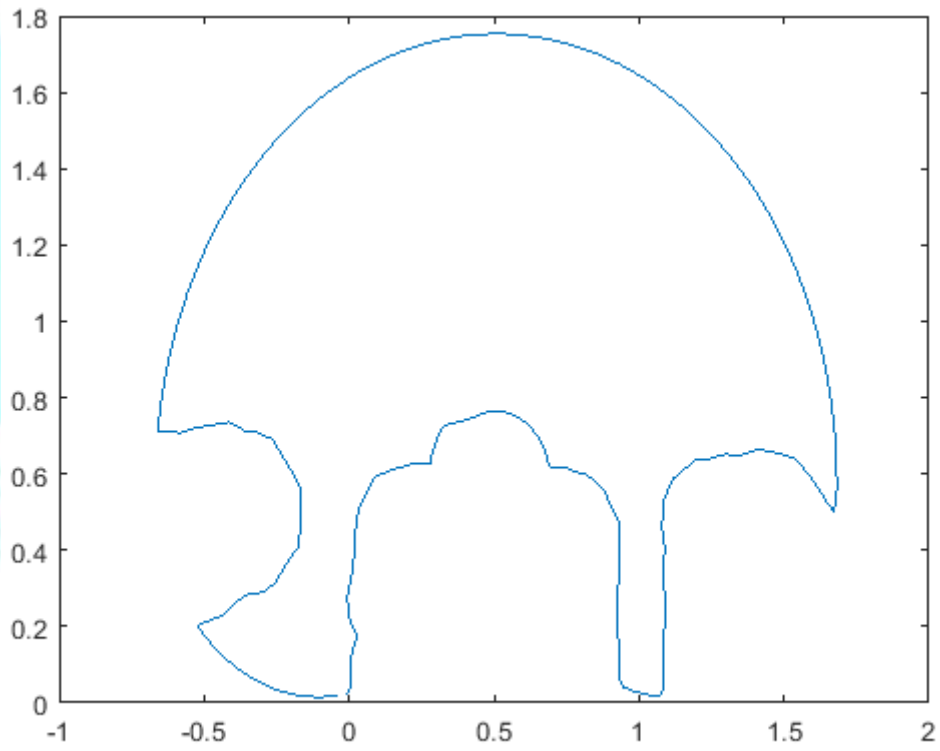
2.2 فضای کاری ربات:

یکی از قسمت‌های مهم برای کنترل ربات، دانستن فضای کاری ربات می‌باشد. به این صورت که برای انجام عملیات Pick and Place نیاز داریم تا بدانیم گریپر ربات به چه موقعیت‌هایی می‌تواند برود تا به درستی بتوانیم به آن دستورهای لازم را بدهیم.

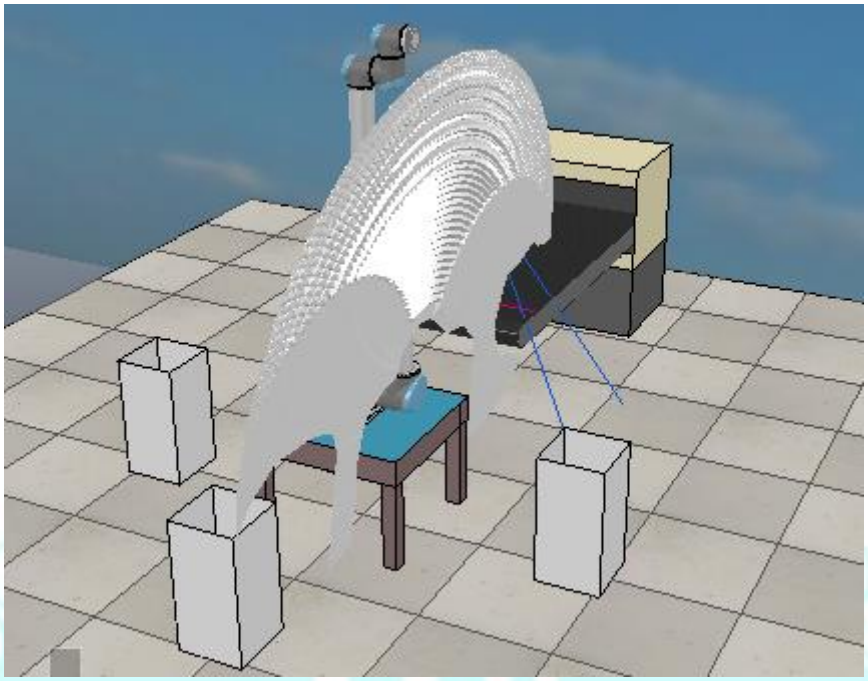
برای این منظور ربات را در دو حالت تنها روی زمین و در محیط کاری بررسی کردیم. فضای کاری ربات با استفاده از شبیه سازی در محیط CoppeliaSim و MATLAB به دست می‌آید که در بخش بحث و بررسی به جزئیات شیوه بدست آوردن آن می‌پردازیم.



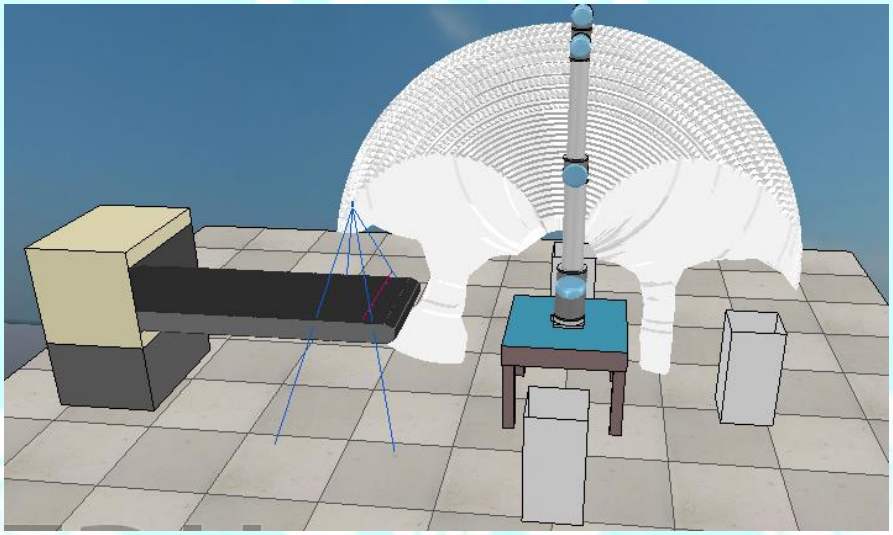
شکل 10 - ربات تنها و فضای کاری آن در محیط شبیه سازی



شکل 11 - فضای کاری ربات در محیط انجام عملیات **Place & Pick**



شکل 12 - تصویر ربات در محیط عملیات و فضای کاری آن



شکل 13 - تصویر ربات در محیط عملیات و فضای کاری آن

در تصاویر فوق قسمت‌های سفید فضای کاری ربات هستند و گریپر ربات می‌تواند در این مکان‌ها قرار بگیرد بدون اینکه ربات با اشیا دیگر برخورد ناخواسته داشته باشد.

2.3 پردازش تصویر:

با استفاده از ممان‌ها[18]ی یک تصویر باینری، می‌توانیم مختصات مرکزی آنرا بدست آوریم. ممان M_{10} برای یکی ماتریس باینری، معادل است با جمع مختصات در راستای x تمام درایه‌های غیر صفر. به طور مشابه ممان M_{01} برای یک ماتریس باینری، معادل است با جمع مختصات در راستای y تمام درایه‌های غیر صفر؛ پس با تقسیم کردن این ممان‌ها بر تعداد نقاط، می‌توانیم میانگین مختصات همه نقاط را بدست آوریم. پس نتیجه می‌شود که این مختصات بدست آمده، مرکز ثقل تصویر است که از میانگین مختصات تمام نقاط آن بدست خواهد آمد که از رابطه زیر بدست می‌آید:

$$\bar{x} = \frac{M_{10}}{M_{00}} \; ; \bar{y} = \frac{M_{01}}{M_{00}}$$

ممان‌های تا مرتبه 3 از طریق cv:Moments قابل دسترسی هستند. بدین ترتیب نیاز است که ابتدا با اعمال یک سری فیلتر روی عکس اصلی، آن را به عکس باینری تبدیل کنیم تا بتوانیم ممان‌های آن را استخراج کنیم و در نتیجه به مشخصات دیگر آن مانند مرکز ثقل دسترسی داشته باشیم.

