ساخت یک بستر کامل حرکتی و ناوبری
برای ربات همه جهته‌ی دانشگاه شهید بهشتی
گزارش کارآموزی

ارایه شده به:

آقای دکتر آرمین سلیمی بدر

توسط:

آریا پرویزی

گروه هوش مصنوعی و رباتیکز

دانشکده مهندسی و علوم کامپیوتر

تابستان ۱۴۰۰

پیش‌گفتار و سپاسگزاری

در این قسمت به دلیل علاقه‌ی وافری که به موضوعات رباتیک، هوش مصنوعی، ریاضیات، آمار، فیزیک و ... داشتم، برای من بهترین و جذاب‌ترین شاخه از رشته‌های مهندسی، رباتیک است. هنگامی که از همکاری اتاق رباتیک دانشکده‌ی مهندسی کامپیوتر در پروژه‌ی ربات بهشت یا ربات دانشگاه شهید بهشتی آگاه شدم، بیدرنگ و مشتاقانه داوطلب شرکت در توسعه‌ی این پروژه شدم. بی‌صبرانه منتظر روزی هستم که این ربات توسعه یافته و در دانشکده با نسل‌های آینده‌ی دانشجویان ارتباط برقرار می‌کند و انگیزه‌ای شود برای جذب نسل‌های آینده به این دانش و تکنولوژی و در نهایت رشد علم رباتیک در ایران.

در این راستا از زحمات دلسوزانه و مشوقانه‌ی جناب آقای دکتر آرمین سلیمی بدر قدردانی و سپاس‌گزاری می‌نمایم. بدون راهنمایی‌های ایشان، این کاراموزی به موفقیت نمی‌رسید.

همچنین از دوستان و اعضای اتاق رباتیک، آقایان نوید مهدیان، کوروش خاوری مقدم و نوید جعفروف برای همفکری و همکاری در این پروژه کمال تشکر را دارم.

آریا پرویزی

پاییز ۱۴۰۰

فهرست

چکیده

ربات‌های چرخ دار امروزی به سمت هوشمند شدن و اتوماسیون در حال توسعه هستند، در این کاراموزی ما در تلاش بودیم تا به ربات ساخته شده در دانشگاه شهید بهشتی، قابلیت‌هایی را بیفزاییم تا این ربات به صورت خودکار و بدون فرمان بتواند کارهای حرکتی و مسیریابی خود را انجام دهد. در مسیرهای ربات می‌تواند موانع ثابت یا متحرکی وجود داشته باشد. در این کاراموزی سعی شد که این مسائل و چالش‌ها را حل کنیم. همچنین در راستای بهبود عملکرد ربات نسبت به کارهای گذشته تلاش‌هایی انجام شد.

۱- مقدمه

امروزه رباتیک به عنوان یک فناوری پیشرو در جهان شناخته شده است. ربات‌ها دارای سه نوع متحرک، صنعتی (بازو مکانیکی) و ترکیبی هستند. از ربات‌های صنعتی که دارای بازوهای متحرک هستند می‌توان از ربات‌های صنعت خودروسازی یاد کرد. از ربات‌های ترکیبی در دانشگاه شهید بهشتی می‌توان به ربات دو پای NAO و ربات هشت پا نیز اشاره کرد. ربات‌های متحرک نیز به چند زیردسته‌ی هوایی، دریایی و زمینی تقسیم می‌شوند که ربات‌های زمینی را می‌توان به دو گروه چرخدار و پادار دسته بندی کرد. ربات ساخت دانشگاه شهید بهشتی نیز از نوع ربات‌های متحرک چرخدار است. این ربات جز دسته‌ی ربات‌های هولونومیک (holonomic) به حساب می‌آید. ربات‌های هولونومیک، ربات‌هایی هستند که تمامی مسیرها را بدون نیاز به انجام مانور مضاعف می‌توانند طی کنند. از ربات‌های غیر هولونومیک می‌توان به خودروی سواری که به دسته‌ی ربات‌های آکرمن متعلق است، اشاره کرد. از انواع ربات‌ها چرخدار، ربات دانشگاه شهید بهشتی در دسته‌ی ربات‌های همه جهته (omni-directional robots) قرار دارد. از ویژگی ربات‌های همه جهته این است که بدون چرخاندن سر، می‌توانند در هر جهتی و هر شکل مسیری حرکت کنند.
معمولا برای ربات‌های متحرک، چالش‌های اصلی موقعیت‌یابی، مسیریابی، عبور و پرهیز از موانع و طرح‌ریزی و ... به صورت خودکار است. در قسمت‌های آینده قصد داریم به بررسی این چالش‌ها برای ربات دانشگاه بپردازیم.

۱-۱- معرفی اتاق رباتیک دانشکده مهندسی و علوم کامپیوتر دانشگاه شهید بهشتی

این اتاق زیر نظر اساتید دانشکده‌ی کامپیوتر، آقایان دکتر سلیمی، دکتر شکفته و دکتر عطارزاده فعالیت دارد. از اعضای فعلی این اتاق می‌توان از آقایان نوید مهدیان، کوروش خاوری مقدم و نگارنده و ... نام برد.

۱-۲- شرح کلی فعالیت اتاق رباتیک دانشکده مهندسی و علوم کامپیوتر دانشگاه شهید بهشتی

در این اتاق پژوهش‌های مربوط به رباتیک و سیستم‌های سایبرفیزیکی است. همچنین در زمینه‌های هوش مصنوعی، پردازش سیگنال، بینایی ماشین و پیاده‌سازی بر روی سخت افزار ربات‌ها فعالیت می‌شود.

۱-۳- شرح فعالیتهای مرتبط اتاق رباتیک دانشکده مهندسی و علوم کامپیوتر دانشگاه شهید بهشتی با رشته تحصیلی

پروژه‌های قبلی و جاری این اتاق عبارتند از:

- ربات نائو (NAO)
- ربات کوآدکوپتر
- ربات شش پا
- ربات کارتپل
- ربات دانشگاه شهید بهشتی

ربات دانشگاه توسط آقای مرسی در دانشکده مکانیک ساخته شد و توسط آقای نوید مهدیان از اتاق رباتیک توسعه یافت.

۲- هدف از کارآموزی

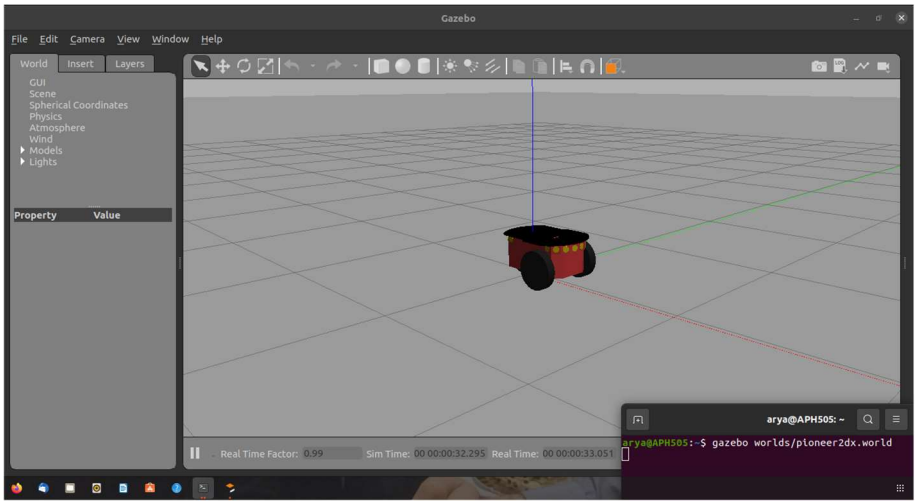
- هدف اصلی از این کارآموزی این است که برای ربات دانشگاه، علاوه بر انتقال این ربات از محیط شبیه سازی گزیبو به ویباتر، بستر ساده و کاملی را برای استفاده‌ی نسل‌های آینده دانشجویان در دانشگاه فراهم کنیم. این بستر دارای ویژگی‌ها و قابلیت‌های متعددی است؛ از جمله، مدل سه بعدی دقیق‌تر و کامل‌تر، محیط فیزیکی بسیار نزدیک به واقعیت، دارای الگوریتم‌های ناوبری کورکورانه، ناوبری همراه با نقشه و مکان‌یابی و نقشه‌سازی همزمان است.
- هدف دیگر از این کارآموزی یادگیری کار با سنسورهای مختلف ربات، مدل سه بعدی آن، مدل حرکتی سینماتیک آن، الگوریتم های مسیریابی کورکورانه، الگوریتم نقشه سازی و موقعیت‌یابی همزمان و کار کردن با میان‌افزار راس است.

۲ – ۱- فعالیت‌های انجام شده

۱. شبیه ساز گزیبو (Gazebo) بر روی لینوکس نصب و تست شد.
۲. مدل سه بعدی بر روی ویباتز (Webots) برده و تست شد.
۳. نرم‌افزارهای ویباتز و گزیبو مقایسه شد و نرم‌افزار ویباتز انتخاب گردید.
۴. سنسورهای تکمیلی و سونار در محیط ویباتز افزوده شد و مدل سه بعدی ربات تکمیل گردید.
۵. کنترلگر و الگوریتم ساده‌ای برای تست مدل انتقال یافته روی ویباتز، نوشته شد.
٦. زبان کنترلگر ربات در محیط ویباتز، از زبان سی (C language) به زبان پایتون (Python language) تغییر یافت.
۷. اندازه‌های واقعی ربات از مرجع [۲] بدست آمد.
۸. سینماتیک مستقیم و معکوس ربات سه چرخ همه جهته بررسی و محاسبه‌ی مجدد شد.
۹. پیاده سازی سینماتیک مستقیم و معکوس ربات در کنترلگر قرار گرفت و مشکلات آن شناسایی و با جزئیات کامل مستند شد.
۱۰. سینماتیک مستقیم و معکوس در محیط شبیه‌ساز ایده‌آل تست و بررسی شد.
۱۱. الگوریتم اولیه‌ای برای تعقیب دیوار ارائه و پیاده‌سازی و مشکلات آن بررسی شد.
۱۲. پارامترهای سنسورهای سونار ربات ابهاماتی داشت، که با تحقیق در مورد پارامترهای مدل واقعی سونار HC-SR04، برطرف شد.
۱۳. برای هر مشکل، اعم از حرکت ربات، تشخیص مانع و اندازه‌ها و پارامترهای فیزیکی ربات راه حلی در نظر گرفته شد و به ترتیب تست شد و مشکلات برطرف گردید. بنابراین الگوریتم اولیه بازنگری شد و الگوریتم کاملتری ارائه شد.
۱۴. برای تست الگوریتم جدید، محیط مجازی پیچیده‌تر شد و ویژگی‌های فیزیکی آن نیز ارتقا یافت.
۱۵. روشهای بدست آوردن نقشه در محیط ویباتز، بررسی شد.
۱۶. الگوریتم‌های مکان‌یابی particle filter، نقشه‌سازی و مکان‌یابی همزمان (SLAM) pose-graph، برنامه‌ریزی و ناوبری بر اساس نقشه RRT*، RRT و A* مطالعه شد.
۱۷. میان‌افزار راس و شبیه‌ساز ویباتز به سیستم عامل لینوکس انتقال یافت.
۱۸. مبانی، مفاهیم و نحوه‌ی ایجاد آن‌ها و عملکرد راس (ROS) مطالعه و بررسی شد.
۱۹. در راستای تغییر معماری نرم‌افزار کنترلر تلاش شد.

۳ – شرح فعالیت‌های کارآموزی

۱. شبیه ساز گزیبو (Gazebo) بر روی لینوکس نصب و تست شد.



۲. مدل سه بعدی بر روی ویباتز (Webots) برده و تست شد.

عمل انتقال ربات از محیط Gazebo به Webots با کمک API در نظر گرفته شده برای این عمل که urdf2webots نام دارد انجام گرفت. هدف این است که توصیف‌ها و طراحی‌های صورت گرفته برای اجزای مختلف ربات را به فرمت proto تبدیل کرده در Webots بیافزایم. بنابراین یک طرح سه بعدی از مدل، ترجیحا به فرمت STL (فرمت های دیگر نظیر STEP و DEA به راحتی قابل تبدیل به STL هستند.) و یک توصیف ترجیحا به فرمت urdf (می توان از xacro هم بهره گرفت) لازم است.
بعد از تغییر جزئی فایل urdf در دسترس و تبدیل فرمت اجزای ربات به فرمت STL، توانستیم که ربات را به صورت proto استخراج کرده و با موفقیت به محیط Webots اضافه کنیم. در ادامه با معرفی proto ی افزوده شده به Webots به عنوان گره ربات، قابلیت تغییر دادن آن در محیط شبیه ساز فعال شد.

۳. نرم‌افزارهای ویباتز و گزیبو مقایسه شد و نرم‌افزار ویباتز انتخاب گردید.

شبیه‌ساز ویباتز با توجه به ویژگی‌های زیر انتخاب گردید.

