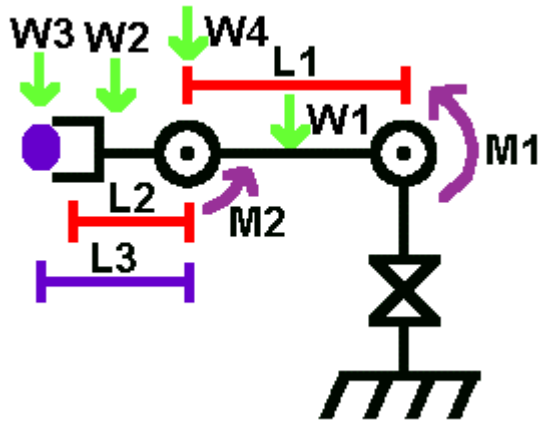


# آموزشی بازوی رباتیک

## محاسبات نیروی مفاصل :

مقصود از محاسبات نیرو و انتخاب موتور ربات است . شما باید اطمینان پیدا کنید موتور انتخابی شما تنها وزن ربات را تحمل نکند بلکه وزن آنچه ربات حمل می کند را نیز تحمل کند . اولین قدم در این کار رسم نمودار جسم آزاد بازوی ربات هست که برای این کار باید بازو را تا بیشترین طول ممکن بازو بکشید .



این پارامتر ها را انتخاب کنید :

1. وزن هر لینک

2. وزن هر مفصل

3. طول هر لینک

4. وزن جسمی که حمل می کند .

بعد از محاسبه بازوی گشتاور ، نیروی رو به پایین وارده بر هر لینک را در طول آن لینک ضرب کنید .

این محاسبات را باید برای هر عملگر بالابری انجام دهید . در این طرح ما فقط دو درجه آزادی داریم و مرکز جرم لینک ها در نصف طول هر لینک قرار دارد .

گشتاور حول مفصل 1 (Joint 1) :

$$M1 = L1/2 * W1 + L1 * W4 + (L1 + L2/2) * W2 + (L1 + L3) * W3$$

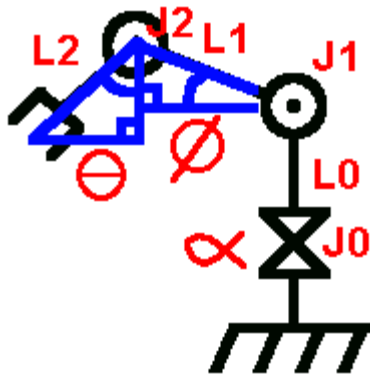
گشتاور حول مفصل 2 (Joint 2) :

$$M2 = L2/2 * W2 + L3 * W3$$

همین طور که می بینید ، با اضافه شدن درجات آزادی تعداد روابط ریاضی نیز بیشتر و پیچیده تر می شود و وزن مفاصل افزایش می یابد . در ضمن مشاهده می شود که شرط لازم برای تولید گشتاور کوچکتر طول بازوی کوچکتر است .

## : Forward Kinematic

این روش برای تعیین موقعیت و جهت عملگر نهایی بازوی رباتیک با داشتن طول لینکها و زاویه آنها است . شما برای این محاسبات نیاز به اطلاعات پایه در جبر و مثلثات هستید . برای ربات نمونه ما ، موقعیت عملگر نهایی را با داشتن زاویه و طول لینکها محاسبه می کنیم . برای تجسم بهتر ما زوایا را با مثلث های آبی مشخص کردیم :



فرض می کنیم که کفه ربات روی موقعیت  $y=0, x=0$  قرار دارد . گام بعدی

تعیین  $x, y$  هر مفصل است .

مفصل 0 (Joint 0) :

$$x_0 = 0$$

$$y_0 = L_0$$

مفصل 1 (Joint 1) :

$$\cos(\psi) = x_1/L_1 \Rightarrow x_1 = L_1 \cdot \cos(\psi)$$

$$\sin(\psi) = y_1/L_1 \Rightarrow y_1 = L_1 \cdot \sin(\psi)$$

مفصل 2 (Joint 2) :

$$\sin(\theta) = x_2/L_2 \Rightarrow x_2 = L_2 \cdot \sin(\theta)$$

$$\cos(\theta) = y_2/L_2 \Rightarrow y_2 = L_2 \cdot \cos(\theta)$$

موقعیت عملگر نهایی (End Effector) :

$$x_0 + x_1 + x_2, \text{ or } 0 + L_1 \cdot \cos(\psi) + L_2 \cdot \sin(\theta)$$

$$y_0 + y_1 + y_2, \text{ or } L_0 + L_1 \cdot \sin(\psi) + L_2 \cdot \cos(\theta)$$

$$z \text{ equals } \alpha, \text{ in cylindrical coordinates}$$

در این طرح زاویه عملگر نهایی برابر با مجموع دو زاویه  $\theta$  و  $\psi$  است .

