Resume

Indholdsfortegnelse

[Indledning 4](#_Toc122096361)

[Redegørelse for lineære transformationer med matricer 4](#_Toc122096362)

[Matricer 4](#_Toc122096363)

[Matrix multiplikation 4](#_Toc122096364)

[Identitets matrix 5](#_Toc122096365)

[Transpose 5](#_Toc122096366)

[Transformationer 5](#_Toc122096367)

[Rotation 6](#_Toc122096368)

[Translation 6](#_Toc122096369)

[Homogene transformation 7](#_Toc122096370)

[Eksempel på homogentransformations matrix 8](#_Toc122096371)

[Beskrivelse af den kinematiske struktur i en Dobot 8](#_Toc122096372)

[Forward kinematik 8](#_Toc122096373)

[Inverse kinematik 11](#_Toc122096374)

[Analyse af Python programmer 16](#_Toc122096375)

[Analyse af Python bibliotek til regning med matricer 16](#_Toc122096376)

[Analyse af program til styring af dobot 19](#_Toc122096377)

[Hvordan er den matematiske viden nødvendig for programmør? 21](#_Toc122096378)

[Konklusion 21](#_Toc122096379)

[Referencer 22](#_Toc122096380)

[Bilag 23](#_Toc122096381)

[Kode fra matrix.py 23](#_Toc122096382)

[Kode fra dobot\_controller.py 24](#_Toc122096383)

# Indledning

# Redegørelse for lineære transformationer med matricer

## Matricer

En vektor er en list af tal som for det meste repræsenter et punkt i et rummet, for eksempel repræsenter

en vektor i rummet det vil sige at den har tre dimension, og mængden af denne vektor er , og hvis den så havde n tal i vektoren ville den til høre denne mængde .

En matrix er så en samling af vektor, som står ved siden af hinanden. De vil sige at den består af en mængde tal de er orginaseteret i rækker og søjler.

Mængden af n x m matricer betegnes med

## Matrix multiplikation

Nu skal der defineres hvordan man ganger to matricer sammen. For at kunne gange to matricer sammen skal den første matrix have det samme antal kolonner som den anden matrix har rækker, ellers er det ikke defineret. For det gør det muligt at tage prik produktet mellem hver række i den første matrix og hver kolonne i den anden matrix, også sættes det givne produkt ind på række nummeret fra det første matrix og kolonne nummeret fra den andet matrix. På Figur 1 ses der et eksempel hvor det illustreret hvilke vektorer der skal prikkes sammen.

Et billede, der indeholder tekst, enhed, meter, måler

Automatisk genereret beskrivelse

Figur 1 Ilustration af matrice multiplikation, billede fra (Pierce, 25)

Dette betyder at hvis og og C er givet ved er .

Sporligt betyder det at det resulterende matrix ved multiplikation mellem to matricer giver en matrix med samme antal rækker som det første matrix og det samme antal kolonner som det andet matrix.

Her er et eksempel på matrix multiplikation

## Identitets matrix

Der findes identitet matricer, og de har den samme effekt som når man ganger med et ettal, i de reelle tal, det vil sige at når man gange med den giver de sig selv tilbage. Identitets matricer er kvadratiske, og har ettaller på diagonalen resten er 0. Det kaldes I. et eksempel på en identitets matrix

De kan komme i alle størrelser så længde de er kvadratiske.

At gange med en identitets matrix giver altid den oprindelige matrix, lad og hver et identitet matrix i den rigtige størrelse

Her ses at når mange gange med en identitets matrix for man de oprindelige matricer bage efter.

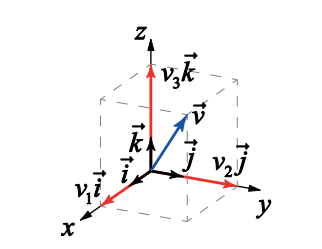
Dette kan også vises med tal eksempel

Det er vigtigt at identitetsmatricen har den rigtige størrelse så det er muligt at finde matrix-matrixproduktet.

## Transpose

## Transformationer

For at lave bruge matricer til at lave på punkter. Bruges et matrix til at beskrive de nye basis vektorer udfra de gamle de gamle basis vektor. En vektor kan beskrives udfra basisvektorene der har en længde på 1 og står ortogonalt på hinanden. I rummet kaldes de og . Og har størrelserne og . Og en vektor i rummet kan beskrives på denne måde

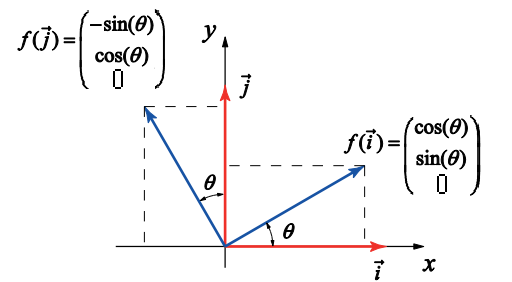
Så hvis man vil transformer et punkt til et andet punkt kan man nøjes med ændre beskrive de nye basis vektor. De kan bare ændre retning og skalars hvis dette findes nødvendigt.

Figur Illustration af basis vektor figur fra (Undervisningsministeriet, 2014)

På måde kan man lave en linær afbilding af et punkt ved hjælp af en matrix. Så i rummet ser sådan en af billedning matrix sådan her ud

Hvor er en linær funktion.

### Rotation

På denne måde kan rotations beskrives ved hjælp af en matrix, for eksempel her en rotation om z-aksen

Figur Ilustration af rotation om z aksen figur fra (Undervisningsministeriet, 2014)

Det ses at ikke ændre sig da vi roteter om z aksen. Se Figur 3. På denne måde kan man beskrive rotation om envære akse. Senere i opgaven skal bruges en matrix som først rotere om x akse, så y aksen og til sidst z aksen. Dette findes ved at gange være enkelte af disse rotationer sammen.

### Translation

For at kunne lave translationer er det ikke nødvendigt med matricer. Da en translation er bare at flytte en vektor med en anden vektor så derfor kan lægges to vektor sammen for at lave en translation. Så hvis punkt skal flyttes med en vektor lægges disse sammen for at få det nye punkt.