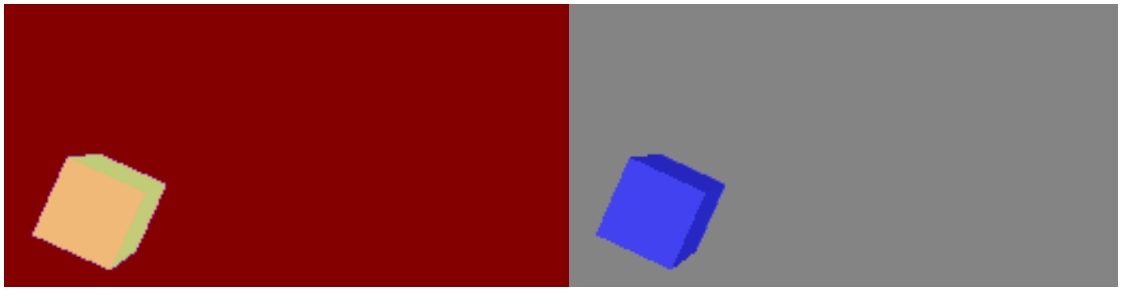
2.3.2 فضای رنگی[19] و تشخیص رنگ خاص:

فضای رنگی RGB، برای تشخیص رنگ مناسب نیست زیرا پارامترهایی نظیر شدت رنگ و میزان روشنایی را در نظر نمی‌گیرد. هرچند که در محیط شبیه سازی چون میزان روشنایی ثابت است می‌تواند بازه رنگی را در فضای رنگی RGB بدست آورد و در نهایت جسم را تشخیص داد، اما در واقعیت به دلیل یکسان نبودن همیشگی میزان روشنایی و همچنین یکسان نبودن شدت رنگ برای تمام مکعب‌ها، پیدا کردن این بازه مشکل است. به همین دلیل از فضای رنگی HSV استفاده می‌کنیم که هر دو پارامتر ذکر شده را داراست و پیدا کردن بازه رنگی برای آن کار راحت تری است. تبدیل فضای رنگی با استفاده از cv.cvtColor امکان پذیر است.

حال با استفاده از متود cv.inRange و دادن بازه‌های مشخص شده به آن، یک ماتریس باینری در اختیار داریم که پیکسل های با رنگ مورد نظر، مقدار 1 و مابقی پیکسل ها مقدار 0 را دارند. بدین ترتیب جسم با رنگ مورد نظر تشخیص داده خواهد شد.

[18] Moments (توضیحات در قسمت ضمیمه)

[19] Color Space

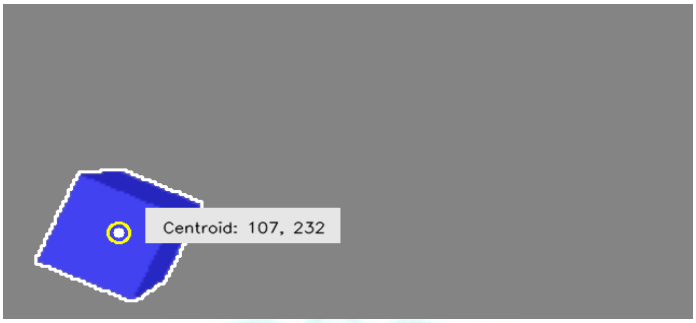


شکل 14 - نمونه عکس گرفته شده توسط دوربین و خروجی آن در فضای رنگی **HSV**

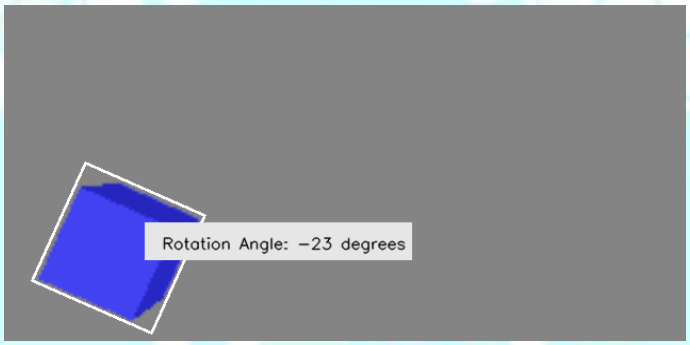
شکل 15 - عکس فیلتر شده باینری با بازه های رنگ آبی

2.3.3 فیلترها:

مورد دیگری که برای بهبود عکس باینری می‌توان انجام داد، استفاده از متودهای Morphological Transformations می‌باشد. این تبدیلات، عملگرهای ساده‌ای هستند که روی عکس دو بعدی باینری اعمال می‌شوند و یک کرنل دارند که عملکرد را تبیین می‌کند. هرچند که در محیط شبیه سازی که نویزی در گرفتن عکس در نظر گرفته نشده، این متود زیاد کارساز نیست، اما در واقعیت برای از بین بردن یک سری نویزهای موجود در تصویر نیاز است که از این متود استفاده کنیم. دو عملگر پایه از این متود، Dilation و Erosion نام دارند. در این بخش از عملگر Dilation استفاده می‌کنیم که هرچند تاثیر خاصی در خروجی ندارد، اما گذاشتن آن که پردازش تصویر را مطابق واقعیت کند، بدون ضرر نیست. این عملگر بدین صورت است که یک کرنل با سایز مشخص را روی تمام پیکسل های تصویر اعمال می کند، به عبارتی کرنل روی تصویر کانوالو می‌شود. اگر در همسایگی یک پیکسل خاص (که این همسایگی به اندازه‌ی سایز کرنل است) حتی یک پیکسل با مقدار یک وجود داشته باشد، آن پیکسل خاص مقدارش یک خواهد شد. لذا هنگامی که در تصویر به علت نویز، بعضی از گوشه‌ها و یا حتی نقطه‌های درون تصویر را نداشته باشیم، با این روش می‌توانیم این نویزها را از بین ببریم و یک عکس یکدست داشته باشیم. بدین ترتیب یک عکس باینری در اختیار داریم که در ناحیه ای که جسم رنگی قرار دارد، پیکسل‌ها مقدار 1 دارند و در مابقی نواحی پیکسل‌ها مقدار 0 دارند.



شکل 16 - کانتور مشخص شده و مرکز ثقل جسم



شکل 17 - مستطیل با مینیمم مساحت شامل جسم و تشخیص چرخش چسم

2.3.4 کانتورها[20]:

ممکن است در یک تصویر چندین مکعب همرنگ قرار گیرد. کانتورها مختصات‌های مرزی نقاط بهم پیوسته هستند که باعث می‌شود بتوانیم در یک تصویر باینری اشکال را تشخیص دهیم و برای این کار از متود cv.findContours استفاده می‌کنیم. با استفاده از توضیحات بخش مربوط به ممان‌ها، ابتدا کانتورهایی را که مساحت قابل قبولی دارند که تعیین می‌کند که این کانتور مکعب مورد نظر است، در نظر می‌گیریم. مرکز ثقل هر کدام از کانتورهای تشخیص داده شده را بدست می‌آوریم تا بتوانیم مختصات مرکز را داشته باشیم. لذا با داشتن مختصات این کانتورها، نزدیک‌ترین مکعب را به عنوان مکعب تشخیص داده شده انتخاب می‌کنیم. بدین ترتیب مختصات نزدیک‌ترین مکعب با رنگ مشخص شده را توانستیم بدست آوریم.

2.3.5 تشخیص زاویه چرخش جسم:

با استفاده از متود cv.minAeraRect می‌توانیم به راحتی مستطیلی با کمینه مساحت پیدا کنیم که مجموعه نقاط را شامل می‌شود. این مستطیل چون مینیمم مساحت را دارد، لذا می‌تواند دارای چرخش باشد که با تشخیص این چرخش می‌توانیم چرخش مکعب‌ها را حدس بزنیم. با استفاده از متود cv.boxPoints، به مختصات چهار گوشه مستطیل دسترسی خواهیم داشت. با داشتن مختصات رئوس این مستطیل، می‌توان زاویه چرخش را از طریق اعمال مثلثاتی بدست آورد.

[20] Contours

جدول 1 - پارامترهای **DH** ربات **UR10**

i	a_i	b_i	α_i	θ_i
1	0	0.1273	$\frac{\pi}{2}$	θ_1
2	0.612	0	0	θ_2
3	0.5723	0	0	θ_3
4	0	0.163941	$\frac{\pi}{2}$	θ_4
5	0	0.1157	$-\frac{\pi}{2}$	θ_5
6	0	0.0922	0	θ_6

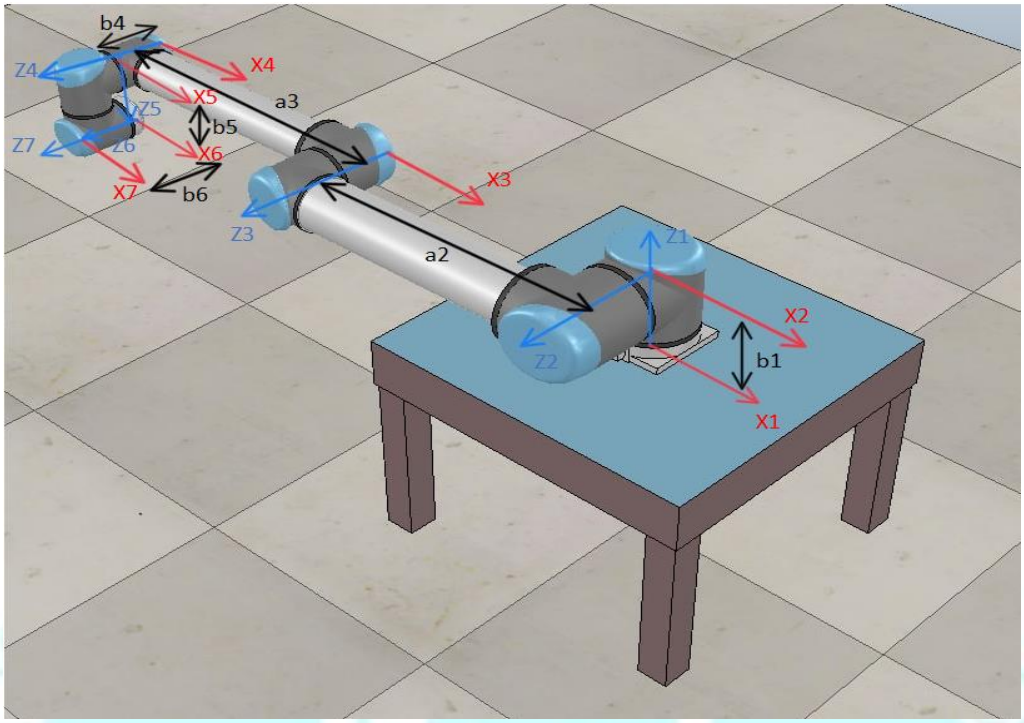
2.4 کنترل ربات:

برای انتقال مجری نهایی و گریپر به مکان بدست آمده از پردازش تصویر نیاز به مجموعهٔ موقعیت (زاویه) برای مفاصل ربات UR10 داریم، سپس مجموعه زوایای بدست آمده توسط MATLAB به کنترلر PID داخلی نرم افزار CoppeliaSim ارسال می‌شود تا زوایای مفاصل به مقدار مطلوب ما برسد. (البته همانطور که گفته شد در این پروژه به طور خاص فقط از کنترلر P استفاده شده است.)