### Inverse Kinematics

این روش برعکس روش قبلی است . وقتی موقعیت خاصی را برای عملگر نهایی در نظر دارید برای رسیدن به آن باید زاویه مورد نیاز مفصل را بدانید . ربات مثل بچه گریه ای می ماند که می خواهد به آن چنگ بزند ؛ هر مفصل چه زاویه ای باید به خود بگیرد تا به آن برسد ؟ اگر چه این روش از روش قبلی سودمند تر است ولی پیچیده تر هم می شود . برای همین این معادلات را روی ربات قبلی طرح نمی کنیم و از یک ربات با پیکربندی دیگر استفاده می کنیم .

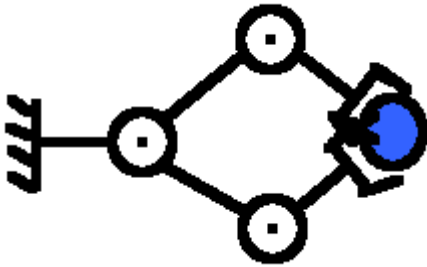
$$\psi = \arccos((x^2 + y^2 - L_1^2 - L_2^2) / (2 * L_1 * L_2))$$

$$\theta = \arcsin((y * (L_1 + L_2 * c_2) - x * L_2 * s_2) / (x^2 + y^2))$$

$$c_2 = (x^2 + y^2 - L_1^2 - L_2^2) / (2 * L_1 * L_2);$$

$$s_2 = \sqrt{1 - c_2^2};$$

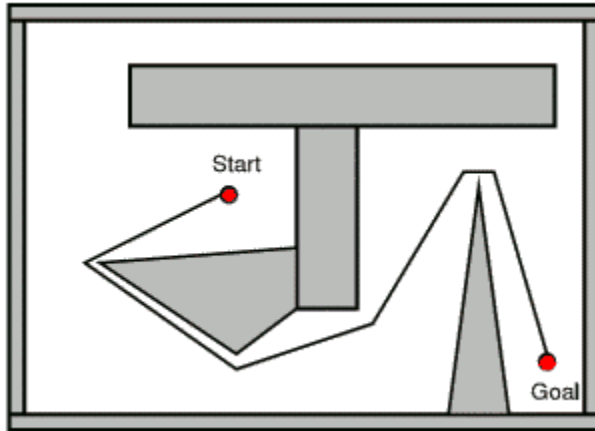
اما چه چیزی باعث سختی این روش تحلیل سینماتیکی می گردد ؟ وجود معادلات غیر خطی هم زمان دلیلی بر این امر است .



ابتدا راه حل های زیادی اعم از تعریف نشده هم وجود دارد همان گونه که در شکل می بینید ؛ بازوی شما کدام یک از حالات را با توجه به گشتاور ، موقعیت قبلی بازو ، زاویه چنگک و ... بهینه می داند ؟

احتمال راه حل های صفر هم موجود است . مثلاً ممکن است موقعیت بیرون از فضای کاری باشد یا نقطه ای درون فضای کار باشد که قرار است توسط چنگک گرفته شود ولی زاویه دست نایافتهی دارد .

نقطه تکین ، یک مکان در شتاب بی نهایت ، می تواند معادلات را به هم بزند یا موتور را در یک حالت باقی بگذارد (یعنی موتور هیچگاه به شتاب بی نهایت نرسد) . در آخر معادلات نمایی برای همیشه در میکروکنترلر محاسبه می شوند . این معادلات پیشرفته هیچ نقطه ای را نمی توانند روی پردازشگر به خوبی نگه دارند .



### نقشه حرکت :

نقشه حرکت یک بازوی رباتیک بدون اغراق واقعاً پیچیده است ، بنابراین ما فقط به بررسی مقدمات آن می پردازیم .

