



دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده مهندسی مکانیک

گزارش پژوهش درس رباتیک

عنوان

تحلیل ربات شش درجه آزادی UR10

نگارش

صادق مهدوی ۹۸۲۶۰۳۰
مهردادی رحمانی ۹۷۲۶۰۳۱

استاد درس

دکتر حامد غفاری راد

تدریسیار درس

مهندس سیدعلی میرحقگوی

بهار ۱۴۰۲

بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِيْمِ
اللّٰهُمَّ اكْفُنْهُ مِنَ الْجَنَّةِ
وَلَا تُنَزِّلْهُ إِلَى الْجَهَنَّمِ
لَا يَرْجِعُ إِلَيْهَا وَلَا يَنْدَمُ

تقدیر و تشکر

بدینویسله مراتب قدردانی و تشکر خود را خدمت،

استاد محترم درس، جناب آقای دکتر غفاری راد بابت آموزش عالی مباحث مربوط به درس رباتیک،
و همچنین جناب آقای مهندس میرحقگوی بابت کمکهای بی‌دربیغ و راهنمایی‌هایشان برای یادگیری
هرچه بهتر نکات درس،

ابراز و از تمامی زحمات ایشان تشکر می‌نمایم.

صفحه	فهرست مطالب	عنوان
۷.....		فصل اول
۷.....		مقدمه.....
۸.....		۱-۱- معرفی ربات.....
۹.....		۱-۲- کاربرد ربات.....
۱۰.....		۱-۳- درجات آزادی مورد بررسی.....
۱۱.....		۱-۴- مشخصات ربات
۱۱.....		۱-۴-۱- ابعاد لینکها.....
۱۲.....		۱-۴-۲- فضای کاری
۱۳.....		فصل دوم
۱۴.....		۲-۱- پیدا کردن درجات آزادی و نام گذاری آنها
۱۵.....		۲-۲- انتخاب فریمها
۱۷.....		۲-۳- استخراج جدول پارامترهای DH
۱۸.....		۲-۴- ماتریس‌های تبدیل همگن
۱۹.....		۲-۵- سینماتیک مستقیم موقعیت و جهت گیری
۲۲.....		۲-۶- بررسی سینماتیک مستقیم برای چندین وضعیت متفاوت
۲۴.....		۲-۷- ترسیم فضای کاری
۲۵.....		۲-۸- صحه گذاری سینماتیک مستقیم
۲۸.....		فصل سوم
۲۹.....		۳-۱- حل تحلیلی و بررسی تعداد جواب
۳۴.....		۳-۲- حل سینماتیک معکوس به کمک روش هندسی
۳۹.....		۳-۳- بررسی سینماتیک معکوس برای چندین وضعیت متفاوت
۴۱.....		۳-۴- آزمایش الگوریتم سینماتیک معکوس
۴۲.....		۳-۵- صحه گذاری سینماتیک معکوس

۴۳.....	فصل چهارم
۴۴.....	۴-۱- استخراج ماتریس ژاکوبین
۴۷.....	۴-۲- بررسی نقاط سینگولاریتی و تقسیم بندی بر حسب موقعیت و جهتگیری
۴۸.....	۴-۳- تحلیل فیزیکی نقاط سینگولاریتی
۵۰.....	۴-۴- راه حل برطرف کردن سینگولاریتی
۵۱.....	فصل پنجم
۵۲.....	۵-۱- مشخصات لینک اول
۵۳.....	۵-۲- مشخصات لینک دوم
۵۴.....	۵-۳- مشخصات لینک سوم
۵۵.....	۵-۴- مشخصات لینک چهارم
۵۶.....	۵-۵- مشخصات لینک پنجم
۵۷.....	۵-۶- مشخصات لینک ششم
۵۸.....	فصل ششم
۵۹.....	۶-۱- توضیحات اولیه بخش دینامیک
۶۰.....	۶-۲- استخراج معادلات دینامیک در فضای متغیرهای مفصلی به روش نیوتن
۶۲.....	۶-۳- استخراج معادلات دینامیک در فضای متغیرهای مفصلی به روش لاگرانژ
۶۳.....	۶-۴- استخراج معادلات دینامیک در فضای کارتزین
۶۴.....	۶-۵- شبیه سازی معادلات دینامیک (فضای متغیرهای مفصلی) در Matlab
۶۶.....	۶-۶- صحه گذاری معادلات دینامیک
۶۷.....	۶-۷- شبیه سازی حرکت ربات در نرم افزار ادمز
۷۰.....	۶-۷-۱- اعمال سرعت زاویه ای ثابت به مفاصل و تحلیل نتایج
۷۳.....	مراجع
۷۴.....	ضمیمه

عنوان	فهرست اشکال	صفحه
شکل ۱- نمایی از ربات UR10	۹	
شکل ۲- نمای انفجاری ربات و نمایش متغیر های مفصلی بر روی آن	۱۰	
شکل ۳- ابعاد لینک ها و آفست های موجود بین آن ها در ربات UR10	۱۱	
شکل ۴- فضای کاری ربات UR10	۱۲	
شکل ۵- درجات آزادی ربات	۱۴	
شکل ۶- فریم گذاری ربات	۱۶	
شکل ۷- وضعیت صفر ربات با ابزار پیترکورک	۲۵	
شکل ۸- صحه سنجی ماتریس تبدیل اندکتور به پایه در وضعیت صفر با پیترکورک	۲۵	
شکل ۹- ماتریس تبدیل اندکتور به پایه در وضعیت صفر به کمک ماتریس های تبدیل	۲۵	
شکل ۱۰- وضعیت ربات در زاویه دلخواه ۶۰ درجه با ابزار پیترکورک	۲۶	
شکل ۱۱- ماتریس تبدیل با زوایای ۶۰ درجه در پیترکورک	۲۶	
شکل ۱۲- ماتریس تبدیل با زوایای ۶۰ درجه با ماتریس های تبدیل	۲۶	
شکل ۱۳- نمایش ربات در زوایای ۶۰ درجه در ادمز	۲۷	
شکل ۱۴- به دست آوردن تندا	۳۴	
شکل ۱۵- به دست آوردن تنتا	۳۵	
شکل ۱۶- به دست آوردن تناع	۳۶	
شکل ۱۷- نمایش ربات در وضعیت صفر در ادمز (fully stretched)	۴۲	
شکل ۱۸- صحه سنجی وضعیت صفر ربات	۴۲	
شکل ۱۹- نمایی از فریم گذاری ربات و حالت سینگولار Elbow	۴۸	
شکل ۲۰- حالت سینگولار Wrist	۴۹	
شکل ۲۱- حالت سینگولار Shoulder	۵۰	
شکل ۲۲- لینک ۱ به همراه فریم اول	۵۲	
شکل ۲۳- ممان های اینرسی لینک ۱ حول مرکز جرم	۵۲	
شکل ۲۴- لینک ۲ به همراه فریم دوم	۵۳	
شکل ۲۵- ممان های اینرسی لینک ۲ حول مرکز جرم	۵۳	
شکل ۲۶- لینک ۳ به همراه فریم سوم	۵۴	
شکل ۲۷- ممان های اینرسی لینک ۳ حول مرکز جرم	۵۴	
شکل ۲۸- لینک چهارم به همراه فریم چهارم	۵۵	
شکل ۲۹- ممان های اینرسی لینک ۴ حول مرکز جرم	۵۵	
شکل ۳۰- لینک ۵ به همراه فریم پنجم	۵۶	

شکل ۳۱ - ممان‌های اینرسی لینک ۵ حول مرکز جرم ۵۶
شکل ۳۲ - لینک ۶ به همراه فریم ششم ۵۷
شکل ۳۳ - ممان‌های اینرسی لینک ۶ حول مرکز جرم ۵۷
شکل ۳۴ - ماتریس M برای تناهای ۴۵ درجه ۶۴
شکل ۳۵ - ماتریس M برای تناهای ۹۰ درجه ۶۴
شکل ۳۶ - تعریف ماتریس C بصورت درایه به درایه ۶۴
شکل ۳۷ - شبیه سازی با فیدبک تنا ۶۵
شکل ۳۸ - شبیه سازی بدون فیدبک تنا ۶۵

عنوان	فهرست نمودارها	صفحه
نمودار ۱- نمودار سرعت زاویه‌ای مراکز جرم ربات در اثر نیروی گرانش	نمودار ۱	۶۷
نمودار ۲- سرعت خطی مراکز جرم ربات در اثر نیروی گرانش	نمودار ۲	۶۸
نمودار ۳- تغییرات انرژی پتانسیل اعضای ربات	نمودار ۳	۶۸
نمودار ۴- تغییرات انرژی جنبش اعضای ربات	نمودار ۴	۶۹
نمودار ۵- نمودار تغییرات ارتفاع اندافکتور	نمودار ۵	۶۹
نمودار ۶- سرعت زاویه‌ای ورودی یک رادیان بر ثانیه به هر مفصل	نمودار ۶	۷۰
نمودار ۷- حرکت اندافکتور تحت سرعت زاویه‌ای ثابت مفاصل	نمودار ۷	۷۰
نمودار ۸- مقدار گشتاور اعمالی به مفاصل در حرکت سقوط آزاد	نمودار ۸	۷۱
نمودار ۹- مقدار گشتاور اعمالی به مفاصل در حرکت با سرعت ثابت	نمودار ۹	۷۱

فصل اول

مقدمه

۱-۱-۱- معرفی ربات

امروزه در صنایع مختلف، کاربرد بازوهای رباتیک رو به گسترش است. از بازوهای رباتیک، جهت مونتاژ، جابجایی قطعات، جوشکاری و ... استفاده می شود. استفاده از ربات‌ها کاهش خطاهای انسانی، افزایش دقت ساخت، کاهش هزینه و افزایش سرعت را در پی دارد.

شرکت یونیورسال ربات، یک شرکت دانمارکی است که در سال ۲۰۰۵ و با ساخت بازوهای رباتیک همکار صنعتی^۱ کوچک و انعطاف پذیر تاسیس شد. در سال ۲۰۰۸ اولین کوبات‌های UR5 این شرکت در بازارهای دانمارک و آلمان در دسترس بودند. در سال ۲۰۱۲ دومین کوبات، UR10، توسعه داده شد. در سپتامبر ۲۰۱۹، این شرکت UR16e را راهاندازی کرد که برای کارهای پر بار، مانند جابجایی مواد سنگین، مراقبت از ماشین‌های سنگین، بسته‌بندی و پالت سازی مناسب است. کوبات‌های UR هم در مشاغل کوچک تا متوسط و هم در شرکت‌های بزرگ و در صنایعی مانند خودروسازی، الکترونیک، فلز و ماشین کاری، داروسازی و... استفاده می شوند.

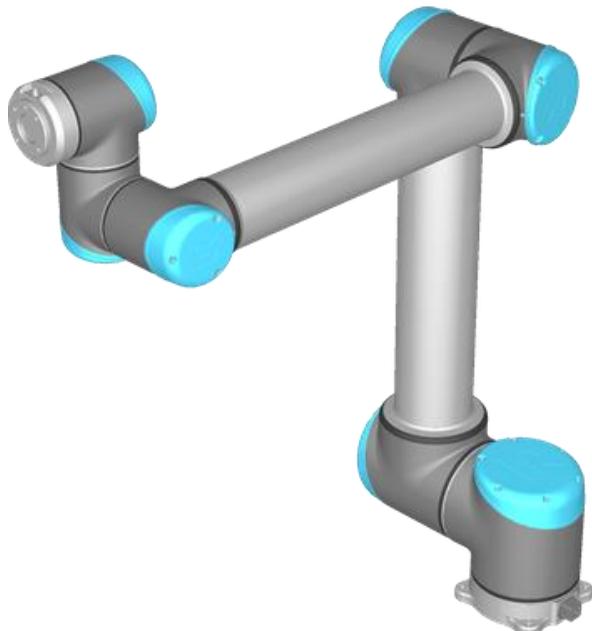
یونیورسال روبات UR10 یک بازوی روباتیک همه کاره است که به طور گسترده در کاربردهای مختلف صنعتی و تحقیقاتی استفاده می شود. بازوی ربات طوری طراحی شده است که برنامه ریزی آسان، انعطاف پذیر و ایمن برای کار در کنار اپراتورهای انسانی باشد. این ربات، یک بازوی رباتیک شش محوره است که قادر است محموله هایی تا وزن ۱۰ کیلوگرم را جابجا کند. دارای برد ۱۳۰۰ میلی متر و دقت تکرار پذیر ۰,۱ میلی متر است. بازوی ربات مجهرز به یک سیستم دید سه بعدی و یک حسگر نیرو/گشتاور است که آن را قادر می‌سازد تا وظایف مختلفی مانند جابجایی، مراقبت از ماشین، مونتاژ و کنترل کیفیت را انجام دهد. در این پروپوزال، یک نمای کلی از ربات UR10، کاربردهای آن، درجات آزادی، بعد لینک‌ها و فضای کاری ارائه خواهیم داد.

^۱ Cobot

۲-۱- کاربرد ربات

ربات UR10 در صنایع مختلفی از جمله خودروسازی، هواپیا، الکترونیک و داروسازی استفاده می شود. همچنین به طور گسترده در تحقیقات و آموزش استفاده می شود. برخی از کاربردهای خاص ربات UR10 عبارتند از:

- **مراقبت از ماشین:** ربات UR10 می تواند برای بارگیری و تخلیه ماشین ها استفاده شود و نیاز اپراتورهای انسانی به انجام این وظایف تکراری و بالقوه خطمناک را کاهش دهد.
- **مونتاژ:** بازوی ربات می تواند پارت ها و قطعات را با دقت بالا مونتاژ کند و خطاهای را کاهش دهد و بهره وری را افزایش دهد.
- **کنترل کیفیت:** ربات UR10 می تواند پارت ها و قطعات را از نظر نقص بررسی کند و اطمینان حاصل کند که فقط محصولات با کیفیت بالا برای مشتریان ارسال می شود.
- **تحقیق:** ربات UR10 در کاربردهای تحقیقاتی مختلفی از جمله تعامل انسان و ربات، دستکاری رباتیک و بینایی کامپیوتری استفاده می شود.



شکل ۱ - نمایی از ربات UR10

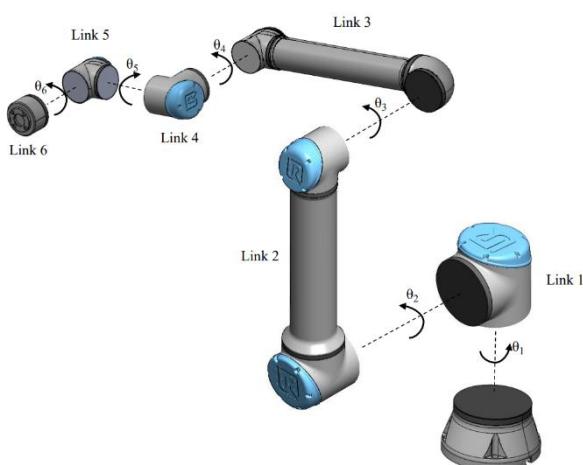
۱-۳- درجات آزادی مورد بررسی

درجات آزادی (DOF) یک ربات درواقع تعداد جهت‌ها و محورهایی است که میتواند به طور مستقل در آن‌ها حرکت داشته باشد. بازوی رباتیک مورد نظر دارای ۶ مفصل رولولوت که هر کدام یک درجه آزادی به ربات ما می‌دهند. در نتیجه UR10 یک ربات ۶ محوره میباشد و ۶ درجه آزادی دارد. محدوده کاری تمامی مفاصل ۳۶۰ درجه می‌باشد و ظرفیت باربرداری آن ۱۰ کیلوگرم است.

هریک از مفصل‌های این ربات یک درجه آزادی دارد و به این معنی که اجزه میدهد که فقط در یک جهت حرکت انتقالی داشته باشد یا حول یک محور دوران داشته باشد. شش مفصل این ربات به صورت سری کنار هم قرار گرفته‌اند و حرکت در هریک از این جوینتها در مکان end effector تاثیر دارد. همچنین آن به گونه‌ای است که به ربات اجزه میدهد که end effector خود را در محدوده گسترهای از مکان‌ها و چرخش‌های مختلف حرکت دهد.

شش درجه آزادی این ربات به شرح زیر میباشد:

- ۱- چرخش حول Base : اجازه میدهد که ربات در یک مسیر دایره‌ای حول base بشیرخشد. (θ_1)
- ۲- چرخش حول Shoulder : اجازه میدهد که ربات بازویش را بالا و پایین ببرد. (θ_2)
- ۳- چرخش حول Elbow : درواقع اجازه میدهد که ربات بازویش را خم کند. (θ_3)
- ۴- چرخش حول محور pitch از Wrist : این به ربات اجازه می‌دهد تا مج دست خود را به سمت بالا و پایین خم کند. (θ_4)
- ۵- چرخش حول محور roll از Wrist : این به ربات اجازه می‌دهد تا مج دست خود را از یک طرف به سمت دیگر بچرخاند. (θ_5)
- ۶- چرخش حول محور yaw از Wrist : این به ربات اجازه می‌دهد تا مج دست خود را حول محور خود بچرخاند. (θ_6)

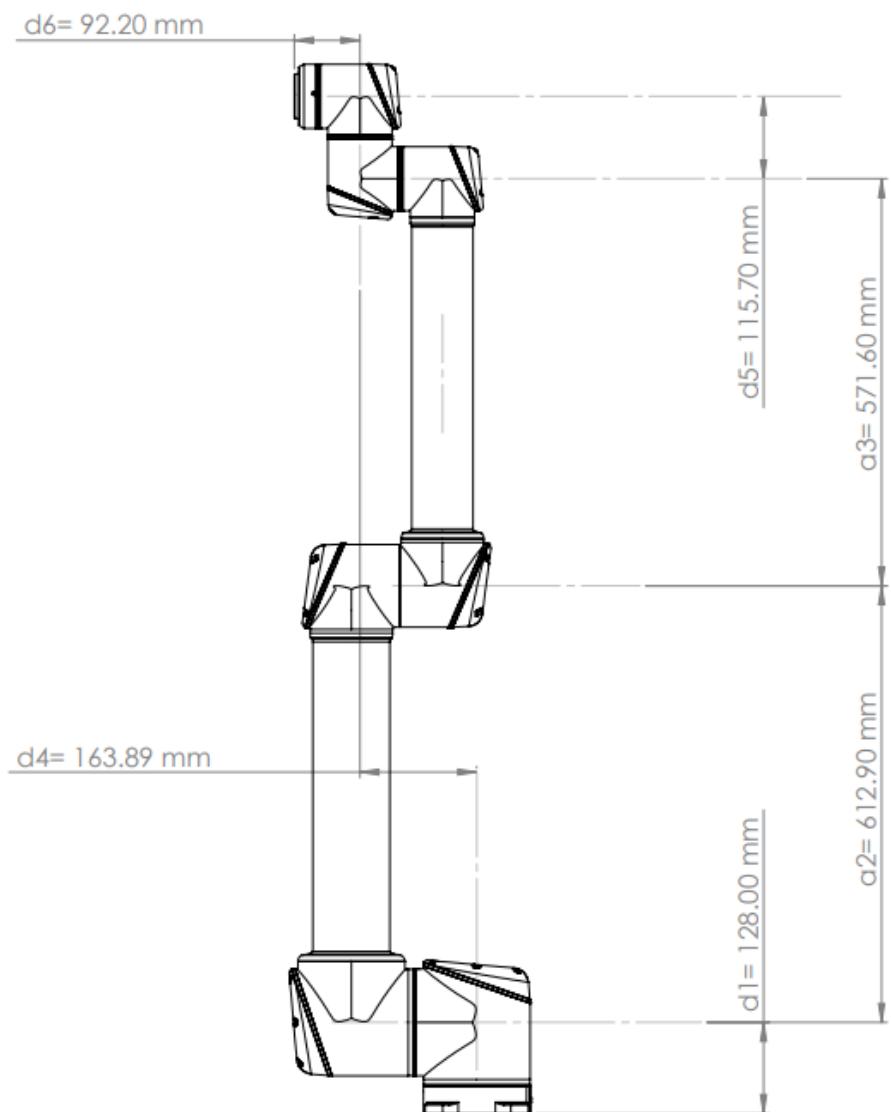


شکل ۲- نمای انفجاری ربات و نمایش متغیرهای مفصلی بر روی آن

۴-۱- مشخصات ربات

۱-۴-۱- ابعاد لینک‌ها

ابعاد با توجه به کاتالوگ‌ها موجود در شکل زیر به وضوح قابل مشاهده می‌باشد. بر اساس این شکل به راحتی می‌توان Offset و سایر موارد لازم در جدول دنويت-هارتبرگ که در کلاس تا حدی تدریس شده است را پیدا کرد.

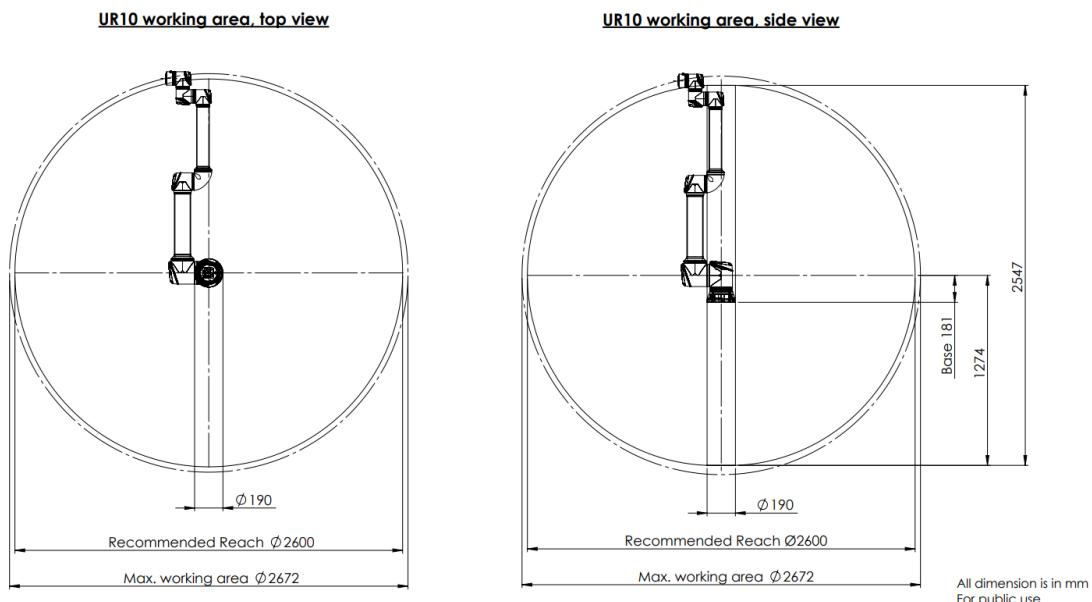


شکل ۳- ابعاد لینک‌ها و آفست‌های موجود بین آن‌ها در ربات UR10

۱-۴-۲- فضای کاری

محدوده کاری تمامی مفاصل این ربات، ۳۶۰ درجه می باشد و ربات UR10 دارای فضای کاری کروی با قطر تقریبی ۲۶۰۰ میلی متر است. این بدان معنی است که بازوی ربات می تواند به هر نقطه ای در این حجم کروی برسد و به آن امکان می دهد طیف گسترده ای از وظایف را انجام دهد.

میتوانید فضای کاری آن را به صورت بهتر و کاملتری در شکل زیر مشاهده کنید.



شکل ۴- فضای کاری ربات UR10

فصل دوم
سینماتیک مستقیم

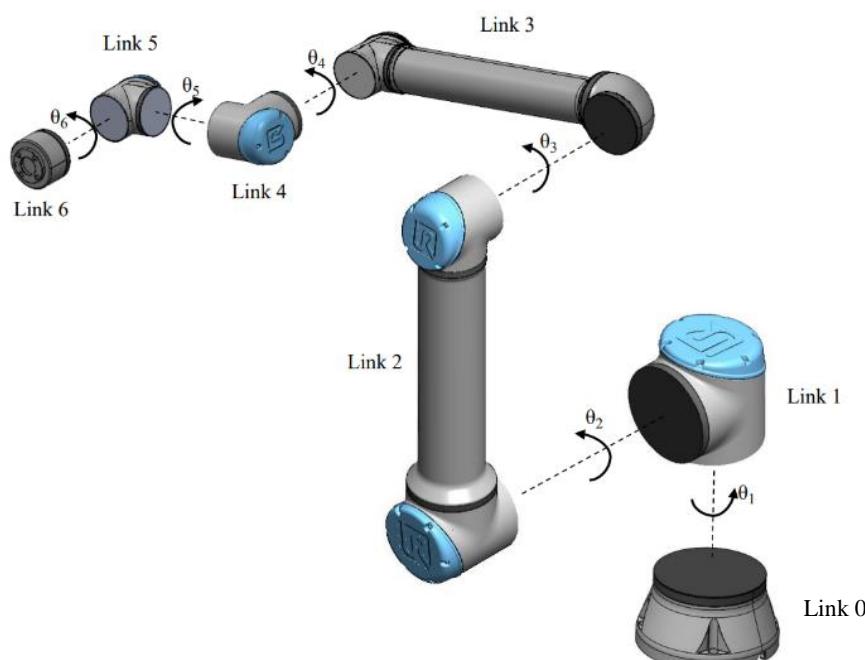
در این قسمت با توجه به اسلایدها و مرج اصلی درس مراحلی را جهت حل مسئله سینماتیک مستقیم مربوط به ربات UR10 پیش میبریم. به طور مختصر این مراحل عبارتند از:

- ۱- پیدا کردن درجات آزادی و نام گذاری آنها و شماره گذاری لینک‌ها
- ۲- انتخاب مناسب فریم‌ها و نمایش آنها بر روی ربات و به دست آوردن ابعاد
- ۳- به دست آوردن جدول DH و همچنین ماتریس تبدیل 0T_6

در ادامه به ترتیب به این مراحل میپردازیم. پس از این مراحل به سراغ صحه گذاری میرویم و مثال‌هایی را با روش‌های مختلف برای این منظور انجام میدهیم. در واقع در این مرحله لازم است که مقدار درجات آزادی مختلف ربات داده شود و سپس به کمک روابط به دست آمده از سینماتیک مستقیم، موقعیت End effector را تعیین نماییم.

۱-۱- پیدا کردن درجات آزادی و نام گذاری آنها

این مرحله درواقع در همان بخش سوم مقدمه انجام شد اما برای کامل بودن سلسله مراحل کار لازم است تا بار دیگر در این قسمت به آن اشاره کنیم. رباتی که با آن سر و کار داریم، یک ربات ۶ درجه آزادی میباشد که همه مفاصل آن از نوع Revolute میباشند. نام گذاری مفاصل و همچنین لینک‌ها در شکل زیر آمده است. لینک‌ها از ۰ تا ۶ شماره گذاری شده اند که لینک ۰ مربوط به Base میباشد. مفاصل هم از θ_1 تا θ_6 شماره گذاری شده‌اند که میتوانید در شکل زیر مشاهده کنید. توضیحات مربوط به هریک از مفاصل در مقدمه آمده است که از تکرار آنها در این قسمت خودداری میکنیم.



شکل ۵- درجات آزادی ربات

۲-۲- انتخاب فریم‌ها

برای انتخاب فریم‌های مناسب براساس اسلایدهای درس پیش میرویم. برای این منظور ابتدا باید فریم گذاری مربوط به لینک ۱ تا ۵ را براساس توضیحات زیر انجام دهیم. (فریم گذاری همه لینک‌ها به جز اولین و آخرین لینک)

- ۱- محور \hat{Z}_i از فریم $\{i\}$ را، \hat{Z}_i می‌نامیم که منطبق بر محوری است که مفصل λ_i حول آن دوران دارد.
- ۲- مبدأ این فریم در محل تقاطع عمود مشترک بین محور مفصل i و مفصل $i+1$ با محور λ_i میباشد. (جایی که a_i متقاطع با \hat{Z}_i میباشد). چنانچه دو محور i و $i+1$ متقاطع باشند، محل قرارگیری مبدأ در همان نقطه تقاطع میباشد.
- ۳- محور \hat{X}_i در امتداد عمود مشترک بین محور i و $i+1$ میباشد یا به عبارتی در امتداد a_i به سمت محور $i+1$ میباشد.
- ۴- محور \hat{Y}_i به کمک قاعده دست راست به دست می‌آید تا فریم λ_i را کامل کند.

دراينجا باتوجه به اينکه مفاصل همگي Revolute اند فقط قاعده مربوط به لينک صفرم و آخر مربوط به مفصل Revolute را كافي است که مرور کنيم.