2.4.1 سینماتیک مستقیم[21]:

از آنجا که برای حل سینماتیک معکوس ربات به سینماتیک مستقیم وابسته است ابتدا به آن می‌پردازیم: برای محاسبات سینماتیک مستقیم به پارامترهای DH ربات نیاز داریم که با توجه به شکل18 به صورت آنچه در جدول 1 آمده است بدست می‌آید.

[21] Forward kinematics



شکل 18 - دستگاه های مختصات مربوط به هر محور و فواصل شان برای یافتن پارامترهای **DH**

معادلات سینماتیک مستقیم با توجه به جدول فوق و آنچه در درس آموخته بودیم به صورت زیر خواهد بود و بردار موقعیت مکانی مجری نهایی به صورت زیر بدست می آید:

$$\overrightarrow{\boldsymbol{p_{EE}}} = \begin{bmatrix} x_{EE} \\ y_{EE} \\ z_{EE} \end{bmatrix}$$

که در آن:

$$\begin{aligned} x_{EE} = {} & 0.163941\, sin(\theta_1) - 0.612\, cos(\theta_2)\, cos(\theta_1) + 0.0922\, cos(\theta_5)\, sin(\theta_1) \\ & + 0.5723\, cos(\theta_1)[sin(\theta_2)\, sin(\theta_3) - cos(\theta_2)\, cos(\theta_3)] \\ & - 0.0922\, sin(\theta_5)\, cos(\theta_1)\, cos(\theta_2 + \theta_3 + \theta_4) \\ & + 0.1157\, cos(\theta_1)\, [cos(\theta_2 + \theta_3)\, sin(\theta_4) + sin(\theta_2 + \theta_3)\, cos(\theta_4)] \end{aligned}$$

$$\begin{aligned} y_{EE} = {} & -0.163941\, cos(\theta_1) \\ & + 0.5723\, sin(\theta_1)\, sin(\theta_2)\, sin(\theta_3) \\ & - 0.0922\, cos(\theta_1)\, cos(\theta_5) - 0.612\, cos(\theta_2)\, sin(\theta_1) \\ & - 0.0922\, sin(\theta_1)\, sin(\theta_5)\, cos(\theta_2 + \theta_3 + \theta_4) + 0.1157\, cos(\theta_2 \\ & + \theta_3)\, sin(\theta_4)\, sin(\theta_1) \\ & + 0.1157\, sin(\theta_1) sin(\theta_2 + \theta_3)\, cos(\theta_4) \\ & - 0.5723\, sin(\theta_1)\, cos(\theta_2)\, cos(\theta_3) \end{aligned}$$

$$z_{EE} = 0.1273 - 0.612\,sin(\theta_2) + 0.1157sin(\theta_2 + \theta_3)\,sin(\theta_4) - 0.5723sin(\theta_2 + \theta_3) - 0.0922sin(\theta_5)[cos(\theta_2 + \theta_3)\,sin(\theta_4) + sin(\theta_2 + \theta_3)\,cos(\theta_4)] - 0.1157\,cos(\theta_4)cos(\theta_2 + \theta_3)$$

همچنین دوران مجری نهایی نیز به صورت زیر حاصل می‌شود:

$$\boldsymbol{Q_{EE}} = \begin{bmatrix} cos(\theta_6)\,\sigma_3 - cos(\theta_1)\,sin(\theta_6)\,\sigma_1 & -sin(\theta_6)\,\sigma_3 - cos(\theta_1)\,cos(\theta_6)\,\sigma_1 & -cos(\theta_1)\,sin(\theta_5)\,\sigma_4 + cos(\theta_5)\,sin(\theta_1) \\ -cos(\theta_6)\,\sigma_2 - sin(\theta_1)\,sin(\theta_6)\,\sigma_1 & sin(\theta_6)\,\sigma_2 - sin(\theta_1)\,cos(\theta_6)\,\sigma_1 & -sin(\theta_1)\,sin(\theta_5)\,\sigma_4 - cos(\theta_5)\,cos(\theta_1) \\ sin(\theta_6)\,\sigma_4 + cos(\theta_5)\,cos(\theta_6)\,\sigma_1 & cos(\theta_6)\,\sigma_4 - cos(\theta_5)\,sin(\theta_6)\,\sigma_1 & -sin(\theta_1)\,\sigma_1 \end{bmatrix}$$

و در آن:

$$\sigma_1 = sin(\theta_2 + \theta_3 + \theta_4)$$
$$\sigma_2 = cos(\theta_1)\,sin(\theta_5) - cos(\theta_2 + \theta_3 + \theta_4)\,cos(\theta_5)\,sin(\theta_1)$$
$$\sigma_3 = sin(\theta_1)\,sin(\theta_5) + cos(\theta_2 + \theta_3 + \theta_4)\,cos(\theta_5)\,cos(\theta_1)$$
$$\sigma_4 = cos(\theta_2 + \theta_3 + \theta_4)$$

2.4.2 سینماتیک معکوس[۲۲]:

برای حل سینماتیک معکوس با توجه به اینکه ربات دارای 6 درجه آزادی است و همانطور که از معادلات سینماتیک مستقیم بدست آمده می‌توان دید، حل سینماتیک معکوس آن به صورت پیچیده خواهد بود؛ در مقاله [7] معادلات سینماتیک معکوس را به صورت هندسی حل و جواب‌های مختلف آن را تحلیل و بررسی کرده است (در بعضی شرایط 8 جواب هم برای آن وجود دارد) اما با توجه به اینکه حالات و شرایط مختلفی وجود دارد که باید در نظر گرفته شوند و چون با روش‌های عددی نیز می‌توان به راحتی به جواب مطلوب رسید و حتی ممکن است برای برخی ربات‌های پیچیده تر عملا امکان حل تحلیلی سینماتیک معکوس وجود نداشته باشد لذا تصمیم به استفاده از روش‌های عددی برای حل آن گرفتیم؛ ما در این پروژه برای این کار از تابع ikunc جعبه ابزار Robotics استفاده کردیم که این تابع نیز خود از جعبه ابزار بهینه‌سازی[۲۳] نرم افزار MATLAB و تابع fminunc برای حل مسئله سینماتیک معکوس که یک مسئله چند متغیره است استفاده می‌کند به این صورت که یک تابع هدف (خطا) تعریف می‌شود به صورت زیر:

$$Error = sumsqr\,((\,T^{-1} \times forward\_kinematic(q) - I) \times \Omega)$$

[22] Inverse Kinematics
[23] Optimization

که اختلاف جواب سینماتیک مستقیم (حاصل از q، زوایایی که تا به حال بدست آمده) و فرم مجری نهایی است. (T ماتریس همگن شده بیانگر موقعیت دورانی و موقعیت مکانی مطلوب ما برای مجری نهایی است که به صورت ورودی به تابع ikunc داده می‌شود؛ همچنین Ω یک ضریب ثابت می‌باشد.)

سپس با استفاده از جعبه ابزار بهینه سازی MATLAB و تابع fminunc، به صورت تکراری[۲۴] و با الگوریتم Quasi-Newton، نقطه کمینه تابع خطای مذکور را می‌یابیم و به این ترتیب و به صورت عددی سینماتیک معکوس ربات برای هر موقعیت دلخواه در فضای کاری ربات بدست می‌آید و با داشتن این جواب مجری‌نهایی را می‌توانیم در هر موقعیت دلخواد در فضای کاری ربات قرار دهیم.

می‌دانیم که در حالت کلی پاسخ سینماتیک معکوس برای یک ربات 6 درجه آزادی یکتا نیست، در اینجا نیز پاسخ نهایی بر اساس حدس اولیه[۲۵] تعیین می‌شود که به صورت پیشفرض $\overrightarrow{q0} = \vec{0}$ است، همچنین می توان آن را به صورت پارامتر نیز به تابع ikunc داد؛ در ابتدا ما این پارامتر را همان مقدار پیشفرض صفر در نظر می‌گرفتیم اما به دلیل مشکلاتی که در بخش بعدی با جزئیات بیشتر آمده است، برخی اوقات به پاسخ درست همگرا نمی‌شد و کنترل ربات با مشکل مواجه می‌شد به همین دلیل در نهایت با تغییر مقدار پیشفرض حدس اولیه مشکل مذکور حل شد و ربات UR10 می‌توانست به درستی عملیات Pick and Place که قبل‌تر توضیح داده شد را انجام دهد.