- دارای قابلیت نصب در تمام بسترها است.
- قابلیت استفاده از تقریبا تمامی زبان‌های برنامه نویسی معروف را دارد.
- با داشتن مدل‌های دینامیکی و شبیه‌ساز دقیق فیزیکی بسیار قدرتمند است.
- امکانات بهتری برای توصیف ساختار ربات و امکانات ساده‌تری برای مدل‌های سه بعدی و تغییر آن‌ها فراهم کرده است.
- دارای امکانات شبیه‌سازی به تعداد دفعات بالا با پیاده‌سازی خودکار است.
- یادگیری برای افراد مبتدی بسیار راحت است.
- تک تک اجزای ربات در کلیه‌ی پروژه‌ها قابلیت استفاده‌ی دوباره نیز دارند.

با توجه به مزایای فوق، این شبیه‌ساز انتخاب شد تا کار کردن با این ربات برای اعضای آینده‌ی آزمایشگاه رباتیک ساده و امکانپذیر باشد.

۴. سنسورهای تکمیلی و سونار در محیط ویباتز افزوده شد.

تعدادی از سنسور ها از قبیل سونار در مدل اولیه‌ی ربات وجود نداشت. در محیط Webots علاوه بر تکمیل قسمت‌های ناقص، سنسورهای GPS و compass که در ربات اصلی نیست اما چون برای مانیتور کردن حرکت و مکان ربات لازم است، افزوده شد.

در ویباتز همه چیز به صورت یک گره یا Node تعریف شده است، این گره ها نیز در فایل‌های proto به صورت درختی و سلسله مراتبی قابل توصیف و مشاهده هستند. مثال تکه‌ای از داخل یک فایل پروتو به شکل زیر است؛

```
#VRML_OBJ R2021b utf8
Robot {
  translation -15.64 -10 0.0690332
  rotation -0.9785662911332222 4.0002311901086377e-07 0.2059320612668402 3.758520688427117e-06
  children [
    Transform {
      children [
        Solid {
          translation -0.0030671 0.00119482 0.03
          rotation -7.0950499999980508e-07 2.234029999993863e-06 -0.9999999999972529 3.14159
          scale 0.0015 0.0015 0.0015
          children [
            Shape {
              appearance PBRAppearance {
              }
              geometry Mesh {
                url [
                  "robot_parts/aluminum profile.stl"
```

سلسله مراتب ربات را به صورت ساده شده‌ی زیر می‌توانیم نمایش دهیم؛

- ربات {
  - آردوینو mega2560
  - ۳ تا رله‌ی L298
  - ۳ عدد موتور PLG32
  - ۳ عدد چرخ سوئدی نود درجه
  - ۳ عدد سونار hc_sr04
  - ۶ عدد سنسور مادون قرمز
  - یک لینک دوربین
  - یک عدد GPS
  - یک عدد قطب نما
  - یک بدنه {
    - دو صفحه‌ی موازی }
  - یک شاسی {
    - ۳ تا پروفایل آلومینیومی
    - یک عدد پروفایل آلومینیومی در راستای عمود
    - ۳ عدد استوانه
    - مرکز شاسی ربات
    - ۶ تا بست‌های قائمه }

}

همانطور که ربات در سلسله مراتب بالا توصیف شد، می‌توان شکل مدل نهایی ربات را به صورت زیر مشاهده کرد. برای جزئیات به پیوست (ت) مراجعه کنید.

با استفاده از سایت‌های اشتراک‌گذاری مدل‌های AutoCAD شماری از قطعات لازم برای ربات که به صورت ابتدایی و ناقص پیاده سازی شده بودند، تکمیل و تصحیح گردیدند. مدل های چرخ های omni-wheel و سونار HC-SR04 و موتور های PLG-32 و ... جمع آوری شدند (برای جزئیات بیشتر به پیوست (ت) مراجعه شود). در ادامه ی پروسه، با تهیه‌ی فایل‌های توصیف آن‌ها، مدل ربات تکمیل و طبیعی تر گردید

۵. کنترلگر و الگوریتم ساده‌ای برای تست مدل انتقال یافته روی ویباتز، نوشته شد.

برای آزمون حرکت ربات الگوریتم ساده‌ای مانند حرکت به جلو و سپس چرخیدن به دور خود و پس از آن حرکت به راست نوشته شد، و ربات انتقال یافته به ویباتز با موفقیت به حرکت در آمد. پس از آن تصمیم بر این شد که برای تست و ناوبری ربات، الگوریتم‌های تعقیب یا کوکورانه را پیاده‌سازی کنیم. در ادامه به چالش‌ها و رویکردهای ما در پیاده‌سازی آن اشاره خواهیم کرد.

٦. زبان کنترلگر ربات در محیط ویباتز، از زبان سی (C language) به زبان پایتون (Python language) تغییر یافت.

در ابتدای امر، حرکات ربات و الگوریتم‌های به زبان C زده شده بودند، اما پس از محاسبات و تحلیل سینماتیکی ربات و همچنین بزرگتر شدن مقیاس و سطوح پروژه تصمیم بر این شد تا از زبان پایتون استفاده شود و قابلیت‌های ساده‌ی این زبان بالخصوص برای محاسبات ماتریسی به کار رود. پس از آن کد تبدیل شده از زبان C تست شد و بدون خطا و مشکل اجرا شد.

۷. اندازه‌های واقعی ربات از مرجع [۲] بدست آمد.

با توجه به اینکه در الگوریتم تعقیب در پیاده‌سازی ابتدایی آن با مشکل مالش ربات به دیوار یا مانع مواجه شده بودیم، برای بررسی علت این مشکل، به مرجع [۲] مراجعه کردیم و مقیاس اندازه‌های ربات را اصلاح کردیم. همچنین برای تحلیل سینماتیکی به اندازه‌های شاسی ربات نیز نیاز داشتیم، در صورتی که اندازه‌ی مطلوب در این مرجع ارائه نشده بود. اندازه‌های مورد نیاز را از نقشه‌های دارای مقیاس مرجع استخراج کردیم. دو اندازه‌ی مهم ربات شعاع چرخ $r = 29mm$ و طول شاسی $l = 220m$ است. مطابق با اشکال زیر؛

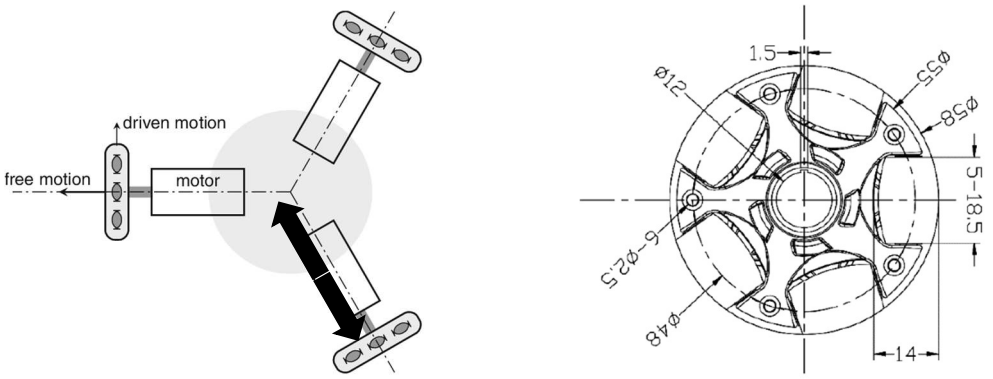


Figure ۱نقشه‌های شاسی و چرخ ربات، باز نشر شده از مرجع ۲

در شکل زیر، مقیاس‌های مدل اولیه‌ی ربات با مدل نهایی آن مقایسه شده است.

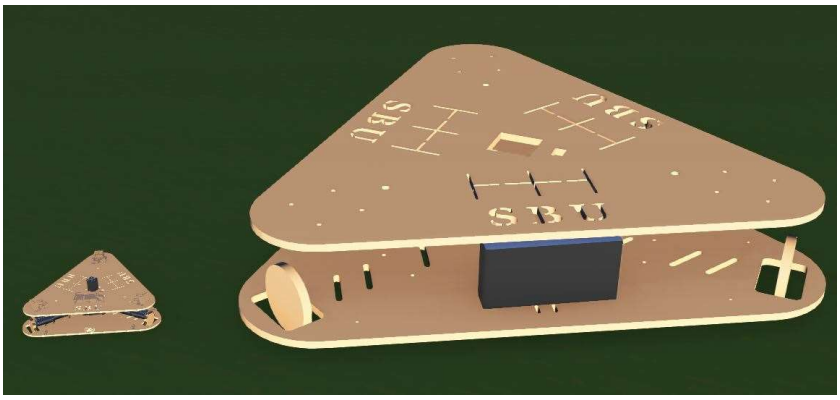


Figure ۲سمت راست مدل اولیه و سمت چپ مدل نهایی را نشان می‌دهد.

۸. سینماتیک مستقیم و معکوس ربات سه چرخ همه جهته بررسی و محاسبه‌ی مجدد شد.

نتیجه‌ی محاسبات

سینماتیک مستقیم (forward kinematics) ربات با فرضیات r = 29mm و l = 220mm، به صورت معادله ماتریسی زیر بدست می‌آید.

$$\dot{\xi}_I = rR^{-1}(\theta)\begin{bmatrix} 0 & \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ \frac{-2}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{-1}{3l} & \frac{-1}{3l} & \frac{-1}{3l} \end{bmatrix}\dot{\phi}\,,$$

و سینماتیک معکوس (inverse kinematics) ربات به صورت معادله‌ی زیر بدست می‌آید.

$$\dot{\phi} = \frac{1}{r}\begin{bmatrix} 0 & -1 & -l \\ \frac{-\sqrt{3}}{2} & \frac{1}{2} & -l \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & -l \end{bmatrix} R(\theta)\dot{\xi}_I\,,$$

که در آن

$$\dot{\phi} = \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{bmatrix},$$

که ماتریس سرعت دورانی چرخ‌ها است. و

$$\dot{\xi}_I = \begin{bmatrix} \dot{x}_I \\ \dot{y}_I \\ \dot{\theta} \end{bmatrix}, \qquad R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\dot{\xi}_R = R(\theta)\,\dot{\xi}_I\,, \qquad R^{-1}(\theta) = R^T(\theta) = R(-\theta)\,.$$

که جزئیات محاسبه در پیوست (الف) است.

۹. پیاده سازی سینماتیک مستقیم و معکوس ربات در کنترلگر قرار گرفت و مشکلات آن شناسایی و با جزئیات کامل مستند شد.

بعد از پیاده‌سازی سینماتیک، ماژول آن را برای حرکت رو به جلو و پیدا کردن دیوار، به الگوریتم تعقیب دیوار اضافه کردیم. خروجی حرکت ربات مطابق حرکت مورد انتظار و پیش‌بینی شده نبود. این حرکت نادرست اما قطعی بود؛ قطعی بودن به این معنا که با تکرار اجرای شبیه‌سازی به ازای ورودی‌های اولیه حرکتی یکسان اما نادرست و نامطلوب انجام می‌شد.

به این منظور برای مستند سازی رفتار حرکتیِ نادرست، در کد کنترلر ویباتز کامنت‌های کاملی برای توصیف رفتار نادرست حرکتی به ازای هر دستور داده شده، نوشته شد. لازم به ذکر است که تمامی حرکت‌های مهم و ساده در این آزمایش‌ها همراه با دستور کوچک یک خطی فراهم شده است. این کد در گیتهاب (مرجع [۶]) نیز در دسترس عموم است. جزئیات بیشتر و توصیفات فیزیکی رفتار مورد انتظار همراه با شکل در پیوست (پ) قرار دارد.