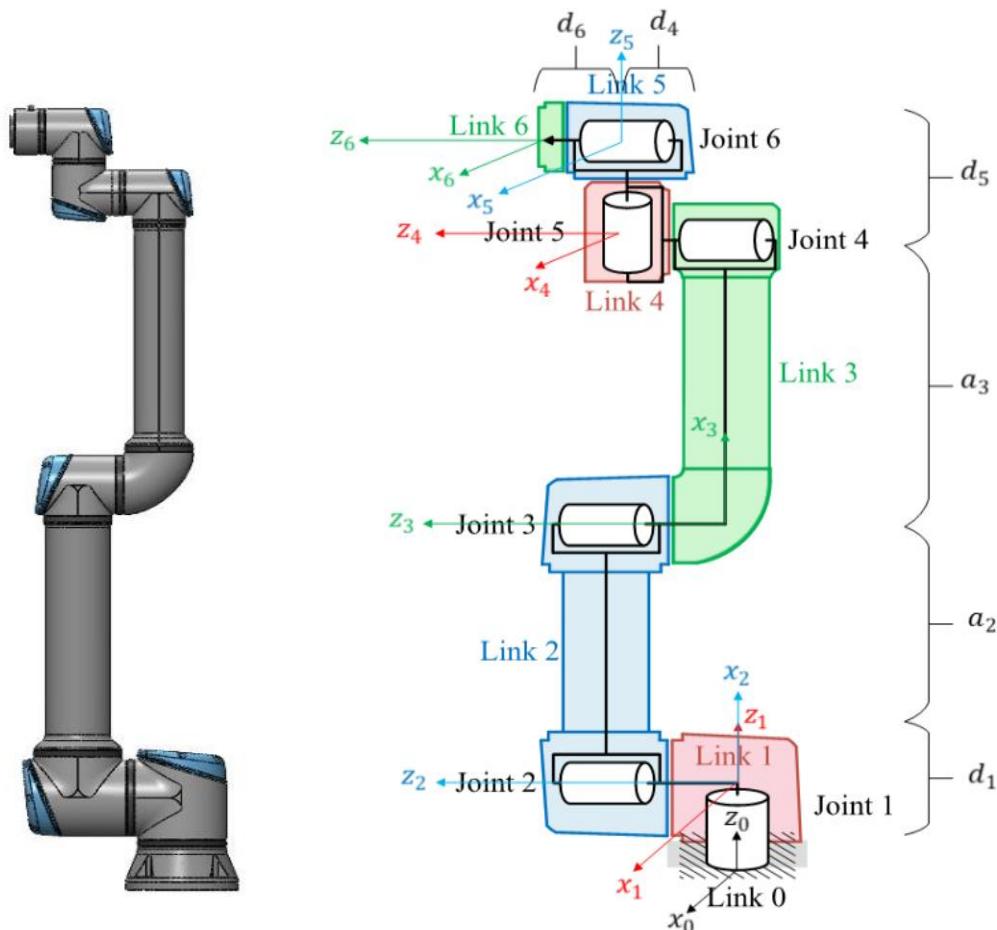
- برای فریم $\{0\}$ باید به صورت زیر عمل کنیم:

این فریم دلخواه است ولی برای آنکه در محاسبات راحت باشیم، \hat{Z}_0 را در جهت محور مفصل ۱ انتخاب میکنیم (مانند \hat{Z}_1) و باید فریم‌های $\{0\}$ و $\{1\}$ به گونه‌ای قرار گیرند که اگر $\theta_1 = 0$ شود، آنگاه این دو فریم روی هم قرار گیرند.

- برای فریم $\{n\}$ بهتر است به صورت زیر عمل میکنیم:

باید \hat{X}_n به گونه‌ای قرار گیرد که چنانچه $\theta_n = 0$ شد، سپس \hat{X}_{n-1} قرار گیرد. همچنین باید به نحوی باشد که بین \hat{X}_n و \hat{X}_{n+1} در راستای \hat{Z}_{n+1} هیچ گونه فاصله‌ای نباشد و در حقیقت $d_n = 0$ باشد.

آنچه در صفحه قبل گفته شد فریم گذاری اسندارد است اما برای مثال به خصوص در فریم $\{0\}$ و $\{6\}$ میتوان کمی متفاوت تر عمل کرد. در نهایت با توجه به توضیحات داده شده در صفحه قبل میتوان فریم گذاری را به صورت زیر انجام داد. دقت شود که x_2 و x_3 در راستای Z_3 فاصله‌شان \cdot است.



شکل ۶- فریم گذاری ربات

۳-۲- استخراج جدول پارامترهای DH

برای این منظور ابتدا لازم است تا ابتدا ابعاد مربوطه را به دست آوریم. برای این کار هم میتوان از کاتالوگ مربوط به این ربات استفاده کرد و هم از فایل CAD اندازه‌ها را به دست آورد. به هرجهت باید اندازه‌ها به نحوی باشد تا بعدا برای صحه سنجی دچار مشکل نشویم. اندازه‌ها در جدول زیر قرار گرفته است.

جدول ۱- ابعاد مربوط به ربات

	اندازه (بر حسب mm)
d_1	128
a_2	612.9
a_3	571.6
d_4	163.9
d_5	115.7
d_6	92.2

برای به دست آوردن جدول DH مطابق اسلاید، تعاریف زیر را برای هر پارامتر داریم:

- \hat{X}_{i-1} : فاصله میان محورهای \hat{Z}_{i-1} و \hat{Z}_i در راستای محور \hat{Z}_{i-1}
- α_{i-1} : زاویه میان محورهای \hat{Z}_{i-1} و \hat{Z}_i حول محور \hat{Z}_{i-1}
- d_i : فاصله میان محورهای \hat{X}_{i-1} و \hat{X}_i در راستای محور \hat{Z}_{i-1}
- θ_i : زاویه میان محورهای \hat{X}_{i-1} و \hat{X}_i در راستای محور \hat{Z}_{i-1}

براین اساس برای جدول DH داریم:

جدول ۲- جدول پارامترهای DH

i	α_{i-1} (rad)	a_{i-1} (mm)	d_i (mm)	θ_i (rad)
1	0	0	128	θ_1
2	$\frac{\pi}{2}$	0	0	θ_2
3	0	612.9	0	θ_3
4	0	571.6	163.9	θ_4
5	$-\frac{\pi}{2}$	0	115.7	θ_5
6	$\frac{\pi}{2}$	0	92.2	θ_6

۴-۲- ماتریس‌های تبدیل همگن

در گام بعدی لازم است تا براساس جدول DH ماتریس‌های تبدیل همگن بین هردو لینک متوالی را بیابیم. برای این منظور باتوجه به اطلاعات هر سطر باید از فرمول کلی زیر استفاده نماییم:

$${}^{i-1}T_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i & c\alpha_{i-1} & c\theta_i & -s\alpha_{i-1} \\ s\theta_i & s\alpha_{i-1} & c\theta_i & c\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

حال براین اساس برای n ‌های ۱ تا ۶ باید این ماتریس‌ها را محاسبه نماییم. کافی است مقادیر α_{i-1} و a_{i-1} و d_i و θ_i را از جدول DH در ماتریس فوق جاگذاری نماییم.

$${}^0T_1 = \begin{bmatrix} \cos \theta_1 & -\sin \theta_1 & 0 & 0 \\ \sin \theta_1 & \cos \theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^1T_2 = \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin \theta_2 & \cos \theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^2T_3 = \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 & a_2 \\ \sin \theta_3 & \cos \theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^3T_4 = \begin{bmatrix} \cos \theta_4 & -\sin \theta_4 & 0 & a_3 \\ \sin \theta_4 & \cos \theta_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^4T_5 = \begin{bmatrix} \cos \theta_5 & -\sin \theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ -\sin \theta_5 & -\cos \theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^5T_6 = \begin{bmatrix} \cos \theta_6 & -\sin \theta_6 & 0 & 0 \\ 0 & 0 & -1 & -d_6 \\ \sin \theta_6 & \cos \theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

۲-۵-۲- سینماتیک مستقیم موقعیت و جهت گیری

برای این منظور باید ابتدا ${}^0T_6^0$ بیابیم. با توجه به ماتریس‌های تبدیل همگن به دست آمده در مرحله قبل و ضرب آن‌ها این امر امکان‌پذیر می‌باشد:

$${}^0T_6 = {}^0T_1 \ {}^1T_2 \ {}^2T_3 \ {}^3T_4 \ {}^4T_5 \ {}^5T_6$$

$${}^0T_6 =$$

$$= \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} c_6(-s_1s_5 + c_1c_5c_{234}) - s_6s_{234}c_1 & -s_6(-s_1s_5 + c_1c_5c_{234}) - s_{234}c_1c_6 & s_1c_5 + s_5c_1c_{234} & c_1(a_2c_2 + a_3c_{23}) + d_4s_1 - d_5s_{234}c_1 + d_6(s_1c_5 + s_5c_1c_{234}) \\ c_6(s_5c_1 + s_1c_5c_{234}) - s_6s_{234}s_1 & -s_6(c_1s_5 + s_1c_5c_{234}) - s_{234}s_1c_6 & -c_1c_5 + s_5s_1c_{234} & s_1(a_2c_2 + a_3c_{23}) - d_4c_1 - d_5s_{234}s_1 - d_6(c_1c_5 - s_5s_1c_{234}) \\ s_6c_{234} + s_{234}c_5c_6 & c_6c_{234} - s_{234}c_5s_6 & s_5s_{234} & a_2s_2 + a_3s_{23} + d_1 + d_5c_{234} + d_6s_5s_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

حال از برابر قرار دادن طرفین میتوان معادلات مربوط به سینماتیک مستقیم موقعیت و جهت گیری را به دست آورد:

$$r_{11} = c_6(-s_1s_5 + c_1c_5c_{234}) - s_6s_{234}c_1$$

$$r_{21} = c_6(s_5c_1 + s_1c_5c_{234}) - s_6s_{234}s_1$$

$$r_{31} = s_6c_{234} + s_{234}c_5c_6$$

$$r_{12} = -s_6(-s_1s_5 + c_1c_5c_{234}) - s_{234}c_1c_6$$

$$r_{22} = -s_6(c_1s_5 + s_1c_5c_{234}) - s_{234}s_1c_6$$

$$r_{32} = c_6c_{234} - s_{234}c_5s_6$$

$$r_{13} = s_1c_5 + s_5c_1c_{234}$$

$$r_{23} = -c_1c_5 + s_5s_1c_{234}$$

$$r_{33} = s_5s_{234}$$

$$p_x = c_1(a_2c_2 + a_3c_{23}) + d_4s_1 - d_5s_{234}c_1 + d_6(s_1c_5 + s_5c_1c_{234})$$

$$p_y = s_1(a_2c_2 + a_3c_{23}) - d_4c_1 - d_5s_{234}s_1 - d_6(c_1c_5 - s_5s_1c_{234})$$

$$p_z = a_2s_2 + a_3s_{23} + d_1 + d_5c_{234} + d_6s_5s_{234}$$

$$\rightarrow {}^0R_6 = \begin{bmatrix} c_6(-s_1s_5 + c_1c_5c_{234}) - s_6s_{234}c_1 & -s_6(-s_1s_5 + c_1c_5c_{234}) - s_{234}c_1c_6 & s_1c_5 + s_5c_1c_{234} \\ c_6(s_5c_1 + s_1c_5c_{234}) - s_6s_{234}s_1 & -s_6(c_1s_5 + s_1c_5c_{234}) - s_{234}s_1c_6 & -c_1c_5 + s_5s_1c_{234} \\ s_6c_{234} + s_{234}c_5c_6 & c_6c_{234} - s_{234}c_5s_6 & s_5s_{234} \end{bmatrix}$$

$$\rightarrow {}^0P_6 = \begin{bmatrix} c_1(a_2c_2 + a_3c_{23}) + d_4s_1 - d_5s_{234}c_1 + d_6(s_1c_5 + s_5c_1c_{234}) \\ s_1(a_2c_2 + a_3c_{23}) - d_4c_1 - d_5s_{234}s_1 - d_6(c_1c_5 - s_5s_1c_{234}) \\ a_2s_2 + a_3s_{23} + d_1 + d_5c_{234} + d_6s_5s_{234} \end{bmatrix}$$

پس از تعیین ماتریس تبدیل عضو آخر ربات نسبت به دستگاه پایه می‌توان ماتریس دوران آن که بیانگر وضعیت دورانی عضو انتهایی ربات است را از ماتریس تبدیل استخراج نمود. برای این موضوع طبق اسلایدها ۴ روش داشتیم که در ادامه به کمک هریک از روش‌ها می‌توان جهت‌گیری نسبت به دستگاه پایه را بیان کرد.

Z-Y-X Euler angles (روش اول)

$$R_{ZYX}(\alpha, \beta, \gamma) = \begin{bmatrix} c\alpha & c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & \\ -s\beta & c\beta s\gamma & c\beta c\gamma & \end{bmatrix}$$

$${}^0R_6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

حال از برابر قرار دادن این دو ماتریس خواهیم داشت:

$$\begin{aligned} \beta &= \text{atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right) \\ \rightarrow \begin{cases} \alpha = \text{atan2}\left(\frac{r_{21}}{\cos \beta}, \frac{r_{11}}{\cos \beta}\right) \\ \gamma = \text{atan2}\left(\frac{r_{32}}{\cos \beta}, \frac{r_{33}}{\cos \beta}\right) \end{cases} \\ \rightarrow \begin{cases} \beta = \text{atan2}\left(-(s_6 c_{234} + s_{234} c_5 c_6), \sqrt{(c_6(-s_1 s_5 + c_1 c_5 c_{234}) - s_6 s_{234} c_1)^2 + (c_6(s_5 c_1 + s_1 c_5 c_{234}) - s_6 s_{234} s_1)^2}\right) \\ \alpha = \text{atan2}\left(\frac{c_6(s_5 c_1 + s_1 c_5 c_{234}) - s_6 s_{234} s_1}{\cos \beta}, \frac{c_6(-s_1 s_5 + c_1 c_5 c_{234}) - s_6 s_{234} c_1}{\cos \beta}\right) \\ \gamma = \text{atan2}\left(\frac{c_6 c_{234} - s_{234} c_5 s_6}{\cos \beta}, \frac{s_5 s_{234}}{\cos \beta}\right) \end{cases} \end{aligned}$$

X-Y-Z Fixed angles (روش دوم)

$$R_{XYZ}(\gamma, \beta, \alpha) = \begin{bmatrix} c\alpha & c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & \\ -s\beta & c\beta s\gamma & c\beta c\gamma & \end{bmatrix}$$

$${}^0R_6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

حال از برابر قرار دادن این دو ماتریس نتایج مثل مرحله قبل خواهد شد و خواهیم داشت:

$$\begin{aligned} & \left\{ \begin{array}{l} \beta = \text{atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right) \\ \alpha = \text{atan2}\left(\frac{r_{21}}{\cos \beta}, \frac{r_{11}}{\cos \beta}\right) \\ \gamma = \text{atan2}\left(\frac{r_{32}}{\cos \beta}, \frac{r_{33}}{\cos \beta}\right) \end{array} \right. \\ \rightarrow & \left\{ \begin{array}{l} \beta = \text{atan2}\left(-(s_6 c_{234} + s_{234} c_5 c_6), \sqrt{(c_6(-s_1 s_5 + c_1 c_5 c_{234}) - s_6 s_{234} c_1)^2 + (c_6(s_5 c_1 + s_1 c_5 c_{234}) - s_6 s_{234} s_1)^2}\right) \\ \alpha = \text{atan2}\left(\frac{c_6(s_5 c_1 + s_1 c_5 c_{234}) - s_6 s_{234} s_1}{\cos \beta}, \frac{c_6(-s_1 s_5 + c_1 c_5 c_{234}) - s_6 s_{234} c_1}{\cos \beta}\right) \\ \gamma = \text{atan2}\left(\frac{c_6 c_{234} - s_{234} c_5 s_6}{\cos \beta}, \frac{s_5 s_{234}}{\cos \beta}\right) \end{array} \right. \end{aligned}$$

روش سوم (Equivalent Angle-Axis)

$$\begin{aligned} & \begin{bmatrix} k_x k_x v\theta + c\theta & k_x k_y v\theta - k_z s\theta & k_x k_z v\theta + k_y s\theta \\ k_x k_y v\theta + k_z s\theta & k_y k_y v\theta + c\theta & k_y k_z v\theta - k_x s\theta \\ k_x k_z v\theta - k_y s\theta & k_y k_z v\theta + k_x s\theta & k_y k_y v\theta + c\theta \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \\ \rightarrow & \left\{ \begin{array}{l} \theta = A \cos\left(\frac{r_{11} + r_{22} + r_{33} - 1}{2}\right) \\ \hat{K} = \frac{1}{2 \sin \theta} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \end{array} \right. \end{aligned}$$

روش چهارم (Euler Parameters)

$$\rightarrow \left\{ \begin{array}{l} \epsilon_4 = \frac{1}{2} \sqrt{1 + r_{11} + r_{22} + r_{33}} \\ \epsilon = \frac{1}{4\epsilon_4} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \end{array} \right.$$

۶-۲- بررسی سینماتیک مستقیم برای چندین وضعیت متفاوت

برای بررسی جواب‌ها ما کد را به زبان پایتون در فایل Jupyter notebook نوشته‌ایم. در ادامه نیز برای چندین حالت کارکرد قسمت سینماتیک مستقیم را بررسی می‌کنیم.

الف) بررسی وضعیت صفر ربات

در این حالت باید تمامی زوایای ورودی را برابر با صفر درجه بگذاریم. در این صورت خواهیم داشت

A) Zero state

- in this state all joint variables have zero value

```
In [10]: theta1 = math.radians(0)
theta2 = math.radians(0)
theta3 = math.radians(0)
theta4 = math.radians(0)
theta5 = math.radians(0)
theta6 = math.radians(0)

desired_theta = [theta1, theta2, theta3, theta4, theta5, theta6]
T_0_6_A = forward(desired_theta)

print('T_0_6 : ')
for row in T_0_6_A:
    print(row)

T_0_6 :
[1.0, 0.0, 0.0, 1184.5]
[0.0, 6.123233995736766e-17, -1.0, -256.1]
[0.0, 1.0, 6.123233995736766e-17, 243.7]
[0.0, 0.0, 0.0, 1.0]
```

ب) بررسی وضعیت کاملاً کشیده

باتوجه به فریم گذاری که داشتیم، زمانی ربات به حالت کشیده در می‌آید که زوایای تنا به صورت زیر باشند:

تنا۱: دلخواه | تنا۲: ۹۰ درجه | تنا۳: ۰ درجه | تنا۴: ۹۰ درجه | تنا۵: ۰ درجه | تنا۶: دلخواه

که تنا۱ را مقدار ۴۵ درجه و تنا۶ را هم مقدار ۶۰ درجه به عنوان مقادیر دلخواه می‌دهیم.

B) fully-stretched state

- We have below values for joint variables in fully-stretched state:

theta1 = arbitrary | theta2 = 0 | theta3 = 90 | theta4 = 0 | theta5 = -90 | theta6 = arbitrary

We consider theta1 = -20 and theta6 = 60 degrees.

```
In [11]: theta1 = math.radians(45)
theta2 = math.radians(90)
theta3 = math.radians(0)
theta4 = math.radians(-90)
theta5 = math.radians(0)
theta6 = math.radians(60)

desired_theta = [theta1, theta2, theta3, theta4, theta5, theta6]
T_0_6_B = forward(desired_theta)

print('T_0_6 : ')
for row in T_0_6_B:
    print(row)

T_0_6 :
[0.3535533905932738, -0.6123724356957946, 0.7071067811865476, 181.09004666187485]
[0.3535533905932739, -0.6123724356957946, -0.7071067811865476, -181.09004666187474]
[0.8660254037844386, 0.5000000000000001, 6.123233995736766e-17, 1428.2]
[0.0, 0.0, 0.0, 1.0]
```

ج) بررسی یک وضعیت دلخواه

برای بررسی وضعیت دلخواه ما زوایای مفاصل را به صورت دلخواه انتخاب کردیم و برابر مقادیر زیر است:

تتا_۱: ۳۰ درجه | تتا_۲: ۴۵ درجه | تتا_۳: ۲۶ درجه | تتا_۴: ۵۰ درجه | تتا_۵: ۶۰ درجه | تتا_۶: ۸۰ درجه

در این صورت خواهیم داشت:

C) Arbitrary state

- We have below values for joint variables in arbitrary state:

```
theta1 = 30 | theta2 = 45 | theta3 = 26 | theta4 = 50 | theta5 = 60 | theta6 = 80
```

```
In [12]: theta1 = math.radians(30)
theta2 = math.radians(45)
theta3 = math.radians(26)
theta4 = math.radians(50)
theta5 = math.radians(60)
theta6 = math.radians(80)

desired_theta = [theta1, theta2, theta3, theta4, theta5, theta6]
T_0_6_C = forward(desired_theta)

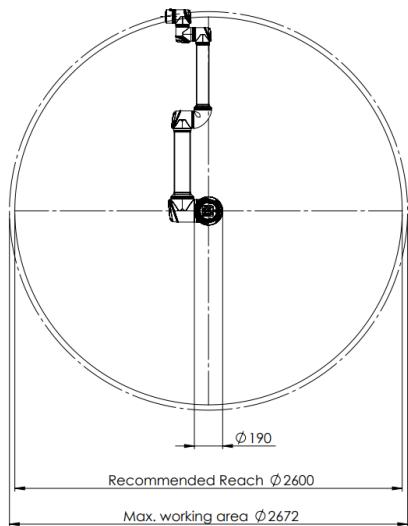
print('T_0_6 : ')
for row in T_0_6_C:
    print(row)

T_0_6 :
[-0.8449695581951862, 0.5171601307605163, -0.1362785561825407, 519.983544410926]
[-0.3141952242140994, -0.6862252123034819, -0.6560307302863893, 57.72552628018109]
[-0.432790719406602, -0.5115079248172085, 0.7423286577013642, 1110.696960974777]
[0.0, 0.0, 0.0, 1.0]
```

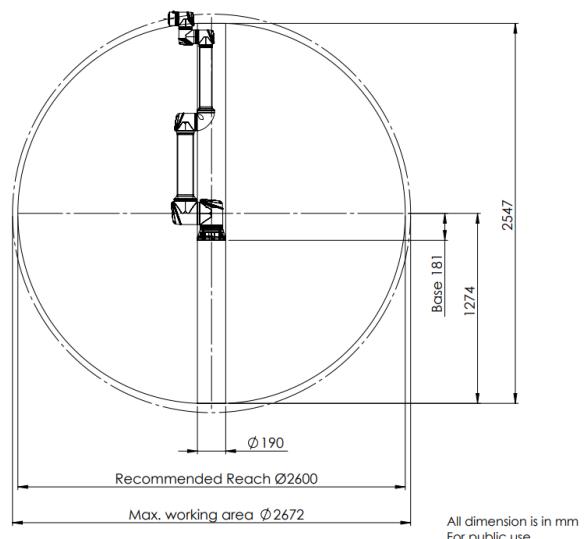
۷-۲- ترسیم فضای کاری

باتوجه به روابط به دست آمده در قسمت‌های قبل برای موقعیت End Effector میتوان به راحتی فضای کاری آن را به ازای ورودی دادن برای تناهای مختلف ترسیم کرد.

UR10 working area, top view

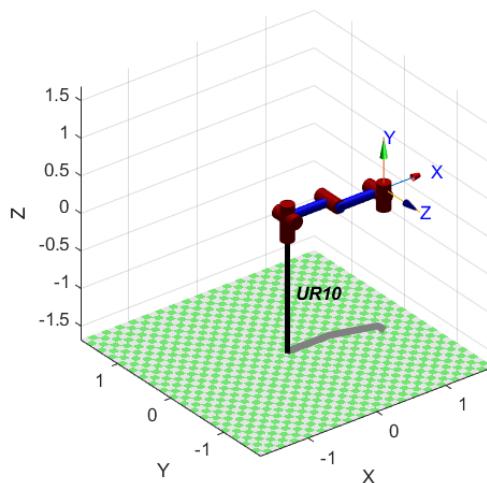


UR10 working area, side view



۲-۸-۲- صحه گذاری سینماتیک مستقیم

در این قسمت به کمک ابزار پیترکورک میتوان به صحه گذاری سینماتیک مستقیم پرداخت.



شکل ۷- وضعیت صفر ربات با ابزار پیترکورک

```

UR10::: 6 axis, RRRRRR, stdDH, slowRNE
+---+-----+-----+-----+-----+
| j | theta | d | a | alpha | offset |
+---+-----+-----+-----+-----+
| 1| q1 | 0.128 | 0 | 0 | 0 |
| 2| q2 | 0 | 0 | 1.5708 | 0 |
| 3| q3 | 0 | 0.6129 | 0 | 0 |
| 4| q4 | 0.16389 | 0.5716 | 0 | 0 |
| 5| q5 | 0.1157 | 0 | -1.5708 | 0 |
| 6| q6 | 0.0922 | 0 | 1.5708 | 0 |
+---+-----+-----+-----+-----+
>> Robot.fkine([0 0 0 0 0 0])

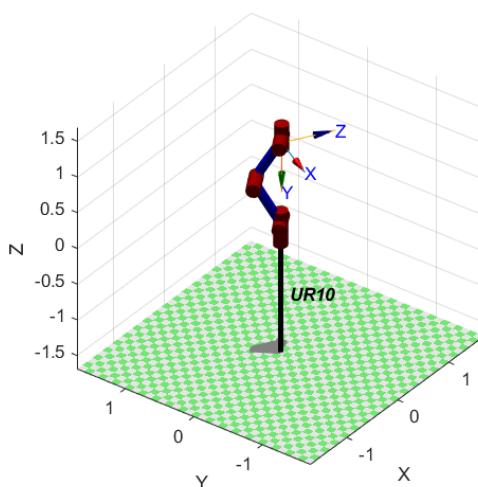
ans =
    1         0         0      1.184
    0         0        -1     -0.2796
    0         1         0      0.2202
    0         0         0         1
>>

```

شکل ۸- صحه سنجی ماتریس تبدیل انداکتور به پایه در وضعیت صفر با پیترکورک

$$T_e = \begin{pmatrix} 1.0000 & 0 & 0 & 1.1845 \\ 0 & 0 & -1.0000 & -0.2561 \\ 0 & 1.0000 & 0 & 0.2437 \\ 0 & 0 & 0 & 1.0000 \end{pmatrix}$$

شکل ۹- ماتریس تبدیل انداکتور به پایه در وضعیت صفر به کمک ماتریس‌های تبدیل



شکل ۱۰- وضعیت ربات در زاویه دلخواه ۶۰ درجه با ابزار پیترکورک

```

UR10:: 6 axis, RRRRRR, stdDH, slowRNE
+---+-----+-----+-----+-----+
| j |     theta |      d |      a |    alpha |   offset |
+---+-----+-----+-----+-----+
| 1|     q1| 0.128|      0|      0|      0|
| 2|     q2|      0|      0| 1.5708|      0|
| 3|     q3|      0| 0.6129|      0|      0|
| 4|     q4| 0.16389| 0.5716|      0|      0|
| 5|     q5| 0.1157|      0| -1.5708|      0|
| 6|     q6| 0.0922|      0| 1.5708|      0|
+---+-----+-----+-----+-----+
>> Robot.fkine([pi/3 pi/3 pi/3 pi/3 pi/3 pi/3])

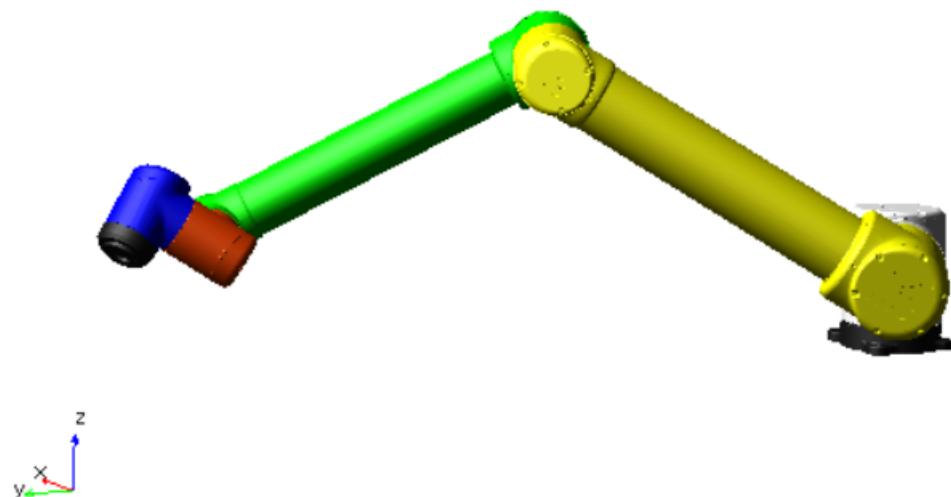
ans =
-0.5000      0     0.8660    0.2318
-0.8660      0    -0.5000    0.1577
      0     -1       0     1.062
      0       0       0       1
>> |

```

شکل ۱۱- ماتریس تبدیل با زوایای ۶۰ درجه در پیترکورک

T_e				
-0.5000	0.8660	-0.0000	0.1523	
-0.0000	-0.0000	-1.0000	-0.1563	
-0.8660	-0.5000	0.0000	1.0381	
0	0	0	1.0000	

شکل ۱۲- ماتریس تبدیل با زوایای ۶۰ درجه با ماتریس های تبدیل



شکل ۱۳- نمایش ربات در زوایای ۶۰ درجه در ادمنز

فصل سوم
سینماتیک معکوس

۳-۱-۳- حل تحلیلی و بررسی تعداد جواب

در مسئله سینماتیک معکوس در حقیقت به دنبال آن هستیم که با داشتن مکان و چرخش End-effector نسبت به فریم پایه، بتوانیم مقادیر زوایای مفاصل را بیابیم و درواقع بیابیم که با چه تناهایی میتوان به موقعیت دلخواه رسید. از نظر تعداد جوابها ممکن است اصلاً برای آن موقعیت خواسته شده، جوابی پیدا نشود یا حتی چندین جواب پیدا شود.

ربات UR10 از آنجایی که محور ۳ جوینت آخر مربوط به Wrist باهم در یک نقطه intersection ندارند، میتوان گفت که مسئله decouple نمیباشد که مانند اسلایدها از روش Piper حل کنیم. در ادامه از روش هندسی و جبری مسئله سینماتیک معکوس را حل کردیم.