## Homogene transformation

For at forsimple operationer som både har en rotation og en translation, kan disse sættes sammen til et enkelt 4 gange 4 matrix. På denne:

Hvor den øverste venstre del er et 3 gange 3 rotation matrix, og kolonner 4 er translation, for at kunne gange punkter med denne matrix så tilføjes der et ettal i slutning så de har en størrelse der passer med at gange med et 4 gange 4 matrix, og dette ettal ændre ikke på resultat et, så det skal bare ses bordt fra når reslutat aflæses. Først sker en rotation så sker translation, men i en og samme operation.

For at forstå det se først hvordan en translation fungere. Et transformations matrice der kun laver en translation, ser så da herud:

Og en transformation med

Når der uden lukkende ses på en translation, er det tydeligt at denne måde at gange dem sammen på giver det samme som at lægge to vektor normalt sammen.

På samme måde kan rotation undersøges. En transformations matrix der kun laver en rotation om z aksen, ser så da herud:

Og en transformation med

Det kan samlingens med hvis der bare ganges med det rotations matrix der tidligere blev fundet

Så ses det at det giver det samme, der for kan en homogen transformation beskrive på ovenstående måde.

### Eksempel på homogentransformations matrix

Figur Eksempel homogentransformations

Her er et eksempel en transformation med en transformation matrice

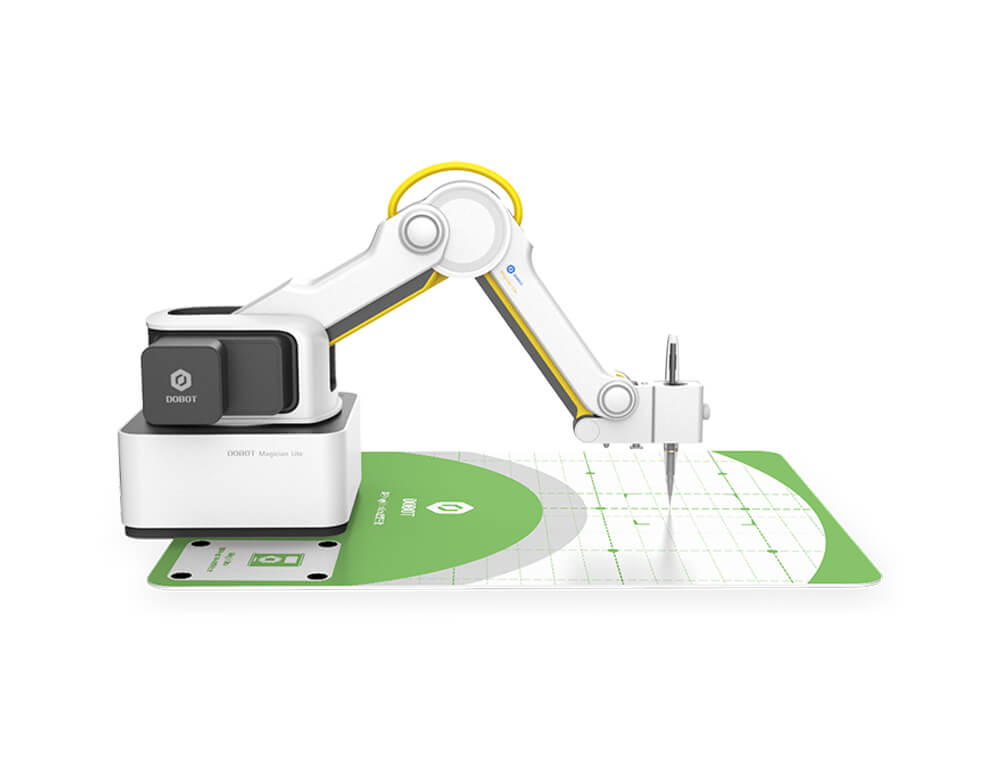
Dette transformation matrice forskyder et punkt med 5 på x aksen og 2 på y aksen, og den forskydes ikke på z aksen, og den roterer 30 grader om z, aksen. Punktet som der transformeres er  det sidste 1 tal til føjes for at de for de rigtige størrelser og det resulterer punkt kalder vi .

Dette eksempel kan ses på Figur 4, P1 er blå, roteret kaldes og er sort, forskyndnings vektoren kaldes Q er grøn, den resultere punkt betegnes og er rød.

Når der arbejdes men transformation er på robotter placer der frames, et andet ord for kordiant system, på alle omdrejnings punkter på robot. Hvert af disse frames postion beskrives ud fra den forige, ved hjælp af en transformations matrix. På denne måde bruges det til at skifte hvilket kordiant system. Så en frame B er beskrevet ud fra frame A, skrives sådan her , så det der står øverst viser hvilket kordnat system den kommer fra, og er det resutlerne kordinat system

# Beskrivelse af den kinematiske struktur i en Dobot

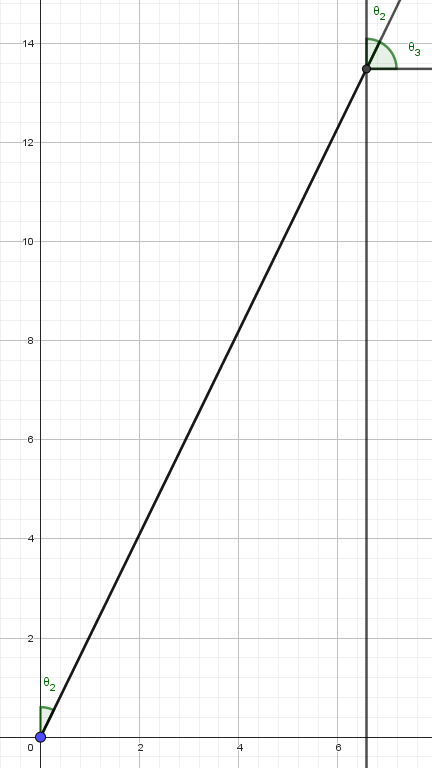
## Forward kinematik

Når den kinematikse struktur af en robot skal beskrive, skal leddenes indbyrdes potion beskrives. En Dobot som ses på Figur 4 har 4 led der er drevet og 2 der mekanisk, som mekanisk holder en vinkel. For at beskrive robotten bruges Denavit-Hartenberg notation, som beskriver to frames indbyrdes postions og rotation. Til at beskrive den næste frame bruges dens for hold til den forrige frame. Så der startes med en stationer frame 0, denne frame står stille i bunden af robotten og den flytter sig aldrig, efter denne frame tælles der op ad så frame 1, 2 og 3 ind til alle led er beskrevet. I starten og slutning kan det være nødevendigt med en en forskydning for der hvor ledene starter ikker i bunden af robotten og eller at tool i robot armens ende punkt ikke er lige i slutning af det sidste led. Denavit-Hartenberg notation beskrives hvert led med 4 parameter . Først roteter man med grader rundt om den er oprindelige x akse, så forskydes med langs den samme x akse, nu roteres denne nye frame med og den nye z akse, det efter en forskydning med hen langs z aksen. Normalt bruges disse to sidste parametere de er led variable, det vil sige at det dem der ændres på når robotten ændrer postion, hvis ledet er et rotations led bruges led variablen til at beksrive rotation, sættes til nul og bliver ikke brugt. Det er omvendt ved led som bevæger sig med linær forskydning. På denne måde kan to frames indbyrdes placering beskrives.