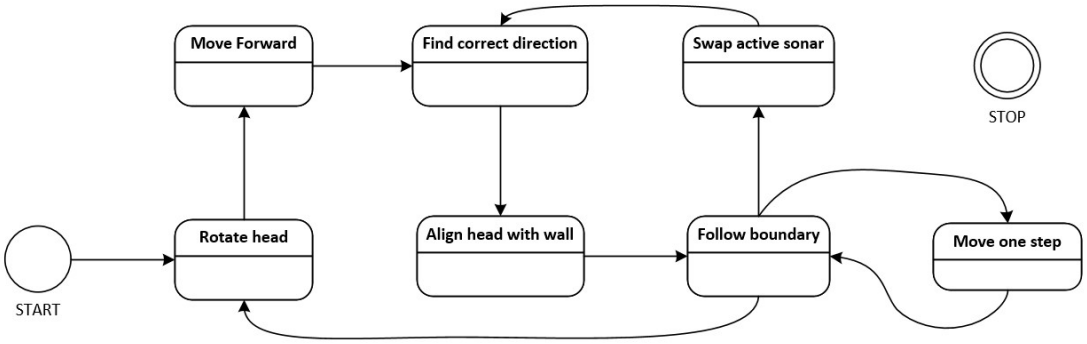
```
266         # experiment 5
267         #robot_omega = [-3, 0, 3] # expected movement: diagonal movement with +60 degrees offset (parallel t
268                                    # unexpected actual movement: first vertical movement along +Y_I axis, then a su
269
270         # experiment 6
271         #robot_omega = [3, 0, -3] # expected movement: diagonal movement with +60 degrees offset (parallel t
272                                  # unexpected actual movement: vertical movement along -Y_I axis.
273
274         # experiment 7
275         #robot_omega = [-3, 1.5, 1.5] # expected movement: (without acceleration): vertical movement along +
276                                        # (considering acceleration in angular movements of wheels): first a slight rotatio
277                                        # unexpected actual movement: first a slight rotation of robot in clockwise directi
```

```
266         # experiment 5
267         robot_omega = [-3, 0, 3] # expected movement: diagonal movement with +60 degrees offset (parallel to wh
268                                    # unexpected actual movement: first vertical movement along +Y_I axis, then a sudde
269
270         # experiment 6
271         #robot_omega = [3, 0, -3] # expected movement: diagonal movement with +60 degrees offset (parallel to w
272                                  # unexpected actual movement: vertical movement along -Y_I axis.
273
274         # experiment 7
275         #robot_omega = [-3, 1.5, 1.5] # expected movement: (without acceleration): vertical movement along +Y_I
276                                        # (considering acceleration in angular movements of wheels): first a slight rotation c
277                                        # unexpected actual movement: first a slight rotation of robot in clockwise direction
```

۱۰. سینماتیک مستقیم و معکوس در محیط شبیه‌ساز ایده‌آل تست و بررسی شد.

کد شبیه‌سازی ایده آل سینماتیک، در کنار پوشه‌های کدهای اصلی در گیتهاب (مرجع [۷]) قرار گرفت. در این شبیه سازی امکان ایجاد شتاب دورانی برای هر چرخ و ایجاد جنبه‌ی اصطکاک نیز برای آن‌ها وجود دارد، اما با چالش‌های اندازه‌گیری این مقادیر در محیط شبیه‌ساز و محیط واقعی رو به رو هستیم و آن را در اولویت‌های بعدی قرار داده‌ایم به دلیل اینکه ممکن است این شبیه سازی غیر ضروری و نا لازم باشد.
نتیاج دقیق هر حرکت در پیوست (ب) با جزئیات دقیق توضیح داده شده است، که اثباتی بر درست بودن مدل سینماتیکی و درست بودن عملیات‌های آن خارج از یک محیط پیچیده‌ی فیزیکی است.

۱۱. الگوریتم اولیه‌ای برای تعقیب دیوار ارائه و پیاده‌سازی و مشکلات آن بررسی شد.



نمودار حالت اولیه کشیده شده با زبان UML

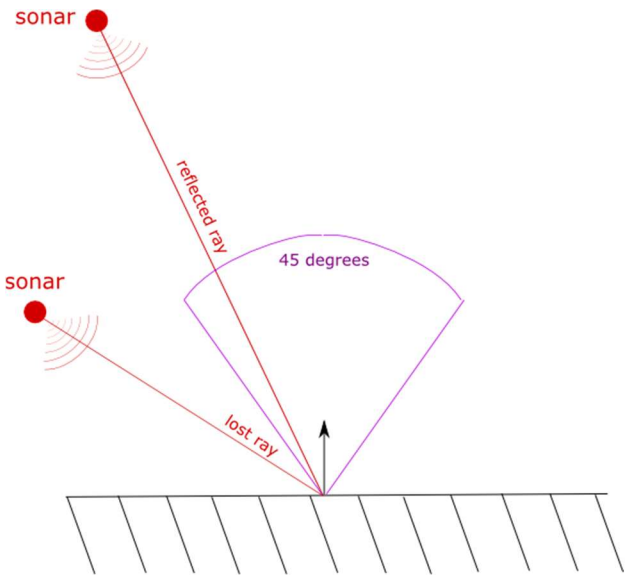
حالت‌های این الگوریتم به شرح زیر است؛

- Start:
- Rotate head:
- Move forward:
- Find correct direction:
- Align head with wall:
- Follow boundary:
- Move one step:
- Swap active sonar:
- Stop:

مشکلات این بخش از کاراموزی شامل موارد زیر بود؛

الف – با وجود مشکل حرکتی که در قسمت‌های قبلی اشاره شد، ربات خطای بسیار زیادی داشت که با گذاشتن یک کنترلر ساده نیز قابل صرف نظر نبود. با توجه به توضیحاتی که در کد به صورت مفصل کامنت شده اند و در بند نهم نیز اشاره شد، در این قسمت سر ربات در حرکت مستقیم انحراف پیدا می‌کرد. با یک کنترلر ساده (کنترلر P) با درجا چرخاندن، توانستیم سر ربات را در جهت مطلوب حفظ کنیم. همچنین حرکت ربات را به صورت گام‌های کوچک که به نوبت، گام کوچکی در جهت افقی، و گام کوچکی در جهت عمودی به میزان لازم برداشته می‌شد تا نتیجه‌ی آن حرکت در زاویه‌ی مشخصی شود. با این راهکار توانستیم با وجود اشکال حرکتی، حرکت ربات را در مسیر مطلوب هدایت کنیم. البته این راه‌حل در این مرحله از کاراموزی با مشاهدات بصری و عددی به خطاهای بزرگی در بلند مدت و مسیرهای طولانی ختم می‌شد. لازم به ذکر است که راهکار اصلی و اساسی در بندهای بعدی به طور کامل بررسی خواهند شد.

ب – امواج صوتی حسگر سونار باید در میدان زاویه‌ای خاصی نسبت به نقطه‌ی برخورد با دیوار داشته باشد، در غیر اینصورت، امواج بازتابی به حسگر باز نخواهند گشت. و این به این معناست که همه‌ی موانع نسبت به سونار نقاط کوری دارند. مطابق شکل زیر؛



سونار پایینی قادر به دیدن مانع نیست چون پرتوی آن خارج از زاویه‌ی دید قرار دارد. اما سونار بالایی مانع را می‌بیند[۱۷].

۱۲. پارامترهای سنسورهای سونار ربات ابهاماتی داشت، که با تحقیق در مورد پارامترهای مدل واقعی سونار HC-SR04، برطرف شد.

پارامترهای فنی حسگر سونار HC-SR04 که برگه‌ی داده‌ی (datasheet) آن در پیوست (ت) قرار دارد، در شبیه‌ساز اعمال شد.

۱۳. برای هر مشکل، اعم از حرکت ربات، تشخیص مانع و اندازه‌ها و پارامترهای فیزیکی ربات راه حلی در نظر گرفته شد و به ترتیب تست شد و مشکلات برطرف گردید. بنابراین الگوریتم اولیه بازنگری شد و الگوریتم کاملتری ارائه شد.

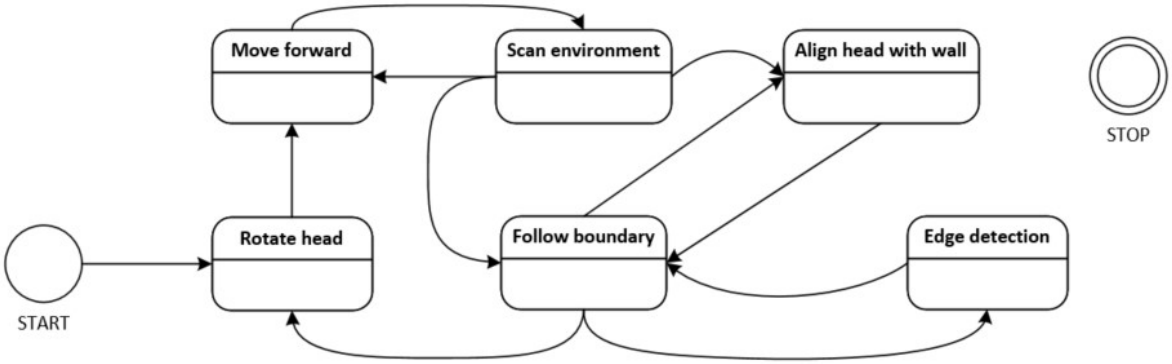
متوجه شدیم که مشکل اصلی حرکت مربوط به لق بودن چرخ‌های ربات بود. این چرخ‌ها را به وسیله‌ی یک مفصل (joint) به شاسی متصل کردیم تا این مشکل برطرف شود. علاوه بر این مدل چرخ‌ها در پروژه‌ی آقای مهدیان به صورت چرخ ساده اما با اصطکاک جانبی صفر، در نظر گرفته شده بود. ما به صورت دستی با همان مفصل‌ها چرخ‌های کوچکتری روی چرخ ساده گذاشتیم تا به صورت یک چرخ همه جهته شود (omni-wheel).

برای مشکل سونار، مجبور شدیم یک تایمر تعریف کنیم تا با استفاده از آن، به نوبت یک قدم مستقیم در جهت مطلوب و در نوبت بعدی یک چرخش در جا انجام می‌دهد اگر دیواری در اطرافش قرار دارد، آن را مشاهده کند. همانطور ک در بند یازدهم اشاره شد، حسگر سونار برای پیدا کردن دیوار، باید در زاویه‌ی خاصی نسبت به آن قرار داشته باشد تا در میدان کور آن نباشد، این چرخش مشکل را رفع می‌کند.

همچنین مشکلات ذکر شده باعث شده بودند تا ربات در پیدا کردن گوشه‌ها و عکس العمل به آن‌ها عملکردی خوبی نداشته باشد. برای رفع این مشکل نیز حرکت نیمه دواری برای ربات در نظر گرفته شد که اگر گوشه‌ای محدب پیدا شد یا در مانعی گیر کرد، این مانور را انجام دهد.

برای کشف گوشه‌های مقعر یا کنترل حرکت در آن‌ها از همان مکانیزم چرخش درجای ربات استفاده می‌شود. در جزئیات پیاده‌سازی این قسمت، متغیر سونار مرکزی (pivot_sonar) را در صورت دیده شدن دیوار دیگری توسط سونار دیگری، از سونار اول به سونار دوم تغییر می‌دهیم. در نتیجه‌ی این تغییر ربات در امتداد دیوار دوم به حرکت ادامه خواهد داد.

الگوریتم ارائه شده را با نمودار حالت زیر نمایش داد.

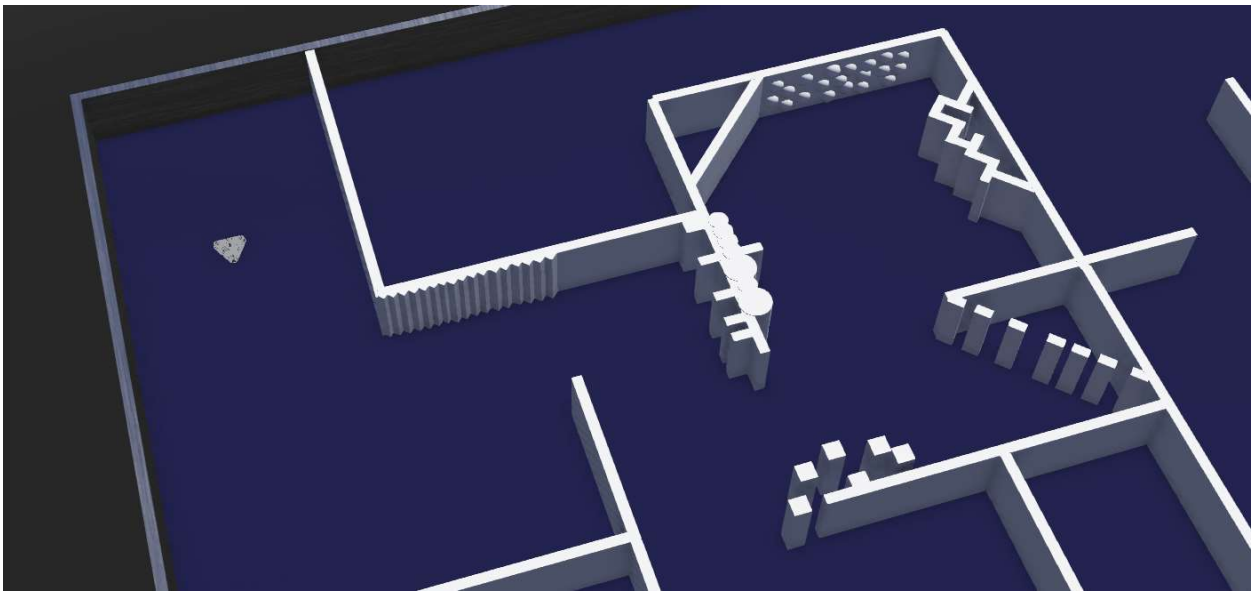


نمودار حالت الگوریتم نهایی، کشیده شده با UML

شرط پایان اجرای این الگوریتم رسیدن به اولین نقطه‌ی برخورد با دیوار است.