ابتدا مسئله سینماتیک معکوس را به روش جبری و با تکنیک separating out variables حل می‌کنیم.

یافتن زاویه θ_1 :

$${}^0T_6 = {}^0T_1 \ {}^1T_2 \ {}^2T_3 \ {}^3T_4 \ {}^4T_5 \ {}^5T_6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

سمت راست رابطه بالا همان ماتریس تبدیل مطلوب و ورودی مسئله می‌باشد. بنابراین مقادیر عددی درایه‌های آن برای ما معلوم است.

رابطه بالا را از سمت چپ در وارون ماتریس 0T_1 ضرب می‌کنیم تا زاویه θ_1 تنها مجهول در سمت راست تساوی باشد.

$${}^1T_6|_{para} = {}^1T_2 \ {}^2T_3 \ {}^3T_4 \ {}^4T_5 \ {}^5T_6 = ({}^0T_1)^{-1} \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

در اینصورت ماتریس تبدیل 1T_6 پارامتریک به شکل زیر حاصل می‌شود:

$$\begin{bmatrix} c_{234}c_5c_6 - s_{234}s_6 & -s_{234}c_6 - c_{234}c_5s_6 & c_{234}s_5 & a_2c_2 + a_3c_{23} - d_5s_{234} + d_6c_{234}s_5 \\ c_6s_5 & -s_6s_5 & -c_5 & -d_4 - d_6c_5 \\ s_{234}c_5c_6 + c_{234}s_6 & c_{234}c_6 - s_{234}c_5s_6 & s_{234}s_5 & a_2s_2 + a_3s_{23} + d_5c_{234} + d_6s_{234}s_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ماتریس تبدیل 1T_6 عددی نیز به فرم زیر می‌باشد:

$${}^1T_6|_{Num} = ({}^0T_1)^{-1} \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$${}^1T_6|_{Num} = \begin{bmatrix} r_{11}c_1 + r_{21}s_1 & r_{12}c_1 + r_{22}s_1 & r_{13}c_1 + r_{23}s_1 & p_xc_1 + p_ys_1 \\ r_{21}c_1 - r_{11}s_1 & r_{22}c_1 - r_{12}s_1 & r_{23}c_1 - r_{13}s_1 & p_yc_1 - p_xs_1 \\ r_{31} & r_{32} & r_{33} & p_z - d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

از تساوی درایه‌های T_{23} و T_{24} دو ماتریس عددی و پارامتریک خواهیم داشت:

$$-c_5 = r_{23}c_1 - r_{13}s_1 \quad (5)$$

$$-d_4 - d_6c_5 = p_yc_1 - p_xs_1 \quad (6)$$

با جایگذاری مقدار عبارت c_5 از رابطه (5) در رابطه (6) خواهیم داشت:

$$-d_4 + d_6(r_{23}c_1 - r_{13}s_1) = p_yc_1 - p_xs_1 \quad (7)$$

با کمی ساده سازی، این معادله به فرم معادله نوع اول مثلثاتی جزوه در می‌آید.

$$(p_y - d_6r_{23})c_1 + (d_6r_{13} - p_x)s_1 = -d_4 \quad (8)$$

پاسخ این معادله از رابطه (9) قابل دسترسی است

$$\theta_1 = 2Atan\left(\frac{b \pm \sqrt{b^2 + a^2 - c^2}}{a + c}\right) \quad (9)$$

که در آن پارامترهای c, b, a تعریفی مطابق زیر دارند:

$$a = \text{ضریب کسینوس زاویه}$$

$$b = \text{ضریب سینوس زاویه}$$

$$c = \text{مقدار ثابت سمت راست تساوی}$$

$$\begin{cases} a = p_y - d_6r_{23} \\ b = d_6r_{13} - p_x \\ c = -d_4 \end{cases}$$

با توجه به علامت \pm موجود در آرگومان تانژات وارون، دو پاسخ برای زاویه θ_1 خواهیم داشت.

یافتن زاویه θ_5 :

چون در قسمت قبل مقدار زاویه θ_1 را پیدا کردیم، اکنون به راحتی می‌توان از رابطه (۵)، زاویه θ_5 را پیدا نمود:

$$-c_5 = r_{23}c_1 - r_{13}s_1 \Rightarrow c_5 = r_{13}s_1 - r_{23}c_1$$

می‌دانیم تابع کسینوس تابعی زوج است، یعنی $c_5 = c_{-5}$

پس به ازای هر مقدار از زاویه θ_1 ، دو مقدار برای θ_5 بدست می‌آید:

$$\theta_5 = \pm A\cos(r_{13}s_1 - r_{23}c_1) \quad (9)$$

با توجه به اینکه θ_1 خود دو مقدار دارد، چهار پاسخ برای θ_5 حاصل می‌شود.

یافتن زاویه $\theta_2 + \theta_3 + \theta_4$:

از این قسمت به بعد، زوایای مفصلی بطور مستقیم از ماتریس تبدیل فریم نهایی به زمین، قابل محاسبه است.

از آنجا که محورهای دوران مفاصل ۲ و ۳ و ۴ موازی اند، ابتدا مقدار مجموع این زوایا بدست می‌آید.

از ماتریس تبدیل فریم نهایی به زمین (0T_6) داریم:

$$\begin{aligned} r_{13} &= s_1c_5 + s_5c_1c_{234} \Rightarrow c_{234} = \frac{r_{13} - s_1c_5}{s_5c_1} \\ r_{33} &= s_5s_{234} \Rightarrow s_{234} = \frac{r_{33}}{s_5} \\ \theta_{234} &= Atan2\left(\frac{r_{33}}{s_5}, \frac{r_{13} - s_1c_5}{s_5c_1}\right) \end{aligned} \quad (10)$$

زاویه θ_1 ، دارای دو تعدد پاسخ و زاویه θ_5 ، دارای چهار تعدد پاسخ می‌باشد.

بنابراین θ_{234} دارای $2 \times 4 = 8$ تعدد پاسخ خواهد بود.

یافتن زاویه θ_6 :

$$r_{32} = c_{234}c_6 - s_{234}c_5s_6$$

$$r_{31} = c_{234}s_6 + s_{234}c_5c_6$$

با کمی دقت می‌توان دریافت که این دستگاه معادلات، به فرم معادلات جبری نوع سوم اند که در جزو درمورد روش حل آنها بحث شده است.

$$k_1 = c_{234}, k_2 = s_{234}c_5$$

$$r_{32} = k_1c_6 - k_2s_6$$

$$r_{31} = k_1s_6 + k_2c_6$$

$$\theta_6 = \text{Atan2}(r_{31}, r_{32}) - \text{Atan2}(s_{234}c_5, c_{234}) \quad (11)$$

یافتن زاویه θ_3 :

$$p_y = s_1(a_2c_2 + a_3c_{23}) - d_4c_1 - d_5s_{234}s_1 - d_6(c_1c_5 - s_5s_1c_{234})$$

$$p_z = a_2s_2 + a_3s_{23} + d_1 + d_5c_{234} + d_6s_5s_{234}$$

با کمی مرتب سازی و گرفتن تغییر متغیر داریم:

$$\frac{p_y + d_4c_1 + d_5s_{234}s_1 + d_6(c_1c_5 - s_5s_1c_{234})}{s_1} = F_2 = a_2c_2 + a_3c_{23}$$

$$p_z - d_1 - d_5c_{234} - d_6s_5s_{234} = F_3 = a_2s_2 + a_3s_{23}$$

$$F_2^2 + F_3^2 = a_2^2 + a_3^2 + 2a_2a_3c_3 \Rightarrow c_3 = \frac{F_2^2 + F_3^2 - a_2^2 - a_3^2}{2a_2a_3}$$

$$\theta_3 = \pm \text{Acos} \left(\frac{F_2^2 + F_3^2 - a_2^2 - a_3^2}{2a_2a_3} \right) \quad (12)$$

یافتن زاویه θ_2 :

با تعریف دو پارامتر Q_1, Q_2 می‌توان به فرم سوم معادلات جبری دست یافت:

$$\begin{cases} Q_1 = a_2 + a_3 c_3 \\ Q_2 = a_3 s_3 \end{cases} \Rightarrow \begin{cases} F_2 = Q_1 c_2 - Q_2 s_2 \\ F_3 = Q_1 s_2 + Q_2 c_2 \end{cases}$$

$$\theta_2 = \text{Atan2}(F_3, F_2) - \text{Atan2}(Q_2, Q_1) \quad (13)$$

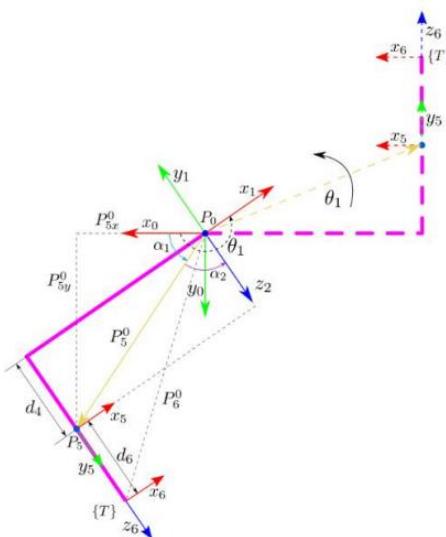
یافتن زاویه θ_4 :

$$\theta_4 = (\theta_2 + \theta_3 + \theta_4) - \theta_2 - \theta_3$$

۳-۲-۳- حل سینماتیک معکوس به کمک روش هندسی

در اینجا با کمک گرفتن از مقالاتی که وجود دارد میتوان به حل سینماتیک معکوس پرداخت.

برای یافتن زاویه‌ی مفصل اول یعنی θ_1 ابتدا به تصویر عمودی ربات که در زیر آمده است دقت کنید. همانطور که در شکل نشان داده شده است ربات حول z_1 به اندازه تتا ۱ به صورت پاد ساعتگرد میچرخد تا از موقعیت اولیه خود که با خط چین نشان داده شده است به موقعیتی که با خط توپر نشان داده است، برسد. در این حالت تصویر بردار P_{5xy}^0 روی محورهای x_0 و y_0 به ترتیب P_{5x}^0 و P_{5y}^0 میباشد.



شکل ۱۴ - به دست آوردن تتا ۱

در نهایت به معادلات زیر میرسیم:

$$\theta_1 = \alpha_1 + \alpha_2 + \frac{\pi}{2}, \quad \alpha_1 = \text{atan2}(y, x) = \text{atan2}(P_{5y}^0, P_{5x}^0)$$

$$\cos(\alpha_2) = \frac{d_4}{P_{5xy}^0}, \quad \alpha_2 = \pm \arccos\left(\frac{d_4}{\sqrt{P_{5x}^0{}^2 + P_{5y}^0{}^2}}\right).$$

$$\theta_1 = \text{atan2}(P_{5y}^0, P_{5x}^0) \pm \arccos\left(\frac{d_4}{\sqrt{P_{5x}^0{}^2 + P_{5y}^0{}^2}}\right) + \frac{\pi}{2}$$

حال در معادله فوق باید مقدار P_{5x}^0 و P_{5y}^0 را تعیین کنیم.

$$P_5^0 = P_6^0 - d_6 \cdot \hat{z}_6$$

$${}_6^0T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, P_6^0 = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}, \hat{z}_6 = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

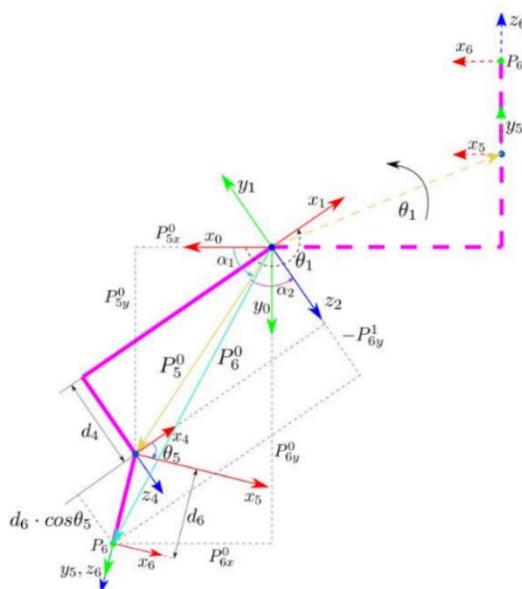
$$\begin{bmatrix} P_5^0 \\ 1 \end{bmatrix} = {}_6^0T \cdot \begin{bmatrix} 0 \\ 0 \\ d_6 \\ 1 \end{bmatrix} = \begin{bmatrix} p_x - d_6 \cdot a_x \\ p_y - d_6 \cdot a_y \\ p_z - d_6 \cdot a_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_{5x}^0 \\ P_{5y}^0 \\ P_{5z}^0 \\ 1 \end{bmatrix}$$

$$P_{5x}^0 = p_x - d_6 \cdot a_x, \quad P_{5y}^0 = p_y - d_6 \cdot a_y$$

حال اگر مقادیر به دست آمده را جایگذاری کنیم در نهایت به معادله زیر میرسیم:

$$\theta_1 = \text{atan2}(p_y - d_6 \cdot a_y, p_x - d_6 \cdot a_x) \pm \arccos\left(\frac{d_4}{\sqrt{(p_x - d_6 \cdot a_x)^2 + (p_y - d_6 \cdot a_y)^2}}\right) + \frac{\pi}{2}$$

در مرحله بعدی به سراغ به دست آوردن θ_5 میرویم. برای این منظور به شکل که یک نمای عمودی میباشد دقت کنید.



شکل ۱۵ - به دست آوردن θ_5

باتوجه به شکل میتوان فهمید که:

$$-P_{6y}^1 = d_4 + d_6 \cdot \cos\theta_5$$

از آنجایی که d_4 و d_6 مشخص هستند میتوان به کمک روش جبری P_{6y}^1 را به دست آورد:

$$P_6^0 = R_1^0 \cdot P_6^1 \rightarrow P_6^1 = R_1^{0^{-1}} \cdot P_6^0$$

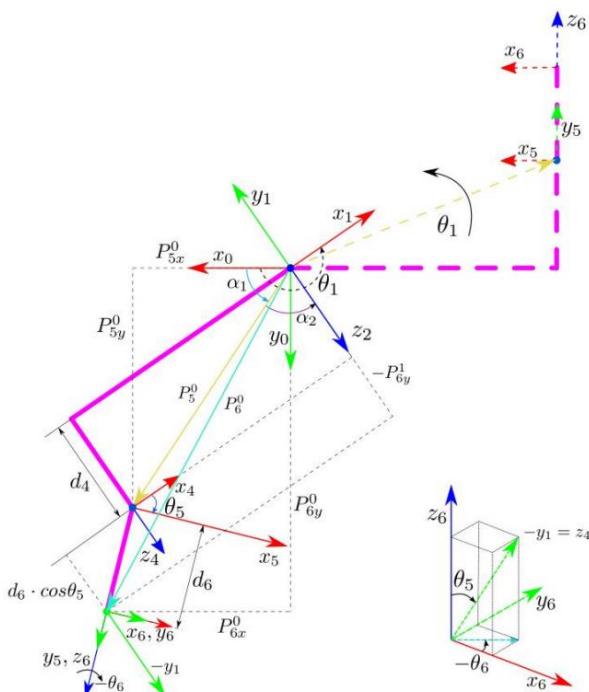
باتوجه به خواص ماتریس‌های دوران که در آن‌ها معکوس ماتریس با Transpose آن یکسان است، داریم:

$$\begin{bmatrix} P_{6x}^1 \\ P_{6y}^1 \\ P_{6z}^1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_{6x}^0 \\ P_{6y}^0 \\ P_{6z}^0 \end{bmatrix} \rightarrow P_{6y}^1 = -P_{6y}^0 \cdot \sin\theta_1 + P_{6y}^0 \cdot \cos\theta_1$$

حال اگر در فرمول اولیه جاگذاری کنیم، داریم:

$$\theta_5 = \pm \arccos \left(\frac{P_{6x}^0 \cdot \sin\theta_1 - P_{6y}^0 \cdot \cos\theta_1 - d_4}{d_6} \right)$$

از آنجایی که محورهای ۲ و ۳ و ۴ با یکدیگر موازی هستند، هر چرخشی در ۲ و ۳ و ۴ باعث میشود که محور y_1 با z_2 و z_3 و z_4 موازی شود. از فرمی که روی EE قرار دادیم کمک میگیریم تا موقعیت بردار \hat{y}_1 را مشخص کنیم.



شکل ۱۶ - به دست آوردن تناع

عکس بردار واحد y_1 روی مختصات $x_6y_6z_6$ میتواند به صورت \hat{y}_1^6 - نمایش داد.

$$-\hat{y}_1^6 = \begin{bmatrix} \sin\theta_5 \cos(-\theta_6) \\ \sin\theta_5 \sin(-\theta_6) \\ \cos\theta_5 \end{bmatrix} \text{ (i.e. } \hat{y}_1^6 = \begin{bmatrix} -\sin\theta_5 \cos\theta_6 \\ \sin\theta_5 \sin\theta_6 \\ -\cos\theta_5 \end{bmatrix}).$$

$$R_1^6 = ((R_1^0)^T \cdot R_6^0)^T$$

$$R_6^0 = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix}$$

$$R_1^0 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_1^6 = \begin{bmatrix} n_x c\theta_1 + n_y s\theta_1 & -n_x s\theta_1 + n_y c\theta_1 & n_z \\ o_x c\theta_1 + o_y s\theta_1 & -o_x s\theta_1 + o_y c\theta_1 & o_z \\ a_x c\theta_1 + a_y s\theta_1 & -a_x s\theta_1 + a_y c\theta_1 & a_z \end{bmatrix}$$

$$\begin{bmatrix} -n_x s\theta_1 + n_y c\theta_1 \\ -o_x s\theta_1 + o_y c\theta_1 \\ -a_x s\theta_1 + a_y c\theta_1 \end{bmatrix} = \begin{bmatrix} -\sin\theta_5 \cos\theta_6 \\ \sin\theta_5 \sin\theta_6 \\ -\cos\theta_5 \end{bmatrix}$$

باتوجه به سطر اول و دوم فرمول داریم:

$$\theta_6 = \text{atan2}\left(\frac{-o_x s\theta_1 + o_y c\theta_1}{s\theta_5}, \frac{n_x s\theta_1 - n_y c\theta_1}{s\theta_5}\right)$$

جوینتهای ۲ و ۳ و ۴ یک بازوی صفحه‌ای RRR ایجاد میکند. اگر ماتریس تبدیل 1T را به دست آوریم داریم:

$${}^1T = {}^1T_3^2 T_4^3 T = \begin{bmatrix} c_{234} & -s_{234} & 0 & a_2 c_2 + a_3 c_{23} \\ 0 & 0 & -1 & -d_4 \\ s_{234} & c_{234} & 0 & a_2 s_2 + a_3 s_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$p_{4x}^1 = a_2 c_2 + a_3 c_{23}$$

$$p_{4z}^1 = a_2 s_2 + a_3 s_{23}$$

$$n_{4x}^1 = c_{234}$$

$$n_{4z}^1 = s_{234}$$

باتوجه به p_{4z}^1 و p_{4x}^1 میتوان مقدار تتا ۳ را پیدا کرد.

$$\theta_3 = \pm \arccos\left(\frac{(p_{4x}^1)^2 + (p_{4z}^1)^2 - a_2^2 - a_3^2}{2a_2 a_3}\right)$$

حال اگر θ_3 را در p_{4x}^1 و p_{4z}^1 جاگذاری کنیم، خواهیم داشت:

$$\theta_2 = \text{atan2}((a_3c_3 + a_2)p_{4z}^1 - a_3s_3p_{4x}^1, (a_3c_3 + a_2)p_{4x}^1 + a_3s_3p_{4z}^1)$$

باتوجه به اینکه مقادیر مربوط به θ_2 و θ_1 محاسبه شد در نهایت به راحتی میتوان θ_4 را پیدا کرد.

$$\theta_4 = \text{atan2}(n_{4z}^1, n_{4x}^1) - \theta_2 - \theta_3$$

حال این موارد از یک سمت به صورت پارامتری بود. لازم است تا ماتریس تبدیل 1T عددی را نیز به دست بیاریم و جاگذاری کنیم:

$${}^1T = ({}^0T)^{-1}({}^0T)({}^5T)^{-1}({}^4T)^{-1}$$

$$({}^0T)^{-1} = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & -d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$({}^5T)^{-1} = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_6 & -c_6 & 0 & -d_6 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$({}^4T)^{-1} = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_5 & c_5 & 0 & -d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

در نهایت میتوان مقادیر مربوط به P_{4x}^1 و P_{4z}^1 و n_{4x}^1 و n_{4z}^1 را یافت.

$$P_{4x}^1 = -d_6(a_xc_1 + a_ys_1) + d_5(o_xc_1c_6 + o_ys_1c_6 + n_xc_1s_6 + n_ys_1s_6) + p_xc_1 + p_ys_1$$

$$P_{4z}^1 = -d_6a_z + d_5o_zc_6 + d_5n_zs_6 + p_z - d_1$$

$$n_{4x}^1 = -s_5(a_xc_1 + a_ys_1) + c_5(n_xc_1c_6 + n_ys_1c_6 - o_xc_1s_6 - o_ys_1s_6)$$

$$n_{4z}^1 = n_zc_5c_6 - o_zc_5s_6 - a_zs_5$$

۳-۳- بررسی سینماتیک معکوس برای چندین وضعیت متفاوت

در این قسمت برای همان حالاتی که در سینماتیک مستقیم مورد بررسی قرار دادیم، سینماتیک معکوس را بررسی میکنیم. برای این منظور ماتریس‌های تبدیل خروجی در سینماتیک مستقیم را به عنوان ورودی بخش سینماتیک معکوس قرار می‌دهیم.

الف) بررسی وضعیت صفر ربات

در این حالت در قسمت سینماتیک مستقیم تمامی زوایای ورودی را برابر با صفر درجه گذاریم. حال اگر ماتریس تبدیل به دست آمده از آن بررسی را در این جا نیز قرار دهیم، انتظار داریم تمامی زوایای θ را به عنوان خروجی بگرداند.

A) Zero state

- We gave zero values to joint variables in forward kinematics and it took us Transformation matrix ($T_{0_6_first}$) as a result. now we give this matrix as an input to inverse function.

```
In [13]: T_0_6_A
Out[13]: [[1.0, 0.0, 0.0, 1184.5],
           [0.0, 6.123233995736766e-17, -1.0, -256.1],
           [0.0, 1.0, 6.123233995736766e-17, 243.7],
           [0.0, 0.0, 0.0, 1.0]]
```

```
In [24]: theta_A = inverse(T_0_6_A)
i = 0
for th in theta_A:
    i += 1
    print("ans", i, " theta1: ", round(degrees(th[0]), 2), " theta2: ", round(degrees(th[1]), 2), " theta3: ", round(degrees(th[2]), 2),
          " theta4: ", round(degrees(th[3]), 2), " theta5: ", round(degrees(th[4]), 2), " theta6: ", round(degrees(th[5]), 2))
```

```
ans 1 theta1: 0.0 theta2: 0.0 theta3: 0.0 theta4: 0.0 theta5: 0.0 theta6: 0.0
ans 2 theta1: 164.24 theta2: 180.0 theta3: 0.0 theta4: -180.0 theta5: 164.24 theta6: 0.0
```

ب) بررسی وضعیت کاملاً کشیده

باتوجه به فریم گذاری گفتیم، زمانی ربات به حالت کشیده در می‌آید که زوایای تتا به صورت زیر باشند:

تتا۱: دلخواه | تتا۲: ۹۰ درجه | تتا۳: ۰ درجه | تتا۴: ۹۰-درجه | تتا۵: ۰ درجه | تتا۶: دلخواه

که تتا۱ را مقدار ۴۵ درجه و تتا۶ را هم مقدار ۶۰ درجه به عنوان مقادیر دلخواه دادیم. حال اگر ماتریس تبدیل نهایی را به تابع سینماتیک معکوس دهیم انتظار داریم که همین زوایای گفته شده را بگیریم.

B) fully-stretched state

- We should use T_0_6_B as input and get the below angles in results:

```
theta1 = 45 | theta2 = 90 | theta3 = 0 | theta4 = -90 | theta5 = 0 | theta6 = 60
```

In [17]: T_0_6_B

```
Out[17]: [[0.3535533905932738,
-0.6123724356957946,
0.7071067811865476,
181.09004666187485],
[0.3535533905932739,
-0.6123724356957946,
-0.7071067811865476,
-181.09004666187474],
[0.8660254037844386, 0.5000000000000001, 6.123233995736766e-17, 1428.2],
[0.0, 0.0, 0.0, 1.0]]
```

```
In [26]: theta_B = inverse(T_0_6_B)
i = 0
for th in theta_B:
    i += 1
    print("ans", i, " theta1: ", round(degrees(th[0]), 2), " theta2: ", round(degrees(th[1]), 2), " theta3: ", round(degrees(th[2]), 2),
          " theta4: ", round(degrees(th[3]), 2), " theta5: ", round(degrees(th[4]), 2), " theta6: ", round(degrees(th[5]), 2))

ans 1 theta1: 45.0 theta2: 90.0 theta3: 0.0 theta4: -90.0 theta5: 0.0 theta6: 60.0
ans 2 theta1: 45.0 theta2: -90.0 theta3: 0.0 theta4: 90.0 theta5: 0.0 theta6: 60.0
ans 3 theta1: 135.0 theta2: 90.0 theta3: 0.0 theta4: -90.0 theta5: 0.0 theta6: 60.0
ans 4 theta1: 135.0 theta2: -90.0 theta3: 0.0 theta4: 90.0 theta5: 0.0 theta6: 120.0
```

ج) بررسی یک وضعیت دلخواه

برای بررسی وضعیت دلخواه ما زوایای مفاصل را به صورت دلخواه انتخاب کردیم و برابر مقادیر زیر بود:

تتا۱: ۳۰ درجه | تتا۲: ۴۵ درجه | تتا۳: ۲۶ درجه | تتا۴: ۵۰ درجه | تتا۵: ۶۰ درجه | تتا۶: ۸۰ درجه

حال اگر ماتریس تبدیل نهایی را به تابع سینماتیک معکوس دهیم انتظار داریم که همین زوایای گفته شده را به عنوان یکی از خروجی‌های قابل قبول بگیریم.

C) Arbitrary state

- We should use T_0_6_B as input and get the below angles in results:

```
theta1 = 30 | theta2 = 45 | theta3 = 26 | theta4 = 50 | theta5 = 60 | theta6 = 80
```

In [27]: T_0_6_C

```
Out[27]: [[-0.8449695581951862,
0.5171601307605163,
-0.1362785561825407,
519.983544410926],
[-0.314195242140994,
-0.6862252123034819,
-0.6560307302863893,
57.725526280181091],
[-0.432790719406602,
-0.5115079248172085,
0.7423286577013642,
1110.6969609747771],
[0.0, 0.0, 0.0, 1.0]]
```

In [28]: theta_C = inverse(T_0_6_C)

```
i = 0
for th in theta_C:
    i += 1
    print("ans", i, " theta1: ", round(degrees(th[0]), 2), " theta2: ", round(degrees(th[1]), 2), " theta3: ", round(degrees(th[2]), 2),
          " theta4: ", round(degrees(th[3]), 2), " theta5: ", round(degrees(th[4]), 2), " theta6: ", round(degrees(th[5]), 2))

ans 1 theta1: 30.0 theta2: 45.0 theta3: 26.0 theta4: 50.0 theta5: 60.0 theta6: -280.0
ans 2 theta1: 30.0 theta2: 70.08 theta3: -26.0 theta4: 76.92 theta5: 60.0 theta6: -280.0
ans 3 theta1: 30.0 theta2: 28.77 theta3: 72.95 theta4: -160.72 theta5: -60.0 theta6: -100.0
ans 4 theta1: 30.0 theta2: 98.77 theta3: -72.95 theta4: -84.82 theta5: -60.0 theta6: -100.0
ans 5 theta1: 175.03 theta2: 82.09 theta3: 66.85 theta4: -65.0 theta5: 131.71 theta6: -58.85
ans 6 theta1: 175.03 theta2: 146.3 theta3: -66.85 theta4: 4.48 theta5: 131.71 theta6: -58.85
ans 7 theta1: 175.03 theta2: 106.62 theta3: 36.99 theta4: -239.67 theta5: -131.71 theta6: -238.85
ans 8 theta1: 175.03 theta2: 142.27 theta3: -36.99 theta4: -201.35 theta5: -131.71 theta6: -238.85
```

۴-۳- آزمایش الگوریتم سینماتیک معکوس

همانطور که در قسمت قبل مشاهده کردید برای حالت وضعیت دلخواه ما سمت زوایای زیر را به عنوان ورودی به سینماتیک مستقیم به عنوان ورودی دادیم:

تتا_۱: ۳۰ درجه | تتا_۲: ۴۵ درجه | تتا_۳: ۶۰ درجه | تتا_۴: ۵۰ درجه | تتا_۵: ۸۰ درجه

در نهایت نتیجه به صورت زیر شد:

C) Arbitrary state

- We have below values for joint variables in arbitrary state:

```
theta1 = 30 | theta2 = 45 | theta3 = 26 | theta4 = 50 | theta5 = 60 | theta6 = 80
```

```
In [12]: theta1 = math.radians(30)
theta2 = math.radians(45)
theta3 = math.radians(26)
theta4 = math.radians(50)
theta5 = math.radians(60)
theta6 = math.radians(80)

desired_theta = [theta1, theta2, theta3, theta4, theta5, theta6]
T_0_6_C = forward(desired_theta)

print('T_0_6 : ')
for row in T_0_6_C:
    print(row)

T_0_6 :
[-0.8449695581951862, 0.5171601307605163, -0.1362785561825407, 519.983544410926]
[-0.3141952242140994, -0.686252123034819, -0.6560307302863893, 57.72552628018109]
[-0.432790719406602, -0.5115079248172085, 0.7423286577013642, 1110.696960974777]
[0.0, 0.0, 0.0, 1.0]
```

سپس همین ماتریس تبدیل به دست آمده را به عنوان ورودی قسمت سینماتیک معکوس دادیم و سپس دیدیم که همان سمت زاویه در قسمت ans1 ظاهر شد. پس صحت کد و روابط به دست آمده تایید شد.