تصور کنید بازوی ربات شما تعدادی شیء در فضای کاری خود دارد ، بازو در فضای کاری چطور حرکت کند که به نقطه مورد نظرش برسد ؟ برای انجام این کار ، فرض کنید که بازوی شما دقیقاً شبیه به ربات متحرکی است که در فضای سه بعدی کنترل می شود .

عملگر نهایی فضای را درست شبیه ربات متحرک می پیماید ،

در عین حال باید مطمئن باشد که مفاصل و لینکها به چیزی برخورد نکنند . انجام این کار خیلی دشوار است .

اگر شما بخواهید با عملگر نهایی ربات خود یک خط مستقیم بکشید چه می کنید ؟

اگر این را این گونه ببینید که از نقطه A روی یک خط راست مستقیم به نقطه B بروید نسبتاً ساده به نظر می رسد . با استفاده

از روش اول یعنی Inverse Kinematics نقاط زیادی بین دو نقطه A و B وجود دارد که ربات از آنها بگذرد . نتیجه حرکت

نهایی یک خط مستقیم ملایم در می آید . شما نمی توانید فقط از این روش برای خطوط مستقیم استفاده کنید و نیز خطوط منحنی .

بازوهای رباتیک پیشرفته و گران قیمت طوری برنامه ریزی شدند که تمام نیازهای شما را جوابگو باشند اینکه چطور فاصله بین

دو نقطه را پیمایش کنند ( روی خط مستقیم ، تا حد ممکن سریع و ..) . برای مطالعه بیشتر شما می توانید از الگوریتم موجی

برای برنامه ریزی مسیر بین دو نقطه کمک بگیرید

الگوریتم موجی = Wave front Algorithm :

از این الگوریتم برای تعیین کوتاه ترین فاصله بین دو نقطه استفاده می شود . در این روش یک چارت داریم که شامل سلولهایی است که کل فضا را پوشش می دهند ، نقطه شروع با 0 موانع با 1 و هدف با 2 شماره گذاری می گردد . از نقطه هدف شروع کرده

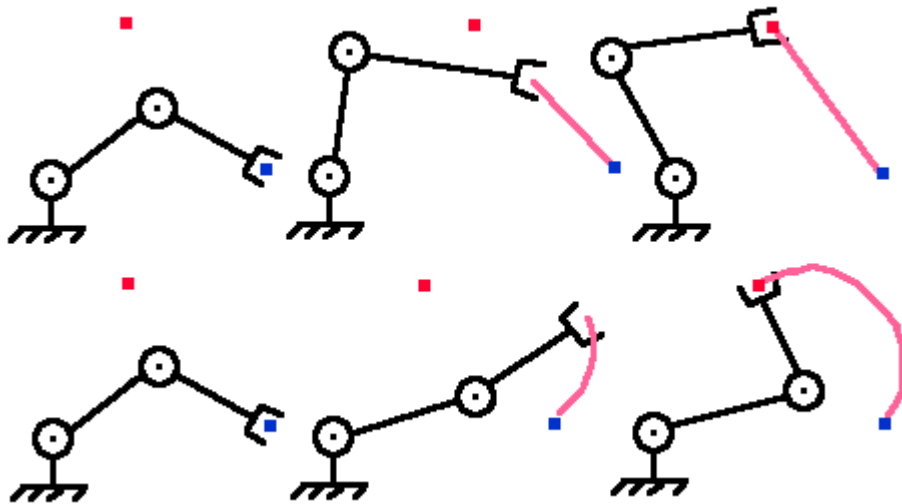
## سرعت و نقشه راه :

محاسبه سرعت عملگر نهایی از لحاظ ریاضی پیچیده است . به همین منظور ما تنها به اصول آن اشاره می کنیم . ساده ترین راه برای این کار این است که فرض کنید بازوی رباتیک شما یک چرخ دوار به قطر  $L$  است . مفصل با سرعت  $Y$  rpm می چرخد ، بنابراین سرعت به این شکل محاسبه می گردد :

$$2 * \pi * \text{radius} * \text{rpm}$$

اگرچه عملگر نهایی تنها حول پایه نمی چرخد ولی می تواند در مسریهای مختلفی حرکت کند . عملگر نهایی می تواند یک خط مستقیم یا منحنی و ... را دنبال کند .