۱۴. برای تست الگوریتم جدید، محیط مجازی پیچیده‌تر شد و ویژگی‌های فیزیکی آن نیز ارتقا یافت.

ربات در این محیط (شکل زیر) توانست دیوارها و موانع را با موفقیت دور بزند و الگوریتم تعقیب دیوار را به خوبی به پایان برساند.



ربات در یک محیط با موانع پیچیده، ربات در قسمت بالا سمت چپ مشاهده می‌شود.

۱۵. روشهای بدست آوردن نقشه در محیط ویباتز، بررسی شد.

با جستجو و بررسی‌های انجام شده به این نتیجه رسیدیم که سه روش متداول نقشه‌های به فرمت ماتریس اشغالی (occupancy grid) وجود دارد.

الف – کشیدن نقشه به صورت دستی با ابزارهای کامپیوتری، و سپس فشرده کردن فایل تصویر آن به کمک روش‌های ساده‌ی پردازش تصویر.

ب – بازسازی نقشه‌ی مجازی (مثلاً یک ماز) در دنیای واقعی. مازهای ساده در یک محیط شبیه‌سازی و مجازی می‌توان با اجزای ساده‌ای یا صفحات پلاستیکی بازسازی کرد.

پ – استفاده از JOSM (مخفف Java OpenStreetMap) که یک نرم‌افزار و API برای نقشه‌های واقعی در فضاهای باز است [۹]. این نقشه‌ها را می‌توان به فرمت مختلف سه بعدی دریافت کرد و با توجه به پشتیبانی کامل ویباتز از این نرم‌افزار نقشه‌ی مکان دلخواه از سرتاسر دنیا را مستقیما در شبیه‌ساز وارد کرد[۱۰]. این نرم‌افزار قابلیت ویرایش نقشه را نیز مهیا کرده است. اما برای محیط‌های بسته همچون فضای ساختمان دانشکده‌ی مهندسی و علوم کامپیوتر، چنین نقشه‌ی سه بعدی در دسترس نیست.

ت – استفاده از کتابخانه‌های آماده [۱۱]. خوشبختانه ویباتز از میان‌افزارهای ROS و ROS2 پشتیبانی کامل می‌کند. برای این دو میان‌افزار بسته‌های آماده‌ای (packages) با نام‌های Gmapping و SLAM-toolbox وجود دارد که به ترتیب برای میان‌افزارهای ROS و ROS2 هستند.

با توجه به روش‌های مذکور تصمیم بر این شد تا از کتابخانه‌های آماده و به صورت خاص بسته‌ی SLAM toolbox برای میان‌افزار استفاده کنیم.

۱۶. الگوریتم‌های مکان‌یابی particle filter، نقشه‌سازی و مکان‌یابی همزمان pose-graph (SLAM)، برنامه‌ریزی و ناوبری بر اساس نقشه RRT*، RRT و A* مطالعه شد [۱۲، ۱۳، ۱۴، ۱۵].

۱۷. میان‌افزار راس و شبیه‌ساز ویباتز به سیستم عامل لینوکس انتقال یافت.

در پروسه‌ی نصب این میان‌افزار و شبیه‌ساز مشکلات متعددی وجود داشت که شرح آن در گیتهاب برای مطالعه‌ی عموم باز است [۱۶].

۱۸. مبانی، مفاهیم و نحوه‌ی ایجاد آن‌ها و عملکرد راس (ROS) مطالعه و بررسی شد.

نحوه‌ی ایجاد مفاهیم راس شامل؛ workspace، package، service، publisher، subscriber و node بررسی شد [۱۸-۲۹].

۱۹. در راستای تغییر معماری نرم‌افزار کنترلر تلاش شد.

معماری نرم‌افزاری که در کاراموزی استفاده شد به صورت ترتیبی و مانند معماری von-neumann بود. برای استفاده از بسته‌ی SLAM ToolBox که از ROS2 استفاده می‌کند باید معماری نرم افزار را به صورت Peer to peer یا message passing تغییر داد که برنامه‌ها در این معماری به صورت موازی اجرا می‌شوند و جزئیات مفصلا در مراجع [۱۸-۲۹] توضیح داده شده است. همانطور که می‌دانیم تغییر معماری نرم‌افزار در فازهای میانی و انتهایی پروژه کار بسیار زمانبر و مشکلی است. برای همین در کاراموزی در راستای تغییر معماری تلاش‌هایی انجام شد و همچنان کار بر روی آن ادامه دارد.

برای استفاده از بسته‌ی SLAM ToolBox، برای ربات دانشگاه یک package و یک workspace مرتبط با ROS ساخته شد و فایل‌ها به فرمت‌های مطلوب برای تولید service، client، publisher و subscriber ساخته شدند و از کد اصلی پروژه به آن‌ها در حال انتقال هستند. سعی شد تا فایل وابستگی‌ها (package.xml و setup.py) به درستی تنظیم شود.

پیشنهادها

- برای ربات جهت بهبود عملکرد و از بین بردن نقاط کور در تشخیص می‌توان به جای الگوریتم پیشنهاد شده که انرژی مضاعفی مصرف می‌کند، می‌توان تعداد حسگرهای سونار را در اضلاع ربات دو برابر کرد. روش دیگر برای بهبود این مشکل، جایگزین کردن حسگرهای سونار با حسگرهای بهتر و مناسب‌تر است (مثلاً برای الگوریتم‌های SLAM معمولاً از حسگر Lidar استفاده می‌شود).
- برای بهینه‌سازی مصرف انرژی ربات بدون اضافه کردن حسگر جدید، می‌توان با الگوریتم‌های یادگیری تقویتی (reinforcement learning) میزان مصرف انرژی را کاهش داد.
- می‌توان با استفاده از الگوریتم یادگیری تقویتی خطاها و یا رفتار دینامیکی ربات را پیش‌بینی کرد و یا تقریب زد.
- برگزاری کارگاه‌هایی حول موضوع میان‌افزار ROS می‌تواند برای داشنجویان علاقه‌مند به کار یا تحقیقات در زمینه‌ی رباتیک بسیار مفید باشد. و هچنین به ایجاد و رشد ROS community در ایران و رایج‌تر شدن این میان‌افزار کمک می‌کند.
- پیشنهاد می‌شود که در راستای ادامه و بهبود معماری نرم‌افزار هدایت ربات و اضافه شدن و تکمیل قابلیت‌های نقشه‌سازی و مکان‌یابی، بسته‌ی ربات دانشگاه در workspace تکمیل شود.
- برای ساده‌تر کردن تغییر معماری و استفاده از ROS پیشنهاد می‌شود از نرم‌افزار سیمولینک برای ادامه‌ی توسعه‌ی ربات استفاده شود. این نرم‌افزار امکانات خوبی از جمله سادگی توصیف و توسعه نرم‌افزار و همچنین قابلیت‌های کدسازی در اختیار توسعه‌دهندگان قرار می‌دهد.

٤- خلاصه

برای جمعبندی این کاراموزی می‌توان به اضافه کردن قابلیت الگوریتم ناوبری بدون نقشه و الگوریتم باگ ۲ برای عبور و پرهیز از موانع اشاره کرد. برای حل مازها نیز می‌توان به الگوریتم تعقیب دیوار توسعه یافته اشاره داشت. همچنین قابلیت‌های بیشتری به این ربات اضافه شد تا بتواند ناوبری و طرح‌ریزی با نقشه، نقشه‌سازی از محیط‌های ناشناس و مکان‌یابی را انجام دهد. از نظر فنی این ربات به نرم‌افزارهای متلب، ROS، Webots متصل است و با هرکدام به درستی می‌تواند ارتباط برقرار کند و از ابزارهای قدرتمند این نرم‌افزارها استفاده کند.

پیوست الف) ریز محاسبات سینماتیک ربات

جزئیات محاسبات سینماتیک ربات به شرح زیر است.

چرخ استاندارد ثابت

در ابتدا به بررسی و تحلیل سینماتیک چرخ استاندارد ثابت (fixed standard wheel) در حالت عمومی می‌پردازیم.

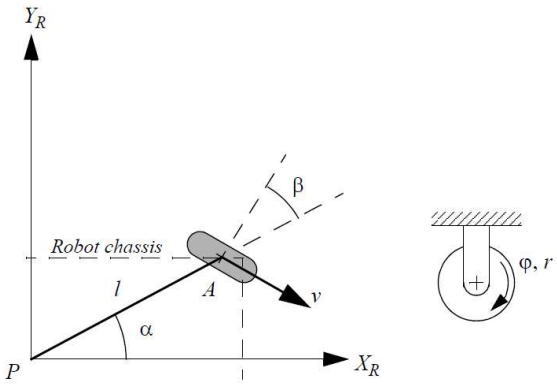
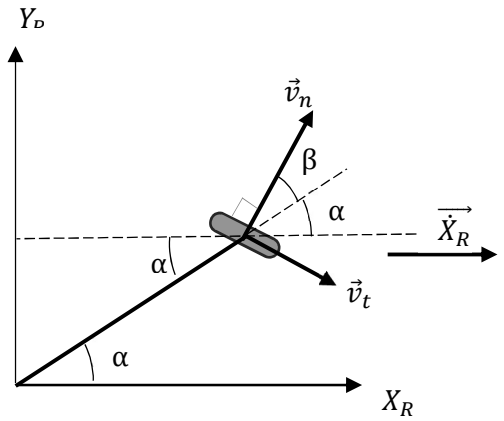


Figure 3 - the fixed standard wheel in robot frame. P is the center of robot.

Reprinted from ref. [1]

ابتدا برای بدست آوردن معادله‌ی غلتش (rolling constraint)، هر مولفه از دستگاه ربات را به دو مولفه‌ی مماس بر حرکت چرخ (با بردار یکه‌ی $\overrightarrow{v_t}$ یا tangent velocity) و محور عمود بر صحفه‌ی چرخ (با بردار یکه‌ی $\overrightarrow{v_n}$ یا normal velocity) یا همان دستگاه چرخ تجزیه می‌کنیم.

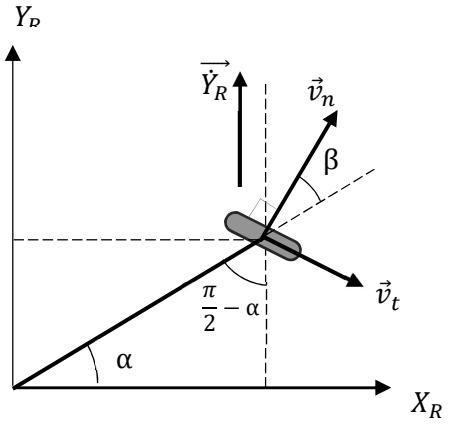
مولفه‌ی اول سرعت در راستای $\dot{x}_R$



$$\vec{\dot{x}}_R = |\vec{\dot{x}}_R| \cos\Big(\frac{\pi}{2} - (\alpha+\beta)\Big)\vec{v}_t + |\vec{\dot{x}}_R| \cos\Big(\alpha+\beta\Big)\vec{v}_n$$

$$\Longrightarrow \qquad \vec{\dot{x}}_R = |\vec{\dot{x}}_R| \sin\Big(\alpha+\beta\Big)\vec{v}_t + |\vec{\dot{x}}_R| \cos\Big(\alpha+\beta\Big)\vec{v}_n$$

مولفه‌ی دوم سرعت در راستای $\dot{y}_R$،

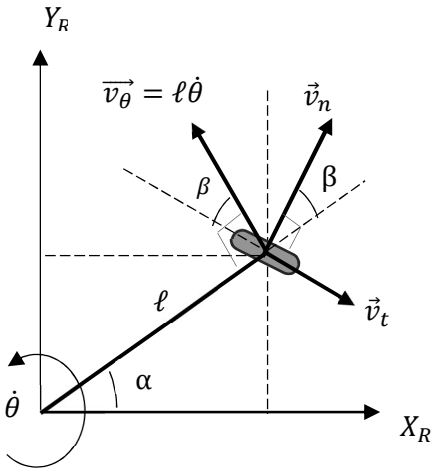


$$\vec{\dot{y}}_R = |\vec{\dot{y}}_R| \cos\Big((\frac{\pi}{2} - \alpha) - \beta\Big)\vec{v}_n - |\vec{\dot{y}}_R| \sin\Big(\frac{\pi}{2} - \alpha - \beta\Big)\vec{v}_t$$

$$\Longrightarrow \qquad \vec{\dot{y}}_R = |\vec{\dot{y}}_R| \sin\Big(\alpha+\beta\Big)\vec{v}_n - |\vec{\dot{y}}_R| \cos\Big(\alpha+\beta\Big)\vec{v}_t$$

مولفه‌ی سوم سرعت در راستای $\dot{\theta}$، سرعت $\overrightarrow{v_\theta}$، سرعت خطی ناشی از سرعت دورانی ربات است. همچنین می‌دانیم،

$$\overrightarrow{v_t} = \vec{\omega} \times \vec{l}$$



سرعت ناشی از سرعت دورانی $\dot{\theta}$ را با $\vec{v}_\theta$ نشان می دهیم:

$$\vec{v}_\theta = -\ell\dot{\theta}\cos\beta\vec{v}_t + \ell\dot{\theta}\cos\left(\frac{\pi}{2}-\beta\right)\vec{v}_n$$

اگر برآیند سرعت‌ها را در راستای $\overrightarrow{v_t}$ محاسبه کنیم، به شرط نداشتن لغزش (rolling constraint)، با سرعت غلتشی چرخ ($r\dot{\phi}$) برابر است،

$$\Big(\dot{x}_R\sin(\alpha+\beta)+\dot{\theta}(-\ell\cos\beta)+\dot{y}_R(-\cos(\alpha+\beta)\Big)\vec{v}_t - \vec{\dot{\phi}}\times\vec{r} = 0$$

$$\dot{\xi}_R = \begin{pmatrix}\dot{x}_R\\ \dot{y}_R\\ \dot{\theta}\end{pmatrix}\ , \qquad \underbrace{\begin{pmatrix}\sin(\alpha+\beta) & -\cos(\alpha+\beta) & -\ell\cos\beta\end{pmatrix}}_{A}\dot{\xi}_R = AR(\theta)\dot{\xi}_I$$

$$\dot{\xi}_I = \begin{pmatrix}\dot{x}_I\\ \dot{y}_I\\ \dot{\theta}\end{pmatrix}\ , \qquad R(\theta) = \begin{pmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}$$

برآیند سرعت‌ها در راستای $\overrightarrow{v_n}$، برابر صفر است، زیرا آزادی حرکت در راستای عمود چرخ وجود ندارد.