2.5 مقایسه دو گریپر RG2 و Baxter Vacuum Cup:

در جداول 2و 3و 4 درصد موفقیت معادل است با نسبت تعداد مکعب‌های انداخته شده درون جعبه‌ها به تعداد کل تعداد مکعب‌های برداشته شده از روی نوار نقاله؛ در هر کدام از این آزمایش‌ها به مدت 5 دقیقه شبیه‌سازی اجرا می‌شد[۲۶] و در این مدت ربات عملیات Pick & Place را انجام می‌داد، در تمام آزمایش‌ها ربات با گریپر مکنده (Baxter Vacuum Cup) کاملاً بدون مشکل مکعب‌ها را برداشته و به جعبه مربوطه منتقل می‌کرد اما گریپر RG2 در برخی موارد معدود به دلیل دقت نسبتاً پایین در تشخیص میزان دوران جسم، مکعب به درستی در چنگک گریپر جای نمی‌گرفت و هنگام برداشتن یا جابجایی از گریپر جدا شده و بر زمین می‌افتاد یا حتی ربات نمی‌توانست مکعب را از روی نوار نقاله بلند کند.

نکته دیگری که مشاهده می‌شود این است که سرعت گریپر مکنده به صورت میانگین حدود 3 برابر سرعت گریپر RG2 است زیرا که در هنگام استفاده از RG2 مکعب‌ها در دو مرحله برداشته می‌شوند، مرحله اول گریپر بالای مکعب و با فاصله از آن قرار می‌گیرد و با تنظیم زاویه در مرحله دوم مکعب را برمی‌دارد و همچنین باز

---

[24] Iterative

[25] Initial guess

[26] توجه داریم که شبیه‌سازی الزاماً Realtime نیست و در اینجا 5 دقیقه منظور 5 دقیقه شبیه‌سازی است که ممکن است اجرای آن بیشتر از 5 دقیقه حقیقی زمان ببرد.

و بسته شدن آن نیز زمانبر است و به طور کلی Baxter Vacuum Cup هم از نظر دقت و هم سرعت گزینه مناسب‌تری است مگر در کاربردهای خاص که به علت جنس جسمی که قرار است جابجا شود نتوان از آن بهره برد.

جدول 2 - نتایج شبیه‌سازی مکنده **Baxter Vacuum Cup**

شماره آزمایش	سرعت (مکعب بر دقیقه)	درصد موفقیت
1	3	100
2	3	100
3	3.2	100
4	3.2	100
5	3.2	100

جدول 3 - نتایج شبیه‌سازی چنگک **RG2 GRIPPER**

شماره آزمایش	سرعت (مکعب بر دقیقه)	درصد موفقیت
1	0.8	100
2	1.2	100
3	1.4	100
4	0.6	75
5	1.2	85

جدول 4 - جمع بندی و مقایسه جدول 2 و 3

نوع گریپر	میانگین سرعت (مکعب بر دقیقه)	میانگین درصد موفقیت
Baxter Vacuum Cup	3.12	100
RG2 Gripper	1.04	92

ربات برای تغییر موقعیت از حالت اولیه به حالت نهایی، خط صاف بین این دو نقطه را طی نمی‌کند و ممکن است از مسیری منحنی به نقطه نهایی برسد که لزوما این مسیر مطلوب نیست. در صورت داشتن مدل دینامیکی ربات، می‌توانیم ربات را به گونه‌ای کنترل کنیم که با داشتن مسیریابی، از نقطه اولیه به نقطه نهایی با سرعت روان‌تری حرکت کند. اما در اینجا صرفا هدف این است که ربات از هر مسیری به نقطه نهایی نرود و همچنین در فضای کاری حرکت کند تا از کنترل خارج نشود زیرا که ربات با مود موقعیت کنترل میشود و سرعت و شتاب کنترل نمی‌شد. فرض می‌کنیم که مدل دینامیکی را در اختیار داریم و در این صورت موقعیت مفاصل به گونه ای است که در زمان اولیه برابر θ_I و در نهایت برابر θ_F باشد و مسیر بین این دو را یک چند جمله ای با درجه 5 تعیین می‌کند. لذا با استفاده از روابط زیر[۲۷] می‌توان چند جمله ای مذکور را بدست آورد:

$$\boxed{s(\tau) = 6\tau^5 - 15\tau^4 + 10\tau^3}$$

$$0 \le s \le 1, \qquad 0 \le \tau = \frac{t}{T} \le 1$$

زاویه، سرعت زاویه‌ای و شتاب زاویه‌ای در هر لحظه با توجه به چندجمله مذکور بدین شکل قابل محاسبه است:

$$\begin{cases} \theta(t) = \theta_I + (\theta_F - \theta_I)s(\tau) \\ \dot{\theta}(t) = (\theta_F - \theta_I)\dfrac{1}{T}s'(\tau) \\ \ddot{\theta}(t) = (\theta_F - \theta_I)\dfrac{1}{T^2}s''(\tau) \end{cases}$$

[27] موجود در کتاب Angeles - اثبات این روابط در بخش ضمیمه آمده است.

۳ بحث و بررسی

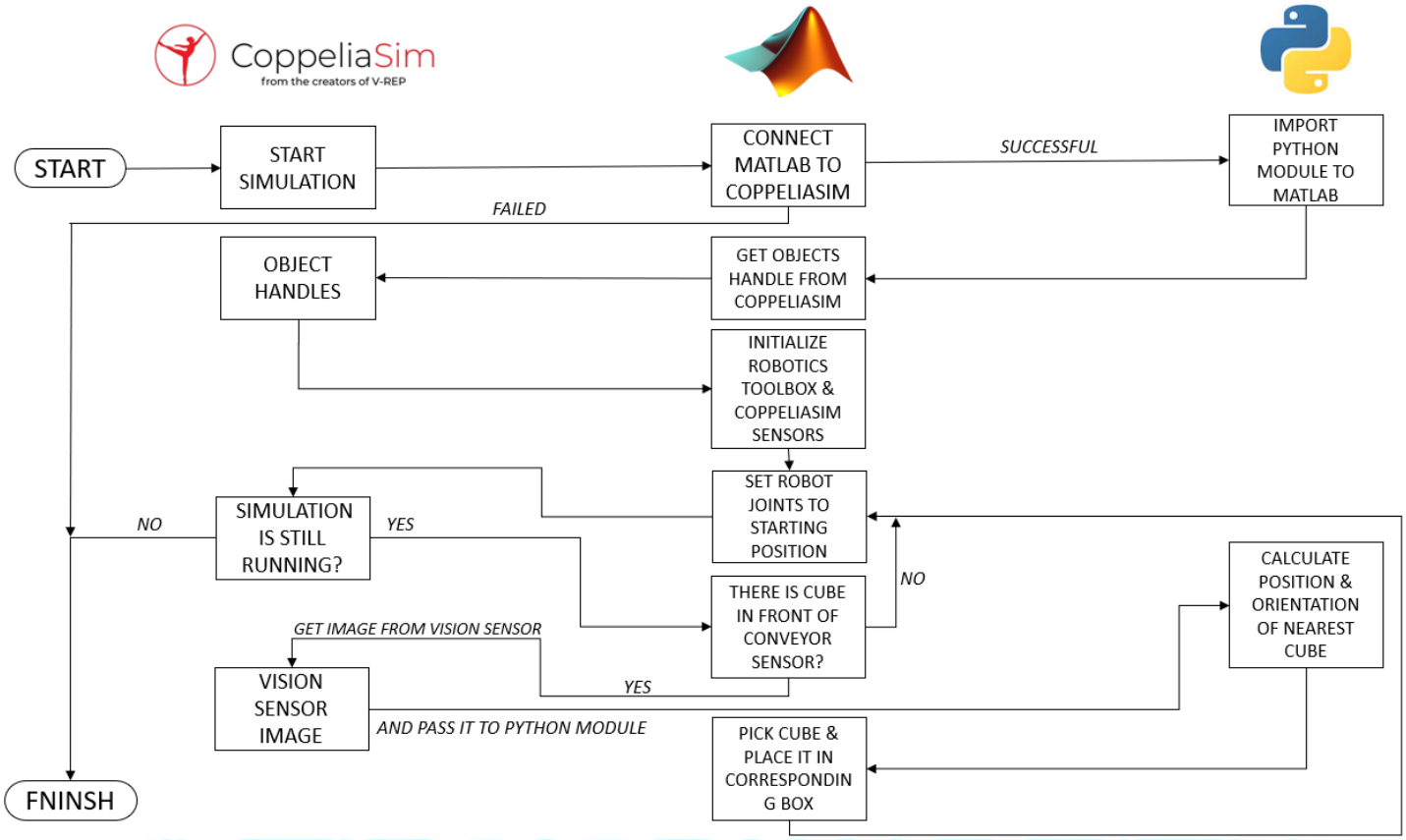
3.1 مراحل پروژه و شبیه سازی:

پس از آماده سازی محیط شبیه سازی و انجام تنظیمات مربوط به آن (مانند تنظیمات مربوط به فرکانس تولید قطعات و... که در بخش قبل توضیح داده شد)، برای ایجاد اتصال بین MATLAB و CoppeliaSim از RemoteApi ارائه شده توسط نرم افزار CoppeliaSim استفاده کردیم، همچنین برای فراخوانی توابع Python مربوط به پردازش تصویر OpenCV از نرم افزار MATLAB، نیاز بود که Module مربوطه را در MATLAB وارد کنیم؛ در شکل19 فلوچارتی در رابطه با نحوه ارتباط این سه محیط آمده است.