برای یک بازوی رباتیک سریع ترین راه بین دو نقطه اغلب یک خط مستقیم نیست . اگر دو مفصل دارای موتورهای مختلفی باشند یا بارهای متفاوتی را حمل کنند می توانند سرعت های مختلفی نیز داشته باشند . زمانیکه شما به بازوی ربات می گوئید از یک نقطه به نقطه ای دیگر برود شما دو انتخاب دارید . یکی اینکه مسیری مستقیم بین دو نقطه را طی کند یا به تمام مفصل بگوئید با بیشترین سرعت ممکن حرکت کنند ، با چشم پوشی از نوسانات و تاب خوردنهای خودسرانه و ناگهانی عملگر نهایی بین دو نقطه . در تصاویر زیر عملگر نهایی بازوی رباتیک در حال حرکت از نقطه آبی به نقطه قرمز است :



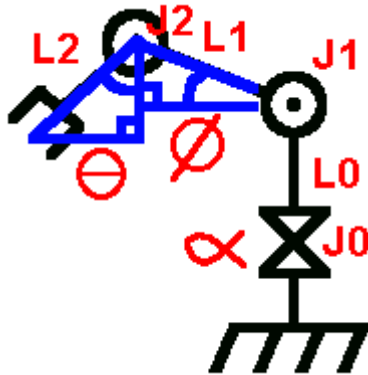
در قسمت بالای این مثال عملگر نهایی خطی مستقیم را پیمایش می کند . این تنها حرکت ممکن است که این بازو برای طی یک خط راست می تواند انجام دهد . در بخش زیرین به ربات گفته شده با بیشترین سرعت ممکن به نقطه قرمز برسد . در این حالت بازو به مفصل اجازه می دهد که با بیشترین سرعت بچرخد ، مسیریهای ممکن زیادی وجود دارد .

و مقادیر سلولها را یک واحد افزایش می دهند . به همین ترتیب برای سلولهای بعدی . الگوریتم انقدر ادامه می یابد که هیچ سلولی که همسایه بزرگتر از 2 دارد باقی نماند . مگر سلولهایی که قابل دسترس نباشند . برای یافتن کوتاه ترین فاصله مسیری را انتخاب می کنیم که با حرکت از سمت میدا به سمت هدف مقادیر عددی سلولها کاهش یابد .

**کدام روش بهتر است ؟** عوامل زیادی در تصمیم گیری دخیل اند . اغلب شما زمانی از خطوط مستقیم استفاده می کنید که شی ای که ربات آن را حرکت می دهد خیلی سنگین است و اندازه حرکت برای جابجایی باید تغییر کند .

اما برای سرعتهای بیشینه (مثلاً زمانی که بازو چیزی را حمل نمی کند یا فقط اجسام سبک را جابجا می کند ) شما به سرعت زیاد مفاصل نیاز دارید .

حالا فرض کنید بازوی ربات شما می خواهد یک سرعت چرخشی خاص بگیرد هر مفصل به چه میزان گشتاور نیاز دارد ؟ اول به نمودار جسم آزاد خود برگردیم :



حال فرض کنید مفصل J0 می خواهد ظرف 2 ثانیه 180 درجه بچرخد ، موتور J0

چه گشتاوری نیاز دارد ؟ خب ، مفصل J0 تحت تاثیر جاذبه نیست و ما به گشتاور

لختی آن نیاز داریم که از معادله زیر به دست می آید :

$$\text{torque} = \text{moment\_of\_inertia} * \text{angular\_acceleration}$$

این معادله را به اجزاء تشکیل دهنده تجزیه می کنیم :

$$\text{torque} = (\text{mass} * \text{distance}^2) * (\text{change\_in\_angular\_velocity} / \text{change\_in\_time})$$

و باز هم جزئی تر :

$$\text{change\_in\_angular\_velocity} = (\text{angular\_velocity1}) - (\text{angular\_velocity0})$$

$$\text{angular\_velocity} = \text{change\_in\_angle} / \text{change\_in\_time}$$

حال فرض می کنیم در لحظه شروع 0 سرعت زاویه ای اولیه نیز صفر است :

$$\text{torque} = (\text{mass} * \text{distance}^2) * (\text{angular\_velocity} / \text{change\_in\_time})$$

که در این فرمول فاصله ، فاصله بین محور چرخش تا مرکز جرم بازو در نظر گرفته می شود :

$$\text{center of mass of the arm} = \text{distance} = 1/2 * (\text{arm\_length})$$