دقت شود که در تتا_۶ مقدار ۲۸۰- به دست آمده است که همان زاویه ۸۰ درجه می باشد.

C) Arbitrary state

- We should use T_0_6_B as input and get the below angles in results:

```
theta1 = 30 | theta2 = 45 | theta3 = 26 | theta4 = 50 | theta5 = 60 | theta6 = 80
```

```
In [27]: T_0_6_C
Out[27]: [[-0.8449695581951862,
 0.5171601307605163,
-0.1362785561825407,
519.983544410926],
[-0.3141952242140994,
-0.686252123034819,
-0.6560307302863893,
57.72552628018109],
[-0.432790719406602,
-0.5115079248172085,
0.7423286577013642,
1110.696960974777],
[0.0, 0.0, 0.0, 1.0]]
```

```
In [28]: theta_C = inverse(T_0_6_C)
i = 0
for th in theta_C:
    i += 1
    print("ans", i, "theta1: ", round(degrees(th[0]),2), "theta2: ", round(degrees(th[1]),2), "theta3: ", round(degrees(th[2]),2),
          "theta4: ", round(degrees(th[3]),2), "theta5: ", round(degrees(th[4]),2), "theta6: ", round(degrees(th[5]),2))
```

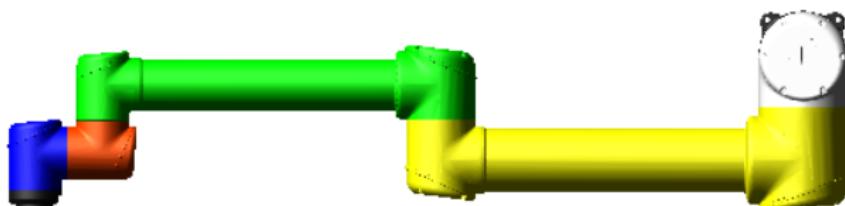
```
ans 1 theta1: 30.0 theta2: 45.0 theta3: 26.0 theta4: 50.0 theta5: 60.0 theta6: -280.0
ans 2 theta1: 30.0 theta2: 70.08 theta3: -26.0 theta4: 76.92 theta5: 60.0 theta6: -280.0
ans 3 theta1: 30.0 theta2: 28.77 theta3: 72.95 theta4: -160.72 theta5: -60.0 theta6: -100.0
ans 4 theta1: 30.0 theta2: 98.77 theta3: -72.95 theta4: -84.82 theta5: -60.0 theta6: -100.0
ans 5 theta1: 175.03 theta2: 82.09 theta3: 66.85 theta4: -65.0 theta5: 131.71 theta6: -58.85
ans 6 theta1: 175.03 theta2: 146.3 theta3: -66.85 theta4: 4.48 theta5: 131.71 theta6: -58.85
ans 7 theta1: 175.03 theta2: 106.62 theta3: 36.99 theta4: -239.67 theta5: -131.71 theta6: -238.85
ans 8 theta1: 175.03 theta2: 142.27 theta3: -36.99 theta4: -201.35 theta5: -131.71 theta6: -238.85
```

۳-۵-۳- صحه گذاری سینماتیک معکوس

در اینجا نیز به کمک ابزار پیترکورک این صحه سنجی را انجام داده ایم. برای وضعیت صفر ربات خواهیم داشت:

```
>> Robot.ikine([1 0 0 1.184; 0 0 -1 -0.2796; 0 1 0 0.2202; 0 0 0 1])  
ans =  
-0.0000 -0.0000 -0.0280 0.0580 -0.0300 0.0000
```

شکل ۱۶- صحه سنجی وضعیت صفر ربات



شکل ۱۷- نمایش ربات در وضعیت صفر در ادمز (*fully stretched*)

فصل چهارم
ژاکوبین

میدانیم که در رباتیک از ژاکوبین برای برقراری رابطه بین سرعت مفاصل و سرعت‌های فضای کارتزین نقطه tip از بازو استفاده می‌شود.

$${}^0v = {}^0J(\theta) \dot{\theta}$$

برای به دست آوردن ماتریس ژاکوبین در کلاس درس ۳ روش متفاوت گفته شد:

- ۱- روش انتشار سرعت
- ۲- مشتق گیری مستقیم
- ۳- روش جیگزین (Alternative Approach)

که ازین این موارد ما روش alternative approach را انتخاب کردیم که در ادامه به آن می‌پردازیم.

۱-۴- استخراج ماتریس ژاکوبین

با توجه به تعریف و رابطه‌ای که در بالا تعریف کردیم، داریم:

$$v = {}^0J(\theta) \dot{\theta} \rightarrow \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} {}^0J_v \\ {}^0J_\omega \end{bmatrix} \dot{\theta} \rightarrow \begin{cases} v = {}^0J_v \dot{\theta} \\ \omega = {}^0J_\omega \dot{\theta} \end{cases}$$

در حقیقت سرعت زاویه‌ای عملگر نهایی ربات برابر با مجموع بردار تمام سرعت زاویه‌های مفاصل ربات نسبت به یک دستگاه مرجع است. همچنین سرعت خطی عملگر نهایی مشتق زمانی بردار موقعیت آن نسبت به یک دستگاه مرجع تعریف می‌شود. به کمک نکات گفته شده می‌توان سرعت عملگر نهایی و به دنبال آن، ژاکوبین را با روش ساده‌تری نسبت به دستگاه مرجع محاسبه نمود.

ستون‌های ماتریس ژاکوبین نسبت به دستگاه پایه بسته به نوع مفصل کشویی یا دورانی تابع بردار محور مفصل، موقعیت دستگاه مفصل و موقعیت دستگاه عملگر نهایی هستند:

$$\begin{cases} Revolute: {}^0J_i = \begin{bmatrix} Z_i \times (O_e - O_i) \\ Z_i \end{bmatrix} \\ Prismatic: {}^0J_i = \begin{bmatrix} Z_i \\ 0 \end{bmatrix} \end{cases}$$

در این روابط زیرond A در حقیقت برای یک ربات با n مفصل از مقدار یک تا مقدار تعداد مفاصل ربات خواهد بود.

در کدی که زده ایم باتوجه به اینکه 0T_i را از قبل محاسبه کردیم، میتوان به راحتی پارامترهای لازم برای محاسبه ماتریس ژاکوبین را استخراج کرد. در حقیقت میتوان گفت که Z_i ها همان مقادیر ۳ سطر اول از ستون سوم ماتریس O_i همان مقادیر ۳ سطر اول ستون چهارم 0T_i ها میباشد.

در نهایت ماتریس J ما به صورت زیر خواهد شد.

$$\begin{bmatrix} -a_2 \sin(\theta_1(t)) \cos(\theta_2(t)) - a_3 \sin(\theta_1(t)) \cos(\theta_2(t) + \theta_3(t)) + d_4 \cos(\theta_1(t)) + d_5 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_1(t)) \\ + d_6 (-\sin(\theta_1(t)) \sin(\theta_5(t)) \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) + \cos(\theta_1(t)) \cos(\theta_5(t))) \\ a_2 \cos(\theta_1(t)) \cos(\theta_2(t)) + a_3 \cos(\theta_2(t) + \theta_3(t)) \cos(\theta_1(t)) + d_4 \sin(\theta_1(t)) - d_5 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos(\theta_1(t)) \\ + d_6 (\sin(\theta_1(t)) \cos(\theta_5(t)) + \sin(\theta_5(t)) \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos(\theta_1(t))) \\ 0 \\ 0 \\ 0 \\ 1 \\ \\ - (a_2 \sin(\theta_2(t)) + a_3 \sin(\theta_2(t) + \theta_3(t)) + d_5 \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_5(t))) \cos(\theta_1(t)) \\ - (a_2 \sin(\theta_2(t)) + a_3 \sin(\theta_2(t) + \theta_3(t)) + d_5 \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_5(t))) \sin(\theta_1(t)) \\ a_2 \cos(\theta_2(t)) + a_3 \cos(\theta_2(t) + \theta_3(t)) - d_5 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_5(t)) \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) \\ \sin(\theta_1(t)) \\ - \cos(\theta_1(t)) \\ 0 \\ \\ - (a_3 \sin(\theta_2(t) + \theta_3(t)) + d_5 \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_5(t))) \cos(\theta_1(t)) \\ - (a_3 \sin(\theta_2(t) + \theta_3(t)) + d_5 \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_5(t))) \sin(\theta_1(t)) \\ a_3 \cos(\theta_2(t) + \theta_3(t)) - d_5 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_5(t)) \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) \\ \sin(\theta_1(t)) \\ - \cos(\theta_1(t)) \\ 0 \\ \\ - (d_5 \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_5(t))) \cos(\theta_1(t)) \\ - (d_5 \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_5(t))) \sin(\theta_1(t)) \\ - d_5 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) + d_6 \sin(\theta_5(t)) \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) \\ \sin(\theta_1(t)) \\ - \cos(\theta_1(t)) \\ 0 \\ \\ d_6 (-\sin(\theta_1(t)) \sin(\theta_5(t)) + \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos(\theta_1(t)) \cos(\theta_5(t))) & 0 \\ d_6 (\sin(\theta_1(t)) \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos(\theta_5(t)) + \sin(\theta_5(t)) \cos(\theta_1(t))) & 0 \\ d_6 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos(\theta_5(t)) \\ - \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos(\theta_1(t)) \\ - \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_1(t)) \\ \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) & \begin{matrix} \sin(\theta_1(t)) \cos(\theta_5(t)) + \sin(\theta_5(t)) \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos(\theta_1(t)) \\ \sin(\theta_1(t)) \sin(\theta_5(t)) \cos(\theta_2(t) + \theta_3(t) + \theta_4(t)) - \cos(\theta_1(t)) \cos(\theta_5(t)) \\ \sin(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin(\theta_5(t)) \end{matrix} \end{bmatrix}$$

همچنین با جایگذاری مقادیر عددی به جای پارامترها و دادن زاویه‌های دلخواه میتوان ماتریس ژاکوبین را محاسبه نمود. تابعی تعریف شده است که پارامترها را به عنوان ورودی گرفته و خروجی آن ماتریس عددی نهایی میباشد.

2-1) J singularities

```
In [7]: J_det = Matrix(J).det()
In [8]: J_det_simp = trigsimp(J_det)
In [9]: J_det_simp
Out[9]: a2a3(a2 cos(θ2(t)) + a3 cos(θ2(t) + θ3(t))) - d5 sin(θ2(t) + θ3(t) + θ4(t))) sin(θ3(t)) sin(θ5(t))
```

میدانیم که زمانی که ربات مج داشته باشد و درواقع Decoupled باشد، باعث میشود که جهت گیری و موقعیت را به روایتی از هم جدا کرد و به صورت Decoupled در آورد و در حقیقت جهت گیری را میتوان به مج نسبت داد و موقعیت را به ۳ درجه اول نسبت داد. چون ربات ما مج ندارد، نمیتوان ماتریس‌های ژاکوبین موقعیت و جهت گیری را به طور جداگانه بررسی کرد.

۴-۲-۴- بررسی نقاط سینگولاریتی و تقسیم بندی بر حسب موقعیت و جهتگیری

در بازوی رباتیک مزایای بسیاری مانند تکرارپذیری بالا، سرعت سریع و دامنه کاری گسترده وجود دارد. با این حال، محدودیتی به نام سینگولاریتی وجود دارد.

می‌توان گفت، سینگولاریتی حالتی است که حرکت خطی ربات در نزدیکی ناحیه سینگولار محدود می‌شود. با توجه به سینماتیک، هیچ جوابی برای سینماتیک معکوس در ناحیه سینگولار وجود ندارد. از نظر ریاضی، سینگولاریتی زمانی اتفاق می‌افتد که رتبه ماتریس ژاکوبین کمتر از رتبه کامل شود، به این معنی که ماتریس معکوس ژاکوبین وجود ندارد. از نظر فیزیکی، در هنگام سینگولار شدن، ربات یک یا چند درجه آزادی را از دست می‌دهد، بنابراین ممکن است نتواند در جهات خاصی حرکت کند.

در واقعیت نیز در هنگامی که با سینگولاریتی رو برو می‌شویم، لازم است که مفصل مورد نظر یک حرکت را با سرعت بالایی انجام دهد و این سرعت بالا یعنی گشتاور بالا نیاز است و گشتاور بالا نیز یعنی موتورها باید جریان زیادی بکشند و این منجر به سوختن موتور می‌شود. بنابراین لازم است که نقاط سینگولار شناسایی و در صورت لزوم برطرف شوند.

- لازم به ذکر است، در اینجا ربات انتخابی ما دارای مج نمیباشد و لذا سینگولاریتی را بر حسب موقعیت و جهت گیری تقسیم بندی کرد.

برای محاسبه نقاط سینگولاریتی همانطور که اشاره شد، میتوان نقاطی را که در آن‌ها دترمینان ماتریس ژاکوبین ۰ می‌شود را معرفی کرد. ماتریس ژاکوبین کلی ما یک ماتریس ۶ در ۶ و مربعی می‌باشد پس میتوان به راحتی به کمک کد نویسی دترمینان آن را محاسبه نمود و برابر با ۰ قرار داد. در صورتی که دترمینان را محاسبه کنیم قطعاً عبارت بسیار بزرگی خواهد شد و لازم است حتماً آن را simplify کنیم تا بتوان آن را برابر با ۰ قرار داد و با حل آن نقاط سینگولار را یافت و همچنین تحلیل کرد.

برای بدست آوردن جهت‌های تکین ربات، می‌بایست از ماتریس ژاکوبین ۶ در ۶ دترمینان گرفت. با مساوی صفر قرار دادن دترمینان، نقاط سینگولار بدست می‌آید.

2-1) J singularities

```
In [7]: J_det = Matrix(J).det()
In [8]: J_det_simp = trigsimp(J_det)
In [9]: J_det_simp
Out[9]: a2*a3*(a2*cos(theta_2(t))+a3*cos(theta_2(t)+theta_3(t))-d5*sin(theta_2(t)+theta_3(t)+theta_4(t)))*sin(theta_3(t))*sin(theta_5(t))
```

$$\det(J) = a_2 a_3 (a_2 c_2 + a_3 c_{23} - d_5 s_{234}) s_3 s_5$$

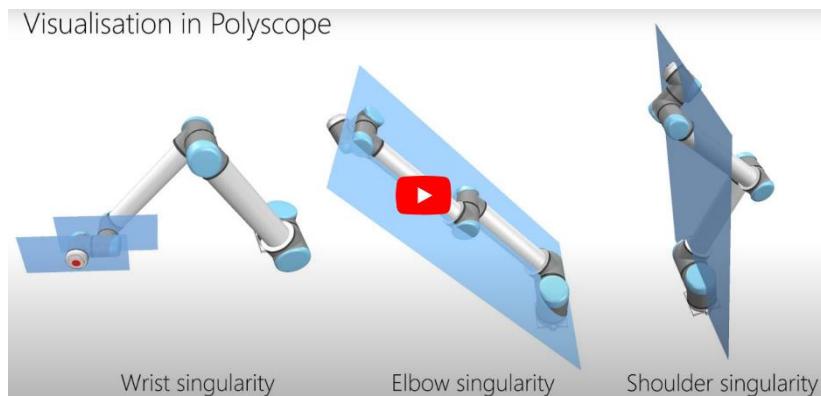
از مساوی صفر قرار دادن عبارت بالا خواهیم داشت:

$$s_3 = 0 \quad \text{یا} \quad s_5 = 0 \quad \text{یا} \quad a_2 c_2 + a_3 c_{23} - d_5 s_{234} = 0$$

۴-۳-۴- تحلیل فیزیکی نقاط سینگولاریتی

همانطور که مشاهده شد، سه حالت وجود دارد که رخ دادن آنها ربات را به وضعیت سینگولار می‌برد. در حقیقت ۳ نوع سینگولاریتی در آن داریم:

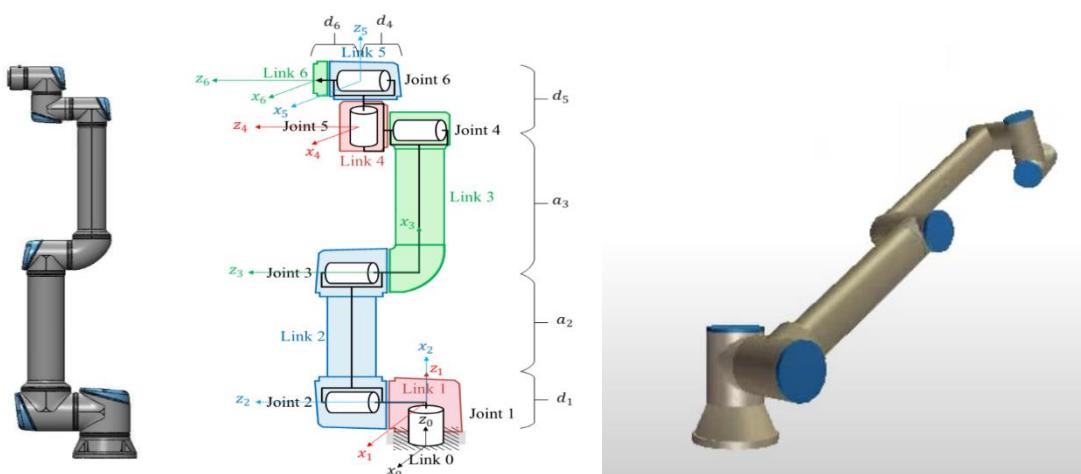
- سینگولاریتی‌های Wrist و Shoulder و Elbow



۱- بررسی حالت اول: حالت Elbow singularities

$$s_3 = 0$$

در این حالت محورهای مفاصل ۲ و ۳ و ۶ در یک صفحه قرار می‌گیرند و باعث می‌شوند elbow در موقعیتش قفل شود. به عبارتی بدین معناست که لینک‌های دوم و سوم، هم جهت ($\theta_3 = 0$) یا خلاف جهت ($\theta_3 = \pm\pi$) یکدیگر قرار دارند. حالتی که زاویه مفصلی صفر است را به اصلاح *Fully stretched* و دیگری را *Folded back* گویند. این سینگولاریتی از نوع سینگولاریتی مزری به حساب می‌آید.



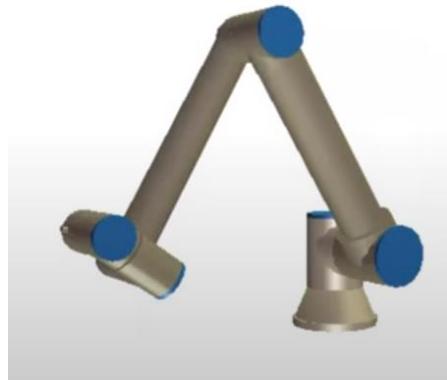
شکل ۱۹- نمایی از فریم گذاری ربات و حالت سینگولار Elbow

۲- بررسی حالت دوم: حالت Wrist Singularities

$$s_5 = 0$$

برای بررسی این حالت نگاهی به فریم گذاری روی ربات می‌اندازیم:

با کمی دقیق می‌توان دریافت که در این وضعیت، هر دو محور Z_6 و Z_4 در یک راستا و موازی قرار گرفته و کار یکسانی را انجام می‌دهند. به عبارت دیگر، جهت گیری که توسط دو درجه آزادی چهارم و ششم ایجاد می‌شود یکسان است و خروجی نهایی جمع یا اختلاف θ_6 و θ_4 خواهد بود. ربات در این وضع یک درجه آزادی خود را ازدست داده و دچار سینگولاریتی می‌شود. این سینگولاریتی از نوع سینگولاریتی داخلی به حساب می‌آید. زمانی رخ میدهد که تناه مقادیر $180^\circ \pm 360^\circ$ یا 0° را اختیار کند.



شکل ۲۰- حالت سینگولار Wrist

۳- بررسی حالت سوم: حالت Shoulder singularities

$$a_2c_2 + a_3c_{23} - d_5s_{234} = 0$$

برای تحلیل این عبارت نیز از شماتیک ربات و فریم گذاری روی آن کمک می‌گیریم:

می‌دانیم جابجایی اندازکتور ناشی از جمع جابجایی‌هایی است که مفاصل قبل ایجاد می‌کنند. اما پیکربندی‌هایی برای ربات وجود دارد که در آن، اثرات این جابجایی‌ها خنثی می‌گردد. حالت سومی که دترمینان ژاکوبین را صفر می‌کند، می‌بین این پیکربندی است. در این وضعیت موقعیت اندازکتور همواره روی محور y_1 قرار داشته و سرعت آن در راستای عمود بر آن یعنی x_1 ، صفر است. سه‌های مختلفی از زوایا در این معادله صدق می‌کند که با توجه به آن، سینگولاریتی می‌تواند از نوع داخلی یا مرزی باشد.

به طور خلاصه این سینگولاریتی زمانی رخ میدهد که محل تقاطع محورهای ۵ و ۶ با صفحه عبوری از محورهای مفاصل ۱ و ۲ همتراز شود.



شکل ۲۱- حالت سینگولار Shoulder

۴-۴- راه حل برطرف کردن سینگولاریتی

همانطور که پیش تر اشاره شد، نقاط سینگولاریتی برای ما نامطلوب هستند و تا جای ممکن باید سعی کرد از آنها دوری کرد و راه حلی یافت تا آنها را برطرف کرد چراکه همانطور که گفتیم در نقاط سینگولار گشتاور زیادی نیاز است که باعث می شود موتور جریان زیادی بکشد و بسوزد. در متن درس نیز اشاره شد که برای برطرف کردن سینگولاریتی راههایی وجود دارد از جمله آنکه در طراحی تغییراتی ایجاد کرد و مثلا در لینکها ایجاد کرد. سینگولاریتی در مرز را می توان با کار نکردن در حد نهایی ربات در مرز ها رفع کرد. همچنین با ایجاد کاری کرد که اند افکتور در راستای مفصل پایه قرار نگیرد. همچنین با Offset دادن، می توان کاری کرد که اند افکتور در راستای مفصل پایه قرار نگیرد.

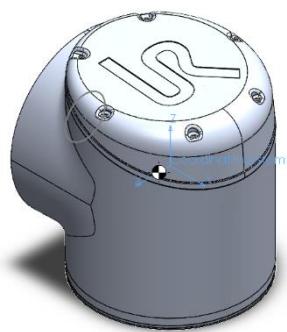
برای توضیحات بیشتر میتوانید به [لينك](#) مراجعه نمایید.

فصل پنجم

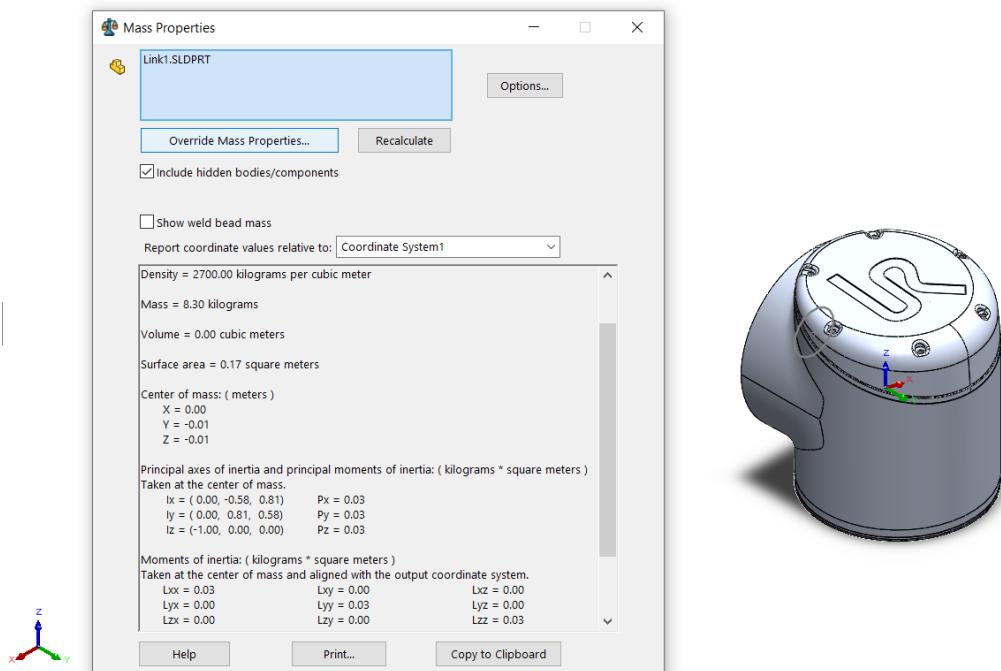
استخراج مشخصات موردنیاز لینک‌ها

قبل از وارد شدن به تحلیل دینامیکی و نوشتن معادلات مربوطه لازم است تا با توجه به فایل‌های Solid که داریم مشخصات لینک‌ها از قبیل جرم، ممان اینرسی و مکان مرکز جرم لینک‌ها را به دست آوریم. این مشخصات برای نوشتن معادلات دینامیکی چه به روش لاگرانژ و چه روش نیوتن اویلر نیاز می‌شود. البته بعضاً در کاتالوگ ربات نیز ممکن است این موارد ذکر شود ولی به هر حال از آنجایی که در نهایت قرار است به کمک همین فایل‌های Solid عملیات صحت سنجی را انجام دهیم فلذًا موارد ذکر شده را از همین فایل‌های طراحی به دست می‌آوریم.