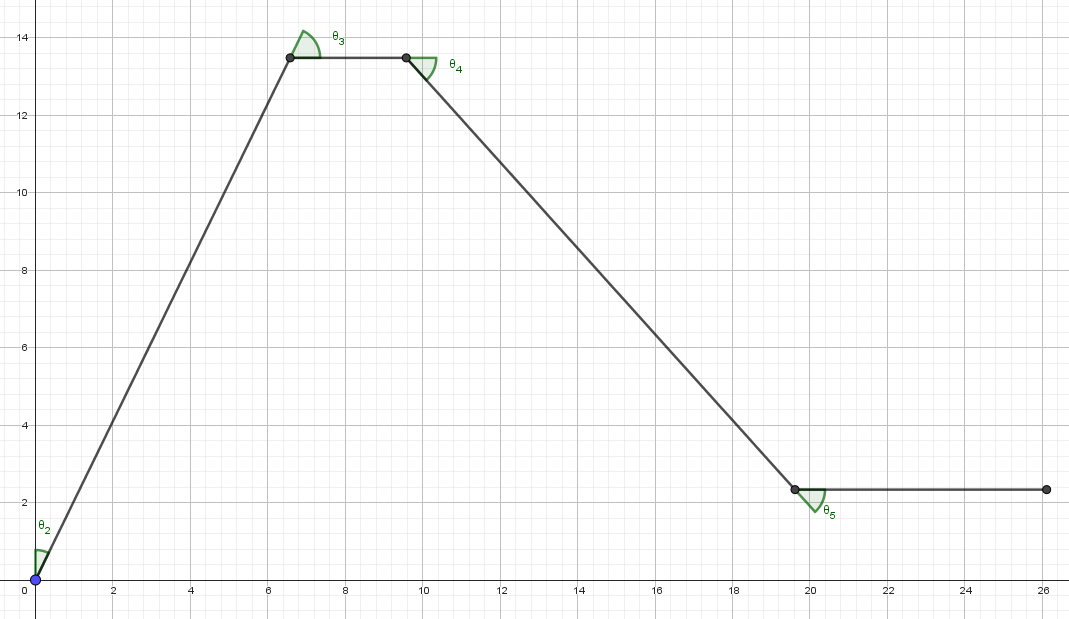
Figur 6 Dobot Magician Lite, billed fra (STEM EDUCATION WORKS, n.d.)

Denne proces består af en rotation om den opridnelige x akse så en translation langs x aksen og så ny rotation om den nye z akse og til sidst en translation langs den nye z akse

Og når disse matricer ganges sammen, får vi en transformation fra den tidligere frame til det nye matrixe i et komplet matrix. Denne transformation kan beskrives med denne matrix

På denne måde kan der laves en forward kinematik model for en robot hvis man kender alle Denavit-hartenberg paramenterne, ved at gange hvert matrix med det forige fås den komplete forward kinematik model. For at beskrive dobotten, startes der med at placeres frame 0 det vil sige den der ikke rykker sig. Jeg placer frame 0 inde i robbotten med postiv lige ud af robbotten, og postiv y lige ud mod venstre, og dermed er z aksen postiv op, og den placere i en højde så det ligger på højde med det første leds rotation punkt. På denne måde kan den første frame bare beskrives med som beskriver rotation af hele robotten om z-aksen. For at beskrive den næste frame (frame 2), skal deres bruges en rotation om frames 1’s x-akse. Den rotateres med det vil sige at for frame 2 er , det er negativ for at sørger for at de næste ikke kommer til at dreje den forkerte vej, da de altid drejer i positiv omløbsretning om rotationsaksen. Når denne rotations er fundet afsted kan den næste led betrages som en planar robbot da de kan beskreves fuldstændigt i et plan. Dette plan kan ses på Figur 6. Ved undersøgelse fandt jeg ud af at når det andet led stod i nul stod den lodret op. Og den vinkel der forventes af transformations matrix, er fra x aksen i postiv omløbsretning. Der for beskrives led variable til den frame 2 som . På denne måde giver det 90 når er 0, så er vinklen mellem y aksen og den først armdel af robotten, dette kan også ses på Figur 6. Frame 3 er først en forskydning langs x aksen dette er selve længden af ledet så af er 150 mm. Rotation af dette frame er fast langt mekanisk i robotten, det vil sige at den kun afhænger af , og den sørger for at den altid er vandret. Dette opnået med at lave et parallelogram af 2 pinde op til dette punkt. For at finde en vinkel der simulerer denne med dennavit-hartberg parameterne. Ved at bestemme hvad vinkel skal være ud fra , skal være så stor at nye linje skal være parrall med x-aksen. Der tegnes en linje der er parralel med y aksen, og da de er parraller kan vinkel overføreres til skæring med robbot arm og linjen. Og for at den nye linje er parralle med x aksen, skal den stå 90 grader på y aksen. Udfra figur ? kan det ses det at . På denne måde kan frame 3 beskrives ud fra frame 2 ved hjælp af . Frame 4 er helt normal ved at det kun har en forskydning på og vinklen vender naturligt den rigtige retning. Frame 5 skal forskydes 150 mm på grund af længden af armenen, og da denne frame på samme mådes som frame 3 skal være parallelt med x-aksen, men her skal den bare være modsat den rotation som blev lavet i frame 4, så derfor . Den sidste frame er ikke, lige som de 4 forrige, en planar så derfor skal roters den så den kommer til at stå som den oprindlige frame, det gøres ved at sætte . Og så har den en helt normal som beskriver rotation af helt spidsen af roboten. De komplete dennavit-hartberg ses i Tabel 1