نمایش ماتریسی قیدها، برای همه‌ی چرخ‌های ربات:

اگر N چرخ استاندارد ثابت داشته باشیم؛ ماتریس سرعت زاویه‌ای چرخ‌ها را به صورت زیر تعریف می‌کنیم؛

$$\dot{\phi} = \begin{bmatrix}\dot{\phi}_1\\ \vdots\\ \dot{\phi}_N\end{bmatrix}$$

ماتریس J_1 را به صورت زیر تعریف می‌کنیم؛

$$J_1 = \begin{bmatrix}A_1\\ \vdots\\ A_N\end{bmatrix}$$

که A_1 تا A_N، ماتریس‌های سطری برگرفته از شرط‌های غلتش هستند. (تعریف ماتریس A در عکس بالا است)

$$J_1R(\theta)\dot{\xi}_I - J_2\dot{\phi} = 0$$

که در آن،

$$J_2 = \begin{bmatrix} r & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & r \end{bmatrix}_{N \times N}$$

و r شعاع هر چرخ است.

برای شرط sliding نمایش ماتریسی می‌توان داشت، اما به خاطر ساختار چرخ ربات دانشکده که از نوع چرخ سوئدی نود درجه یا همان چرخ همه جهته (omni-wheel, omni-directional wheel or 90 degrees Swedish wheel)، این شرط وجود ندارد.

چرخ سوئدی

Figure 4 - 90 degree Swedish wheel, SBU ROBOT

Reprinted from ref. [2]

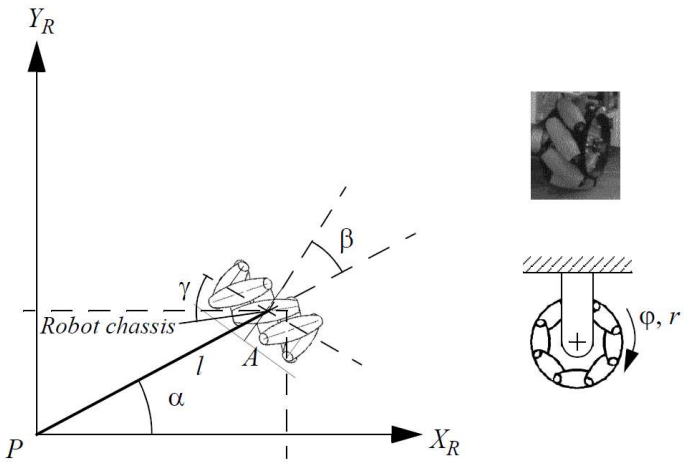


Figure 3- a Swedish wheel and its parameters, indicating wheel axes.

Reprinted from ref. [1]

بدون بررسی و ارائه‌ی اثبات روابط شروط غلتش و لغزش، این روابط را از مرجع [۱]، بازنویسی می‌کنیم؛

$$\left[\sin(\alpha+\beta+\gamma) \;\; -\cos(\alpha+\beta+\gamma) \;\; (-l)\cos(\beta+\gamma)\right] R(\theta)\dot{\xi}_I - r\dot{\varphi}\cos\gamma \;=\; 0\,.$$

$$\left[\cos(\alpha+\beta+\gamma) \;\; \sin(\alpha+\beta+\gamma) \;\; l\sin(\beta+\gamma)\right] R(\theta)\dot{\xi}_I - r\dot{\varphi}\sin\gamma - r_{sw}\dot{\varphi}_{sw} \;=\; 0\,.$$

در ربات دانشکده، $\gamma = 0$ و قید لغزش وجود ندارد، یعنی چرخ، در راستای عمود بر صفحه‌ی چرخ آزادانه حرکت می‌کند و سرعت حرکت عمودی آن با برآیند سرعت غلتش چرخ‌های کوچک چرخ سوئدی شکل ۳ برابر است. همچنین در قید غلتش چون $\gamma = 0$ است، کاملا مشابه با چرخ استاندارد ثابت عمل می‌کند و تفاوت آن فقط در قید لغزش آن است.

سینماتیک ربات همه جهته،

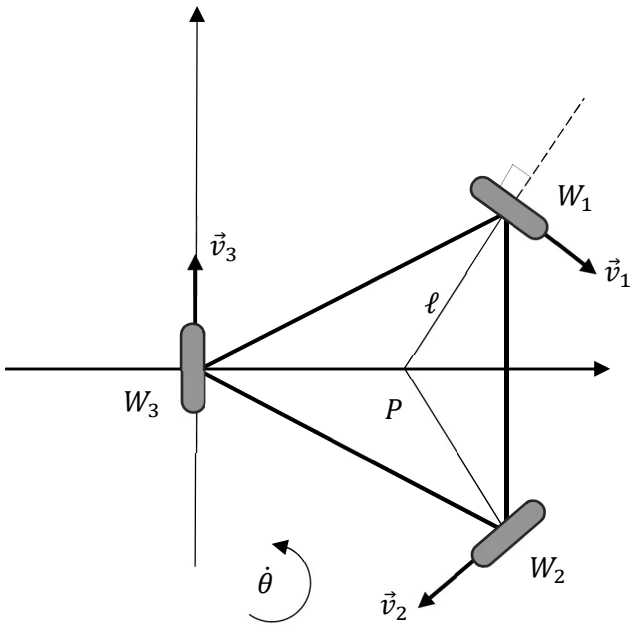


Figure 6 - structure of Omni-robot.

با توجه به شکل بالا، زوایای α و β برای چرخ‌ها به صورت زیر است،

$$\alpha_1 = 0\,,$$

$$\alpha_2 = \frac{-2\pi}{3}\,,$$

$$\alpha_3 = \frac{2\pi}{3}\,,$$

$$\beta_1 = \beta_2 = \beta_3 = 0\,,$$

برای بدست آوردن زوایای α و β، توجه داشته باشید که α زاویه‌ی خط اتصال بین مرکز چرخ و نقطه‌ی P (chassis axle، محور شاسی)، با جهت مثبت محور X_R است. برای یافتن β، جهت سرعت خطی تمام چرخ‌ها را چنان در نظر می‌گیریم که ربات در جهت منفی $\dot{\theta}$ دوران کند. در ضمن $\dot{\theta}$ را در جهت پادساعتگرد فرض می‌کنیم. بدین ترتیب اگر سرعت خطی هر چرخ را نود درجه پادساعتگرد بچرخانیم، زاویه‌ی آن با جهت محور شاسی، زاویه‌ی β است؛ مطابق با شکل زیر،

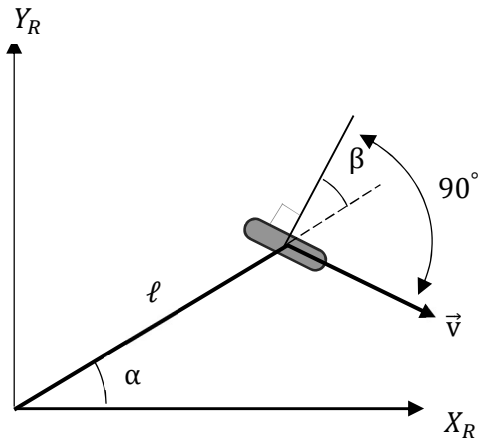


Figure 7- β angle definition.

حال اگر رابطه‌ی قید غلتش را برای ربات دانشکده بازنویسی کنیم، روابط زیر را خواهیم داشت،

$$J_1 R(\theta)\dot{\xi}_I - J_2\dot{\phi} = 0$$

$$J_1 = \begin{bmatrix} \sin(\alpha_1+\beta_1) & -\cos(\alpha_1+\beta_1) & -l\cos(\beta_1) \\ \sin(\alpha_2+\beta_2) & -\cos(\alpha_2+\beta_2) & -l\cos(\beta_2) \\ \sin(\alpha_3+\beta_3) & -\cos(\alpha_3+\beta_3) & -l\cos(\beta_3) \end{bmatrix} = \begin{bmatrix} 0 & -1 & -l \\ \frac{-\sqrt{3}}{2} & \frac{1}{2} & -l \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & -l \end{bmatrix}$$

$$\dot{\xi}_I = R^{-1}(\theta)\, J_1^{-1} J_2 \dot{\phi}\,, \qquad J_1^{-1} = \begin{bmatrix} 0 & \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ \frac{-2}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{-1}{3l} & \frac{-1}{3l} & \frac{-1}{3l} \end{bmatrix}, \qquad J_2 = \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix} = r\, I_{3\times 3}\ ,$$

بنابراین رابطه‌ی نهایی بصورت زیر در می‌آید،

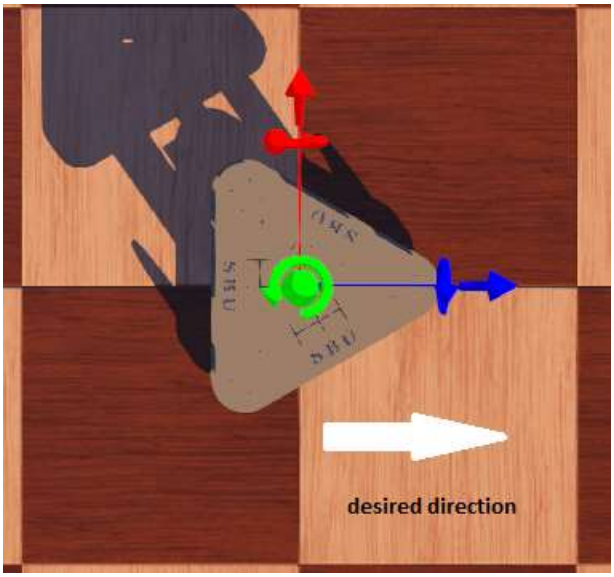
$$\dot{\xi}_I = r\, R^{-1}(\theta) \begin{bmatrix} 0 & \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ \frac{-2}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{-1}{3l} & \frac{-1}{3l} & \frac{-1}{3l} \end{bmatrix} \dot{\phi}\,.$$

پیوست ب) تحلیل و آزمون مدل سینماتیک

در این کد سعی شده تمامی حرکات ساده و مهم به نمایش گذاشته شوند، در تمامی حرکات، جهت سر ربات در راستای مثبت محور X_I ها است و زاویه‌ی θ که در پیوست الف معرفی شده است، صفر است. در هر شکل زیر جهت دلخواه حرکت ربات کشیده شده است و زیر آن کد مربوط به زبان پایتون و نتیجه‌ی آن نیز در زیر آن مشخص شده است. نمودارهای شکل نتیجه نیز از نقطه‌ی ابتدایی که با دایره نمایش داده شده شروع، و با نقطه انتهایی که با ستاره مشخص شده است، برنامه متوقف شده است (تعداد حلقه‌های اجرایی برنامه ۲۵۰ تا و گام زمان هر حلقه ۶۴ میلی ثانیه است). برای نمایش درستی مدل سینماتیک اعداد و مقادیر هر چرخ، پس از اجرای هر بار برنامه نوشته شده است. کد این شبیه سازی ساده نیز در گیتهاب قابل دسترسی است.