۱-۵- مشخصات لینک اول

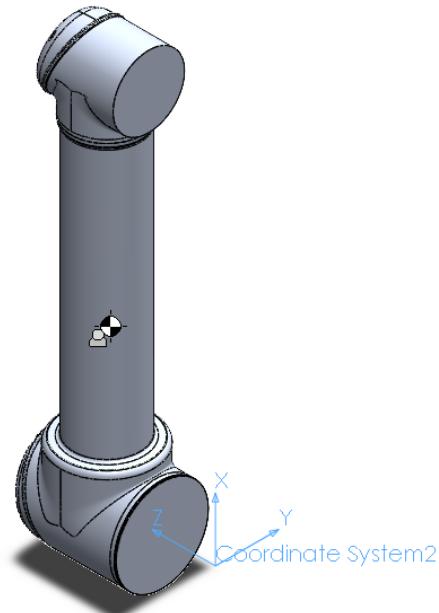


شکل 22 - لینک ۱ به همراه فریم اول

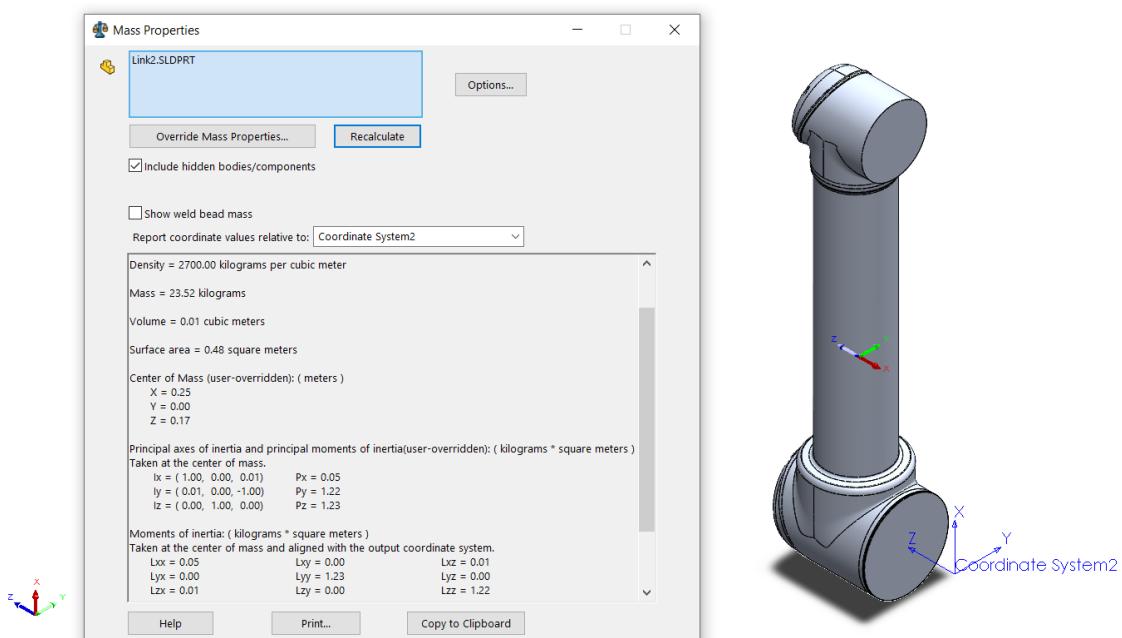


شکل 23 - ممان‌های اینرسی لینک ۱ حول مرکز جرم

۲-۵- مشخصات لینک دوم

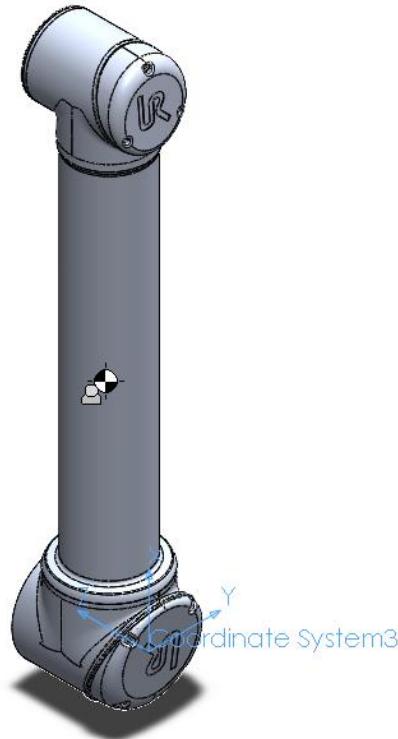


شکل ۲۴ - لینک ۲ به همراه فریم دوم

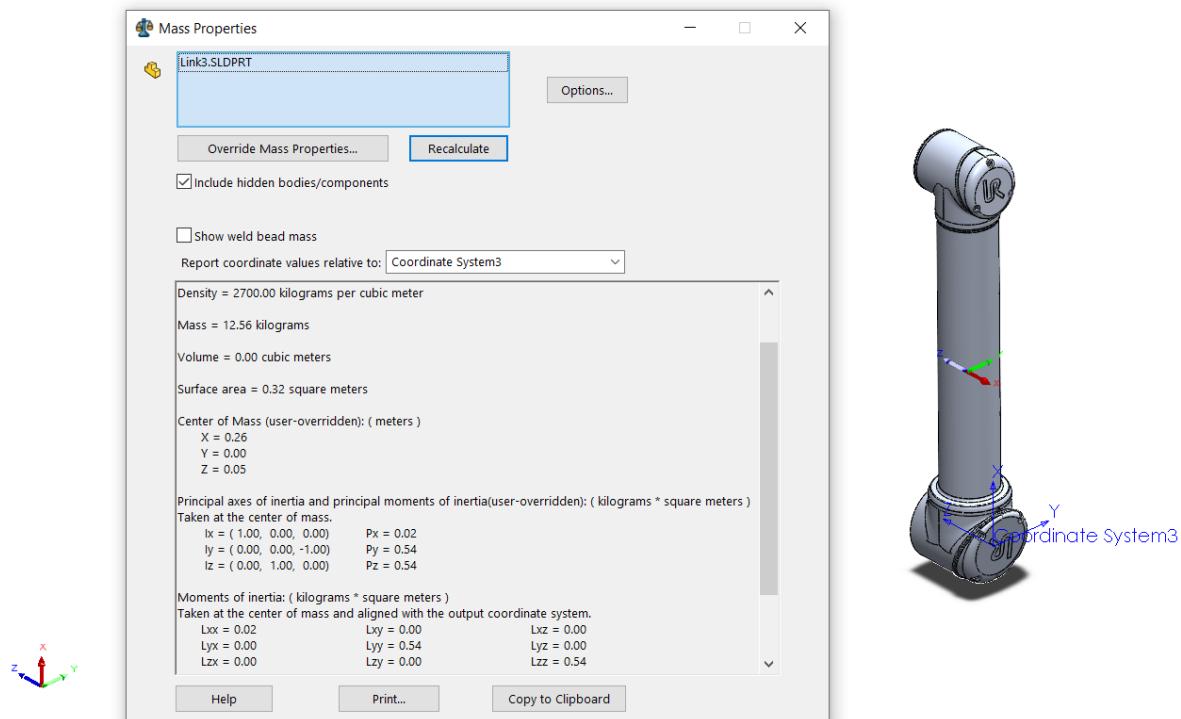


شکل ۲۵ - ممان‌های اینرسی لینک ۲ حول مرکز جرم

۳-۵- مشخصات لینک سوم

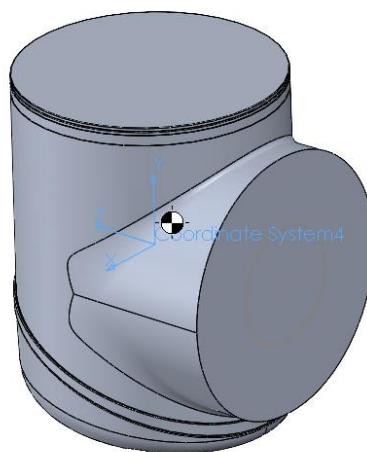


شکل 26 - لینک ۳ به همراه فریم سوم

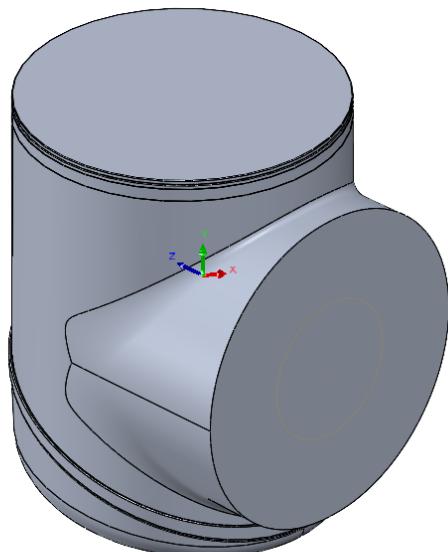
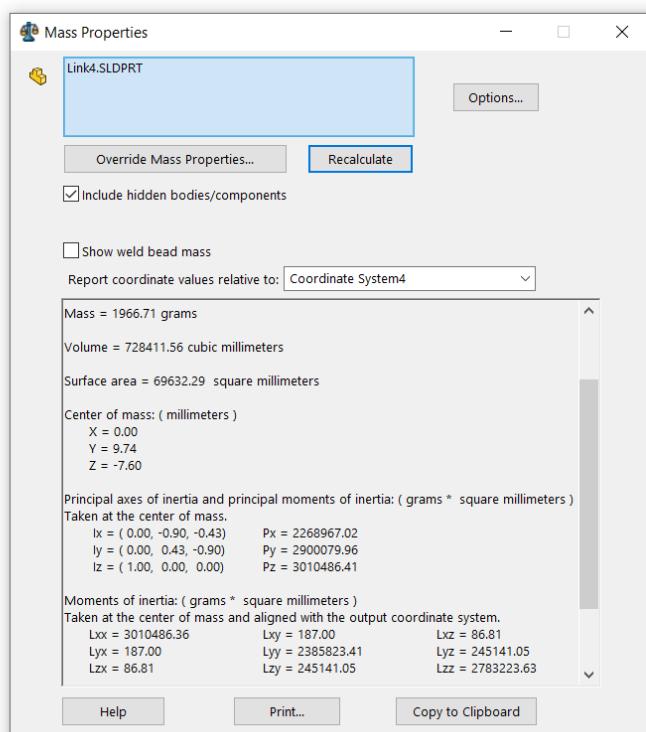


شکل 27 - ممان‌های اینرسی لینک ۳ حول مرکز جرم

۴-۵- مشخصات لینک چهارم

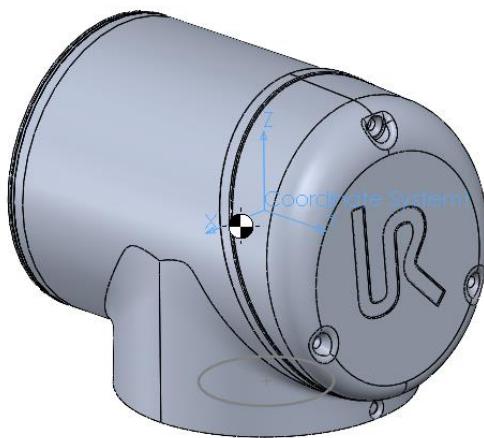


شکل 28 - لینک چهارم به همراه فریم چهارم

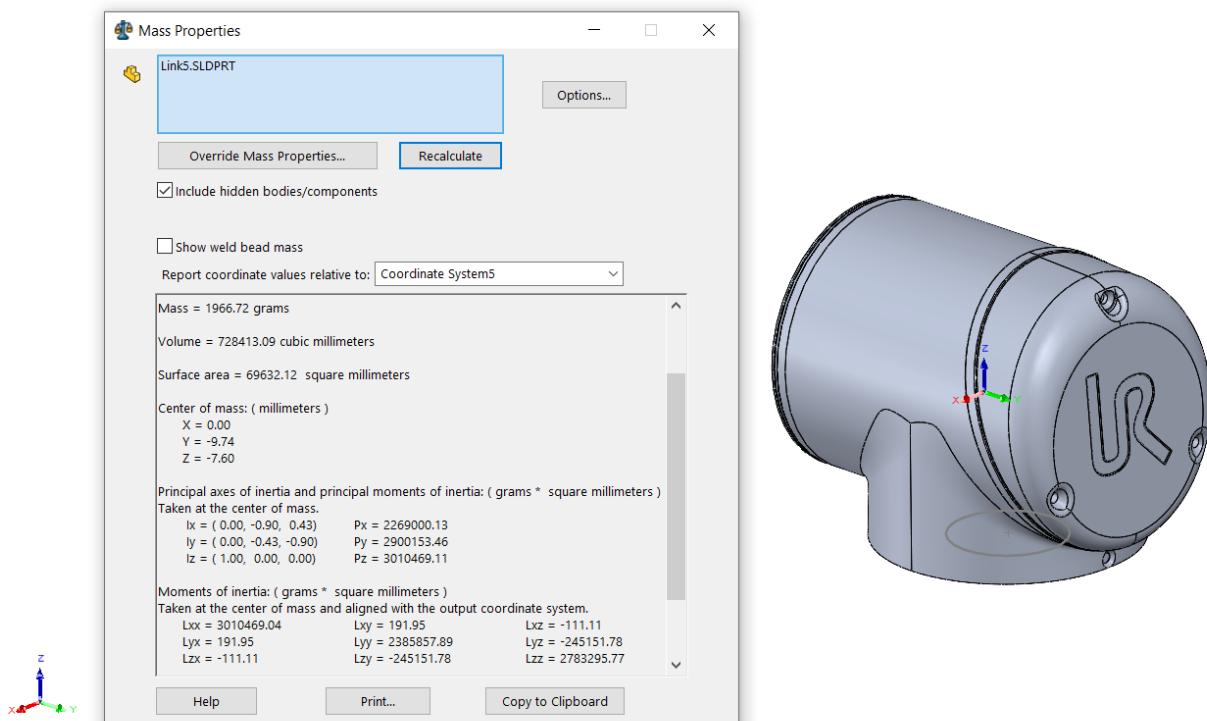


شکل 29 - ممان‌های اینرسی لینک ۴ حول مرکز جرم

۵-۵- مشخصات لینک پنجم

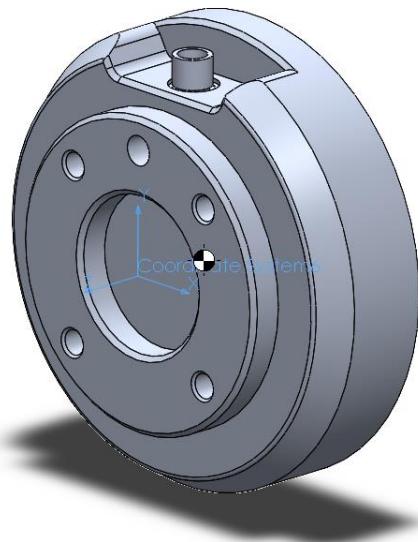


شکل ۳۰ - لینک ۵ به همراه فریم پنجم

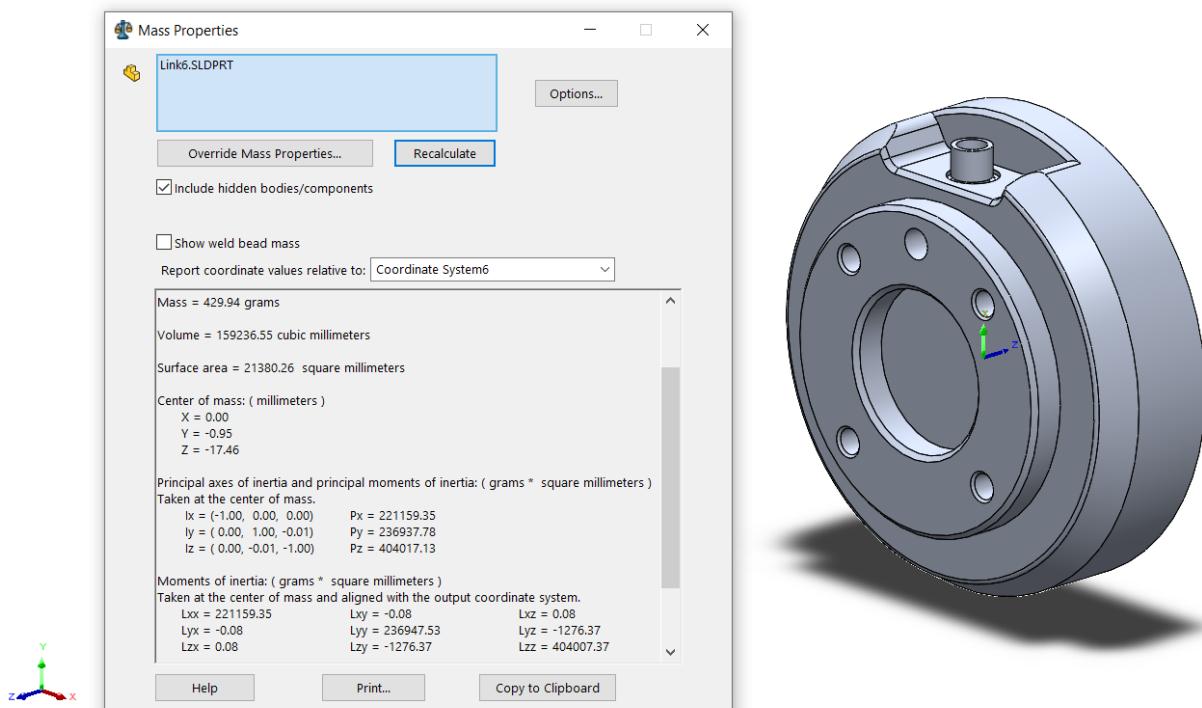


شکل ۳۱ - ممان‌های اینرسی لینک ۵ حول مرکز جرم

۶-۵- مشخصات لینک ششم



شکل 32 - لینک ۶ به همراه فریم ششم



شکل 33 - ممان‌های اینرسی لینک ۶ حول مرکز جرم

فصل ششم
دینامیک

۶-۱- توضیحات اولیه بخش دینامیک

دینامیک در حقیقت بررسی نیروهایی است که منجر به حرکت ربات می‌شوند. حرکت ربات ما میتواند دو تا منشا داشته باشد:

- گشتاورهای اعمال شده توسط Actuatorها
- نیروهای خارجی واردہ به ربات

در این قسمت قصد داریم تا معادلات دینامیکی ربات را به دست آوریم. در رابطه دینامیک ربات با دو مسئله مواجه هستیم:

- ورودی ما مقادیر θ و $\dot{\theta}$ باشد و ما براساس آن‌ها بتوانیم مقادیر مورد نیاز برای گشتاورهای مفاصل را بیابیم. که به آن Inverse Dynamics گویند و برای کنترل ربات به کار می‌رود.
- ورودی ما بردار گشتاور τ باشد و سپس براساس آن مقادیر θ و $\dot{\theta}$ را بیابیم که به آن Forward Dynamics گویند و در شبیه سازی ربات به کار می‌رود.

طبق مطالبی که در درس مطالعه کردیم، برای به دست آوردن معادلات دینامیک دو روش «نیوتون-اویلر» و «لاگرانژ» را داریم که ما در اینجا با هردو روش معادلات را به دست می‌آوریم. لازم به ذکر است که مشخصات لینک‌ها مانند جرم و اینرسی و مرکز جرم و ... در قسمت قبل به کمک فایل‌های سالید به دست آمدند و در این قسمت از آن‌ها افاده خواهیم کرد.

۶-۲- استخراج معادلات دینامیک در فضای متغیرهای مفصلی به روش نیوتن

باتوجه با قواعد و روابطی که برای انتشار سرعت در ربات به دست آوردیم، در گام بعد به کمک مشتق گیری از آنها میتوان شتاب خطی و زاویه‌ای را به دست آورد سپس به کمک دو معادله نیوتون برای حرکت خطی مرکز جرم و معادله اویلر برای دوران جسم صلب حول مرکز جرم معادلات دینامیکی ربات در روش نیوتون-اویلر به دست خواهند آمد.

از آنجایی که تمامی مفاصل ربات ما Revolute میباشد میتوان آن به کمک روابط زیر که در اسلایدهای درس آمده است به راحتی معادلات را استخراج کرد:

Outward iterations

$$\begin{aligned} {}^{i+1}\omega_{i+1} &= {}^{i+1}R_i \ {}^i\omega_i + \dot{\theta}_{i+1} {}^{i+1}\hat{Z}_{i+1} \\ {}^{i+1}\dot{\omega}_{i+1} &= {}^{i+1}R_i \ {}^i\dot{\omega}_i + {}^{i+1}R_i \ {}^i\omega_i \times \dot{\theta}_{i+1} {}^{i+1}\hat{Z}_{i+1} + \ddot{\theta}_{i+1} {}^{i+1}\hat{Z}_{i+1} \\ {}^{i+1}\dot{v}_{i+1} &= {}^{i+1}R_i ({}^i\dot{\omega}_i \times {}^iP_{i+1} + {}^i\omega_i \times ({}^i\omega_i \times {}^iP_{i+1})) + {}^i\dot{v}_i \\ {}^{i+1}\dot{v}_{C_{i+1}} &= {}^i\dot{\omega}_i \times {}^{i+1}P_{C_{i+1}} + {}^{i+1}\omega_{i+1} \times ({}^{i+1}\omega_{i+1} \times {}^{i+1}P_{C_{i+1}}) + {}^{i+1}\dot{v}_{i+1} \\ {}^{i+1}F_{i+1} &= m_{i+1} {}^{i+1}\dot{v}_{C_{i+1}} \\ {}^{i+1}N_{i+1} &= {}^{C_{i+1}}I_{i+1} {}^{i+1}\dot{\omega}_{i+1} + {}^{i+1}\omega_{i+1} \times {}^{C_{i+1}}I_{i+1} {}^{i+1}\omega_{i+1} \end{aligned}$$

در این مرحله باتوجه به اینکه ۶ درجه آزادی داریم، اندیس‌های i از ۰ تا ۵ مقدار دهی میشوند.

Inward Iterations:

$$\begin{aligned} {}^i f_i &= {}^i R_{i+1} {}^{i+1}f_{i+1} + {}^i F_i \\ {}^i n_i &= {}^i N_i + {}^i R_{i+1} {}^{i+1}n_{i+1} + {}^i P_{C_i} \times {}^i F_i + {}^i P_{i+1} \times {}^i R_{i+1} {}^{i+1}f_{i+1} \\ \tau_i &= {}^i n_i^T \hat{Z}_i \end{aligned}$$

در این مرحله، اندیس‌های i از ۱ تا ۶ مقدار دهی میشوند.

دقت شود که سرعت‌های خطی و زاویه‌ای فریم ۰ نسبت به رفرنس بیان شده در همان فریم برابر با ۰ میباشد. همچنین نیروها و گشتاورهای خارجی به ربات وارد نمیشوند پس:

$$\begin{cases} {}^7 f_7 = 0 \\ {}^7 n_7 = 0 \end{cases}$$

در روابط R_i^{i+1} ماتریس‌های دوران داخل ماتریس‌های تبدیلی است که در سینماتیک مستقیم به دست آورده‌یم منتها باید حواسمن باشد که Transpose شده است. همچنین P_{i+1}^i ستون چهارم آن ماتریس‌های تبدیل می‌باشد.

در نهایت با توجه به معادلات به دست آمده می‌توان ماتریس M و بردارهای G و V را استخراج کرد. میدانیم که ضرایب $\ddot{\theta}$ همان درایه‌های ماتریس M را می‌سازند. پس با توجه به معادلات به دست آمده می‌توان از آن‌ها نسبت به هریک از $\dot{\theta}_1$ تا $\dot{\theta}_6$ مشتق گرفت و ترم‌های دیگر \cdot می‌شوند و ضرایب آنها پیدا می‌شود.

برای یافتن بردار V کافی است تا مقدار $\ddot{\theta}$ ‌ها و همچنین g را در عبارات پارامتری \cdot نماییم تا آنچه باقی می‌ماند فقط V باشد چراکه V فقط تابع θ و $\dot{\theta}$ است. در نهایت برای یافتن G نیز می‌توان از معادلات برحسب g مشتق گرفت چرا که G فقط ترم‌هایی را در بر دارد که g در آن‌ها ظاهر شده باشد.

$$\tau = M(\theta)\ddot{\theta} + V(\theta, \dot{\theta}) + G(\theta)$$

به این ترتیب معادلات در فضای مفصلی استخراج می‌شوند. کد مربوط به این قسمت با پایتون زده شده است که در پیوست قرار گرفته است.

۶-۳- استخراج معادلات دینامیک در فضای متغیرهای مفصلی به روش لاغرانژ

در روش لاغرانژ مبنا براین است تا با نوشتن انرژی پتانسیل و انرژی جنبشی و مشتق گرفت برحسب مختصات‌های تعیین یافته، معادلات دینامیک را استخراج کرد. معادله کلی لاغرانژ به صورت زیر می‌باشد:

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_k} - \frac{\partial L}{\partial q_k} = \tau_k, \quad k = 1, \dots, n$$

در نهایت چیزی که ما نیاز داریم آن است که ماتریس M و بردارهای C و G را به دست آوریم. برای این منظور روابط زیر را داریم:

$$M(\theta) = \sum_{i=1}^n \left\{ m_i J_{v_{c_i}}^T(\theta) J_{v_{c_i}}(\theta) + J_{\omega_{c_i}}^T(\theta) R_i(\theta)^T I_i R_i^T(\theta) J_{\omega_i}(\theta) \right\}$$

$$c_{kj} = \sum_{i=1}^n c_{ijk}(\theta) \dot{\theta}_i = \sum_{i=1}^n \frac{1}{2} \left\{ \frac{\partial m_{kj}}{\partial \theta_i} + \frac{\partial m_{ki}}{\partial \theta_j} + \frac{\partial m_{ij}}{\partial \theta_k} \right\} \dot{\theta}_i \rightarrow C(\theta, \dot{\theta}) \sqrt{}$$

$$\begin{cases} g_k = \frac{\partial P}{\partial \theta_k} \\ G(\theta) = [g_1(\theta) \dots g_n(\theta)] \end{cases}$$

پس از به دست آوردن ماتریس و بردارهای فوق به راحتی می‌توان به State Space form زیر رسید:

$$M(\theta) \ddot{\theta} + C(\theta, \dot{\theta}) \dot{\theta} + G(\theta) = \tau$$

در این روش خوبیش آن است که ماتریس‌ها و بردارها به صورت مستقیم به دست می‌آیند و نیازی نیست تا آن‌ها را تفکیک کنیم. در نهایت هم می‌توان این ماتریس‌ها را در شبیه ساز مطلب و سیمولینک استفاده کرد تا مسئله Forward Dynamics را به راحتی حل کنیم.

کد مربوط به این قسمت در متلب زده شده و در پیوست قرار گرفته است.

۴-۶- استخراج معادلات دینامیک در فضای کارتزین

در قسمت قبل توانستیم در نهایت معادلات دینامیک را در فضای مفاصل بنویسیم و به فرمت زیر برسیم:

$$\tau = M(\theta)\ddot{\theta} + V(\theta, \dot{\theta}) + G(\theta)$$

اکنون میخواهیم بین Joint Space و شتاب Cartesian رابطه برقرار کنیم. با توجه به بخش‌های قبل میدانیم که بین سرعت در فضای کارتزین و فضای مفصلی به کمک ژاکوبین رابطه زیر را داشتیم:

$$\dot{X} = J(\theta)\dot{\theta}$$

حال اگر از طرفین یک مشتق زمانی بگیریم خواهیم داشت:

$$\ddot{X} = J(\theta)\ddot{\theta} + j(\theta)\dot{\theta}$$

و اگر در طرفین آن J^{-1} را ضرب کنیم:

$$\ddot{\theta} = J^{-1}(\theta)\ddot{X} - J^{-1}(\theta)j(\theta)\dot{\theta}$$

حال اگر این معادله به دست آمده را در همان رابطه‌ی ابتدایی بخش جایگذاری کنیم خواهیم داشت:

$$\tau = M(\theta)(J^{-1}\ddot{X} - J^{-1}j(\theta)\dot{\theta}) + V(\theta, \dot{\theta}) + G(\theta)$$

حال میتوان معادله جدید را به صورت زیر تعریف کرد:

$$\tau = M_X(\theta)\ddot{\theta} + V_X(\theta, \dot{\theta}) + G_X(\theta)$$

که در آن روابط زیر بین پارامترهای فعلی و معادله قبل برقرار است:

$$\begin{cases} M_X(\theta) = M(\theta)J^{-1}(\theta) \\ V_X(\theta, \dot{\theta}) = V(\theta, \dot{\theta}) - M(\theta)J^{-1}(\theta)j(\theta)\dot{\theta} \\ G_X(\theta) = G(\theta) \end{cases}$$

دقیق کنید که $J(\theta)$ باید در فریم یکسانی با \ddot{X} نوشته شده باشد. به این ترتیب به راحتی با ۳ رابطه به دست آمده میتوان معادلات دینامیک در فضای کارتزین را نیز به دست آورد.

که مربوطه در مطلب نوشته شده است.

۶-۵- شبیه سازی معادلات دینامیک (فضای متغیرهای مفصلی) در Matlab

ماتریس های M و بردارهای V و G بسیار بزرگ هستند و کار با آنها بسیار سخت و زمانگیر میباشد. میتوان نشان داد که ماتریس M تا حد خوبی ثابت میماند و به ازای مقدار مختلف ترا میتوان مقدار ثابتی برای آن در نظر گرفت.