Figur Udsnit af planar model af dobot



Figur Model af planar del af dobot

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1 | 0 | 0 | 0 |  |
| 2 |  | 0 | 0 |  |
| 3 |  | 150 | 0 |  |
| 4 | 0 | 30 | 0 |  |
| 5 | 0 | 150 | 0 |  |
| 6 |  | 65 | 0 |  |

Tabel Denavit-Hartenberg parameter for en dobot

Nu opstilles der en transformationsmatrix fra være frame til den næste ud Denavit-Hartenberg parameterne og det matrix som beskriver transformation fra et frame til en anden. Så transformation fra frame 0(basen) til frame 1 ser sådan her ud

På samme måde kan man få resten

For at få den komplette transformation fra bunden af robotten til værktøjet på robotten, ganges disse matricer samme for at for en matrix der beskriver

Med denne matrix kan positionen af værktøjet på robotten beregnes hvis alle vinklerne kendes, det vil sige at dette er den fremadrettede kinematek beskrevet.

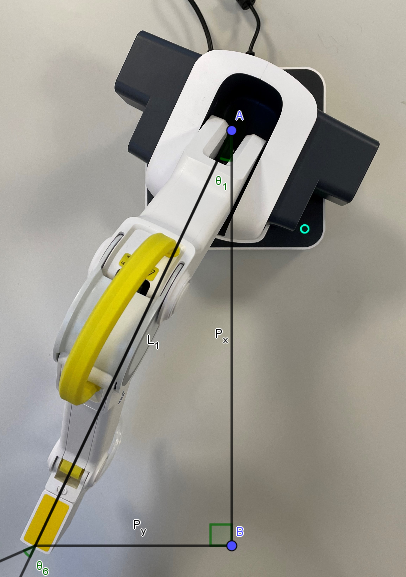
## Inverse kinematik

Nu kendes den fremadrettede kinematik for dobotten, men for at kunne sige til robotten at den skal bevæge sig et bestem sted hen i rummet ud fra et koordinater sæt og en rotation af værktøjet. Er det nødvendigt at kunne gå den anden vej, det vil sige fra et positions matrix til 4 vinkler de 4 led, dette er også kaldet inverse kinematik. Lad os sige at vi gerne vil til position P.

Grunden til at denne matrix er ikke særlig mange rotationsmuligheder, fordi dobotten kun kan roter om z aksen, da den mekanisk låst på alle andre akser, rotation af værktøjet er beskrevet med . For at komme fra P til de 4 vinkler vil jeg bruge geometrisk argumentation. Det er også muligt at løse inverse kinematik problemer med algebra men i dobottens tilfælde er der det simplere at gøre det ved med geometrisk argumentation. Vinkel er relativ simple da den ikke afhænger af andet en position af slutpunktet. Der kan tegnes en trekant hvor den en katete er og den anden katete er . Det kan ses på Figur 7. På denne måde kan vi tage den inverse tangens, så derfor kan beskrives på denne måde

er kun afhængig den ønskede vinkel og fordi og ikke har nogle betydning for vinkel på værktøjet. Da der er mekanisk lavet så dele kommer til at være vandret ret. For at beregnere , opstilles denne ligning som beskriver vinklen ud fra og , hvor er vinkel af værktøjet på robotten. Det kan ses på Figur 7 at de to langt sammen giver den endelige rotation af værktøjet på robotten.

For at finde trækkes fra på begge sider.



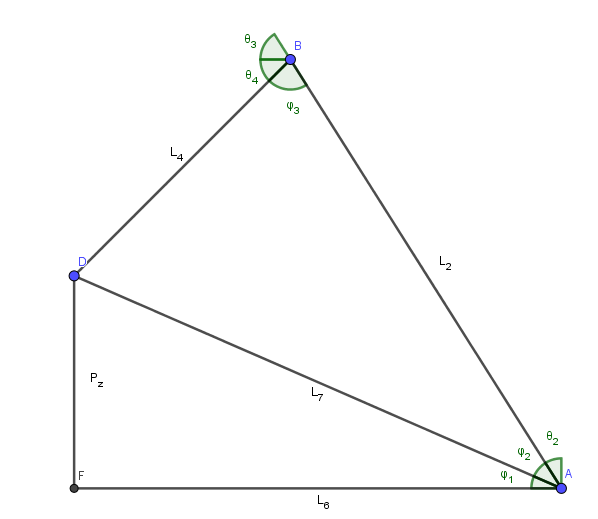
Figur Visualisering af θ1 og θ6

For at finde de sidste to vinkler ud fra den ønskede position, skal vi betragte den arm i planet for at forsimple det. Dette kan lade sig gøre fordi at og kun bevæger sig om en akse. På Figur 8 er der sat punkter ind i alle frames position i planet. til er længder der er kendt ud fra udformning af robotten. Længden er afstanden fra frame 0 til robottens værktøj position, projekteret ned på xy-planet. Det vil sige at hvis er 0 så er , og for at finde til alle positioner kan Pythagoras læresætning om retvinklede trekanter bruges, hvor er hypotenusen og og er kateter, denne trekant ses på Figur 7. Så findes ved



Figur Plan model for dobot

Den form der ses på Figur 8 er ret kompliceret, derfor forsimpler jeg den så det kommer til at ligne en helt almindelige plan robot med 2 rotationsled. For at gøre dette trækker jeg og fra trekanten, det vil sige at jeg samler det til én trekant. Med andre ord fjerner jeg den flade runde sektion, mellem punkt B og C, og sektion mellem D og E, når jeg fjerne disse sektioner, samler det hele sig for at jeg ikke for nogen huler i figuren. Denne ses på Figur 9. Jeg har også tilføjet et linjestykke fra A til D. Den gør at jeg har en lukket trekant ABD.



Figur Forsimplet model for en plan Dobot

er den tidligere men jeg har fjernet nogle stykker fra den, så længden af den er nu givet ved:

For at jeg kan begynde at regne på ABD trekanten skal jeg først kende tre værdi i trekanten. Jeg kender og da de er konstante begge to givet ud fra robottens fysiske dimensioner, denne spisefikse robot er de begge , men jeg forsætter med at arbejde med dem generelt. Det eneste ukendte side længde i trekanten ABD er , den er hypotenusen i en ret vinkelret trekant, hvor begge kateter er kendt så kan findes ved hjælp af Pythagoras.