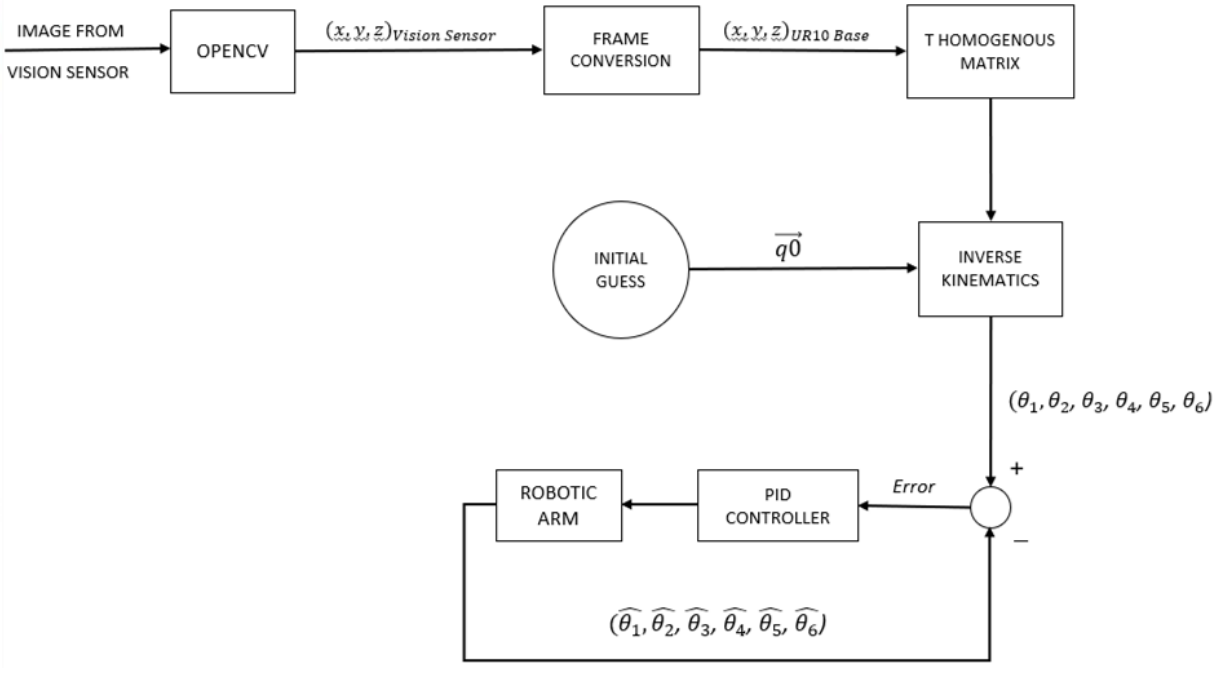
پس از ایجاد ارتباط بین MATLAB و دو نرم افزار دیگر، لازم است که Object Handleها که برای کنترل اشیای موجود در محیط شبیه سازی (مثل فرمان دادن به Jointها یا گرفتن تصویر ازVision Sensor یا...) را بدست آوریم در نهایت با توجه به پارامترهای DH ربات که بدست آوردیم ربات UR10 را در MATLAB با جعبه ابزار Robotics تعریف می‌کنیم و در نهایت حلقه اصلی شبیه سازی آغاز می‌شود؛ این حلقه تا زمانی که شبیه سازی در جریان است ادامه دارد و به این ترتیب است که در ابتدا ربات به حالت اولیه خود (زوایای مفاصل همه صفر) برمی‌گردد و اگر سنسور مجاورت نوار نقاله تشخیص دهد که یک مکعب مقابل آن قرار دارد تصویر آن توسط MATLAB ذخیره شده و بعد تابع Python را فراخوانی کرده و این تابع نیز موقعیت مکانی و دورانی نزدیک‌ترین مکعب به ربات (در صورتی که به هر دلیلی چندین مکعب در زاویه دید Vision Sensor باشد) را به MATLAB بر می‌گرداند؛ این مختصات در فریمVision Sensor است که با تبدیل مختصات، مختصات مکعب مذکور در فریم مرجع ربات UR10 خواهیم داشت.

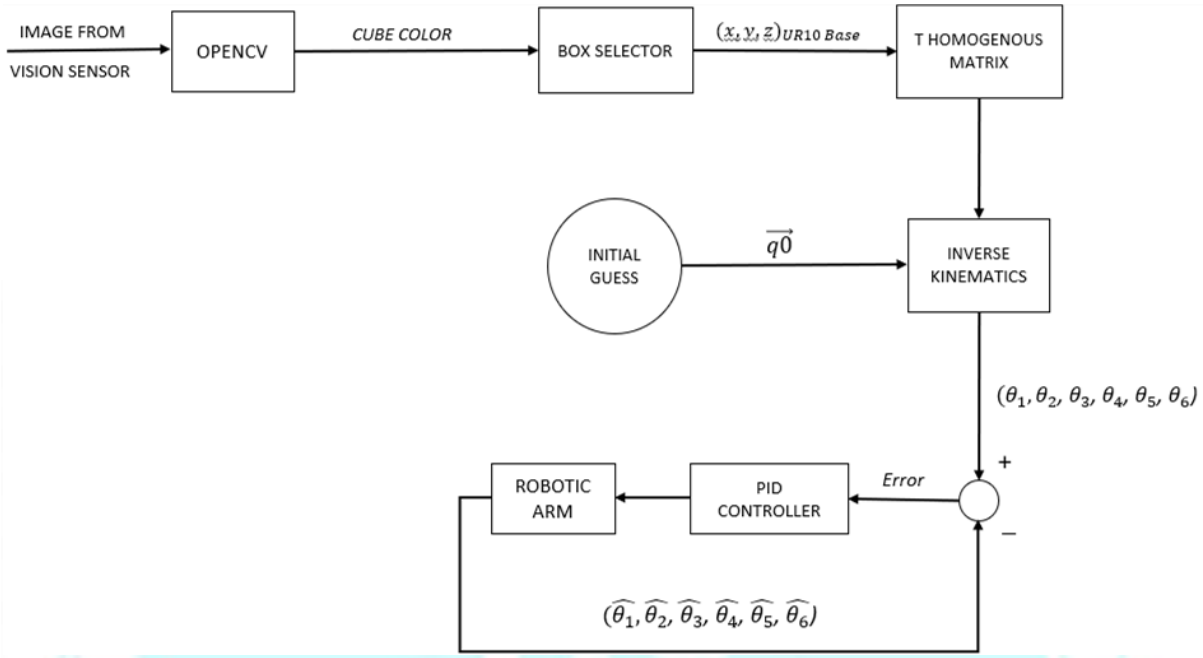
با داشتن مختصات مکعب و با کمک جعبه ابزار Robotics، مسئله سینماتیک معکوس را برای مختصات مربوط به مکعب حل می‌کنیم تا زوایای مطلوب Jointها را بدست آوریم، در نهایت این زوایا را به کنترل کننده PID داخلی CoppeliaSim داده تا زوایای Jointها به مقدار مطلوب برسند و در نتیجه مجری نهایی به محل مکعب می‌رسد، در نهایت مکعب توسط ربات UR10 برداشته می‌شود و به این ترتیب عملیات Pick انجام می‌گیرد؛ به طور مشابه هر مکعب با توجه به رنگش به مختصات مربوط به جعبه با همان رنگ منتقل می‌شود و به این ترتیب عملیات Place انجام می‌گیرد و ربات UR10 دوباره به موقعیت اولیه خود باز می‌گردد و منتظر مکعب بعدی برای انجام عملیات Pick & Place می‌ماند. (حلقه اصلی شبیه‌سازی به همین ترتیب ادامه می‌یابد.)[۲۸]

[28] توابع مورد استفاده در MATLAB و OpenCV در بخش ضمیمه آمده است.



شکل 19 - فلوچارت کلی شبیه‌سازی



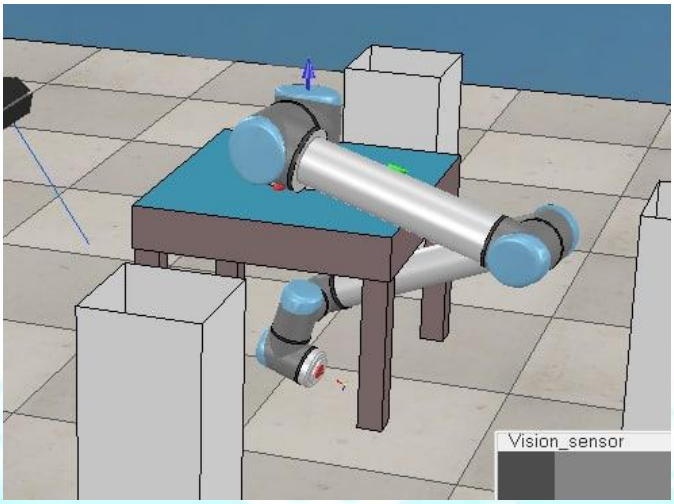


شکل 20 - فلوچارت مربوط پردازش تصویر و کنترل مفاصل در عملیات‌های **Pick** و **Place**

3.2 چالش‌ها

نخستین چالشی که با آن مواجه شدیم این بود که علیرغم امکانات بسیاری که این نرم افزار دارد و یک محیط قدرتمند برای شبیه سازی ارائه می‌دهد اما منحنی یادگیری[۲۹] آن کمی پیچیده است؛ برای مثال بعضاً دارای باگ هایی است که باعث کند شدن روند پیشرفت پروژه می‌شد زیرا مثلا امکان تغییر برخی تنظیمات از طریق رابط گرافیکی[۳۰] برنامه ممکن نبود و نیاز به تغییر اسکریپت های Lua مربوطه بود و از طرفی نمی‌خواستیم قسمت‌های اصلی پروژه وابسته به محیط شبیه‌سازی باشد به عبارت دیگر نمی‌خواستیم بخش-های مربوط به پردازش تصویر و کنترل ربات با اسکریپت Lua برنامه ریزی شود؛ این موارد در ابتدا باعث کند شدن پروسه‌ی پیشرفت پروژه بود که قبل از انجام پروژه پیش‌بینی نمی‌کردیم که چنین موردی برای ما به صورت چالش مطرح شود.