میدانیم که M تابع تنا میباشد. حال به ازای تناهای 45° و 90° درجه برای M خواهیم داشت:

```
M(pi/4)
[ 5.271, -1.845, -0.5544, 0.02976, 0.004875, 0.0001888]
[ -1.845, 16.35, 4.977, -0.2353, 0.03281, 0.0005992]
[ -0.5544, 4.977, 2.653, -0.06294, 0.01071, 0.0004741]
[ 0.02976, -0.2353, -0.06294, 0.03599, -0.003871, 0.0002748]
[ 0.004875, 0.03281, 0.01071, -0.003871, 0.005619, 2.159e-5]
[0.0001888, 0.0005992, 0.0004741, 0.0002748, 2.159e-5, 0.0004004]
```

شکل ۳۴ - ماتریس M برای تناهای ۴۵ درجه

```
M(pi/2)
[ 3.432, -1.795, 0.008376, 0.008376, -0.02378, -0.0003334]
[ -1.795, 11.52, 2.499, -0.1548, -0.0001862, -0.0002198]
[ 0.008376, 2.499, 2.531, -0.1234, -0.0001862, 3.053e-5]
[ 0.008376, -0.1548, -0.1234, 0.03756, 4.726e-5, 3.053e-5]
[ -0.02378, -0.0001862, -0.0001862, 4.726e-5, 0.005609, 0]
[ -0.0003334, -0.0002198, 3.053e-5, 3.053e-5, 0, 0.0004004]
```

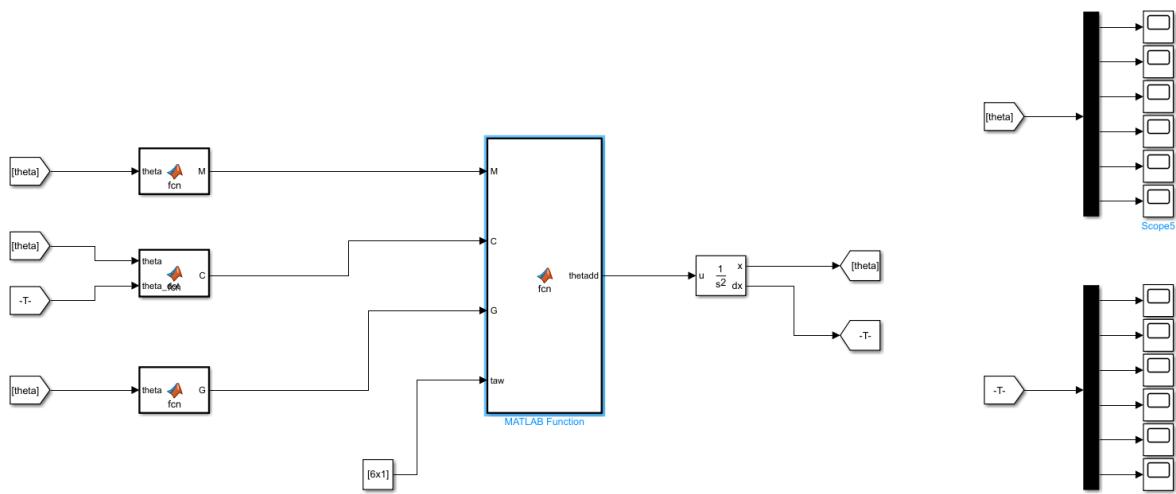
شکل ۳۵- ماتریس M برای تناهای ۹۰ درجه

```

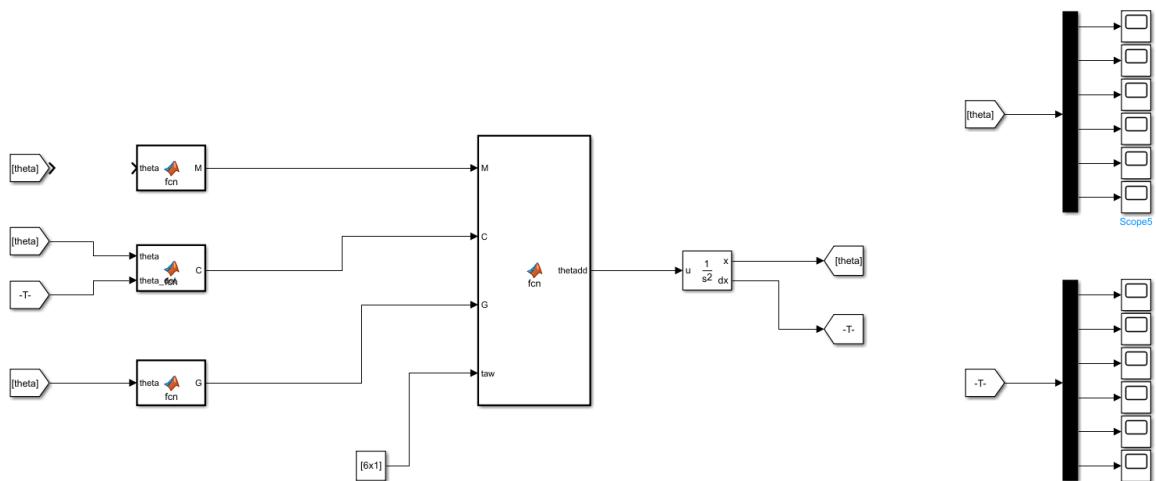
MATLAB Function1
Simulation_in_Simulink MATLAB Function1
50
51 c51= - td2*((49450651*sin(t2 + t3 + t4 - t5))/250000000000 - (7474967197*sin(t2 + t3 + t4 - 2*t5))/40000000000000 - (7474967197*sin(t2 + t3 + t4 + 2*t5))/40000000000000 + (784477*sin(t5)^2*t6^2));
52 c52=td6*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)^2)/80000000000 - (817*d6*cos(t5)*sin(t6));
53 c53=td6*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)^2)/80000000000 - (817*d6*cos(t5)*sin(t6));
54 c54=td6*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)^2)/80000000000 - (817*d6*cos(t5)*sin(t6));
55 c55=td6*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)^2)/80000000000 - (817*d6*cos(t5)*sin(t6));
56 c56=td6*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)^2)/80000000000 - (817*d6*cos(t5)*sin(t6));
57 c57=td6*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)^2)/80000000000 - (817*d6*cos(t5)*sin(t6));
58 c58=td6*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)^2)/80000000000 - (817*d6*cos(t5)*sin(t6));
59 c59= - (784477*t6*d6*sin(2*t6))/80000000000;
60
61 c56=td2*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)^2)/80000000000 - (817*d6*cos(t5)*sin(t6));
62
63 c61=td4*((713241*sin(t2 + t3 + t4 + t5 - t6))/800000000000 - (713241*sin(t2 + t3 + t4 + t5 + t6))/800000000000 - (713241*sin(t2 + t3 + t4 - t5 + t6))/800000000000 - (784477*sin(t2 + t3 + t4 - t5 + t6))/800000000000);
64
65 c62=td2*((784477*cos(t5)^2*cos(t6)*sin(t6))/40000000000 - (817*d5*sin(t6))/2000000 - (784477*cos(t6)*sin(t5))/40000000000 + (817*a3*sin(t4)*sin(t6))/2000000 + (713241*cos(t5)*cos(t6));
66
67 c63=td1*((713241*sin(t2 + t3 + t4 + t5 - t6))/800000000000 - (713241*sin(t2 + t3 + t4 + t5 + t6))/800000000000 - (713241*sin(t2 + t3 + t4 - t5 + t6))/800000000000 - (784477*sin(t2 + t3 + t4 - t5 + t6))/800000000000);
68
69 c64=td1*((713241*sin(t2 + t3 + t4 + t5 - t6))/800000000000 - (713241*sin(t2 + t3 + t4 + t5 + t6))/800000000000 - (713241*sin(t2 + t3 + t4 - t5 + t6))/800000000000 - (784477*sin(t2 + t3 + t4 - t5 + t6))/800000000000);
70
71 c65=(784477*td5*sin(2*t6))/80000000000 - td2*((16015523*sin(t5))/80000000000 + (713241*cos(t5)*sin(t6))/100000000000 + (784477*cos(t6)^2*sin(t5))/80000000000 - (784477*sin(t5)*sin(t6)));
72
73 c66=0;
74
75 C=[c11 c12 c13 c14 c15 c16
76     c21 c22 c23 c24 c25 c26
77     c31 c32 c33 c34 c35 c36
78     c41 c42 c43 c44 c45 c46
79     c51 c52 c53 c54 c55 c56
80     c61 c62 c63 c64 c65 c66];
81
82 end

```

شکل ۳۶- تعریف ماتریس C بصورت درایه به درایه



شکل ۳۷- شبیه سازی با فیدبک تنا



شکل ۳۸- شبیه سازی بدون فیدبک تنا

فایل های متلب و سیمولنک این قسمت قرار داده شده است.

۶-۶- صحه گذاری معادلات دینامیک

باتوجه به اینکه از هر دو روش لگرانژ و نیوتون-اویلر معادلات را به دست آوردهیم میتوان با جاگذاری تناها و تنتاداتها ماتریس و بردارهای M و V و G را با یک دیگر مقایسه نمود.

در این دو روش مقادیر مربوط به تناها را برابر با \cdot قرار دادیم. همانطور که دیده میشود بردارها و ماتریس‌های مورد نظر تا حد زیادی مطابقت دارند. همچنین ماتریس M یک ماتریس متقاضن شده است که مورد انتظار ما میباشد.

- با کمک روش نیوتون در کد پایتون به نتایج زیر رسیدیم:

```
In [33]: M_num
Out[33]: [[ 12.782265545108, -0.0511883109442, -0.0511883109442, -0.0511883109442, 0.0140356429114, 9.7480355 · 10-5 ],
           [-0.0511883109442, 12.6494261034502, 3.5311272994102, 0.0344342513142, -0.005508900948, 0.000353124625 ],
           [-0.0511883109442, 3.5311272994102, 2.0002698082102, 0.0344342513142, -0.005508900948, 0.000353124625 ],
           [-0.0511883109442, 0.0344342513142, 0.0344342513142, 0.0344342513142, -0.005508900948, 0.000353124625 ],
           [ 0.0140356429114, -0.005508900948, -0.005508900948, 0.0056286136372, 3.053129 · 10-5 ],
           [ 9.7480355 · 10-5, 0.000353124625, 0.000353124625, 0.000353124625, 3.053129 · 10-5, 0.000400388075 ]]

In [34]: V_num
Out[34]: [[ 1.94289029309402 · 10-16 ],
           [ 0.33300296241 ],
           [ 0.33300296241 ],
           [ 0.33300296241 ],
           [ -0.06076103591 ],
           [ -0.00048386825 ]]

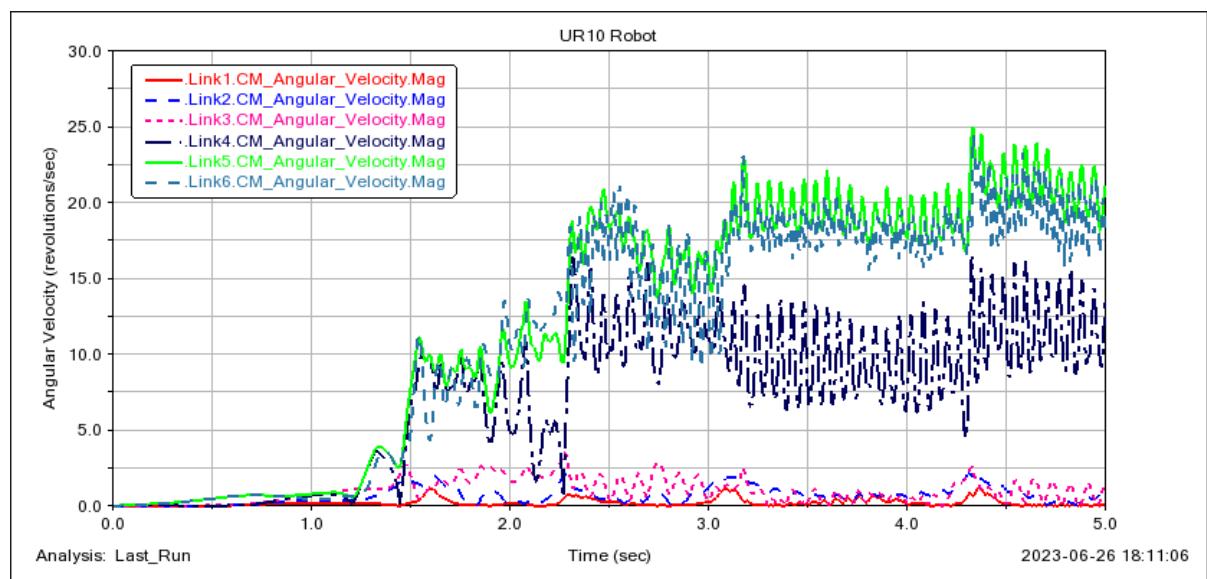
In [35]: G_num
Out[35]: [[ 0 ],
           [ 126.3167737356 ],
           [ 24.50271168 ],
           [ 0 ],
           [ 0 ],
           [ 0 ]]]
```

- با کمک روش لگرانژ نیز در کد مطلب با همان شرایط گفته شده به مقادیر زیر رسیدیم:

Command Window	
M(0)	
[19.81, -0.05119, -0.05119, -0.05119, 0.01404, 9.748e-5]	
[-0.05119, 18.96, 6.38, 0.03443, -0.005509, 0.0003531]	
[-0.05119, 6.38, 2.849, 0.03443, -0.005509, 0.0003531]	
[-0.05119, 0.03443, 0.03443, 0.03443, -0.005509, 0.0003531]	
[0.01404, -0.005509, -0.005509, -0.005509, 0.005629, 3.053e-5]	
[9.748e-5, 0.0003531, 0.0003531, 0.0003531, 3.053e-5, 0.0004004]	
Command Window	Command Window
V	G(0)
0	0
0.333	22.01*g
0.333	5.76*g
0.333	0
-0.06076	0
-0.0004839	0

۷-۶- شبیه سازی حرکت ربات در نرم افزار ادمز

برای شبیه سازی این قسمت، حالت ساده شده ای را درنظر گرفتیم که در آن، ربات بطور عمودی (خلاف جهت گرانش) قرار داشته و بدون هیچ شرط اولیه سرعت و جابجایی ای رها می شود. درحالت ایدهآل انتظار می رود در نبود اغتشاشاتی مثل باد و ... ربات در وضع تعادل خود بماند. اما پس از لحظاتی بعد از رها شدن، تعادل خود را از دست داده و فرو می ریزد. این حرکات و نوسات بسیار کوچک از سمت اندافکتور آغاز شده و به کل ربات سرایت می کند. بعد از اینکه ربات از وضع تعادل خود خارج شد، حرکات پیش بینی نشده ای دارد و این حرکات بخاطر عدم وجود هیچ نیروی اتلاف کننده ای مثل اصطکاک ویسکوز مفاصل تا ابد ادامه می یابد.

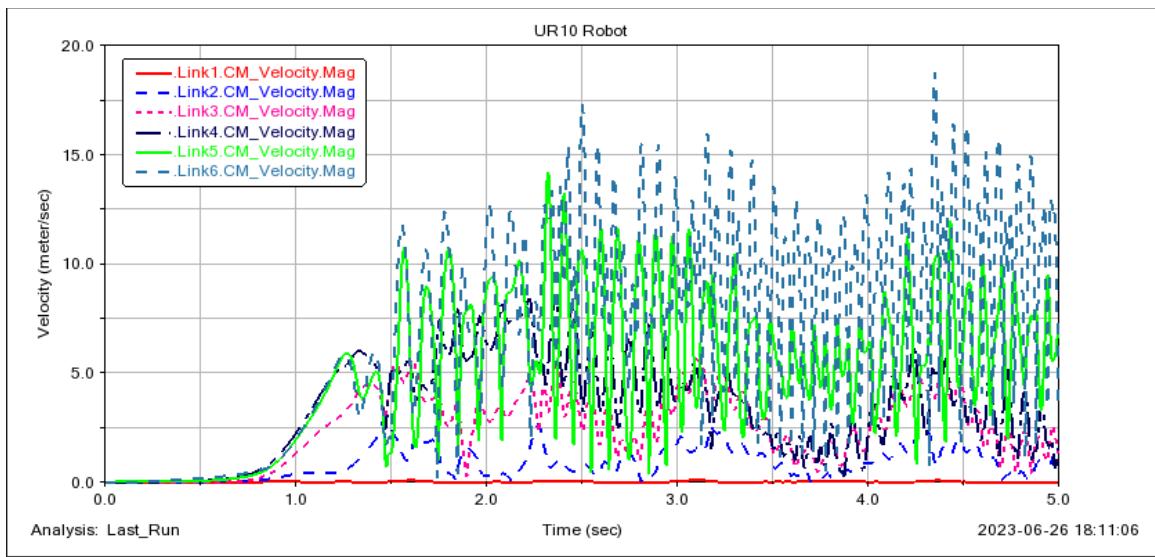


نمودار ۱- نمودار سرعت زاویه‌ای مرکز جرم ربات در اثر نیروی گرانش

در نمودار بالا و همه نمودارهای بعدی، شبیه سازی تا ثانیه پنجم صورت گرفته است. همانطور که از روی نمودار بالا مشخص است، ابتدا لینک‌های نزدیک به اندافکتور که اینرسی دورانی کمتری دارند، سرعت گرفته و پیوسته به سرعت زاویه ای آنها افروده می شود.

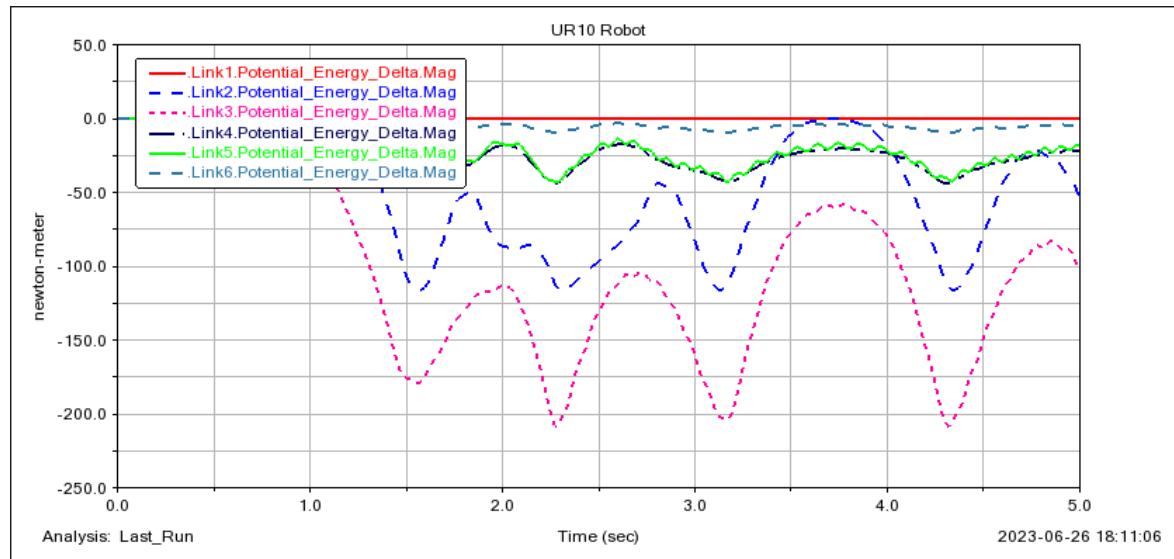
یکی از دلایلی که باعث از دست رفتن تعادل ربات در عدم حضور نیروی خارجی و اغتشاشات می شود، بخاطر حکاکی‌های روی موتورهای نزدیک اندافکتور است. بنظر می آید همین عدم تقارن بسیار کم باعث ایجاد چنین حرکتی شده است.

در فایل آپلود شده، در پوشه Adams ویدئویی از این شبیه سازی ضبط شده است.



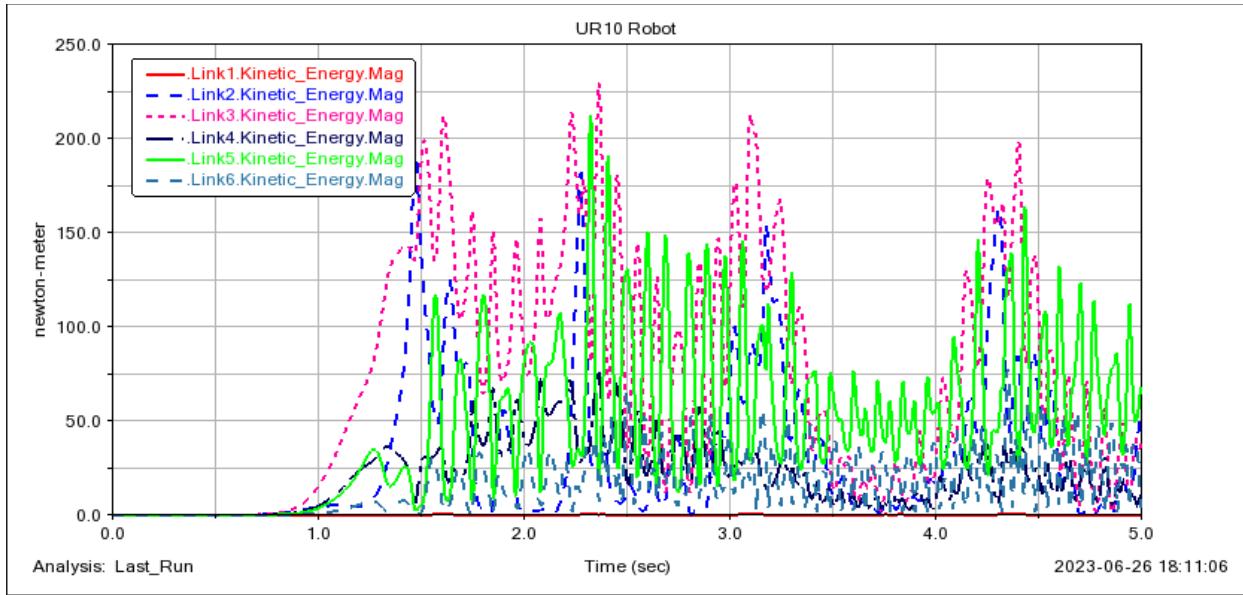
نمودار ۲- سرعت خطی مراکز جرم ربات در اثر نیروی گرانش

این نمودار نیز نشان دنده سرعت خطی مراکز جرم در حضور نیروی گرانش و بدون اعمال هیچ گونه گشتاور خارجی ای به مفاصل می‌باشد. نمودار قرمز رنگ، سرعت خطی مرکز جرم لینک اول است که به علت بسیار نزدیک بودن مرکز جرم به محور دوران لینک اول، سرعتی بسیار کم و در حدود صفر دارد.

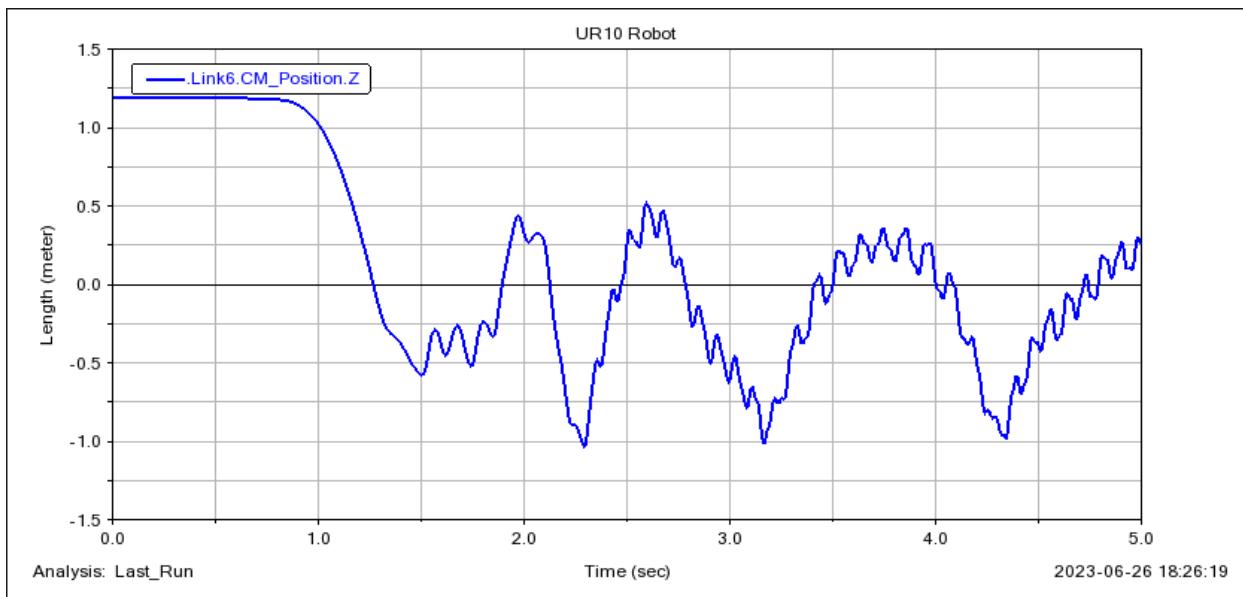


نمودار ۳- تغییرات انرژی پتانسیل اعضای ربات

نمودار رسم شده در بالا، انرژی پتانسیل لینک‌های ربات را نشان می‌دهد که فرم کلی آن مثل نمودار سرعت خطی مراکز جرم ولی درجهت قرینه است. از این موضوع می‌توان اینطور برداشت کرد که برای این ربات که هیچ گونه میرایی ای در آن لحاظ نشده، پایستاری انرژی برقرار بوده و انرژی‌های جنبشی و پتانسیل بطور پیوسته در حال تبدیل شدن به یکدیگر اند.



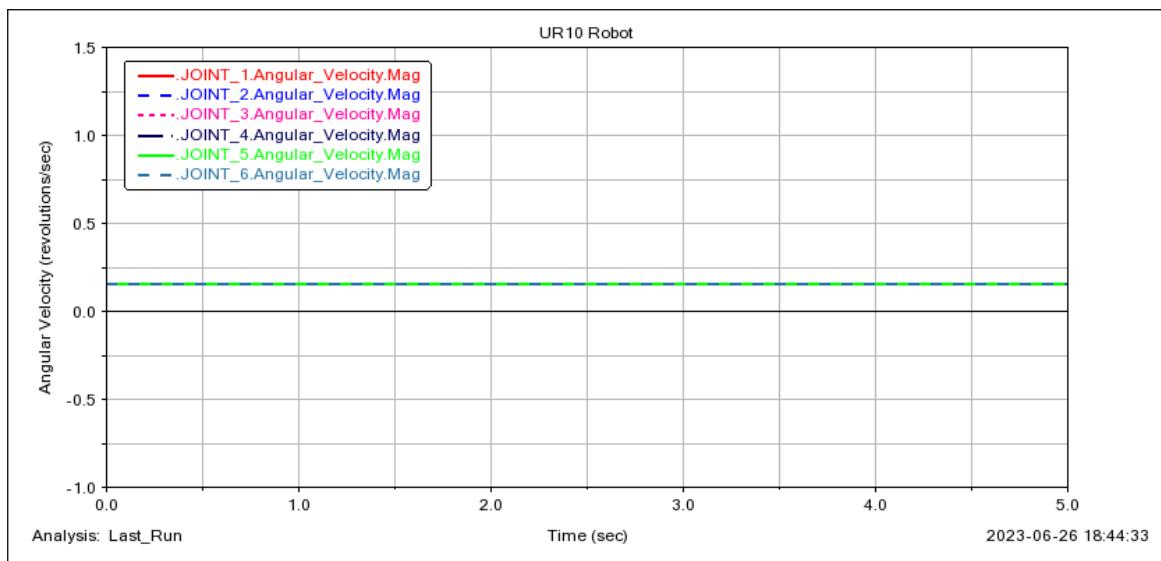
نمودار ۴- تغییرات انرژی جنبش اعضای ربات



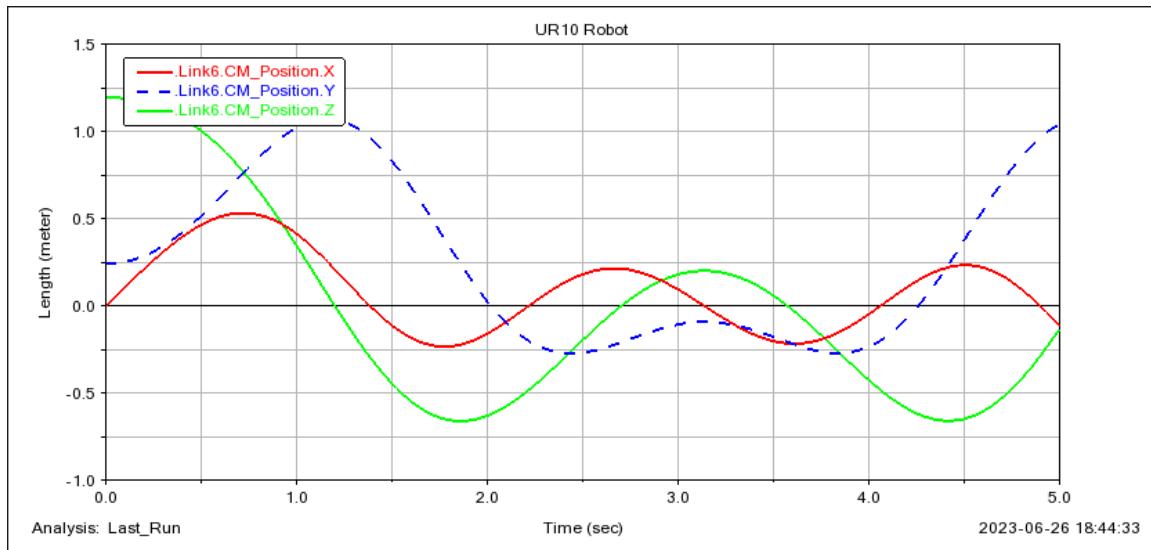
نمودار ۵- نمودار تغییرات ارتفاع اندافکتور

همانطور که در قبل گفته شد، ربات پس از رها شدن در میدان گرانش، برای لحظاتی تعادل خود را حفظ کرده ولی با خاطر عدم تقارن اسم حک شده در پشت موتور اندافکتور، گشتاور خالصی به آن وارد شده که باعث سقوط یکباره ربات می‌گردد. بعد از این اتفاق، اندافکتور دیگر به مکان اولیه خود برنگشته و حول ارتفاع صفر، نوسانات نامنظمی خواهد داشت.