ولی شما باید مرکز جرم جسمی که بازو می خواهد آن را نگاه دارد را نیز محاسبه کنید :

$$\text{center of mass of the object} = \text{distance} = \text{arm\_length}$$

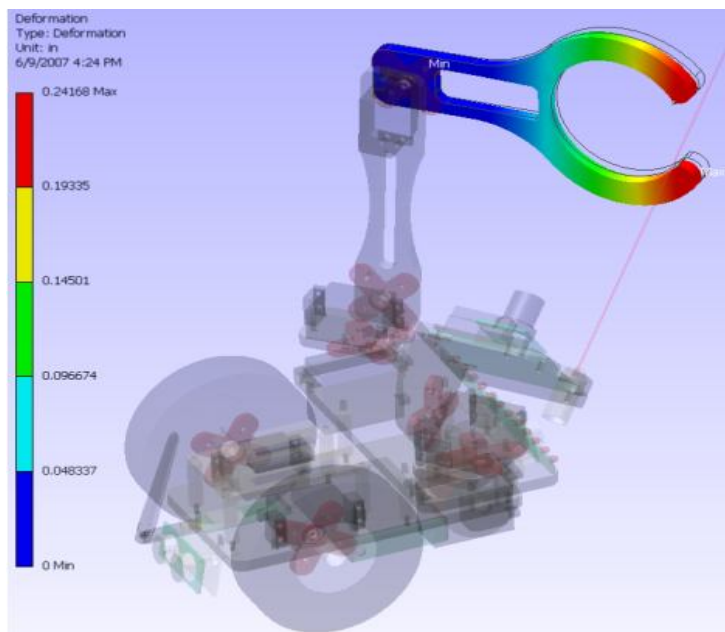
و بعد گشتاور را برای هر دو ، بازو و جسم ، حساب کرده و مجموع آنها نیز به دست آورید :

$$\text{torque}(\text{of\_object}) + \text{torque}(\text{of\_arm}) = \text{torque}(\text{for\_motor})$$

و البته اگر بازو تحت تاثیر نیروی جاذبه باشد باید گشتاور لازم برای بلند کردن بازو نیز محاسبه شود تا به سرعت دلخواه خود برسد .

### تاب خوردگی بازو :

تاب خوردگی بازو یک معضل در نتیجه طراحی بد بازوهای رباتیک است . این مشکل مربوط به وقتی است که بازو بیش از حد دراز یا سنگین باشد در این صورت موقع کشیده شدن دچار خمش می شود . موقع طراحی بازو مطمئن شوید که بازوی شما سبک و مستحکم است . یک تحلیل المان محدود برای ربات خود انجام دهید و خمش ، پیچش و تنش را محاسبه کنید .



سنگین ترین قطعات ربات را نگاه دارید تا پایه بازو را ببندید . این برای مفصل میانی ربات ایده خوبیست تا مانند کمر بند یا زنجیری شود که توسط موتوری که در پایه ربات هست کنترل شود . مشکل تاب خوردگی گاهی تا حدی وخیم است که در حرکت از نقطه شروع تا انتها ربات تلوتلو می خورد . راه حل این مشکل استفاده از کنترلر PID است که تا قبل از توقف کامل ربات حرکت را آهسته تر کند .

### سنسورها :

برخی بازوها فقط سنسورهای داخلی دارند مثل انکدرها ، ولی دلایل زیادی برای استفاده از سنسورهای بیشتری نیز هست مثل : سنسورهای تماسی ، تصویری ، لمسی و ... .

یک بازوی ربات بدون سنسورهای تصویری مثل هنرمند نقاشی است که چشمهایش را بسته است . با استفاده از الگوریتمهای بینایی فیدبک دار ساده ربات خودش می تواند بدون کمک موقعیتهای برنامه ریزی شده از یک نقطه به نقطه دیگر برود .

ترجمه : زهره دارابیان

منبع : [www.societyofrobots.com](http://www.societyofrobots.com)

سایت مرجع این مطلب : <http://nrec.ir>

کمیته ملی مهندسی رباتیک

واژگان :

Forward Kinematics = سینماتیک مستقیم

Inverse Kinematics = سینماتیک معکوس

در ترجمه این متن از فایل آپلود شده قبلی در همین سایت نیز استفاده شد .