[29] Learning Curve
[30] GUI – Graphical User Interface



شکل 21 - مشاهده می‌شود که ربات از کنترل خارج شده و مجری نهایی آن در مکانی نامطلوب قرار گرفته است.

چالش دیگری که در بخش قبل هم مطرح شد این بود که حل عددی سینماتیک معکوس برخی اوقات به پاسخ درست همگرا نمی‌شد (تابع fminunc نیز به صورت یک اخطار[۳۱] این موضوع را بیان می‌کرد که به جواب صحیح همگرا نشده و احتمالا در یک کمینه محلی[۳۲] قرار گرفته است) و حتی باعث می‌شد ربات از ناحیه کاری اصلی‌اش (ناحیه بدون تکینگی) خارج شود و به طور کلی ربات از کنترل خارج شود که یک نمونه از این موارد در شکل21 آمده است.

برای رفع چالش مذکور با توجه به اینکه مکان جعبه‌ها و نیز مکان نوار نقاله از پیش مشخص و ثابت است، متوجه شدیم که با استفاده از یک حدس اولیه مناسب‌تر (غیر از مقدار پیشفرض صفر) می‌توان این مشکل را حل کرد و همینطور هم شد و با انتخاب یک حدس اولیه مناسب الگوریتم ما به سرعت و با دقت بالایی همگرا می‌شد (تابع خطا به مقدار کمینه اش می‌رسید) و ربات در ناحیه کاری مطلوب ما می‌توانست به درستی عملیات Pick and Place که قبل‌تر جزئیات آن توضیح داده شده بود را انجام دهد.

بعد از رفع مشکل فوق در هنگام انجام تست متوجه شدیم که باز هم این موضوع پیش می‌آمد که ربات دوباره از کنترل خارج می‌شد و بعضاً حالت نوسانی پیدا می‌کرد البته این اتفاق نسبت به حالت قبل کمتر رخ می‌داد اما همچنان در بعضی مواقع باعث کنترل ناپذیری ربات می‌شد؛ در ابتدا به همان مشکل قبل شک بردیم که شاید باز هم ایراد در قسمت حل سینماتیک معکوس باشد اما در نهایت متوجه شدیم مشکل از وزن مکعب‌ها است زیرا که در هنگام آماده سازی شبیه‌سازی به وزن آن‌ها توجه کافی نداشتیم و چون وزنشان زیاد بود و ما با نیز به روش سینماتیکی و با دادن موقعیت[۳۳] ربات را کنترل می‌کردیم باعث می‌شد در بعضی حالات ربات از کنترل خارج شده و ناپایدار و نوسانی شود که با کاهش چگالی و در نتیجه وزن مکعب‌ها این مشکل

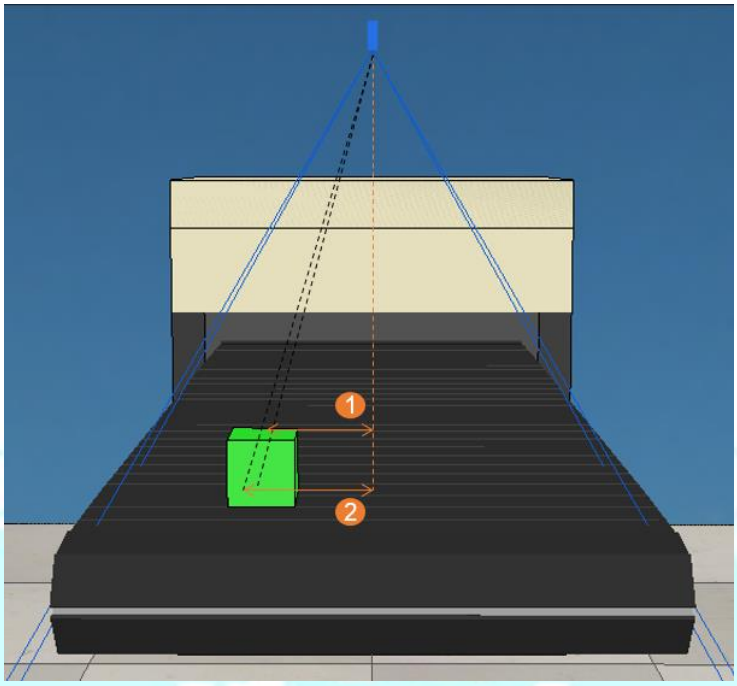
---

[31] Warning
[32] Local minimum
[33] position

هم برطرف شد؛ این مشکل شاید ساده به نظر برسد اما به دلیل اینکه پروژه در محیط شبیه سازی بود باعث می‌شد که پی بردن به علت آن چالش برانگیز باشد.

روش دیگر کنترلی که مطرح بود روش کنترل دینامیکی با استفاده از کنترل گشتاور موتورها بود که بدین ترتیب می‌توانستیم گشتاور و نیروی مجری نهایی را تعیین کنیم و علاوه بر آن، از مسیریابی استفاده کنیم که هم ربات از مسیر مورد نظر به جسم برسد هم با شتاب و سرعت نرم و روان تری به نقطه مورد نظر برسد و ضربه کمتری به ربات و مفاصل آن وارد آید. اما بدست آوردن مدل دینامیکی ربات پیچیده‌تر و وقت گیرتر بود لذا تصمیم گرفتیم که همان کنترل سینماتیکی را پیش بگیریم؛ اما در نسخه دیگری از پروژه، مسیریابی 345 را پیاده سازی کردیم که در صورتی که بخواهیم موتورها را در مد گشتاور کنترل کنیم، بتوانیم سرعت و گشتاور مجری نهایی روان‌تری داشته باشیم. اطلاع از مشخصات دینامیکی ربات و مسیریابی می‌توانست مشکلات ناشی از وزن مکعب‌ها و همچنین از کنترل خارج شدن‌ها را به نحوی از بین ببرد. در این نسخه، مسیر بین دو نقطه اولیه و نهایی، به چند نقطه دیگر تقسیم بندی شده و از یک چند جمله ای درجه 5 پیروی می‌کند. بدین ترتیب ربات در هر بازه زمانی در یکی از این نقاط تقسیم بندی شده باید قرار گیرد. مشکل این روش آن بود که فرآیند قرارگیری مفاصل ربات در هر کدام از این موقعیت های بین نقطه اول و آخر، بسیار زمان می‌برد و عملاً برای آنکه بتوانیم حرکت ربات را ببینیم باید تعداد نقاط را کمتر می‌کردیم که این کار هم باعث می‌شد که حرکت ربات پیوسته نباشد و لذا در نهایت حرکت شکسته شکسته‌ای را شاهد باشیم؛ همچنین تلاش کردیم که با کنترل سرعت مفاصل با استفاده از سرعت بدست آمده از مسیریابی 345 نیز ربات را کنترل کنیم اما دستوری که با استفاده از آن می‌توانستیم از سرعت مفاصل فیدبک بگیریم در RemoteApi که برای MATLAB ارائه شده بود قابل اجرا نبود و با خطا مواجه می‌شدیم و به همین دلیل این روش هم قابل استفاده نبود و در نهایت نسخه بدون مسیریابی را برای کنترل موقعیت در شبیه سازی، بهتر و روان‌تر یافتیم.

چالش دیگری که برای ادامه کار مسئله ساز بود، دید هرمی دوربین بود. در حقیقت تصویر دو بعدی‌ای که دوربین به ما می‌داد، ارتفاع جسم را درنظر نمی‌گرفت و چون جسم ما ارتفاع داشت، مرکزی که دوربین تشخیص میداد کمی دورتر از مرکز واقعی بود. برای درک بهتر این موضوع شکل22 آورده شده است که در آن مشخص است که مرکز جسم تشخیص داده شده توسط دوربین نسبت به یک سطح مشترک با مختصات اصلی (که همان کف مکعب است)، فاصله شماره2 است ولی فاصله اصلی که دوربین باید تشخیص دهد، فاصله شماره1 است. که این اختلاف ناشی از دوبعدی بودن تصویر پردازش شده است که ارتفاع در آن لحاظ نشده.



شکل 22 - فاصله از مرکز تشخیص داده شده توسط دوربین و فاصله واقعی

همچنین این امر باعث شده بود که فکر کنیم ربات خیلی دقت پایینی دارد و نمی‌تواند به درستی به مختصاتی که به آن داده می‌شد برود؛ لذا چالش بزرگی برایمان بوجود آورده بود. برای حل این مشکل تنها لازم بود که مختصات بدست آمده را نسبت به ارتفاع جسم، اسکیل کنیم و بدین ترتیب فاصله از مرکز واقعی را داشته باشیم. این موضوع سبب شد که ربات با دقت خیلی بهتری به مختصات جسم برود.