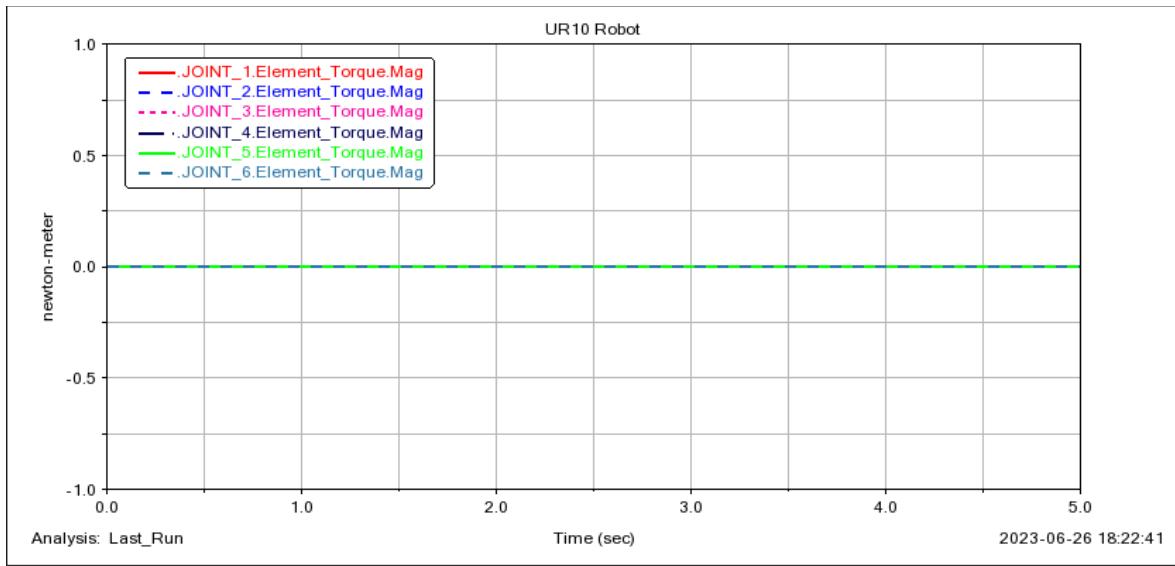
۶-۷-۱- اعمال سرعت زاویه‌ای ثابت به مفاصل و تحلیل نتایج



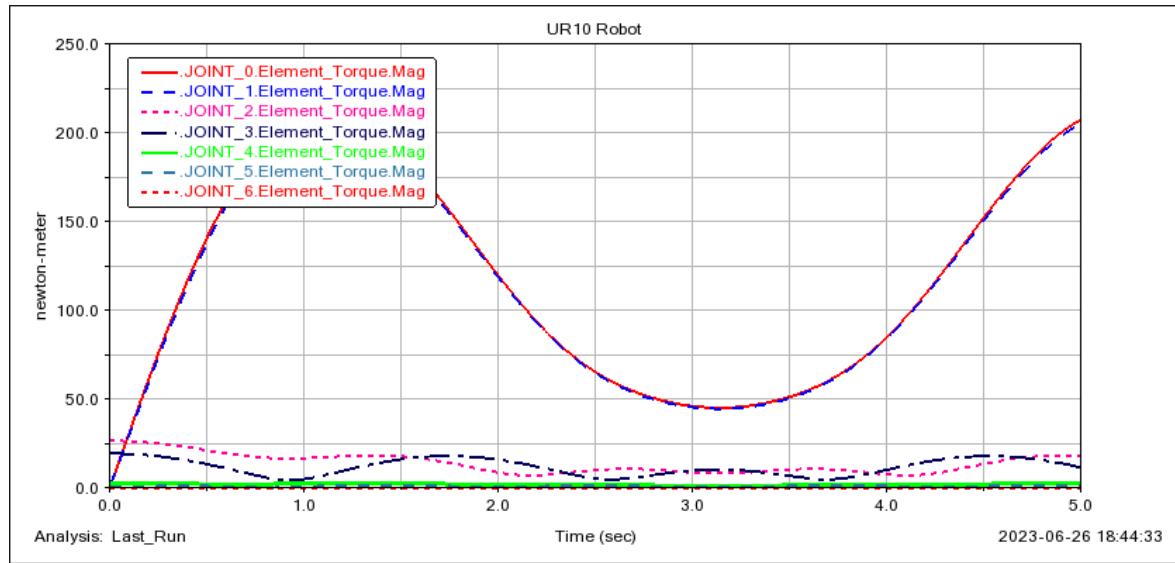
نمودار ۶- سرعت زاویه‌ای ورودی یک رادیان بر ثانیه به هر مفصل



نمودار ۷- حرکت اندازه‌گیر تحت سرعت زاویه‌ای ثابت مفاصل



نمودار ۱- مقدار گشتاور اعمالی به مفاصل در حرکت سقوط آزاد



نمودار ۹- مقدار گشتاور اعمالی به مفاصل در حرکت با سرعت ثابت

همانطور که انتظار می‌رفت، در شبیه سازی حرکت ربات تحت نیروی گرانش، هیچ گونه گشتاور را مفاصل تحمل نکرده و حرکت آنها آزادانه صورت می‌گیرد. این گشتاور در واقع همان گشتاوری است که موتورهای ربات به مفاصل وارد می‌کنند و صفر بودن آنها نشان از عدم دخالت و مقاومت موتورها می‌باشد.

در قسمت حرکت با سرعت ثابت، گشتاورها دیگر صفر نیستند چرا که باید یک گشتاور کنترل کننده از سمت موتور وجود داشته باشد تا حرکت با سرعت ثابت رخ دهد. در نمودار مربوط به این قسمت، ۲ نمودار قرمز و آبی برهم منطبق شده اند که نمودار آبی مقدار گشتاور اعمالی به مفصل اول و دیگری نماینده گشتاوری است که از سمت پایه (مفصل صفرم) باید تحمل شود تا دوران صورت بگیرد. این همان قانون عمل و عکس العمل

نیوتن است که بخاطر ترسیم اندازه گشتاورها، این ۲ نمودار قرینه نشده اند. همانطور که از شکل هم پیداست، هرچه لینک‌ها به پایه نزدیک‌تر باشند، ابعاد و جرم بیشتری داشته، و هم باید لینک‌های بعدی را دوران دهد، درنتیجه مقدار گشتاوری که باید به آنها اعمال شود هم بیشتر است.

مراجع

- [1] John J. Craig, “Introduction to Robotics: Mechanics and Control”
- [2] Mark W. Spong, Seth Hutchinson, M. Vidyasagar, ”Robot Modeling and Control”
- [3] Qiang Liu, Daoguo Yang, Weidong Hao, Yao Wei, ” Research on Kinematic Modeling and Analysis Methods of UR Robot”.
- [4] <https://ieeexplore.ieee.org/document/8740681>
- [5] https://www.google.com/url?sa=i&url=https%3A%2F%2Fs3-eu-west-1.amazonaws.com%2Fur-support-site%2F41472%2F1000700.PDF&psig=AOvVaw08Hq3ibK5y_GzqLvZe6v1-&ust=1679395284725000&source=images&cd=vfe&ved=2ahUKEwj8mquMquer9AhWx2rsIHTtuAMEQr4kDegUIARC_AQ
- [6] https://www.pishrobot.com/wp-content/uploads/2017/04/ur10_details.pdf



Technical details

UR10

Performance

Repeatability	±0.1 mm / ±0.0039 in (4 mils)
Temperature range	0-50°
Power consumption	Min 90W, Typical 250W, Max 500W
Collaboration operation	15 advanced adjustable safety functions. TÜV NORD Approved Safety Function Tested in accordance with: EN ISO 13849:2008 PL d

Specification

Payload	10 kg / 22 lbs
Reach	1300 mm / 51.2 in
Degrees of freedom	6 rotating joints
Programming	Polyscope graphical user interface on 12 inch touchscreen with mounting

Movement

Axis movement robot arm	Working range	Maximum speed
Base	± 360°	± 120°/Sec.
Shoulder	± 360°	± 120°/Sec.
Elbow	± 360°	± 180°/Sec.
Wrist 1	± 360°	± 180°/Sec.
Wrist 2	± 360°	± 180°/Sec.
Wrist 3	± 360°	± 180°/Sec.
Typical tool		1 m/Sec. / 39.4 in/Sec.

Features

IP classification	IP54
ISO Class Cleanroom	5
Noise	Comparatively noiseless
Robot mounting	Any
I/O ports	Digital in 2 Digital out 2 Analog in 2 Analog out 0
I/O power supply in tool	12 V/24 V 600 mA in tool

Physical

Footprint	Ø 190mm
Materials	Aluminium, PP plastics
Tool connector type	M8
Cable length robot arm	6 m / 236 in
Weight with cable	28.9 kg / 63.7 lbs

* The robot can work in a temperature range of 0-50°C. At high continuous joint speed, ambient temperature is reduced.

CONTROL BOX

Features

IP classification	IP20
ISO Class Cleanroom	6
Noise	<65dB(A)
I/O ports	Digital in 16 Digital out 16 Analog in 2 Analog out 2
I/O power supply	24V 2A
Communication	TCP/IP 100Mbit, Modbus TCP, Profinet, EthernetIP
Power source	100-240 VAC, 50-60 Hz

Physical

Control box size (WxHxD)	475mm x 423mm x 268mm / 18.7 x 16.7 x 10.6 in
Weight	17 kg / 37.5 lbs

Materials Steel

TEACH PENDANT

Features

IP classification	IP20
--------------------------	------

Physical

Materials	Aluminium, PP
Weight	1.5 kg / 3.3 lbs
Cable length	4,5 m / 177 in



Robotics project

Analyzing 6-DOF UR10 robot arm



Import libraries

```
In [2]: from sympy import *
import numpy as np
import math
from math import degrees
```

Part1

Forward Kinematics

```
In [122]: # Define the symbolic variables
num_symbols = 6 # Number of symbols to generate
alpha_names = [f"alpha{i}" for i in range(num_symbols+1)]
a_names = [f"a{i}" for i in range(num_symbols+1)]
d_names = [f"d{i}" for i in range(num_symbols+1)]
theta_names = [f"theta{i}" for i in range(num_symbols+1)]

alpha = symbols(alpha_names)
a = symbols(a_names)
d = symbols(d_names)
theta = symbols(theta_names)

# Define DH table
DH_param = [[0, 0, d[1], theta[1]],
             [pi/2, 0, 0, theta[2]],
             [0, a[2], 0, theta[3]],
             [0, a[3], d[4], theta[4]],
             [-pi/2, 0, d[5], theta[5]],
             [pi/2, 0, d[6], theta[6]]]

# find homogeneous transformations(T_i_i-1)
T = []
for i in range(6):
    T_temp = [[cos(DH_param[i][3]), -sin(DH_param[i][3]), 0, DH_param[i][1]],
               [sin(DH_param[i][3])*cos(DH_param[i][0]), cos(DH_param[i][3])*cos(DH_param[i][0]), -sin(DH_param[i][0]), -sin(DH_param[i][0])*cos(DH_param[i][2])],
               [sin(DH_param[i][3])*sin(DH_param[i][0]), cos(DH_param[i][3])*sin(DH_param[i][0]), cos(DH_param[i][0]), cos(DH_param[i][0])*cos(DH_param[i][2])],
               [0, 0, 0, 1]]
    T.append(T_temp)

# print homogeneous transformations(T_i_i-1)
for i in range(len(T)):
    print('T_{:,i}'.format(i+1), ': \n')
    pprint(Matrix(np.array(T[i])))
    print()
```

T_0_1 :

$$\begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_1_2 :

$$\begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_2_3 :

$$\begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & a_2 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_3_4 :

$$\begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & a_3 \\ \sin(\theta_4) & \cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_4_5 :

$$\begin{bmatrix} \cos(\theta_5) & -\sin(\theta_5) & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ -\sin(\theta_5) & -\cos(\theta_5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_5_6 :

$$\begin{bmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 & 0 \\ 0 & 0 & -1 & -d_6 \\ \sin(\theta_6) & \cos(\theta_6) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
In [123]: # find T_0_6 and print it
result = T[0]
for n in range(1,len(T)):
    # Multiply the matrices
    result = [[sum(result[i][k] * T[n][k][j] for k in range(4)) for j in range(4)] for i in range(4)]
# Simplify the result according to cosine multiplication rules
simplified_result = [[trigsimp(expr) for expr in row] for row in result]

# Print the simplified result
print('T_0_6 :')
Matrix(np.array(simplified_result))

T_0_6 :
```

$$\begin{bmatrix} (-\sin(\theta_1)\sin(\theta_5) + \cos(\theta_1)\cos(\theta_5)\cos(\theta_2 + \theta_3 + \theta_4))\cos(\theta_6) - \sin(\theta_6)\sin(\theta_2 + \theta_3 + \theta_4)\cos(\theta_1) & -(-\sin(\theta_1)\sin(\theta_5) + \cos(\theta_1)\cos(\theta_5)\cos(\theta_2 + \theta_3 + \theta_4))\sin(\theta_6) - \sin(\theta_2 + \theta_3 + \theta_4)\cos(\theta_1) \\ (\sin(\theta_1)\cos(\theta_5)\cos(\theta_2 + \theta_3 + \theta_4) + \sin(\theta_5)\cos(\theta_1))\cos(\theta_6) - \sin(\theta_1)\sin(\theta_6)\sin(\theta_2 + \theta_3 + \theta_4) & -(\sin(\theta_1)\cos(\theta_5)\cos(\theta_2 + \theta_3 + \theta_4) + \sin(\theta_5)\cos(\theta_1))\sin(\theta_6) - \sin(\theta_1)\sin(\theta_2 + \theta_3 + \theta_4) \\ \sin(\theta_6)\cos(\theta_2 + \theta_3 + \theta_4) + \sin(\theta_2 + \theta_3 + \theta_4)\cos(\theta_5)\cos(\theta_6) & -\sin(\theta_6)\sin(\theta_2 + \theta_3 + \theta_4)\cos(\theta_5) + \cos(\theta_6)\cos(\theta_2 + \theta_3 + \theta_4) \\ 0 & 0 \end{bmatrix}$$

```
In [124]: def forward(theta):
    DH = [[0, 0, 0.128, theta[0]],
          [pi/2, 0, 0, theta[1]],
          [0, 0.6129, 0, theta[2]],
          [0, 0.5716, 0.1639, theta[3]],
          [-pi/2, 0, 0.1157, theta[4]],
          [pi/2, 0, 0.0922, theta[5]]]
    # find homogeneous transformations(T_i_i-1)
    T = []
    for i in range(6):
        T_temp = [[cos(DH_param[i][3]), -sin(DH_param[i][3]), 0, DH_param[i][1]],
                  [sin(DH_param[i][3])*cos(DH_param[i][0]), cos(DH_param[i][3])*cos(DH_param[i][0]), -sin(DH_param[i][0]), -sin(DH_param[i][0])*(DH_param[i][2])],
                  [sin(DH_param[i][3])*sin(DH_param[i][0]), cos(DH_param[i][3])*sin(DH_param[i][0]), cos(DH_param[i][0]), cos(DH_param[i][0])*DH_param[i][2]],
                  [0, 0, 0, 1]]
        T.append(T_temp)

    # find T_0_6 and print it
    result = T[0]
    for n in range(1,len(T)):
        # Multiply the matrices
        result = [[sum(result[i][k] * T[n][k][j] for k in range(4)) for j in range(4)] for i in range(4)]
    #T_res = np.array(T[0])@np.array(T[1])@np.array(T[2])@np.array(T[3])@np.array(T[4])@np.array(T[5])
    return result
```

```
In [112]: def forward_version2(theta):
    theta1, theta2, theta3, theta4, theta5, theta6 = theta
    FK = Matrix(np.array(simplified_result)).subs({'d1':0.128, 'd4': 0.1639, 'd5':0.1157, 'd6':0.0922, 'a2':0.6129, 'a3':0.5716,
                                                    'theta1':theta1, 'theta2':theta2, 'theta3':theta3, 'theta4':theta4,
                                                    'theta5':theta5, 'theta6':theta6})
    return np.array(FK).tolist()
```

Part2

Inverse Kinematics

```
In [125]: def inverse(T):
    # define DH table
    DH = [[0, 0, 0.128, theta[0]],
          [pi/2, 0, 0, theta[1]],
          [0, 0.6129, 0, theta[2]],
          [0, 0.5716, 0.1639, theta[3]],
          [-pi/2, 0, 0.1157, theta[4]],
          [pi/2, 0, 0.0922, theta[5]]]

    # define theta database
    theta_db = []

    # define variables for each element of T
    [[r11, r12, r13, px],
     [r21, r22, r23, py],
     [r31, r32, r33, pz]] = T[0:3]

    # step1) find theta1
    a = px-DH[5][2]*r23
    b = DH[5][2]*r13-px
    c = -DH[3][2]
    theta_db.append(2*math.atan((b+math.sqrt(b**2+a**2-c**2))/(a+c)))
    theta_db.append(2*math.atan((b-math.sqrt(b**2+a**2-c**2))/(a+c)))

    # step2) find theta2
    theta_temp = []
    for theta1 in theta_db:
        c5 = r13*math.sin(theta1)-r23*math.cos(theta1)
        s5 = [math.sqrt(1-(c5)**2), -math.sqrt(1-(c5)**2)]
        theta_temp.append([theta1, math.atan2(s5[0], c5)])
        theta_temp.append([theta1, math.atan2(s5[1], c5)])
    theta_db = theta_temp

    # step3) find theta2+theta3+theta4
    theta_temp = []
    flag = []
    for theta_list in theta_db:
        theta1 = theta_list[0]
        theta5 = theta_list[1]
        theta234 = []
        if round(theta5, 4) == 0:
            gamma = math.atan2(r31, r32)
            Q1 = r31
            Q2 = r11*math.cos(theta1) + r21*math.sin(theta1)
            b = Q1*math.sin(gamma) + Q2*math.cos(gamma) - 1
            a = Q2*math.sin(gamma) - Q1*math.cos(gamma)
            theta234 = [2*math.atan((b+math.sqrt(b**2+a**2))/a), 2*math.atan((b-math.sqrt(b**2+a**2))/a)]
            flag.append('0')
            flag.append('0')
        elif round(theta5, 4) == pi or round(theta5, 4) == -pi:
            gamma_prim = math.atan2(-r31, r32)
            Q1 = r31
            Q2 = r11*math.cos(theta1) + r21*math.sin(theta1)
            b_prime = Q1*math.sin(gamma_prim) - Q2*math.cos(gamma_prim) - 1
            a_prime = Q2*math.sin(gamma_prim) + Q1*math.cos(gamma_prim)
            theta234 = [2*math.atan((b_prime+math.sqrt(b_prime**2+a_prime**2))/a_prime), 2*math.atan((b_prime-math.sqrt(b_prime**2+a_prime**2))/a_prime)]
            flag.append('180')
            flag.append('180')
        else:
            c234 = (r13*math.sin(theta1)*math.cos(theta5))/(math.sin(theta5)*math.cos(theta1))
            s234 = r33/math.sin(theta5)
            c234 = [math.sqrt(1-s234**2), -math.sqrt(1-s234**2)]
            theta234 = [math.atan2(s234[0]), math.atan2(s234[1])]
            flag.append('ok')
            flag.append('ok')
        theta_temp.append([theta1, theta5, theta234[0]])
        theta_temp.append([theta1, theta5, theta234[1]])

    theta_db = theta_temp

    # step4) find theta6
    theta_temp = []
    i = 0
    for theta_list in theta_db:
        theta1, theta5, theta234 = theta_list
        theta6 = None
        if flag[i] == '0':
            gamma = math.atan2(r31, r32)
            theta6 = gamma - theta234
        elif flag[i] == '180':
            gamma_prim = math.atan2(-r31, r32)
            theta6 = theta234 - gamma_prim
        else:
            k1, k2 = math.cos(theta234), math.sin(theta234)*math.cos(theta5)
            theta6 = math.atan2(r31, r32)-math.atan2(k2, k1)
        theta_temp.append([theta1, theta5, theta234, theta6])
        i+=1
    theta_db = theta_temp

    # step5) find theta3, theta2, theta4
    theta_temp = []
    i=0
    for theta_list in theta_db:
        theta1, theta5, theta234, theta6 = theta_list

        d1, d4, d5, d6 = DH[0][2], DH[3][2], DH[4][2], DH[5][2]
        a2, a3 = DH[2][1], DH[3][1]
        if round(theta1, 4) == 0 or round(theta1, 4) == -pi:
            F1 = -d5*math.sin(theta234)*math.cos(theta1)+d6*math.sin(theta5)*math.cos(theta1)*math.cos(theta234)
            F2 = d1+d5*math.cos(theta234)+d6*math.sin(theta5)*math.sin(theta234)
            X = (px - F1)/math.cos(theta1)
            Z = pz - F2
            # 5.1) find theta3
            c3 = (X**2 + Z**2 - a2**2 - a3**2)/(2*a2*a3)
            s3 = [math.sqrt(1-c3**2), -math.sqrt(1-c3**2)]
            theta_3_list = [math.atan2(s3[0], c3), math.atan2(s3[1], c3)]
            # 5.2) find theta2 and theta4
            for theta3 in theta_3_list:
                Q1 = a2 + a3*math.cos(theta3)
                Q2 = a3*math.sin(theta3)
                theta2 = math.atan2(Z, X) - math.atan2(Q2, Q1)
                theta4 = theta234 - theta3 - theta2
                theta_temp.append([theta1, theta5, theta234, theta6, theta3, theta4])

        else:
            F2 = (py+d4*math.cos(theta1)+d5*math.sin(theta234)*math.sin(theta1)+d6*(math.cos(theta1)*math.cos(theta5)-math.sin(theta1)*math.sin(theta5)*math.cos(theta234)))/m
            F3 = pz-(d1+d5*math.cos(theta234)+d6*math.sin(theta5)*math.sin(theta234))

            # 5.1) find theta3
            c3 = ((F2)**2 + (F3)**2 - (a2)**2 - (a3)**2)/(2*(a2)*(a3))
            s3 = [math.sqrt(1-(c3)**2), -math.sqrt(1-(c3)**2)]
            theta_3_list = [math.atan2(s3[0], c3), math.atan2(s3[1], c3)]
            # 5.2) find theta2 and theta4
            for theta3 in theta_3_list:
                Q1 = a2 + a3*math.cos(theta3)
                Q2 = a3*math.sin(theta3)
                theta2 = math.atan2(F3, F2) - math.atan2(Q2, Q1)
                theta4 = theta234 - theta3 - theta2
                theta_temp.append([theta1, theta5, theta234, theta6, theta3, theta4])

    theta_db = theta_temp
    return theta_db
```

Inverse Kinematics (Geometric approach)

```
In [126]: def inverse2(T_inp):
    # define DH table
    DH = [[0, 0, 0.128, theta[0]],
          [pi/2, 0, 0, theta[1]],
          [0, 0.6129, 0, theta[2]],
          [0, 0.5716, 0.1639, theta[3]],
          [-pi/2, 0, 0.1157, theta[4]],
          [pi/2, 0, 0.0922, theta[5]]]

    # define theta database
    theta_db = []

    # define variables for each element of T
    [[nx, ox, ax, px],
     [ny, oy, ay, py],
     [nz, oz, az, pz]] = T_inp[0:3]

    # step1) find theta1
    d6 = DH[5][2]
    d4 = DH[3][2]
    theta1 = [math.atan2(py-d6*ay, px-d6*ax)+math.acos(d4/(math.sqrt((px-d6*ax)**2+(py-d6*ay)**2)))+(math.pi/2),
              math.atan2(py-d6*ay, px-d6*ax)-math.acos(d4/(math.sqrt((px-d6*ax)**2+(py-d6*ay)**2)))+(math.pi/2)]
    theta_db.append(theta1[0])
    theta_db.append(theta1[1])

    # step2) find theta5
    d4 = DH[3][2]
    p_0_6_x = px
    p_0_6_y = py
    theta_temp = []
    for th1 in theta_db:
        theta5 = [math.acos((p_0_6_x*math.sin(th1) - p_0_6_y*math.cos(th1))/d4),
                  -math.acos((p_0_6_x*math.sin(th1) - p_0_6_y*math.cos(th1))/d4)/d6]
        theta_temp.append([th1, theta5[0]])
        theta_temp.append([th1, theta5[1]])
    theta_db = theta_temp

    # step3) find theta6
    theta6 = []
    theta_temp = []
    for th in theta_db:
        theta1, theta5 = th
        if round(theta5,4)==0 or round(theta5,4)==math.pi or round(theta5,4)==-math.pi or round(theta5,4)==2*math.pi:
            continue
        th6=math.atan2((-ox*math.sin(theta1)+oy*math.cos(theta1))/math.sin(theta5),(nx*math.sin(theta1)-ny*math.cos(theta1))/math.sin(theta5))
        theta_temp.append([theta1, theta5, th6])
        theta6.append(th6)

    if len(theta6) == 0:
        print('Theta5 is singular')
        return 0
    theta_db = theta_temp

    # step4) find theta 2,3,4
    a2 = DH[2][1]
    a3 = DH[3][1]
    theta_temp = []
    for th in theta_db:
        theta1, theta5, theta6 = th
        T_0_1 = T[0]
        T_4_5 = T[4]
        T_5_6 = T[5]
        T_0_1_inv = Matrix(T_0_1).inv().subs({'d1':DH[0][2], 'theta1': theta1}).evalf()
        T_4_5_inv = Matrix(T_4_5).inv().subs({'d5':DH[4][2], 'theta5': theta5}).evalf()
        T_5_6_inv = Matrix(T_5_6).inv().subs({'d6':DH[5][2], 'theta6': theta6}).evalf()
        T_1_4 = T_0_1_inv*Matrix(T_inp)*T_5_6_inv*T_4_5_inv
        [[n4x, ox, ax, p4x],
         [ny, oy, ay, py],
         [nz, oz, az, p4z]] = np.array(T_0_1_inv).tolist()[0:3]
        theta2 = [math.acos((p4x**2+p4z**2-a2**2-a3**2)/(2*a2*a3)), -math.acos((p4x**2+p4z**2-a2**2-a3**2)/(2*a2*a3))]
        theta2 = [math.atan2((a3*math.cos(theta3[0])+a2)*p4z-a3*math.sin(theta3[0])*p4x, (a3*math.cos(theta3[0])+a2)*p4z+a3*math.sin(theta3[0])*p4x),
                  math.atan2((a3*math.cos(theta3[1])+a2)*p4z-a3*math.sin(theta3[1])*p4x, (a3*math.cos(theta3[1])+a2)*p4z+a3*math.sin(theta3[1])*p4x)]
        theta3 = [math.atan2(n4z,n4x)-theta2[0]-theta3[0], math.atan2(n4z,n4x)-theta2[1]-theta3[1]]
        theta_temp.append([theta1, theta2[0], theta3[0], theta4[0], theta5, theta6])
        theta_temp.append([theta1, theta2[1], theta3[1], theta4[1], theta5, theta6])

    theta_db = theta_temp
    return theta_db
```

Some examples of forward kinematics

A) Zero state

- in this state all joint variables have zero value

```
In [115]: theta1 = math.radians(0)
theta2 = math.radians(0)
theta3 = math.radians(0)
theta4 = math.radians(0)
theta5 = math.radians(0)
theta6 = math.radians(0)

desired_theta = [theta1, theta2, theta3, theta4, theta5, theta6]
T_0_6_A = forward_version2(desired_theta)

print('T_0_6 : ')
for row in T_0_6_A:
    print(row)

T_0_6 :
[1, 0, 0, 1.184500000000000]
[0, 0, -1, -0.256100000000000]
[0, 1, 0, 0.012300000000000]
[0, 0, 0, 1]
```

B) fully-stretched state

- We have below values for joint variables in fully-stretched state:

theta1 = arbitrary | theta2 = 0 | theta3 = 90 | theta4 = 0 | theta5 = -90 | theta6 = arbitrary

We consider theta1 = -20 and theta6 = 60 degrees.

```
In [116]: theta1 = math.radians(45)
theta2 = math.radians(90)
theta3 = math.radians(0)
theta4 = math.radians(-90)
theta5 = math.radians(0)
theta6 = math.radians(60)

desired_theta = [theta1, theta2, theta3, theta4, theta5, theta6]
T_0_6_B = forward_version2(desired_theta)

print('T_0_6 : ')
for row in T_0_6_B:
    print(row)

T_0_6 :
[0.353553390593274, -0.612372435695795, 0.707106781186547, 0.181090046661875]
[0.353553390593274, -0.612372435695794, -0.707106781186548, -0.181090046661875]
[0.866025403784439, 0.500000000000000, 0, 1.196800000000000]
[0, 0, 1]
```

C) Arbitrary state

- We have below values for joint variables in arbitrary state:

```
theta1 = 30 | theta2 = 45 | theta3 = 26 | theta4 = 50 | theta5 = 60 | theta6 = 80
```

```
In [117]: theta1 = math.radians(30)
theta2 = math.radians(45)
theta3 = math.radians(26)
theta4 = math.radians(50)
theta5 = math.radians(60)
theta6 = math.radians(80)

desired_theta = [theta1, theta2, theta3, theta4, theta5, theta6]
T_0_6_C = forward_version2(desired_theta)

print('T_0_6 : ')
for row in T_0_6_C:
    print(row)

T_0_6 :
[-0.69458525014751, -0.335708401191927, 0.636278556182541, 0.762988161563082]
[-0.574667490714495, 0.790986417214830, -0.209994673498050, 0.198024307407300]
[-0.432790719406602, -0.511507924817209, -0.742328657701364, 1.09299136702883]
[0, 0, 0, 1]
```

Some examples of inverse kinematics

A) Zero state

- We gave zero values to joint variables in forward kinematics and it took us Transformation matrix (T_0_6_first) as a result. now we give this matrix as an input to inverse function.