Vinkel kan findes ved hjælp af den inverse tanges

For at finde bruger jeg cosinusrelationerne, og jeg kender alle side længderne i trekanten så på denne måde kan jeg finde.

På samme måde kan jeg finde

Nu er alle værdi kendt der skal bruges til at finde og .

Og for at finde skal jeg bruge og da jeg lavede den fremad rettede kinematik fandt jeg at

Ude fra Figur 9 ses det at

Jeg isoler

Og indsætter definition af ud fra

Så hæver jeg minus parentesen, ved at vende alle fortegnene i parentesen

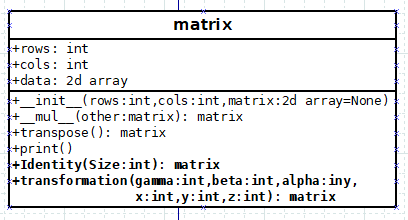
Til sidst reducerer jeg

Ved at trække nogle få af ligninger oven for sammen får man disse ligninger som beskriver den inverse kinematiken dobotten:

Hvor

# Analyse af Python programmer

## Analyse af Python bibliotek til regning med matricer

For at lave et bibliotek til at regne med matricer, skal klassen have noget data om matricen, den skal kende størrelsen af matricen, det vil sige antallet at af rækker og kolonner, så skal den også opbevare selve data i matricen, det vil sige de tal der er i matrixen. Det er alt klassen opbevarer, dette kan ses på klassediagrammet på Figur 10. Data variable er et to dimensionale liste, det betyder at det er en liste af lister. Dette stemmer godt overens med at man kan set et matrix som en vektor af vektor. Ud fra klasse diagrammet på Figur 10, kan det ses at klassen 3 normale funktioner der skal af et instans af klassen nemlig \_\_mul\_\_, transpose og print. Ud over det er det en konstruktor funktion som laver objektet \_\_init\_\_. Til sidst er der de to der er mærket med fed, identity og transformation, fordi det er klasse metoder, det betyder at det ikke er en instans af klassen der skal bruges til at kalde disse funktioner, men i stedet skal de kaldes direkte på klassen. Grund til at de er lavet på denne måde er fordi det er fabriks metoder, det betyder at de bruges til lave nogle spisefikke matricer som man ofte har brug for at lave, på denne måde bliver det lettere for dem som skal bruge biblioteket.

Figur Klasse diagram over matrixklassen

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Figur Construktor

Lad os starte med kigge på constructoren i klassen, det ses på linje 5 i Figur 11 at init metoden kan tage op til 3 parameter, den skal have antal af rækker og søjler, men behøves ikke at have et matrix fordi den har standard værdi, som er none, det betyder at den er tom. I andre sprog end Python har man mulighed for at lave flere constructor, så man kan lave en som tager et antal rækker og søljer, og så laver en tom matrix og så en anden der kun tager en 2d liste, også selv regner antal af rækker og søljer ud, men i python er det nødvendigt at lave det i en constructor, som har nogle standard værdier. På linje 8 tjekkes der om den har fået en matrix, hvis den ikke har, initialiser den bare data til at være et 2d list fyldt med 0. Men hvis den har fået noget tjekker den på linje 11 om det er et 2d liste, hvis ikke raise den en exeption, for at fortælle at programmøren at vedkommet har gjort noget forkert ved ikke at sende et 2d liste som argument til matrix. Men hvis det er en 2d liste så sætter den data varialblen til at være det matrix der er blevet sendt med til constructoren. Til sidst på linje 14 og 15 sætter den rows and cols til at være højden og bredten af matricen.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Figur Matrix multiplikation metode

For at implemntere en funktion til at gange to matricer sammen bruges nølgeordet \_\_mul\_\_ på linje 17 på Figur 12, ved at kalde den dette fortæller det python at denne funktion skal kaldes når der sættes gange tegn mellem to matrix objekter. Det første der bliver tjekket på linje 18 er at det er muligt gange de to matricer sammen, da martix-matrixproduktet kun er definert når antallet af kolonner i den første matrix er det samme som antal rækker i den anden matrix. I hvis sætning spørge dem om de ikke er det samme, og hvis de ikke er det sammen raiser den en exeption, for at fortælle programøren at de matricer vedkommet forsøgte at gange sammen ikke kan lade sig gøre. Nu ved programmet at det er muligt at finde produkt mellem de to matricer, så det forsætter ved at går igennem 2 løkker, og på denne måde regner resultat ud for hver postion i det resulternde matrix. På linje 24 og 25 lægger den de to vektor der skal prikkes ud i variablerne vec1 og vec2. På linje 26 findes produktet ved at går igennem være værdi i vektoren og gange dem sammen, og til sidst tage summen. Dette gøres ved alle tal i den resultaterne matrix, og til sidst return funktion den nye matrix.



Figur 15 Tranpose metode

Funktion der transponere funktion, bruger 2 inline for løkker til at spejle matricen over diagonalen, dette kan ses på Figur 13 på linje 35.



Figur Manglende invert metode

Invert funktion er ikke implentert i dette biblotekt, da det ville ikke helt simplet at gøre, men det er den første funktion der skulle tilføjes, hvis biblotiktet skulle udvides. Og da det ikke er nødvedingt i demstartions programmet der arbejder med dobotten, har jeg valgt ud lade den forholdvis væstelige funktion fra biblotket.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Figur 17 Print metode

Matrixklassen har en print funktion som printe matricen til konsolen, dette bruges for at lettere kunne debugge koden. Funktionen fungerer ved at for hver rækker laver den string, som består at dataen fra den række, så printer denne string. For at udvide den string der arbejds på bruges += opratern som bare tilføjer den nye string til enden af variablen string, dette ses på Figur 15 op linje 41. Når der printes til konsollen laver den et linjeskiftet, på denne måde kommer være række på hver sin linje. På Figur 16 er der et eksempel på hvordan det kan ses ud når den printer en matrix til konsollen, i dette tilfælde er den en 32 matrix.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Figur Eksempel på print af matrix