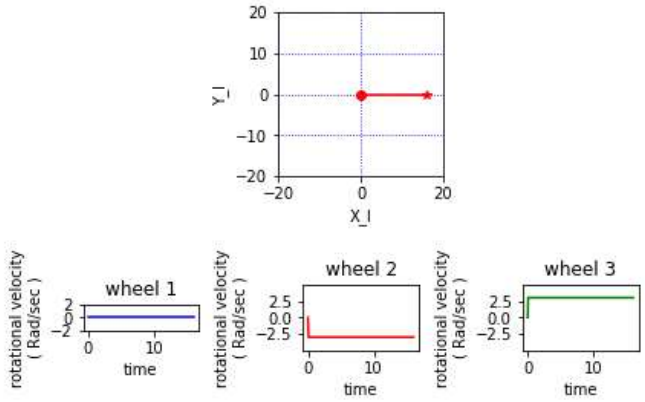
حر کت در راستای محور هر چرخ،

چرخ ۱:

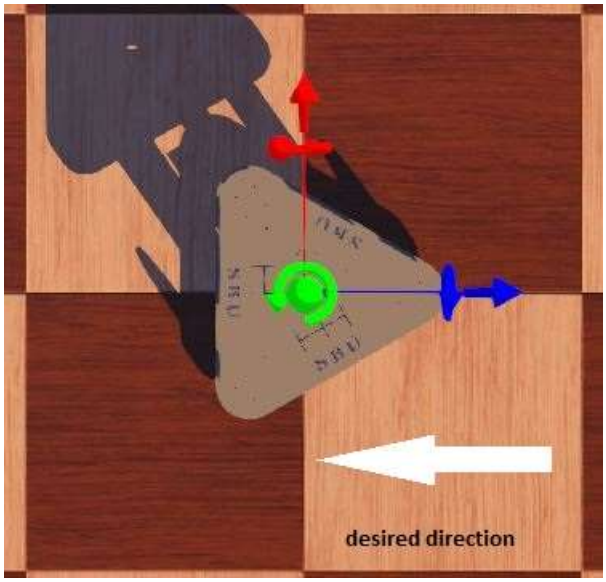



```
desired_angle = math.radians(0)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along wheel 1 normal vector
```

```
robot's last position = (16.00, 0.00)
wheels' rotational velocity:
"wheel 1" = 0.00,        "wheel 2" = -2.99,       "wheel 3" = 2.99.
```

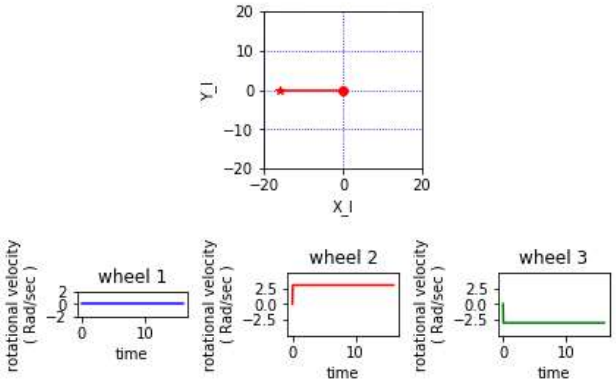


در راستای معکوس محور چرخ ۱،



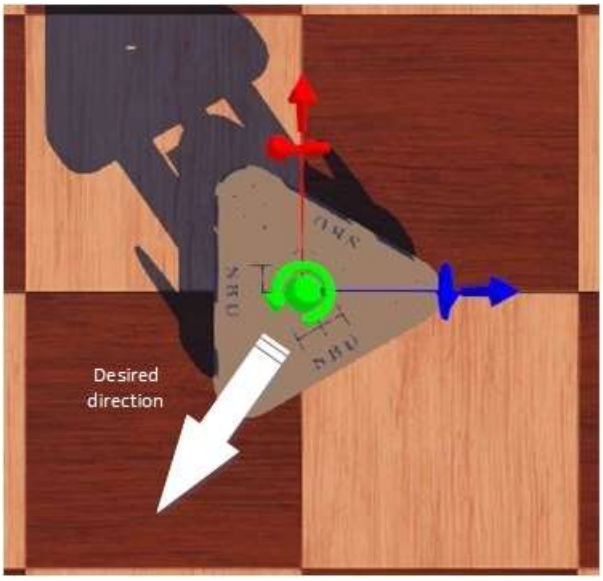

```
desired_angle = math.radians(180)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along wheel 1 normal vector
```

```
robot's last position = (-16.00, 0.00)
wheels' rotational velocity:
"wheel 1" = -0.00,       "wheel 2" = 2.99,       "wheel 3" = -2.99.
```



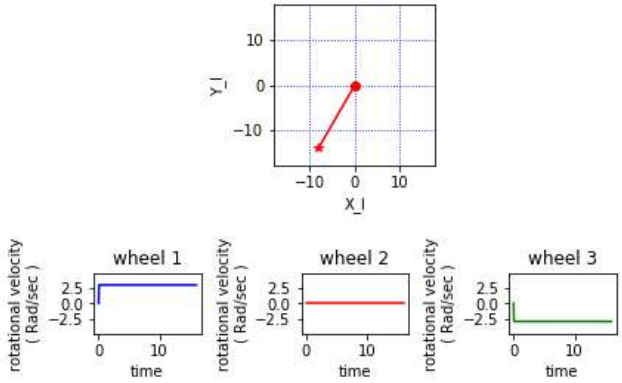
همانطور که در شکل مشاهده می‌کنید، اندازه‌ی هر چرخ در طول زمان و همچنین مقدار آن با دقت دو رقم اعشار مشخص شده است. این عددها در پیوست پ با جزئیات کامل تحلیل شده‌اند.

چرخ ۲:

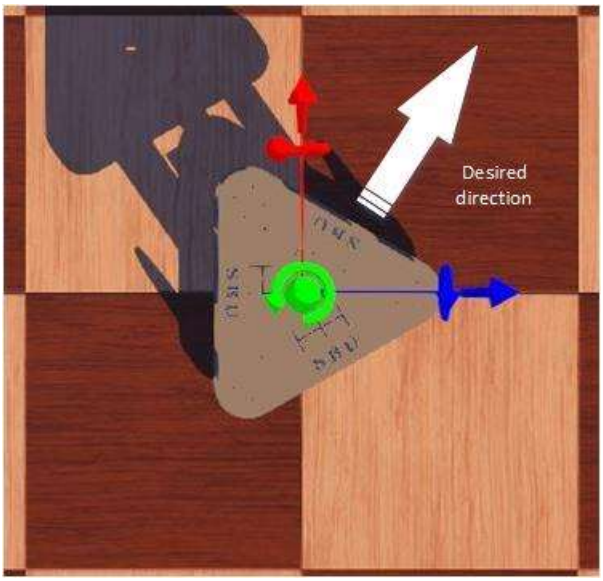



```
desired_angle = math.radians(-120)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along wheel 2 normal vector
```

```
robot's last position = (-8.00, -13.86)
wheels' rotational velocity:
"wheel 1" = 2.99,        "wheel 2" = 0.00,       "wheel 3" = -2.99.
```

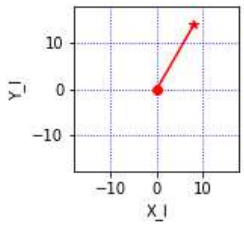


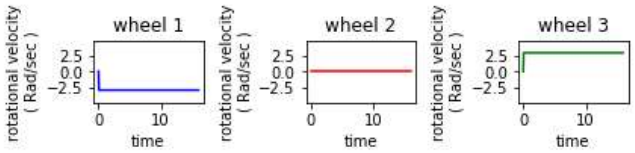
در راستای معکوس محور چرخ ۲،



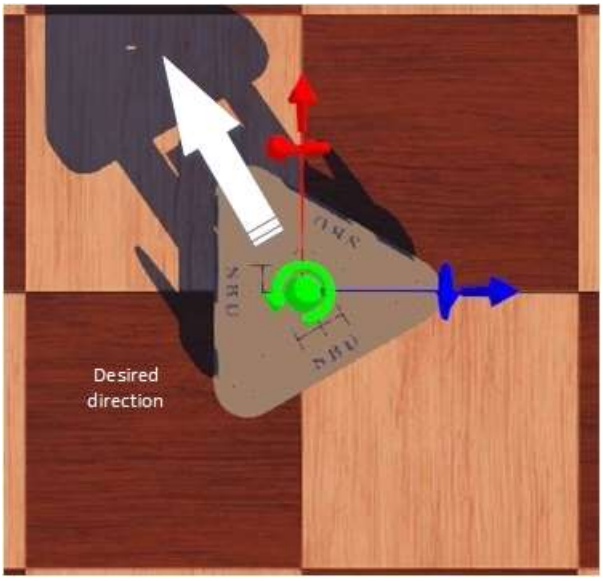

```
desired_angle = math.radians(60)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along wheel 2 normal vector
```

```
robot's last position = (8.00, 13.86)
wheels' rotational velocity:
"wheel 1" = -2.99,       "wheel 2" = -0.00,      "wheel 3" = 2.99.
```



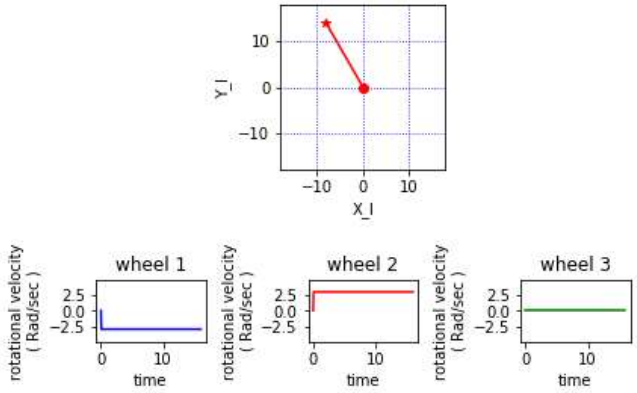


چرخ ۳:

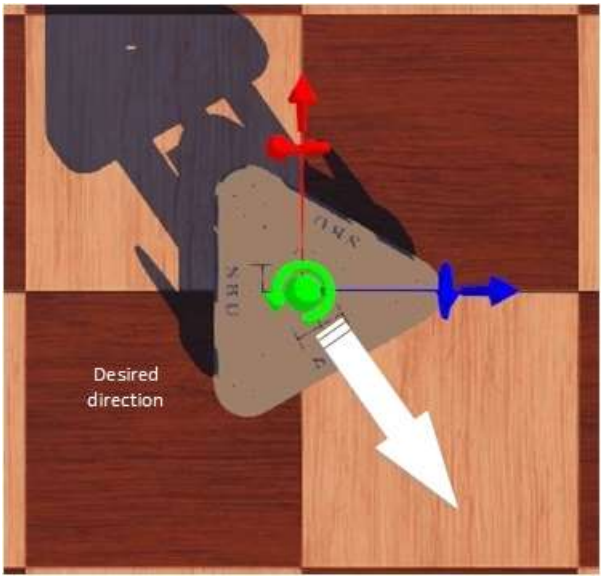



```
desired_angle = math.radians(120)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along wheel 3 normal vector
```

```
robot's last position = (-8.00, 13.86)
wheels' rotational velocity:
"wheel 1" = -2.99,        "wheel 2" = 2.99,        "wheel 3" = -0.00.
```

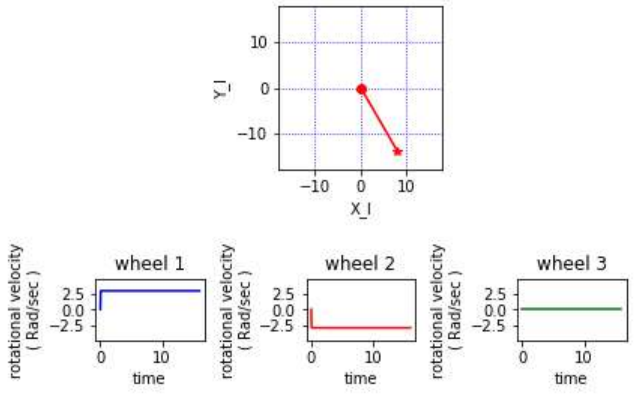


در راستای معکوس محور چرخ ۳،



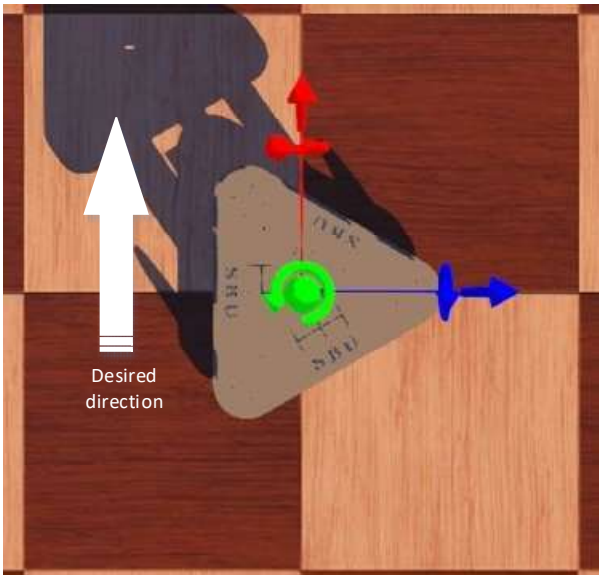

```
desired_angle = math.radians(-60)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along wheel 3 normal vector
```

```
robot's last position = (8.00, -13.86)
wheels' rotational velocity:
"wheel 1" = 2.99,        "wheel 2" = -2.99,      "wheel 3" = 0.00.
```