```
In [118]: T_0_6_A
```

```
Out[118]: [[1, 0, 0, 1.184500000000000],  
[0, 0, -1, -0.256100000000000],  
[0, 1, 0, 0.0123000000000000],  
[0, 0, 0, 1]]
```

```
In [119]: theta_A = inverse2(T_0_6_A)
i = 0
for th in theta_A:
    i += 1
    print("ans", i, " theta1: ", round(degrees(th[0]), 2), " theta2: ", round(degrees(th[1]), 2), " theta3: ", round(degrees(th[2]), 2),
          " theta4: ", round(degrees(th[3]), 2), " theta5: ", round(degrees(th[4]), 2), " theta6: ", round(degrees(th[5]), 2))

ans 1 theta1: 164.24 theta2: -155.4 theta3: 168.25 theta4: 167.15 theta5: 164.24 theta6: 0.0
ans 2 theta1: 164.24 theta2: -24.6 theta3: -168.25 theta4: 372.85 theta5: 164.24 theta6: 0.0
ans 3 theta1: 164.24 theta2: -155.4 theta3: 168.25 theta4: 167.15 theta5: -164.24 theta6: 180.0
ans 4 theta1: 164.24 theta2: -24.6 theta3: -168.25 theta4: 372.85 theta5: -164.24 theta6: 180.0
```

B) fully-stretched state

- We should use T_0_6_B as input and get the below angles in results:

```
theta1 = 45 | theta2 = 90 | theta3 = 0 | theta4 = -90 | theta5 = 0 | theta6 = 60
```

```
In [120]: T_0_6_B
```

```
Out[120]: [[0.353553390593274, -0.612372435695795, 0.707106781186547, 0.181090046661875],  
[0.353553390593274,  
-0.612372435695794,  
-0.707106781186548,  
-0.181090046661875],  
[0.866025403784439, 0.500000000000000, 0, 1.19680000000000],  
[0, 0, 0, 1]]
```

```
In [121]: theta_B = inverse2(T_0_6_B)
i = 0
for th in theta_B:
    i += 1
    print("ans", i, " theta1: ", round(degrees(th[0]), 2), " theta2: ", round(degrees(th[1]), 2), " theta3: ", round(degrees(th[2]), 2),
          " theta4: ", round(degrees(th[3]), 2), " theta5: ", round(degrees(th[4]), 2), " theta6: ", round(degrees(th[5]), 2))
```

Theta5 is singular

```
-----
TypeError                                 Traceback (most recent call last)
Input In [121], in <cell line: 3>()
      1 theta_B = inverse2(T_0_6_B)
      2 i = 0
----> 3 for th in theta_B:
      4     i += 1
      5     print("ans", i, " theta1: ", round(degrees(th[0]), 2), " theta2: ", round(degrees(th[1]), 2), " theta3: ", round(degrees(th[2]), 2),
      6           " theta4: ", round(degrees(th[3]), 2), " theta5: ", round(degrees(th[4]), 2), " theta6: ", round(degrees(th[5]), 2))

TypeError: 'int' object is not iterable
```

C) Arbitrary state

- We should use T_0_6_B as input and get the below angles in results:

```
theta1 = 30 | theta2 = 45 | theta3 = 26 | theta4 = 50 | theta5 = 60 | theta6 = 80
```

```
In [88]: T_0_6_C
```

```
Out[88]: [[-0.844969558195186,  
0.517160130760516,  
-0.136278556182541,  
0.51997854410926],  
[-0.314195224214099,  
-0.686225212303482,  
-0.656030730286389,  
0.0577341865342189],  
[-0.432790719406602, -0.511507924817209, 0.742328657701364, 1.11069696097478],  
[0, 0, 0, 1]]
```

```
In [94]: theta_C = inverse(T_0_6_C)
i = 0
for th in theta_C:
    i += 1
    print("ans", i, " theta1: ", round(degrees(th[0]), 2), " theta2: ", round(degrees(th[1]), 2), " theta3: ", round(degrees(th[2]), 2),
          " theta4: ", round(degrees(th[3]), 2), " theta5: ", round(degrees(th[4]), 2), " theta6: ", round(degrees(th[5]), 2))

ans 1 theta1: 30.0 theta2: 27.51 theta3: 63.13 theta4: -31.64 theta5: 60.0 theta6: -179.53
ans 2 theta1: 30.0 theta2: 88.18 theta3: -63.13 theta4: 33.94 theta5: 60.0 theta6: -179.53
ans 3 theta1: 30.0 theta2: 45.0 theta3: 26.0 theta4: 50.0 theta5: 60.0 theta6: -280.0
ans 4 theta1: 30.0 theta2: 70.08 theta3: -26.0 theta4: 76.92 theta5: 60.0 theta6: -280.0
ans 5 theta1: 30.0 theta2: 28.77 theta3: 72.95 theta4: -160.72 theta5: -60.0 theta6: -100.0
ans 6 theta1: 30.0 theta2: 98.77 theta3: -72.95 theta4: -84.82 theta5: -60.0 theta6: -100.0
ans 7 theta1: 30.0 theta2: 42.44 theta3: 59.0 theta4: -222.45 theta5: -60.0 theta6: 0.47
ans 8 theta1: 30.0 theta2: 99.18 theta3: -59.0 theta4: -161.18 theta5: -60.0 theta6: 0.47
ans 9 theta1: 175.03 theta2: 82.09 theta3: 66.85 theta4: -65.01 theta5: 131.71 theta6: -58.85
ans 10 theta1: 175.03 theta2: 146.3 theta3: -66.85 theta4: 4.48 theta5: 131.71 theta6: -58.85
ans 11 theta1: 175.03 theta2: 82.18 theta3: 63.86 theta4: -49.97 theta5: 131.71 theta6: -40.68
ans 12 theta1: 175.03 theta2: 143.55 theta3: -63.86 theta4: 16.38 theta5: 131.71 theta6: -40.68
ans 13 theta1: 175.03 theta2: 102.93 theta3: 44.86 theta4: -231.72 theta5: -131.71 theta6: -220.68
ans 14 theta1: 175.03 theta2: 146.14 theta3: -44.86 theta4: -185.21 theta5: -131.71 theta6: -220.68
ans 15 theta1: 175.03 theta2: 106.62 theta3: 36.99 theta4: -239.67 theta5: -131.71 theta6: -238.85
ans 16 theta1: 175.03 theta2: 142.27 theta3: -36.99 theta4: -201.35 theta5: -131.71 theta6: -238.85
```

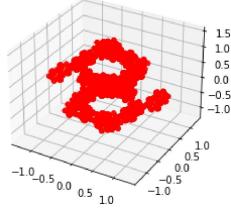
As we can see, the answer that we give in forward kinematics, is appear in ans1. So we evaluate the forward and inverse kinematics code.

Workspace

```
In [132]: ee_point = []
for t1 in range(-180,180,90):
    for t2 in range(-180,180,90):
        for t3 in range(-180,180,90):
            for t4 in range(-180,180,90):
                for t5 in range(-180,180,90):
                    for t6 in range(-180,180,90):
                        theta = [math.radians(t1), math.radians(t2), math.radians(t3), math.radians(t4),
                                math.radians(t5), math.radians(t6)]
                        EE_T = forward_version2(theta)
                        ee_point.append(np.array(EE_T)[:3,3])
```

```
In [137]: import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(np.array(ee_point)[:,0], np.array(ee_point)[:,1], np.array(ee_point)[:,2], zdir='z', c= 'red')
```

```
Out[137]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1e9164386a0>
```



```
In [ ]:
```

Robotics project

Analyzing 6-DOF UR10 robot arm

Phase2 (Inverse velocity and Dynamics)

Import libraries

```
In [1]: from sympy import *
import numpy as np
import math
from math import degrees
```

Forward Kinematics

```
In [2]: # Define the symbolic variables
num_symbols = 6 # Number of symbols to generate
alpha_names = [f"alpha{i}" for i in range(num_symbols+1)]
a_names = [f"a{i}" for i in range(num_symbols+1)]
d_names = [f"d{i}" for i in range(num_symbols+1)]
theta_names = [f"theta{i}" for i in range(num_symbols+1)]

alpha = symbols(alpha_names)
a = symbols(a_names)
d = symbols(d_names)

# our thetas are function of time
theta = []
t = Symbol('t')
for i in theta_names:
    theta.append(function(i)(t))

# Define DH table
DH_param = [[0, 0, d[1], theta[1]],
            [pi/2, 0, 0, theta[2]],
            [0, a[2], 0, theta[3]],
            [0, a[3], d[4], theta[4]],
            [-pi/2, 0, d[5], theta[5]],
            [pi/2, 0, d[6], theta[6]]]

# find homogeneous transformations(T_i_i-1)
T = []
for i in range(6):
    T_temp = [[cos(DH_param[i][3]), -sin(DH_param[i][3]), 0, DH_param[i][1]],
               [sin(DH_param[i][3])*cos(DH_param[i][0]), cos(DH_param[i][3])*cos(DH_param[i][0]), -sin(DH_param[i][0]), -sin(DH_param[i][0])*cos(DH_param[i][2])],
               [sin(DH_param[i][3])*sin(DH_param[i][0]), cos(DH_param[i][3])*sin(DH_param[i][0]), cos(DH_param[i][0]), cos(DH_param[i][0])*cos(DH_param[i][2])],
               [0, 0, 0, 1]]
    T.append(T_temp)
```

Part1

Finding Jacobian Matrix

1-1) Find T_0_i (like T_0_1, T_0_2, ...)

```
In [3]: T_0_i = [T[0]]
result = T[0]
for n in range(1,len(T)):
    # Multiply the matrices
    result = [[sum(result[i][k] * T[n][k][j] for k in range(4)) for j in range(4)] for i in range(4)]
    # Simplify the result according to cosine multiplication rules
    simplified_result = [[trigsimp(expr) for expr in row] for row in result]
    T_0_i.append(simplified_result)
```

1-2) Find zi and oi

```
In [4]: z = []
o = []
for T0i in T_0_i:
    z.append(np.array(T0i)[:3,2][np.newaxis].T)
    o.append(np.array(T0i)[:3,3][np.newaxis].T)
```

1-3) Create J(theta)

Here all joints are Revolute. So we use below formula for creating J(theta):

```
Ji = [[zi*(oe-oi)], [zi]]
```

```
In [5]: J = []
for i in range(len(z)):
    Jvi = np.cross(z[i].T, (o[i].T - o[i]).T)
    Jwi = z[i]
    Ji = np.append(Jvi, Jwi)
    J.append(Ji)
J = np.array([[trigsimp(expr) for expr in row] for row in J]).T
```

```
In [6]: Matrix(J)
```

```
Out[6]: 
-a2 sin(theta_1(t)) cos(theta_2(t)) - a3 sin(theta_1(t)) cos(theta_2(t) + theta_3(t)) + d4 cos(theta_1(t)) + d5 sin(theta_2(t) + theta_3(t) + theta_4(t)) sin(theta_1(t)) - (a2 sin(theta_2(t)) + a3 sin(theta_2(t) + theta_3(t)) + d5 cos(theta_2(t) + theta_3(t) + theta_4(t))) sin(theta_1(t))
+ d6 (-sin(theta_1(t)) sin(theta_5(t)) cos(theta_2(t) + theta_3(t) + theta_4(t)) + cos(theta_1(t)) cos(theta_5(t)))
a2 cos(theta_1(t)) cos(theta_2(t)) + a3 cos(theta_2(t) + theta_3(t)) cos(theta_1(t)) + d4 sin(theta_1(t)) - d5 sin(theta_2(t) + theta_3(t) + theta_4(t)) cos(theta_1(t)) - (a2 sin(theta_2(t)) + a3 sin(theta_2(t) + theta_3(t)) + d5 cos(theta_2(t) + theta_3(t) + theta_4(t))) cos(theta_1(t))
+ d6 (sin(theta_1(t)) cos(theta_5(t)) + sin(theta_5(t)) cos(theta_2(t) + theta_3(t) + theta_4(t)) cos(theta_1(t)))
0
0
0
1
a2 cos(theta_2(t)) + a3 cos(theta_2(t) + theta_3(t)) - d5 sin(theta_2(t) + theta_3(t) + theta_4(t)) sin(theta_1(t))
sin(theta_1(t))
-cos(theta_1(t))
0
```

Part2

Find Singularities

2-1) J singularities

```
In [7]: J_det = Matrix(J).det()
```

```
In [8]: J_det_simp = trigsimp(J_det)
```

In [9]: J_det_simp

Out[9]: $a_2 a_3 (a_2 \cos(\theta_2(t)) + a_3 \cos(\theta_2(t) + \theta_3(t)) - d_5 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t))) \sin(\theta_3(t)) \sin(\theta_5(t))$

2-2) Jv singularities

```
In [10]: Jv = J[:3,:]
simple_JvJvT = simplify(Jv@(Jv.T))
```

In [15]: simple_JvJvT.shape

Out[15]: (3, 3)

```
In [17]: Jv_det = Matrix(simple_JvJvT).det()
```

```
In [21]: np.array(Jv_det)
```

```
In [29]: Jv_det_simp = simplify(Jv_det)
Jv_det_simp
```

Out[29]: [Math Processing Error]

2-3) Jw singularities

```
In [22]: Jw = J[3:,:]
        simple_JwJwT = simplify(Jw@(Jw.T))
```

In [23]: simple_JwJwT.shape

Out[23]: (3, 3)

```
In [24]: Jw_det = Matrix(simple_JwJwT).det()
```

```
In [26]: Jw_det_simp = simplify(Jw_det)
Jw_det_simp
```

Out[26]:

Part3

Dynamics Equations (Newton-Euler method)

3-1) Define suitable variables

- We have Rotation matrixes from forward kinematics. Here we need both R and R transpose.

```
In [7]: DOF = 6
# Some variables which we had them from Phase1 and robot structure
T_num = T
T_num = [Matrix(T_num[i]).subs({'d1':0.128, 'd4': 0.16389, 'd5':0.1157, 'd6':0.0922, 'a2':0.6129, 'a3':0.5716}) for i in range(len(T_num))]
R = [np.array(T_num[i])[:,3:3] for i in range(DOF)]
R_trans = [np.array(T_num[i])[:,3:,3].T for i in range(DOF)]
# Also we add R_n_e because we need it in inward iterations (Note: here We assume we dont have end effector
# and here we just set R_6_7 for preventing from error in code)
R_6_7 = np.array([[1, 0, 0],[0, 1, 0],[0, 0, 1]])
R.append(R_6_7)
R_trans.append(R_6_7)
```

- We need Inertia tensors of the links. We find them from Solidworks file

```
In [8]: # Units : kg.m^2
I_c0_0 = np.array([[0, 0, 0],[0, 0, 0],[0, 0, 0]])
I_c1_1 = np.array([[0.03, 0, 0],[0, 0.03, 0],[0, 0, 0.03]])
I_c2_2 = np.array([[0.05, 0, 0.01],[0, 1.23, 0],[0.01, 0, 1.23]])
I_c3_3 = np.array([[0.02, 0, 0],[0, 0.54, 0],[0, 0, 0.54]])
I_c4_4 = np.array([[0.003, 0, 0],[0, 0.0024, 0.00025],[0, 0.00025, 0.0028]])
I_c5_5 = np.array([[0.003, 0, 0],[0, 0.0024, -0.00025],[0, -0.00025, 0.0028]])
I_c6_6 = np.array([[0.00022, 0, 0],[0, 0.00024, 0],[0, 0, 0.0004]])
I = [I_c0_0, I_c1_1, I_c2_2, I_c3_3, I_c4_4, I_c5_5, I_c6_6]
```

- We need mass of each link. We find them from Solidworks file.

```
In [9]: # Units : kg
m0, m1, m2, m3, m4, m5, m6 = 0, 8.3, 23.52, 12.56, 1.967, 1.967, 0.43
m = [m0, m1, m2, m3, m4, m5, m6]
```

- We need the distance of center of mass of link i from frame i. (P_{i_ci})

```
In [10]: P_0_c0 = np.array([[0],[0],[0]])
P_1_c1 = np.array([[0],[-0.01],[-0.01]])*10**(-3)
P_2_c2 = np.array([[0.25],[0],[0.17]])*10**(-3)
P_3_c3 = np.array([[0.26],[0],[0.05]])*10**(-3)
P_4_c4 = np.array([[0],[9.74],[-7.6]])*10**(-3)
P_5_c5 = np.array([[0],[-9.74],[-7.6]])*10**(-3)
P_6_c6 = np.array([[0],[0.95],[-17.46]])*10**(-3)
Pc = [P_0_c0, P_1_c1, P_2_c2, P_3_c3, P_4_c4, P_5_c5, P_6_c6]
```

- We need the distance between frame i+1 and i. We can find them from fourth column of forward kinematics transformation matrix

```
In [11]: p_0_1 = np.array(T_num[0][:3,3])
p_1_2 = np.array(T_num[1][:3,3])
p_2_3 = np.array(T_num[2][:3,3])
p_3_4 = np.array(T_num[3][:3,3])
p_4_5 = np.array(T_num[4][:3,3])
p_5_6 = np.array(T_num[5][:3,3])
# we define p_6_7 only for preventing from Error in inward iteration step
p_6_7 = np.array([[0],[0],[0]])
P = [p_0_1, p_1_2, p_2_3, p_3_4, p_4_5, p_5_6, p_6_7]
```

- We need some parametric variables like theta_dot and theta_dotdot

```
In [12]: theta_d = [i.diff(t) for i in theta]
theta_dd = [i.diff(t) for i in theta_d]
```

```
Out[12]: '\nnum_symbols = 6 # Number of symbols to generate\ntheta_d_names = [f"theta_d{i}" for i in range(num_symbols+1)]\ntheta_d = symbols(theta_d_names)\ntheta_dd_names = [f"theta_dd{i}" for i in range(num_symbols+1)]\ntheta_dd = symbols(theta_dd_names)'
```

- Some lists are needed and we should use them in Outward iterations.

```
In [13]: w = [np.array([[0],[0],[0]])]
w_dot = [np.array([[0],[0],[0]])]
g = symbols('g')
v_dot = [np.array([[0],[0],[g]])]
vc_dot = []
F = [np.array([[0],[0],[0]])]
N = [np.array([[0],[0],[0]])]
```

- Some lists are needed and we should use them in Inward iterations

```
In [14]: f = [np.array([[0],[0],[0]])]
n = [np.array([[0],[0],[0]])]
tauw = []
```

3-2) Outward iterations

```
In [15]: for i in range(DOF):
    #print(i)
    w_i1 = R_trans[i]@w[i] + np.array([[0],[0],[theta_d[i+1]]])
    #w_i1 = simplify(w_i1)
    w_dot_i1 = R_trans[i]@w_dot[i] + np.cross(R_trans[i]@w[i], np.array([[0],[0],[theta_d[i+1]]]),axis=0) + np.array([[0],[0],[theta_dd[i+1]]])
    #w_dot_i1 = simplify(w_dot_i1)
    v_dot_i1 = R_trans[i]@np.cross(w_dot[i], P[i],axis=0)+np.cross(w[i],np.cross(w[i],P[i], axis=0), axis=0)+v_dot[i]
    #v_dot_i1 = simplify(v_dot_i1)
    vc_dot_i1 = np.cross(w_dot_i1, P[i+1], axis=0)+np.cross(w_i1,np.cross(w_i1,P[i+1], axis=0), axis=0)+v_dot_i1
    F_i1 = m[i+1]*vc_dot_i1
    F_i1 = m[i+1]*vc_dot_i1
    N_i1 = I[i+1]@w_dot_i1 + np.cross(w_i1, I[i+1]@w_i1, axis=0)

    w.append(w_i1)
    w_dot.append(w_dot_i1)
    v_dot.append(v_dot_i1)
    vc_dot.append(vc_dot_i1)
    F.append(F_i1)
    N.append(N_i1)
```

3-3) Inward iterations

```
In [16]: for i in range(DOF,0,-1):
    fi = R[i]@f[i-1] + F[i]
    ni = N[i] + R[i]@n[len(f)-1]+np.cross(Pc[i],F[i], axis=0)+np.cross(P[i], R[i]@f[len(f)-1], axis=0)
    tawi = ni[2]
    f.append(fi)
    n.append(ni)
    tawi.append(tawi)
f.reverse()
n.reverse()
tawi.reverse()
```

```
In [17]: tauw_copy = taw
replacements = []
for i in range(1,len(theta)):
    replacements.append((theta[i].diff(t).diff(t), Symbol(f'ddt{theta[i]}')))
    replacements.append((theta[i].diff(t), Symbol(f'dt{theta[i]}')))
    replacements.append((theta[i], Symbol(f'th{theta[i]}')))
tauw_replace = []
for eq in tauw_copy:
    tauw_replace.append(eq[0].subs(replacements))
```

```
In [18]: num_symbols = 6 # Number of symbols to generate
th_names = [f"th{i}" for i in range(num_symbols+1)]
th = symbols(th_names)
th_d_names = [f"thd{i}" for i in range(num_symbols+1)]
th_d = symbols(th_d_names)
th_dd_names = [f"thdd{i}" for i in range(num_symbols+1)]
th_dd = symbols(th_dd_names)
```

```
In [25]: M = []
for tau_i in tau_replace:
    m = []
    for i in range(1, len(th_dd)):
        m.append(tau_i.diff(th_dd[i]))
    M.append(m)
```

```
In [26]: G = []
for tau_i in tau_replace:
    G.append(tau_i.diff(g))
```

```
In [27]: tau_replace_cop3 = tau_replace
V = Matrix(tau_replace_cop3).subs({'ddth1':0, 'ddth2':0, 'ddth3':0, 'ddth4':0,
                                    'ddth5':0, 'ddth6':0, 'g':0})
```

check matrices with some values

```
In [32]: init_th = [0, 0, 0, 0, 0, 0]
init_dth = [1, 0, 0, 0, 0, 0]
init_ddth = [0, 0, 0, 0, 0, 0]
M_copy = M
M_num = Matrix(M_copy).subs({'th1':init_th[0], 'th2':init_th[1], 'th3':init_th[2], 'th4':init_th[3], 'th5':init_th[4],
                             'th6':init_th[5], 'dth1':init_dth[0], 'dth2':init_dth[1], 'dth3':init_dth[2], 'dth4':init_dth[3],
                             'dth5':init_dth[4], 'dth6':init_dth[5]})

V_copy = V
V_num = V_copy.subs({'th1':init_th[0], 'th2':init_th[1], 'th3':init_th[2], 'th4':init_th[3], 'th5':init_th[4],
                     'th6':init_th[5], 'dth1':init_dth[0], 'dth2':init_dth[1], 'dth3':init_dth[2], 'dth4':init_dth[3],
                     'dth5':init_dth[4], 'dth6':init_dth[5]})

G_copy = G
G_num = 9.81*Matrix(G_copy).subs({'th1':init_th[0], 'th2':init_th[1], 'th3':init_th[2], 'th4':init_th[3], 'th5':init_th[4],
                                   'th6':init_th[5], 'dth1':init_dth[0], 'dth2':init_dth[1], 'dth3':init_dth[2], 'dth4':init_dth[3],
                                   'dth5':init_dth[4], 'dth6':init_dth[5], 'g':9.81})
```

```
In [33]: M_num
```

```
Out[33]: 
[ 12.782265545108 -0.0511883109442 -0.0511883109442 -0.0511883109442 0.0140356429114 9.7480355 · 10⁻⁵
 -0.0511883109442 12.6494261034502 3.5311272994102 0.0344342513142 -0.005508900948 0.000353124625
 -0.0511883109442 3.5311272994102 2.0002698082102 0.0344342513142 -0.005508900948 0.000353124625
 -0.0511883109442 0.0344342513142 0.0344342513142 0.0344342513142 -0.005508900948 0.000353124625
  0.0140356429114 -0.005508900948 -0.005508900948 -0.005508900948 0.0056286136372 3.053129 · 10⁻⁵
  9.7480355 · 10⁻⁵ 0.000353124625 0.000353124625 0.000353124625 3.053129 · 10⁻⁵ 0.000400388075 ]
```

```
In [34]: V_num
```

```
Out[34]: 
[ 1.94289029309402 · 10⁻¹⁶
  0.33300296241
  0.33300296241
  0.33300296241
 -0.06076103591
 -0.00048386825 ]
```

```
In [35]: G_num
```

```
Out[35]: 
[ 0
  126.316773756
  24.50271168
  0
  0
  0 ]
```

Solving dynamics equations

```
In [100]: from scipy.integrate import odeint
```

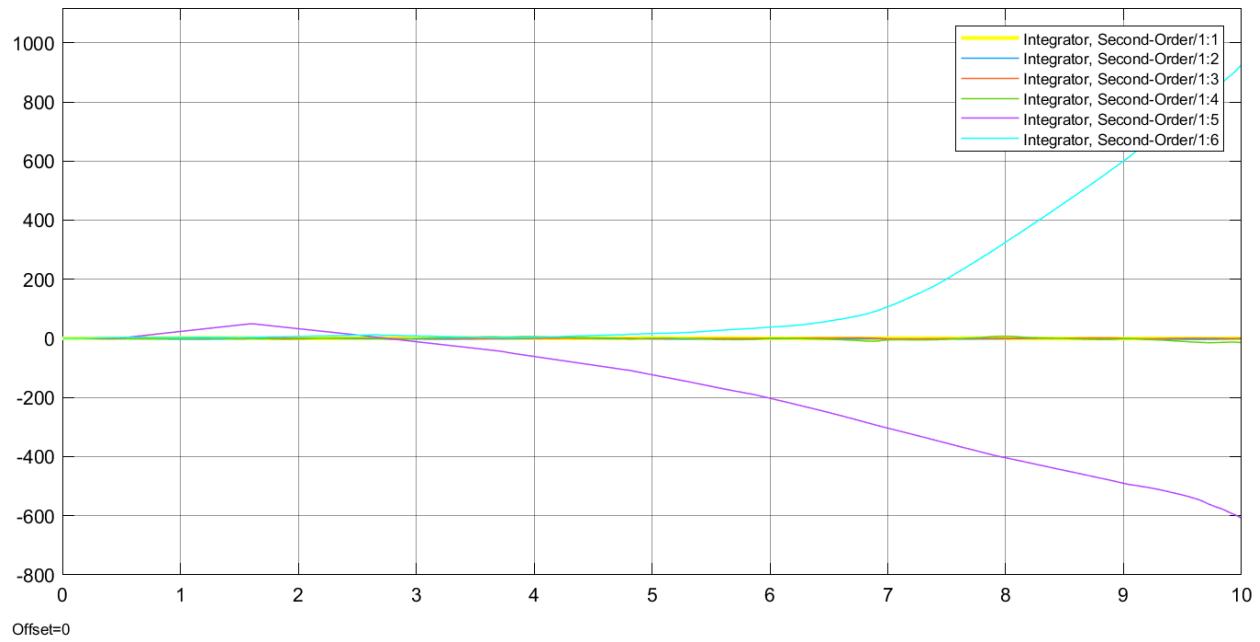
```
In [110]: def model(z,t):
    M_copy = M
    M_num = Matrix(M_copy).subs({'th1':z[0], 'th2':z[2], 'th3':z[4], 'th4':z[6], 'th5':z[8], 'th6':z[10],
                                 'dth1':z[1], 'dth2':z[3], 'dth3':z[5], 'dth4':z[7], 'dth5':z[9], 'dth6':z[11]})

    V_copy = V
    V_num = Matrix(V_copy).subs({'th1':z[0], 'th2':z[2], 'th3':z[4], 'th4':z[6], 'th5':z[8], 'th6':z[10],
                                 'dth1':z[1], 'dth2':z[3], 'dth3':z[5], 'dth4':z[7], 'dth5':z[9], 'dth6':z[11]})

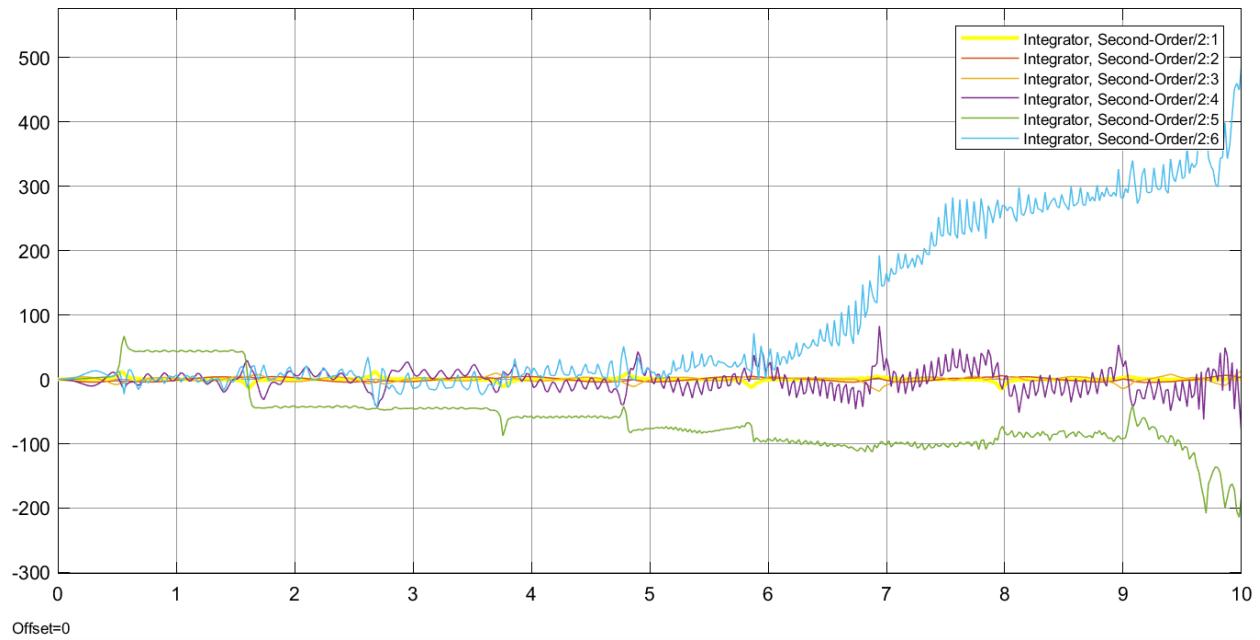
    G_copy = G
    G_num = Matrix(G_copy).subs({'th1':z[0], 'th2':z[2], 'th3':z[4], 'th4':z[6], 'th5':z[8], 'th6':z[10],
                                 'dth1':z[1], 'dth2':z[3], 'dth3':z[5], 'dth4':z[7], 'dth5':z[9], 'dth6':z[11],
                                 'g':9.81})

    M_num_inv = M_num.inv()
    theta_dd = np.array(M_num_inv)@(np.array(tau_num_input) - np.array(V_num) - np.array(G_num))
    return [z[1], theta_dd[0][0], z[3], theta_dd[1][0], z[5], theta_dd[2][0], z[7], theta_dd[3][0], z[9],
            theta_dd[4][0], z[11], theta_dd[5][0]]
```

```
In [ ]: tau_num_input = Matrix([0, 1, 1, 1, 1, 1])
theta_init = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
t = np.linspace(0, 1, 5)
theta_solved = odeint(model, theta_init, t)
```



جابجایی دورانی به ازای گشناور و شرایط اولیه صفر



سرعت دورانی به ازای گشناور و شرایط اولیه صفر