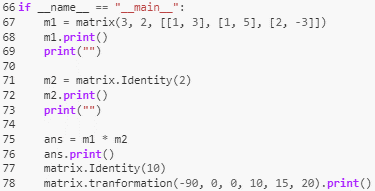
De sidste to funktioner i klassen er klassemetoder, betydende at de ikke kræver et instance af klassen. Det kan ses ved at der står @classmethod før begge metoder, se Figur 17 på linje 44 og 53. Når det er klasse metode, skal den alt modtage en reference til selve klassen som det første parrameter i funktion, dette er i modsætning til normale metoder i et objekt i python som skal have et parameter til en refernce til sig selv, dette kan ses på Figur 15 på linje 37. I disse fabriks metoder bruger denne klasse refernce ikke, men de skal have den alligevel fordi python sender den med uanset hvad. Identity funktion laver en identitets matrix i størrelsen den modtager i size paremeteret. På linje 46 laver den en tom kvadradisk matrix, da den siger at den skal havde det samme antal rækker som søjler. For at lave dette om til en identitets matrix sætter den 1 ind på diagonalen, ved hjælp af for løkken på linje 47 og 48.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Figur Fabriks metoder til at lave identites matricer og transformations matricer

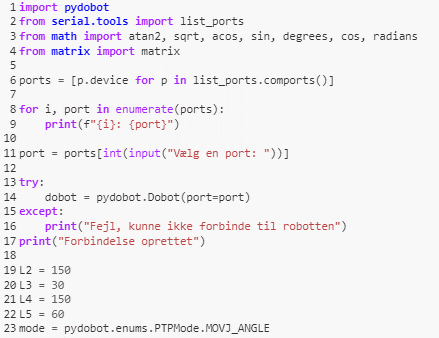
Transformations metode laver en homogen transformations matrix, udfra 3 vinker beskirver rotations med 3 fast låse akser, først en rotation om x-aksen så y-aksen til sidst z-aksen, og transaltion med x, y, z. Jeg bruger matricen fundet i afsnit Rotation. På linje 55 til 57 som ses Figur 17, laves vinklerne fra grader til radianer fordi pythons math biblotek regner med radiane i de trigonomgiske funktioner. På linje 63 laves denne 2d liste om til et matrix og samtidigt bliver det rundet af til 4 decimaler.



Figur Test af Python bibliotek

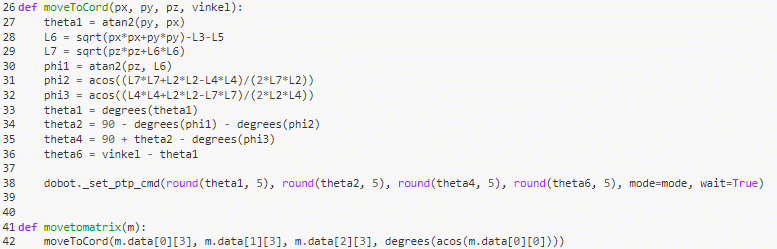
For at teste et bibliotek i Python, kan lave en hvis sætning som på linje 66 på Figur 18, til at tjekke om filen bliver kørt selv stædigt eller om det er er blevet inkluderet af en anden Python program, så linjerne 67 til 78 kører kun hvis den bliver kørt selvstændigt, på denne måde kan man test om biblioteket gør som det forventes. I dette bibliotek er er det ikke implementeret så den selv ved om testen er rigtig, men der skal programmøren tjekke, men her kunne det udvides med unit testning, hvor man tester være enkelt funktion, med noget kendt input, så man også kender resultatet.

## Analyse af program til styring af dobot

For at kunne styre robot gennem et Python program skal der først oprettes kontant til robot. Computeren og robotten snakker sammen gennem en helt almindelige seriel port. Det gøres som det første i programmet fra linje 6 til 17 på Figur 19. Inden da importes alle bibloteker programmet skal bruges, på linje 1 til 4. Læg særlig mærke til at matrix-klassen fra biblioteket importeres på linje 4.

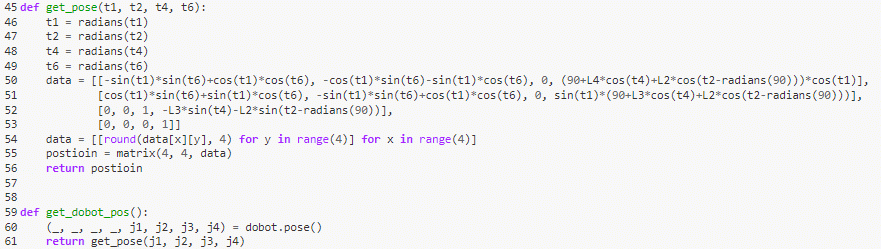
Figur Opsætning af program til styring af dobot

Der laves på linje 14 et dobot objekt som håndterer al kommunikation med dobotten fra computeren. Det forsætter med opsætning af programmet når der på linje 19 til 22 bliver oprettet varibler som har beskriver den fysiske størrelse af robbotten. Disse variabler har de samme navne som der blev brugt da, den inverse kinematik blev lavet. Det gøre at det bliver at overføre matematikken til programmet. Den sidste opsætning der skal til er at vælge hvilken mode dobotten skal bevæge sig med. Dette sker på linje 23. MOVJ\_ANGLE betyder at den for koordinaterne til position i joint rummet, det vil sige at de 4 værdi der sendes til den svar til vinkel til hver af det 4 led på robotten og j’et betyder at den skal bevæger sig fra et punkt til et andet bare ved at gå fra den ene vinkel til den for være at motorerne. Det betyder man ikke kender hvilken vej robotten kommer hen til det punkt man ønsker. Mod sat kunne man sætte et L i stedet for j’et, men for at den kan gøre det skal dobotten lave inverse kinematik også giver det ikke mening jeg selv har lavet det.[[1]](#footnote-3)



Figur Funktion der gør brug af den inversekinematik

På Figur 20 ses funktionen moveToCord som får dobotten til at bevæge sig hen til et givent punkt, et koordinat sæt og en vinkel af værktøjets rotation. Linjerne 27 til 36 på Figur 20 svar en til en med de ligninger jeg fandt i afsnit Inverse kinematik, de står i samme rækkefølge som i afslutning af dette afsnit. Den eneste ting som er ændret, er at der bruges funktion degrees til at omregne fra radianer til grader på linjerne 33 til 36. Her ses det at når formlerne er fundet er det helt simpelt at skrive den ind et program og kører efter dem. På linje 38 sendes kommandoen til dobot til at få den til at bevæge sig hen til de udregnet vinkler så værktøjet på robotten kommer til at være på pointet sendt i parametrene til funktionen, alle vinkelværdierne bliver afrundet til 5 decimaler, da jeg har oplevet at robotten gik i fejl hvis den fik for mange decimaler. Her bruges den mode der blev valgt på linje 23.