3.3 روش بدست آوردن فضای کاری ربات:

با نوشتن اسکریپت در بخش مربوط به ربات UR10 در CoppeliaSim، مفصل های 2،3،4 و 6 که مربوط به تعیین مکان مجری نهایی در یک صفحه هستند را در تمام حالت های ممکن قرار می‌دهیم، به این صورت تابعی به نام jointSweep نوشتیم که با گرفتن شماره مفصل‌ها و موقعیت ابتدایی آن‌ها و تعیین تعداد گام‌ها موقعیت‌هایی که هر مفصل در هر مرحله باید در آن قرار بگیرد را به عنوان خروجی می‌دهد. منظور از تعداد گام این است که مثلا اگر مفصل شماره دو قرار است 360 درجه دوران کند، و تعداد گام برای این مفصل برابر 100 است در هر مرحله 3.6 درجه نسبت به مرحله قبل دوران می‌کند. واضح است که هر چه این گام‌ها بیشتر باشند نقاط بیشتری پوشش داده می‌شود و فضای کاری بدست آمده دقیق‌تر خواهد بود. سپس با استفاده از خروجی این تابع در هر گام موقعیت مفصل‌های مربوطه را تغییر می‌دهیم و به کمک تابع sim.checkCollision بررسی می‌کنیم که آیا قسمتی از ربات با چیزی برخورد[۳۴] دارد یا خیر. اگر در این موقعیت برخوردی وجود نداشت، این نقطه (مختصات کنونی مکان مجری نهایی) را به عنوان قسمتی از فضای

[34] collision

کاری ذخیره می‌کنیم، و اگر برخورد وجود داشت ربات به موقعیت بعدی می‌رود. و این مراحل تا هنگامی که تمام موقعیت‌های مشخص شده مورد بررسی قرار گرفت انجام می‌شود. در نهایت این نقاط را در فایلی با پسوند xyz. ذخیره می‌کنیم که 3 ستون اول همان x,y,z ربات و 3 ستون بعدی بیانگر زاویه مجری نهایی است. در ادامه به کمک MATLAB فضای کاری ربات را رسم کرده و همچنین فایل stl. تولید می‌کنیم تا فضای کاری در محیط شبیه سازی مورد مشاهده قرار گیرد.

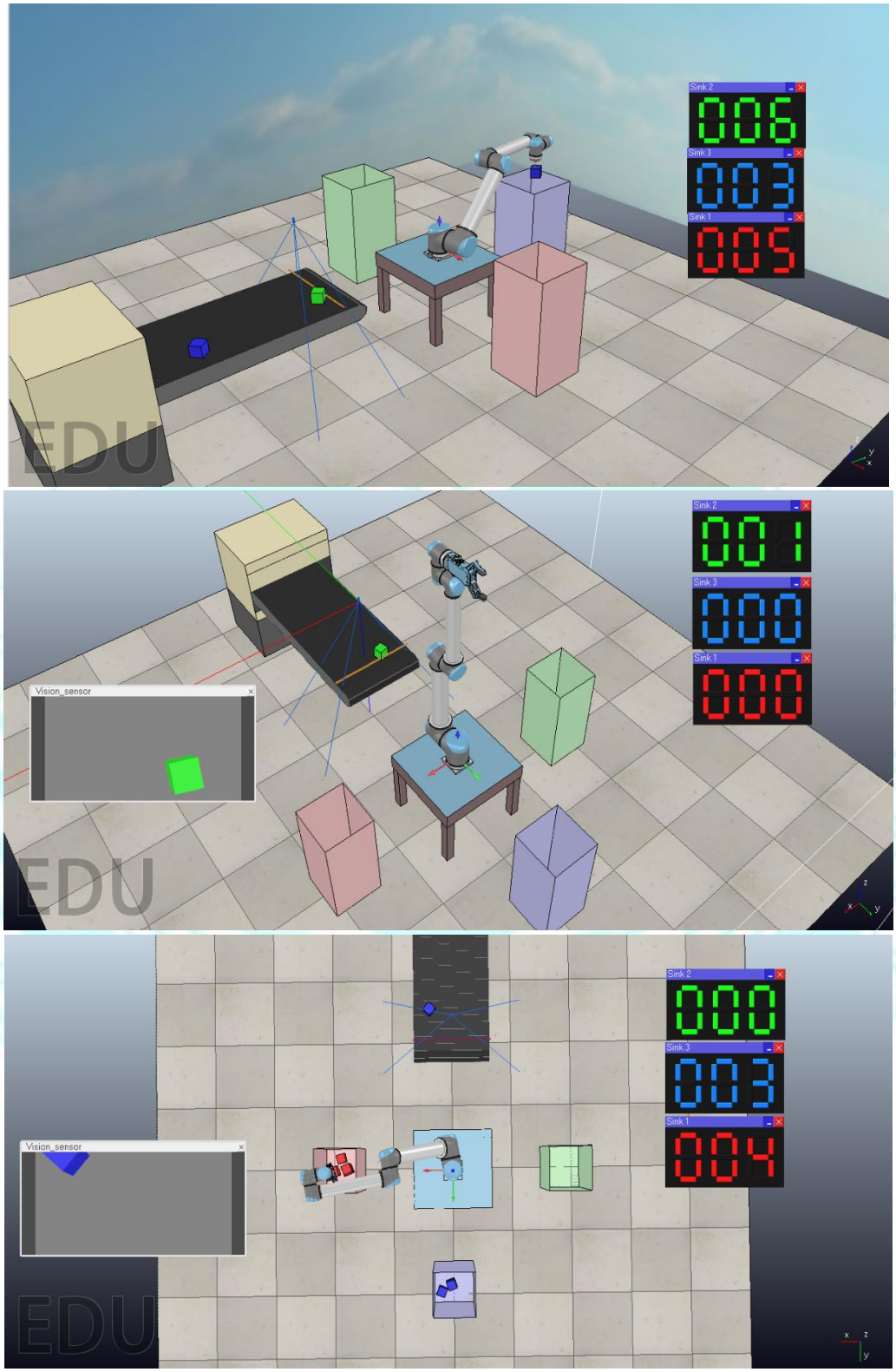
توضیحات کد MATLAB: فایلی را که به وسیله کد در CoppeliaSim ذخیره کرده بودیم در MATLAB لود می‌کنیم، سپس به کمک دستور های alphaShape و boundaryFacets کانتور فضای کاری را بدست می آوریم و در ادامه آن را رسم می‌کنیم، سپس به کمک دستور stlwrite این فضای کاری را در فایلی با فرمت stl. ذخیره کرده و آن را در محیط شبیه سازی import می‌کنیم.

3.4 نتیجه گیری

در این پروژه ما با کمک جعبه ابزارها، کتابخانه‌ها و نرم افزارهایی نظیر MATLAB، OpenCV و CoppeliaSim یک ربات UR10 را برای کاربرد Pick and Place برای مرتب و جداسازی بلوک‌های رنگی بر مبنای پردازش تصویر در خط تولید، شبیه سازی کردیم و موفق شدیم ربات UR10 را برای انجام این عملیات با سرعت و دقت بالا با کمک مدل سینماتیکی آن کنترل کنیم؛ در طول این پروژه با استفاده از دانش‌هایی که در طول ترم آموختیم (مواردی چون جدول DH یک ربات و حل مسائل سینماتیک مستقیم و معکوس و...) ربات را کنترل کردیم و کاربرد عملی این آموخته‌ها را در طول این پروژه مشاهده کردیم، همچنین علاوه بر آن ما در زمینه‌هایی چون شبیه سازی در محیط CoppeliaSim و نحوه ارتباط آن با MATLAB و نیز ارتباط MATLAB با Python و کتابخانه پردازش تصویرOpenCV و... مطالب زیادی را آموختیم و تجربه زیادی در این زمینه‌ها کسب کردیم.

یکی از مهم ترین دستاوردهای این پروژه برای ما، روبرو شدن با چالش¬های یک پروژه رباتیکی بود، اینکه می‌بایست هم مهارت‌های نرم افزاری و کامپیوتری و برنامه نویسی را تقویت می‌کردیم، در کنار آن می‌بایست با نحوه مکانیزم و مکانیک ربات آشنایی پیدا می‌کردیم و همچنین بحث کنترل ربات که از بحث‌های بسیار مهم است و می‌بایست با چالش‌های آن روبرو می‌شدیم.

در کل می‌توان گفت که بهتر متوجه شدیم که یک ربات ترکیبی از دانش‌های مکانیک، کنترل، برق و کامپیوتر را نیاز دارد و به نوعی زمینه‌ای بین رشته‌ای است و لذا نیازمند کار گروهی متشکل از تخصص‌های مختلف است و می‌تواند در آینده ای نزدیک با پیشرفت بیشتر تکنولوژی به زمینه‌ای اثرگذارتر از آنچه امروز شاهد آن هستیم، تبدیل شود.