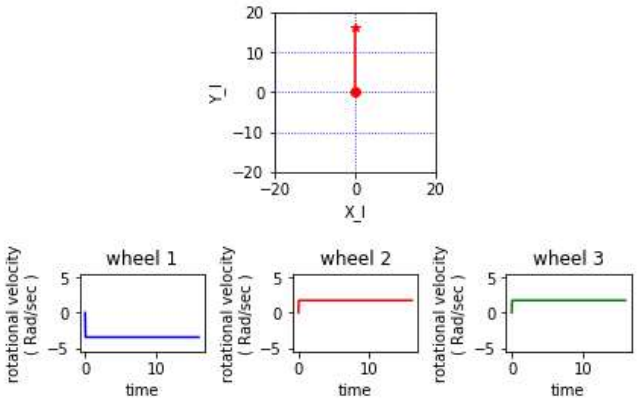
در راستای اضلاع ربات:

ضلع ۲،۳:

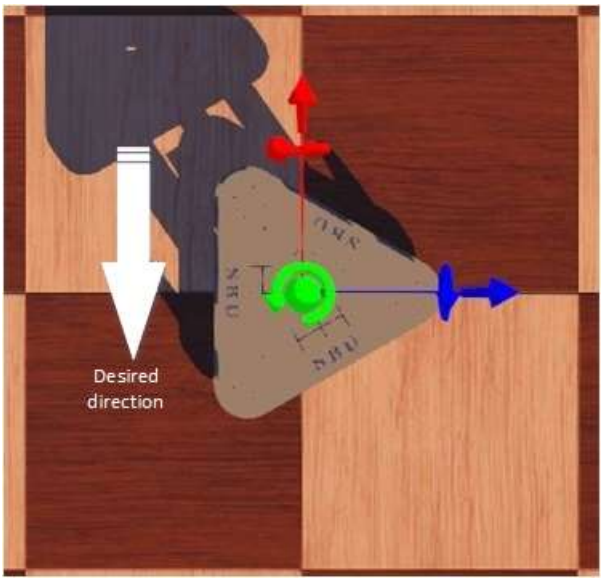



```
desired_angle = math.radians(90)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along +Y_I axis
```

```
robot's last position = (-0.00, 16.00)
wheels' rotational velocity:
"wheel 1" = -3.45,       "wheel 2" = 1.72,       "wheel 3" = 1.72.
```

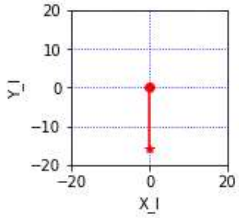


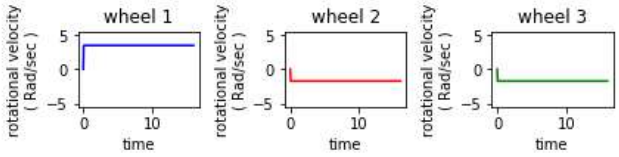
راستای معکوس:



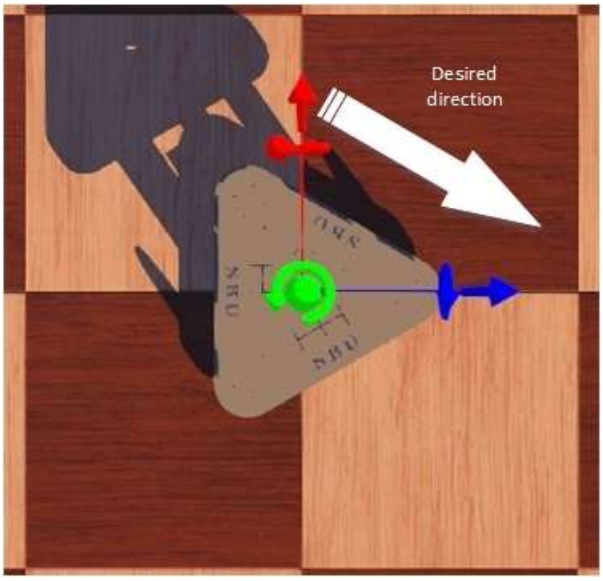

```
desired_angle = math.radians(-90)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along -Y_I axis
```

```
robot's last position = (0.00, -16.00)
wheels' rotational velocity:
"wheel 1" = 3.45,        "wheel 2" = -1.72,      "wheel 3" = -1.72.
```



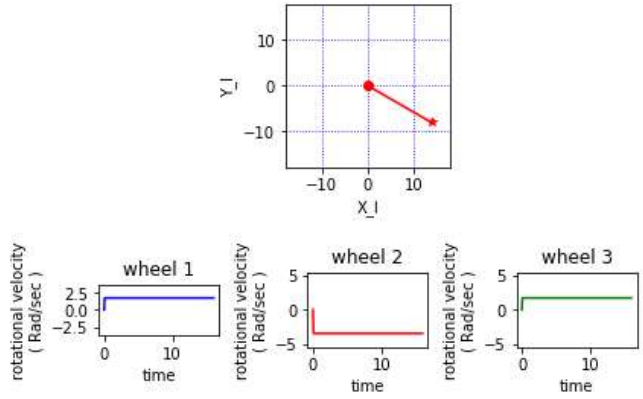


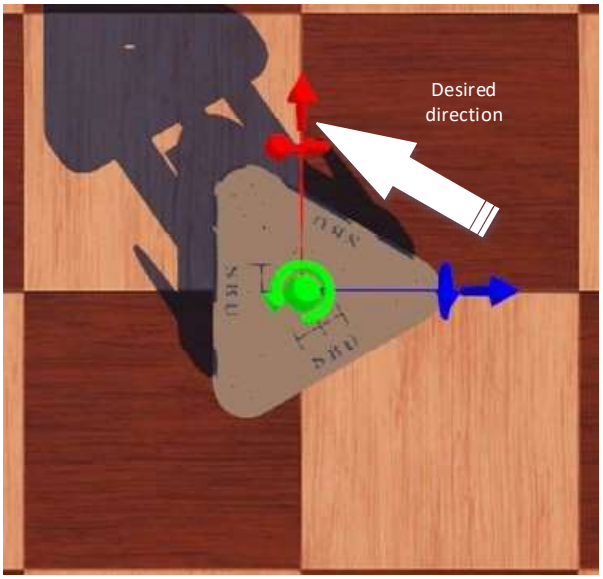
ضلع ۱،۳:




```
desired_angle = math.radians(-30)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along +(1,3) edge.
```

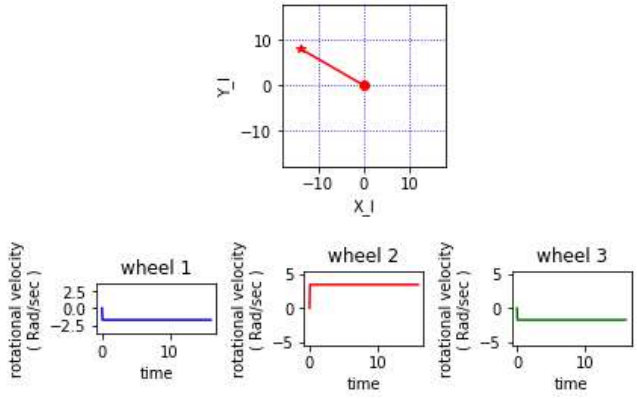
```
robot's last position = (13.86, -8.00)
wheels' rotational velocity:
"wheel 1" = 1.72,        "wheel 2" = -3.45,      "wheel 3" = 1.72.
```



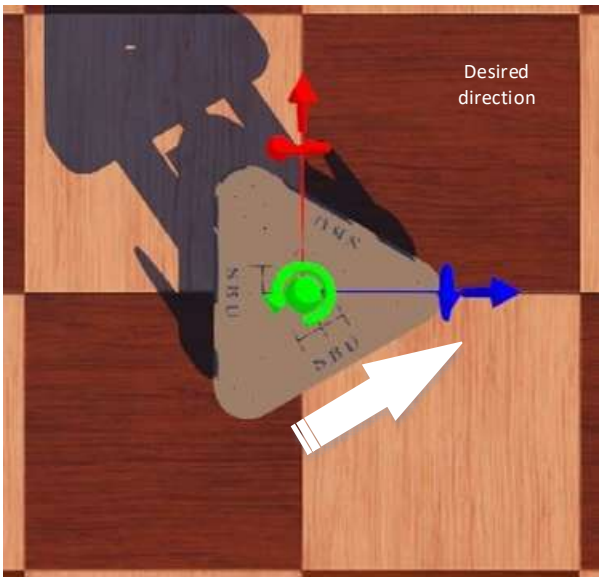



```
desired_angle = math.radians(150)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along -(1,3) edge.
```

```
robot's last position = (-13.86, 8.00)
wheels' rotational velocity:
"wheel 1" = -1.72,       "wheel 2" = 3.45,       "wheel 3" = -1.72.
```

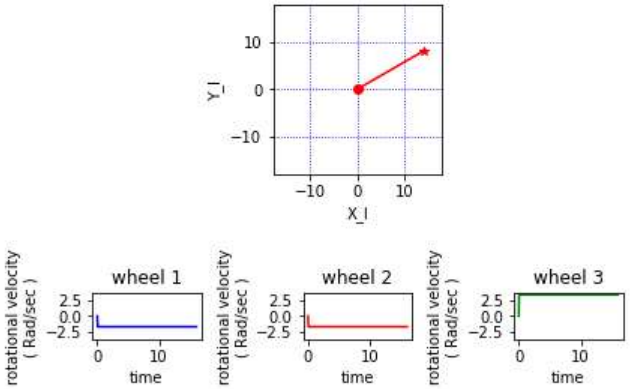


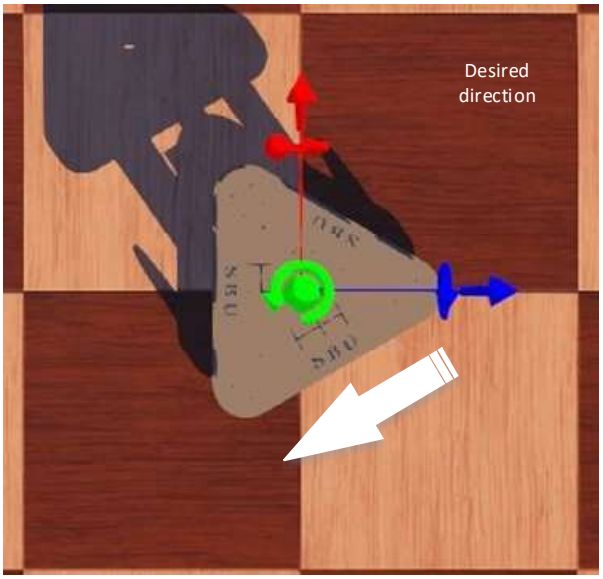
ضلع ۱،۲:




```
desired_angle = math.radians(30)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along +(1,2) edge.
```

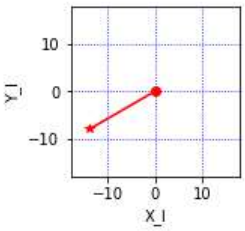
```
robot's last position = (13.86, 8.00)
wheels' rotational velocity:
"wheel 1" = -1.72,       "wheel 2" = -1.72,       "wheel 3" = 3.45.
```






```
desired_angle = math.radians(-150)
model(np.array([math.cos(desired_angle),
                math.sin(desired_angle),
                0.0]))
# movement along -(1,2) edge.
```

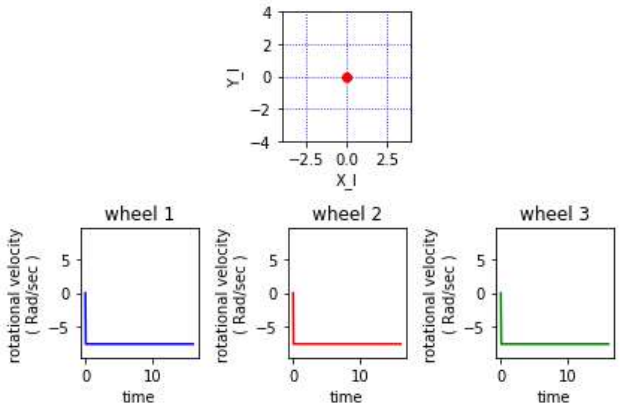
```
robot's last position = (-13.86, -8.00)
wheels' rotational velocity:
"wheel 1" = 1.72,        "wheel 2" = 1.72,        "wheel 3" = -3.45.
```



چرخش حول مرکز ربات در صفحه x , y:

```
model(np.array([0.0, 0.0, 1.0]))
```

```
robot's last position = (-0.00, -0.00)
wheels' rotational velocity:
"wheel 1" = -7.59,        "wheel 2" = -7.59,       "wheel 3" = -7.59.
```

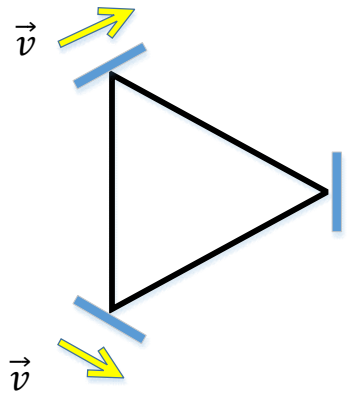


همانطور که در شکل بالا مشهود است، چرخش سر ربات با دادن مقدار سرعت دورانی به رادیان بر ثانیه در مولفه‌ی سوم مدل به عنوان ورودی نتیجه‌ی مقدار چرخ‌های هم اندازه در جهت یکسان را خروجی میدهد، همچنین ربات در سر جای خودش باقی می‌ماند و فقط در مکان اولیه‌ی خود می‌چرخد.