Figur 23 Forward kinematik funktion

Funktion get\_pose der ses på Figur 21, får 4 vinkler fra robotten, også regner det om til en matrix der beskriver postion, af robbotten. For at gøre det bruger funktion den forward kinematik matrix jeg fandt i afsnittet Forward kinematik. Denne matrix er skrevet direkte inden på linjerne 50 til 53. For at kunne brug de trigonmisek funktioner om regnes der inden til radianer på linjerne 46 til 49.

Der er en hjælpe funktion begge ovenstående funktioner , der gør dem, lettere at bruge, på Figur 20 på linje 41 og 42 er der en funktion movetomatrix som har et matrix som parameter og finder den de relaveante data fra matricen og sender dem videre til moveToCord funktion. Og på Figur 21 på linje 59 til 61, ses get\_dobot\_pos funktion, som bruger dobot biblotiekt posetions funktion til at for vinklerne på motorerne lige nu, disse vinkler sender den videre til get\_pos funktion som laver fremadrettet kinematik på dem. Dette er alle funktioner der skal bruges til at komadere dobot til at gøre hvad man vil havde den til.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Figur Test af dobot styrering

For at teste disse funktioner afsluttes programet med en kort test. På Figur 22 på linjerne 64 til 74 testets matrix regningen ved at på linje 64 fortæller at den skal kører til 300, -100, 0 og vinkel på . Derefter aflæser position af robot, dette returners som en matrix, så på linje 68 laves en transformations matrix med fabriks metoden. Disse to matricer ganges sammen og så flytter robotten hen til det nye punkt på linje 74. Det viser at der muligt dobotten i forhold til den nuværende potion og rotation. Linjerne 77 til 83 kører dobotten fremad og tilbage i en lige linje mellem to punkter for at hvis at inverse kinematik model virker.

# Hvordan er den matematiske viden nødvendig for programmør?

Som det ses i den ovenstående del af opgaven er det et stor arbejde der ligger i alt lave fremadrettet kinematek og den inversekinematik, så nu er det relevant at undersøge hvor stor del af det arbejde er nødvendigt at programmøren laver. Der er flere måder at løse inversekinematik problemt på, i denne opgaven er der bruget en analytisk metode som har fundet en lukket form, så en 1 til 1 funktion som kan regne postion af ledene udfra mål postion. Det er også muligt at bruge numeriske løsninger, hvor man bruger en iterativ metoder, hvor man prøve nogle vinkler, og ser hvor tættet robboten er på mål postion, udfra denne information ændre man på vinklerne for at komme tættere på. De findes forskellige metoder til lave iterativ inversekinematik, nogle er helt simple andre indeholder mere matematik for at forsimple optimerngings problemet. Når man skal arbejde med at progammere robotter er der flere til gange man kan lave alt matematikken selv, men det er meget tidskrævende og for holdsvist svært. Men man kan finde et bibloteker der er lavet til at løser inverse kinematik problemer, beskrive den robot der arbejdes med også vil den højst sandsyndligt kunne løse det problemet for programøren, dette gøre det meget lettere at komme i gang, dette er især smart hvis det er hobby projektter hvor det ikke er nødvendingt med den matematiske forståelse som opnås ved at selv at lave det hele.[[2]](#footnote-4) Hvis der arbejdes i industrien hvor det handler hurtigst muligt at få noget der virker i produktion, er der produkter som robodk lever som er lavet for at hurtigst muligt for et produkt til at virke, de har en masse industri robotter som kan programmers let fra computeren også kan roboten med det samme gå i gang. De udtaler sig på deres blog at man ikke skal lave matematik hvis ikke det ikke er dig selv der har bygget robotten, for ellers har nogle andre lavet løst problemtet og det er ikke nødevendigt at gører igen.[[3]](#footnote-5)

# Konklusion

# Referencer

Craig, J. J. (2005). *Introduction to Robotics mechanics and control* (3 ed.). Upper Saddle River, United States of America: Pearson Prentice Hall.

luismesas, knutsun, ZdenekM, retospect, ksketo, & remintz. (8. februar 2020). *ptpMode.py*. Hentet 15. December 2022 fra github.com/luismesas/pydobot: https://github.com/luismesas/pydobot/blob/master/pydobot/enums/ptpMode.py

Pierce, R. (2021. August 25). *How to Multiply Matrices*. Hentet 8. December 2022 fra Math Is Fun: http://www.mathsisfun.com/algebra/matrix-multiplying.htm

STEM EDUCATION WORKS. (u.d.). *DOBOT MAGICIAN LITE*. Hentet 12. December 2022 fra stemeducationworks: https://stemeducationworks.com/product/dobot-magician-lite/

Undervisningsministeriet. (27. august 2014). Lineære afbildninger.