شکل 23- چند تصویر از ربات در حالت انجام عملیات **Pick & Place** در محیط شبیه‌سازی

مراجع

1. https://opencv.org/
2. https://www.coppeliarobotics.com/helpFiles
3. Remote API functions (MATLAB) (coppeliarobotics.com)
4. J.Angeles, Fundamentals of Robotic Mechanical Systems, Fourth Edition, New York: Springer, 2014, pp. 255-265.
5. https://petercorke.com/toolboxes/robotics-toolbox/
6. Lecture Notes
7. K. P. Hawkins. Analytic Inverse Kinematics for the Universal Robots UR5/UR10 Arms, 2013.
8. https://www.universal-robots.com/case-stories/allied-moulded-products-inc/
9. https://www3.panasonic.biz/ac/e/fasys/appli_search/data/images/apl0331.png

ضمیمه ها

ممان‌ها:

ممان‌ها کمیت‌های قابل سنجشی هستند که یک تابع را توصیف و خصوصیات مهم آن را ثبت می‌کنند. همچنین به صورت گسترده در علم آمار برای توصیف تابع چگالی احتمال استفاده می‌شوند. به زبان ساده، ممان‌های یک عکس مجموعه‌ای از پارامترهای آماری هستند که چگالی پیکسل‌ها و همچنین شدت آن‌ها را معین می‌کنند. از دید ریاضی، ممان تصویر M_{ij} که از مرتبه‌ی (i,j) است، برای یک عکس خاکستری[۳۵]، به صورت زیر محاسبه می‌شود:

$$M_{ij} = \sum_{x}\sum_{y} x^i y^j I(x,y)$$

بدین ترتیب که y و x، نشان دهنده اندیس سطر و ستون هستند و $I(x,y)$، شدت رنگ را در مختصات (x,y) نشان می‌دهد.

حال فرض کنیم که می‌خواهیم مساحت یک تصویر را برای یک ماتریس باینری، بدست آوریم. با استفاده از فرمول ذکر شده، ممان مرتبه صفر (M_{00})را بدست خواهیم آورد:

$$M_{00} = \sum_{x}\sum_{y} I(x,y)$$

برای یک تصویر باینری که شدت رنگ 0 یا 1 است، ممان صفر، نشان دهنده‌ی تعداد تمام درایه‌های غیر صفر می‌باشد که به عبارتی معادل است با مساحت جسم تشخیص داده شده زیرا که مقدار $I(x,y)$ در یک ماتریس باینری، یا صفر است و یا یک که بدین ترتیب تعداد درایه‌های با مقدار یک را به ما خواهد داد. برای یک عکس خاکسرتی، ممان صفر معادل جمع شدت رنگ پیکسل‌ها می‌باشد.

[35] grayscale

توابع Python مورد استفاده در پردازش تصویر:

```
def get_contour_angle(cnt):
    rect = cv2.minAreaRect(cnt)
    angle = rect[-1]
    width, height = rect[1][0], rect[1][1]
    ratio_size = float(width) / float(height)
    if 1.25 > ratio_size > 0.75:
        if angle < -45:
            angle = 90 + angle
    else:
        if width < height:
            angle = angle + 180
        else:
            angle = angle + 90

        if angle > 90:
            angle = angle - 180

    return math.radians(angle)

def find_contours(frame, lower_limit, upper_limit):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    _, lightThresh = cv2.threshold(gray, 230, 255, cv2.THRESH_BINARY_INV)
    frame = cv2.bitwise_and(frame, frame, mask=lightThresh)
    blur = cv2.GaussianBlur(frame,(3,3),0)
    #Resize and coverting BGR to HSV
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
    #Finding the range in image
    out = cv2.inRange(hsv, lower_limit, upper_limit)
    dilated = cv2.dilate(out, cv2.getStructuringElement(cv2.MORPH_RECT,ksi
ze=(3,3)))
    contours, _ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.CHAIN_APPRO
X_SIMPLE)
    return contours
```

```
def get_pos(frame, lower_limit, upper_limit):
    x_res, y_res = 256, 128
    frame = cv2.resize(frame, (x_res,y_res))
    contours = find_contours(frame, lower_limit, upper_limit)
    result = frame.copy()
    cv2.drawContours(result, contours, -1, (255,255,255), 3)
    cx, cy = 0,0
    min_x, min_y = -10000,-10000
    rotation = -10000;
    if len(contours) > 0:
        for _,cnt in enumerate(contours):
            M = cv2.moments(cnt)
            if M['m00'] != 0:
                area = cv2.contourArea(cnt)
                if area>500:
                    cx = int(M['m10']/M['m00'])
                    cy = int(M['m01']/M['m00'])
                    if (cy > min_y):
                        min_x = cx
                        min_y = cy
                    cv2.circle(result, (cx, cy), 5, (255, 255, 255), -1)
                    cv2.circle(result, (cx,cy), 10, (0,255,255), 3)
                    rotation = get_contour_angle(cnt)
                    cv2.imwrite('out.png', result)
    else: return (-10000, -10000, -10000)  #default values
    return (min_x,min_y,rotation)
```

کد MATLAB مربوط به حلقه اصلی شبیه‌سازی و توابع اصلی :

```
  while (vrep.simxGetConnectionId(id) == 1)

      [~,state,~,Cuboid,~]=vrep.simxReadProximitySensor(id,ConveyorSensor,
      vrep.simx_opmode_streaming);
      if (state == 1) %if there is cube in front of Conveyor Sensor go and
pick it
          %pick
[~,~,color]=GotoNearestCube(id,vrep,Robot,Joints,Camera,ConveyorSensor);
          CloseVaccum(id,vrep,Cuboid,EE)
          %place
          GotoBasket(id,vrep,Robot,Joints,color)
          %release the cuboid
          OpenVaccum(id,vrep,Cuboid)
          %go to starting point
          RotateJoints(id,vrep,Joints,JointsStartingPos);
      end
  end

  function[p,rotation,color]=GotoNearestCube(id,vrep,Robot,Joints,Camera,C
onveyorSensor)

      [x,y,rotation,color]=GetPositionFromPy(id,vrep,Camera,ConveyorSensor
      );
      z = 0.405;
      p = [x,y,z];
      fprintf('coordinates: [%i,%i,%i] m\n',p(1),p(2),p(3));
      fprintf('rotation: %i degrees\n',rotation*180/pi);
      fprintf('color: %s\n',color);
      T = transl(p);
      theta_x = 0;
      theta_y = pi;
      theta_z = pi/2;
      T(1:3,1:3) = RotationMatrix(theta_z,theta_y,theta_x,'ZYX',true);
      %initial guess
```

```
    q0 = [1.2807    0.6263    1.5280   -0.5836    1.5708    0.2901];
    TargetPos = Robot.ikunc(T,q0); %1*6 vector
    TargetPos(6) = TargetPos(6) + rotation;
    RotateJoints(id, vrep, Joints, TargetPos);
End

function GotoBasket(id,vrep,Robot,Joints,color)
    if (color == "r")
        x = 0.9;
        y = 0;
        q0 = [0.17,-0.78,-0.81,-3.14,1.59,1.39];
    elseif (color == "b")
        x = 1e-3;
        y = 0.9;
        q0 = [1.75,-0.78,-0.81,-3.13,1.59,-0.19];
    elseif (color == "g")
        x = -0.9;
        y = 0;
        q0 = [0 0 0 0 0 0];
    end
    z = 0.65;
    p = [x,y,z];
    T = transl(p);
    %q0 is an initial guess
    TargetPos = Robot.ikunc(T,q0);
    RotateJoints(id, vrep, Joints, TargetPos);
end
```

اثبات روابط مسیریابی 345:

همانطور که گفته شد یک چندجمله‌ای از درجه 5 به صورت زیر داریم که زاویه، سرعت زاویه‌ای و شتاب زاویه‌ای در هر لحظه با توجه به آن قابل محاسبه است.

$$s(\tau) = a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f$$

$$0 \le s \le 1, \qquad 0 \le \tau = \frac{t}{T} \le 1$$

$$\begin{cases} \theta(t) = \theta_I + (\theta_F - \theta_I)s(\tau) \\ \dot{\theta}(t) = (\theta_F - \theta_I)\frac{1}{T}s'(\tau) \\ \ddot{\theta}(t) = (\theta_F - \theta_I)\frac{1}{T^2}s''(\tau) \end{cases}$$

با توجه به اینکه می‌خواهیم از $\theta(0) = \theta_I$ درنهایت به $\theta(T) = \theta_F$ برسیم و همچنین می‌خواهیم در ابتدا و انتهای حرکت سرعت و شتاب صفر داشته باشیم تا به ربات ضربه و ... وارد نشود و ربات آسیب نبیند، داریم:

$$\dot{\theta}(T) = \ddot{\theta}(T) = \dot{\theta}(0) = \ddot{\theta}(0) = 0$$

پس باید شرایط زیر حاکم باشد:

$$\begin{cases} s(0) = s'(0) = s''(0) = s'(1) = s''(1) = 0 \\ s(1) = 1 \end{cases}$$

بنابراین با توجه به این شرایط فوق ضرایب $s(\tau)$ را بدست می‌آوریم:

$$s'(\tau) = 5a\tau^4 + 4b\tau^3 + 3c\tau^2 + 2d\tau + e$$

$$s''(\tau) = 20a\tau^3 + 12b\tau^2 + 6c\tau + 2d$$

$$\Rightarrow \begin{cases} \boxed{f = e = d = 0} \\ a + b + c = 1 \\ 5a + 4b + 3c = 0 \\ 20a + 12b + 6c = 0 \end{cases} \Rightarrow \boxed{a = 6, b = -15, c = 10}$$

$$\Rightarrow \boxed{s(\tau) = 6\tau^5 - 15\tau^4 + 10\tau^3}$$