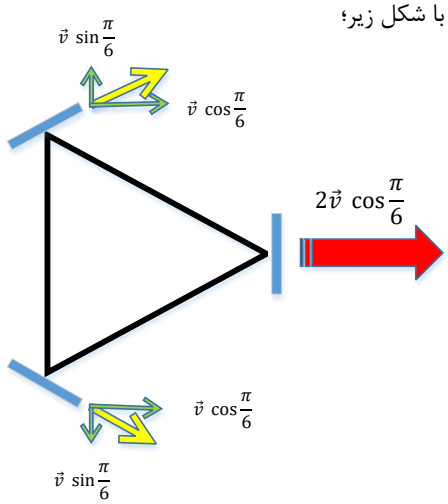
پیوست پ) تحلیل برداری مدل سینماتیک

در پیوست (پ) قصد داریم که تحلیل برداری ساده‌ای با رسم شکل برای توصیف رفتاری سینماتیک ربات و همچنین نیم نگاهی بر تحلیل دینامیکی و چالش‌های احتمالی آن برای مدل سینماتیکی داشته باشیم. تا رفتار حرکتی ربات و سیستم را بهتر متوجه شویم.

حرکت در راستای محور چرخ



همانطور که در شکل نیز مشاهده می‌کنید می‌توان به دو چرخ از سه چرخ ربات، طوری سرعت زاویه‌ای داد که سرعت خطی در راستای چرخ داشته باشند و اندازه‌ی آن‌ها با یکدیگر برابر باشد. بر اساس برآیند برداری سرعت‌های عامل بر ربات سرعتی با اندازه‌ی $2v\cos(30^{\circ})$ در ربات در راستای محور xها به وجود خواهد آمد. مطابق با شکل زیر؛

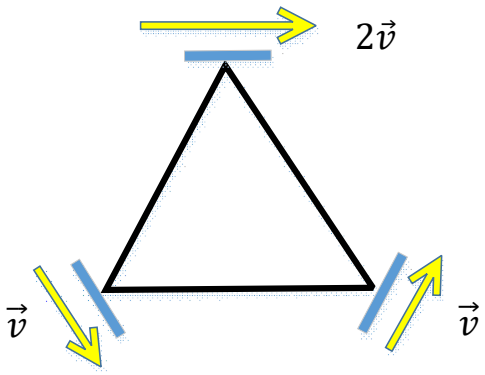


همچنین نمایش برداری تحلیل بالا به صورت زیر خواهد بود؛

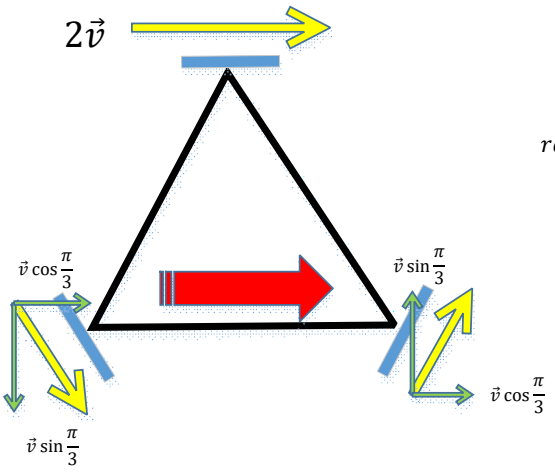
$$robot\ velocity\ = \left(v\sin\frac{\pi}{6} - v\sin\frac{\pi}{6}\right)\vec{j} + \left(v\cos\frac{\pi}{6} + v\cos\frac{\pi}{6}\right)\vec{i}$$

$$= 2\,v\cos\frac{\pi}{6}\,\vec{i}$$

حرکت در راستای ضلع ربات



در این حرکت به دو چرخ از سه چرخ، در یک راستا (مثلاً ساعتگرد) سرعت زاویه‌ای می‌دهیم به طوری که سرعت خطی آن‌ها همانند شکل با یکدیگر برابر باشد. چرخ سوم را نیز با سرعت خطی معادل دو برابر سرعت خطی دو چرخ دیگر قرار می‌دهیم. در این صورت با توجه به شرایط در نظر گرفته شده، سرعت ربات مانند شکل از چپ به راست خواهد بود.



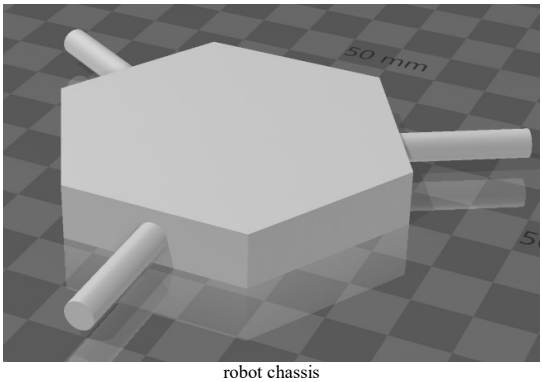
$$robot\ velocity = \left(v\sin\frac{\pi}{3} - v\sin\frac{\pi}{3}\right)\vec{j} + \left(v\cos\frac{\pi}{3} + v\cos\frac{\pi}{3}\right)\vec{i}$$

$$robot\ velocity = 2\,v\cos\frac{\pi}{3}\vec{i}$$

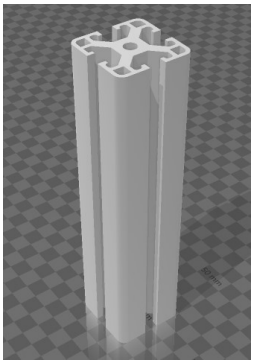
پیوست ت) اجزای مدل سه بعدی ربات

بدنه و شاسی ربات:

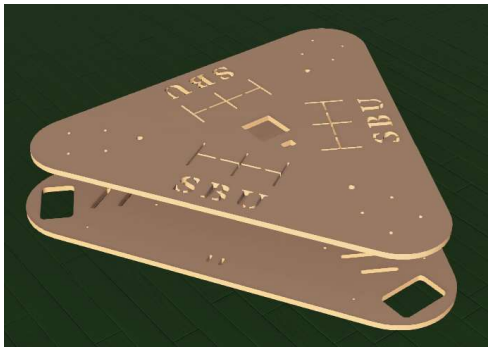
نمای کلی شاسی ربات



robot chassis



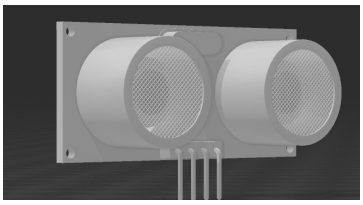
aluminium profile



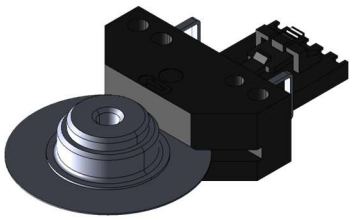
upper and lower plates

cast corner 90 degrees

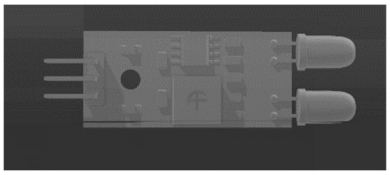
سنسورهای ربات:



- ultrasonic, HC-SR04۴ Figure

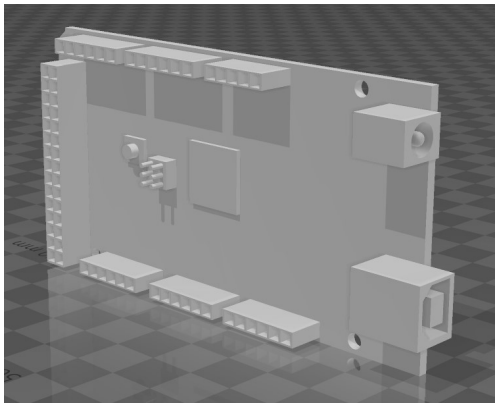


HEDS 9040

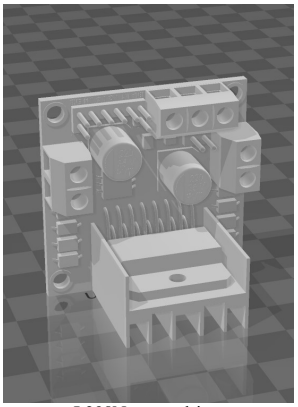


IR سنسور ۵ Figure

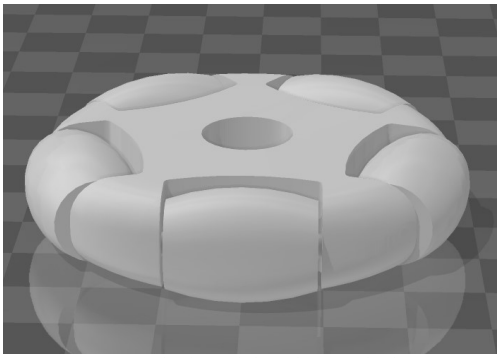
اجزای دیگر:



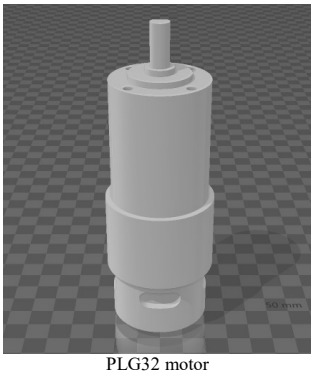
Arduino Mega2560



L298N motor driver



omni-wheel, swedish 90 degrees wheel



PLG32 motor

Tech Support: services@elecfreaks.com

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

Electric Parameter

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

منابع

1. Siegwart, R., Nourbakhsh, I. R. & Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots* (2nd ed.). The MIT Press.

2. Mersi, R. (2020). *Design, Fabrication and Setting up an Omni-Directional Mobile Robot.* Undergraduate dissertation, Shahid Beheshti University, Faculty of Mechanical and Energy Engineering.
3. https://www.arduino.cc/en/reference/board
4. https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/
5. https://www.arduino.cc/reference/en/
6. https://github.com/ph504/ROBOTICS_PROJECT/blob/main/SBU_robot_design/controllers/move_bot_test/move_bot_test.py
7. https://github.com/ph504/ROBOTICS_PROJECT/blob/main/test_modules/kinematics_testing.ipynb
8. https://josm.openstreetmap.de/
9. https://cyberbotics.com/doc/automobile/openstreetmap-importer
10. https://github.com/SteveMacenski/slam_toolbox
11. Pitt, M. K., Shephard, N. (1999). Filtering via simulation: auxiliary particle filters. *J. Amer. Statist. Assoc.* 94, no. 446, 590–599.
12. https://www.mathworks.com/videos/series/autonomous-navigation.html
13. https://www.mathworks.com/videos/autonomous-navigation-part-3-understanding-slam-using-pose-graph-optimization-1594984678407.html
14. https://www.mathworks.com/videos/autonomous-navigation-part-4-path-planning-with-a-and-rrt-1594987710455.html
15. https://github.com/ph504/webots-ROS-installation_man/blob/main/README.md
16. https://cyberbotics.com/doc/reference/distancesensor#sonar-sensor
17. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Turtlesim/Introducing-Turtlesim.html
18. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Understanding-ROS2-Nodes.html
19. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Topics/Understanding-ROS2-Topics.html
20. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Services/Understanding-ROS2-Services.html
21. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Parameters/Understanding-ROS2-Parameters.html
22. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Understanding-ROS2-Actions.html
23. https://github.com/cyberbotics/webots_ros2/wiki/Tutorial-E-puck-for-ROS2-Beginners
24. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Workspace/Creating-A-Workspace.html
25. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Creating-Your-First-ROS2-Package.html
26. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Writing-A-Simple-Py-Publisher-And-Subscriber.html

27. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Writing-A-Simple-Py-Service-And-Client.html
28. http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Custom-ROS2-Interfaces.html