Bilag

## Kode fra matrix.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | **from** **math** **import** radians, cos, sin  **class** matrix:  **def** \_\_init\_\_(self, rows, cols, matrix=None):  self.rows = rows  self.cols = cols  **if** matrix **is** None:  self.data = [[0 **for** \_ **in** range(cols)]**for** \_ **in** range(rows)]  **else**:  **if** **not** isinstance(matrix, list) **or** **not** isinstance(matrix[0], list):  **raise** **Exception**("Matrix must be a 2d list")  self.data = matrix  self.rows = len(matrix)  self.cols = len(matrix[0])  **def** \_\_mul\_\_(self, other):  **if** self.cols != other.rows:  **raise** **Exception**(  "The first matrix must have the same numbers of coloms as the other has rows")  m = matrix(self.rows, other.cols)  **for** row **in** range(self.rows):  **for** col **in** range(other.cols):  vec1 = self.data[row]  vec2 = [other.data[i][col] **for** i **in** range(other.rows)]  dotproduct = sum(vec1[i] \* vec2[i] **for** i **in** range(self.cols))  m.data[row][col] = dotproduct  **return** m  **def** invert(self):  **pass**  **def** transpose(self):  **return** matrix(self.cols, self.rows, [[self.data[row][col] **for** row **in** range(self.rows)] **for** col **in** range(self.cols)])  **def** print(self):  **for** row **in** range(self.rows):  string = ""  **for** col **in** range(self.cols):  string += f'{self.data[row][col]} '  **print**(string)  @classmethod  **def** Identity(cls, size):  m = matrix(size, size)  **for** i **in** range(size):  m.data[i][i] = 1  **return** m  *# fixed angels X-Y-Z in degrees, alpha around z, beta around y, gamma around x*  @classmethod  **def** tranformation(cls, gamma, beta, alpha, x, y, z):  gamma = radians(gamma)  beta = radians(beta)  alpha = radians(alpha)  data = [[cos(alpha)\*cos(beta), cos(alpha)\*sin(beta)\*sin(gamma)-sin(alpha)\*cos(gamma), cos(alpha)\*sin(beta)\*cos(gamma)+sin(alpha)\*sin(gamma), x],  [sin(alpha)\*cos(beta), sin(alpha)\*sin(beta)\*sin(gamma)+cos(alpha)\*cos(gamma), sin(alpha)\*sin(beta)\*cos(gamma)-cos(alpha)\*sin(gamma), y],  [-sin(beta), cos(beta)\*sin(gamma), cos(beta)\*cos(gamma), z],  [0, 0, 0, 1]  ]  **return** matrix(4, 4, [[round(data[x][y], 4) **for** y **in** range(4)] **for** x **in** range(4)])  **if** \_\_name\_\_ == "\_\_main\_\_":  m1 = matrix(3, 2, [[1, 3], [1, 5], [2, -3]])  m1.**print**()  **print**("")  m2 = matrix.Identity(2)  m2.**print**()  **print**("")  ans = m1 \* m2  ans.**print**()  matrix.Identity(10)  matrix.tranformation(-90, 0, 0, 10, 15, 20).**print**() |
|  |  |

## Kode fra dobot\_controller.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86 | **import** **pydobot**  **from** **serial.tools** **import** list\_ports  **from** **math** **import** atan2, sqrt, acos, sin, degrees, cos, radians  **from** **matrix** **import** matrix  ports = [p.device **for** p **in** list\_ports.comports()]  **for** i, port **in** enumerate(ports):  **print**(f"{i}: {port}")  port = ports[int(input("Vælg en port: "))]  **try**:  dobot = pydobot.Dobot(port=port)  **except**:  **print**("Fejl, kunne ikke forbinde til robotten")  **print**("Forbindelse oprettet")  L2 = 150  L3 = 30  L4 = 150  L5 = 60  mode = pydobot.enums.PTPMode.MOVJ\_ANGLE  **def** moveToCord(px, py, pz, vinkel):  theta1 = atan2(py, px)  L6 = sqrt(px\*px+py\*py)-L3-L5  L7 = sqrt(pz\*pz+L6\*L6)  phi1 = atan2(pz, L6)  phi2 = acos((L7\*L7+L2\*L2-L4\*L4)/(2\*L7\*L2))  phi3 = acos((L4\*L4+L2\*L2-L7\*L7)/(2\*L2\*L4))  theta1 = degrees(theta1)  theta2 = 90 - degrees(phi1) - degrees(phi2)  theta4 = 90 + theta2 - degrees(phi3)  theta6 = vinkel - theta1  *# print(f'j1:{theta1} j2:{theta2} j3:{theta4} j4:{theta6}')*  dobot.\_set\_ptp\_cmd(round(theta1, 5), round(theta2, 5), round(theta4, 5), round(theta6, 5), mode=mode, wait=True)  (x, y, z, r, j1, j2, j3, j4) = dobot.pose()  *# print(f'x:{x} y:{y} z:{z} r:{r} j1:{j1} j2:{j2} j3:{j3} j4:{j4}')*  **print**(f'Diff: x: {round(px-x,2)} y: {round(py-y,2)} z: {round(pz-z,2)} r: {round(vinkel-r,2)}')  **def** movetomatrix(m):  moveToCord(m.data[0][3], m.data[1][3], m.data[2][3], degrees(acos(m.data[0][0])))  **def** get\_pose(t1, t2, t4, t6):  t1 = radians(t1)  t2 = radians(t2)  t4 = radians(t4)  t6 = radians(t6)  data = [[-sin(t1)\*sin(t6)+cos(t1)\*cos(t6), -cos(t1)\*sin(t6)-sin(t1)\*cos(t6), 0, (90+L4\*cos(t4)+L2\*cos(t2-radians(90)))\*cos(t1)],  [cos(t1)\*sin(t6)+sin(t1)\*cos(t6), -sin(t1)\*sin(t6)+cos(t1)\*cos(t6), 0, sin(t1)\*(90+L3\*cos(t4)+L2\*cos(t2-radians(90)))],  [0, 0, 1, -L3\*sin(t4)-L2\*sin(t2-radians(90))],  [0, 0, 0, 1]]  data = [[round(data[x][y], 4) **for** y **in** range(4)] **for** x **in** range(4)]  postioin = matrix(4, 4, data)  **return** postioin  **def** get\_dobot\_pos():  (\_, \_, \_, \_, j1, j2, j3, j4) = dobot.pose()  **return** get\_pose(j1, j2, j3, j4)  moveToCord(300, -100, 0, 50)  m = get\_dobot\_pos()  m.**print**()  transformation = matrix.tranformation(0, 0, 0, -100, 40, 40)  reslut = m \* transformation  **print**("")  reslut.**print**()  movetomatrix(reslut)  step\_count = 7  p1 = (170, 150, 110)  p2 = (290, -150, -30)  step = ((p2[0]-p1[0])/step\_count, (p2[1]-p1[1]) / step\_count, (p2[2]-p1[2])/step\_count)  **while**(True):  **for** i **in** list(range(step\_count+1))+list(reversed(range(step\_count+1))):  moveToCord(p1[0]+step[0]\*i, p1[1]+step[1]\*i, p1[2]+step[2]\*i, 90) |

1. (luismesas, et al., 2020) [↑](#footnote-ref-3)
2. <https://roboticsbackend.com/should-you-learn-mathematics-to-program-robots/> [↑](#footnote-ref-4)
3. <https://robodk.com/blog/inverse-kinematics-in-robotics-what-you-need-to-know/> [↑](#footnote-ref-5)