Technion – Israel Institute of Technology

Faculty of Mechanical Engineering



HW2

Kinematics, Dynamics, and Control of Robots

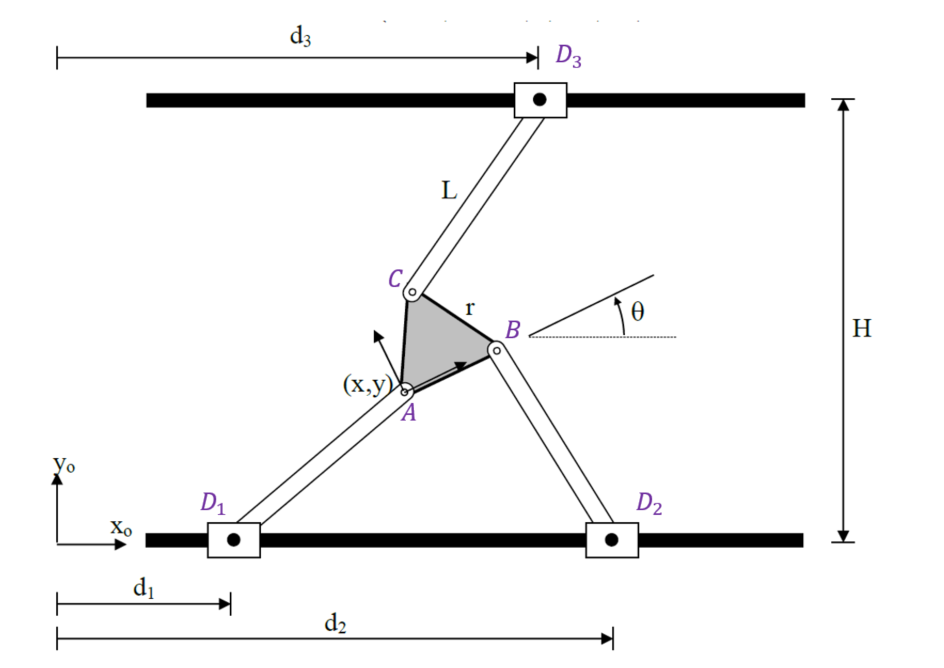
036026

|  |  |  |
| --- | --- | --- |
| Alon Spinner | 305184335 | alonspinner@gmail.com |
| Shahar Tsadok | 203783519 | Shahar507@gmail.com |

December 2, 2020

# Finding the Inverse Kinematics of the Robot

We mark points of interest on the mechanism:



Triangle Points:

Prism Points:

Kinematic constraints:

Extracting from each of the three kinematic constraints:

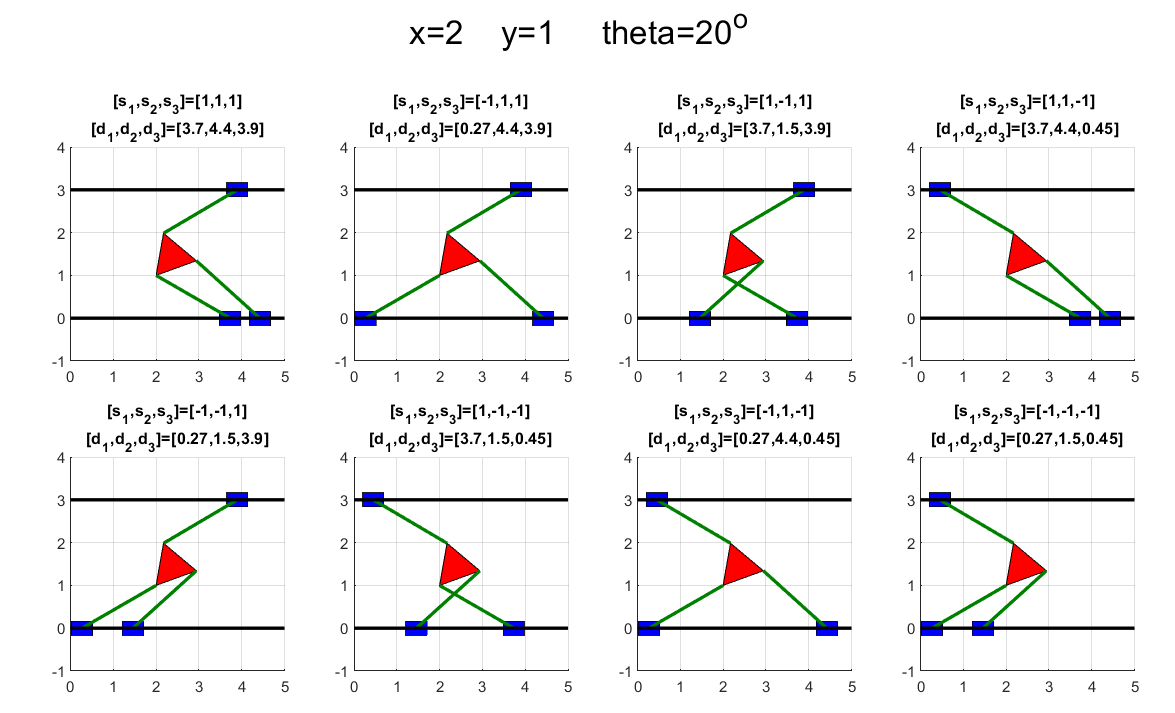
## Calculating Solutions for

As there are inverse kinematics solutions for each combination, we defined them as follows:

We created a function *Inverse\_Kinematics* that accepts vectors and and returns the joint values with which we can draw the mechanism. Code and results follow.

[d1,d2,d3]=Inverse\_kinematics(x,y,theta,elbows,varargin)

DrawMechanism(ax,x,y,theta,d1,d2,d3)



# Finding the Direct Kinematics

Most of the work in this section was done via MATLAB’s symbolic toolbox with the terms being long and uninformative. For that reason, we will show the flow and not explicit results.  
**Note: the code shown in this section is not as it appears in the scripts. We removed distracting functions that are necessary but provide no value in explaining the process.**

We start with the three kinematic constraint equations developed in the previous section:

syms d1 d2 d3 x y theta L r H real

Eq1=(x-d1)^2+y^2==L^2;

Eq2=(x+r\*cos(theta)-d2)^2+(y+r\*sin(theta))^2==L^2;

Eq3=(x+r\*cos(theta+pi/3)-d3)^2+(y+r\*sin(theta+pi/3)-H)^2==L^2;

Deducting equation from and equation from we find that the output terms are only a function of allowing us to create a linear system of equations.

Eq12=Eq1-Eq2;

Eq13=Eq1-Eq3;

[A,b] = equationsToMatrix([Eq12;Eq13],[x;y]);

Solving the linear system of equations to find ,

X=linsolve(A,b);

Xt=X(1);

yt=X(2);

Defining a substitution parameter for to create rational terms

syms T;

sinT=2\*T/(1+T^2);

cosT=(1-T^2)/(1+T^2);

xT=subs(xt,[sin(theta),cos(theta)],[sinT,cosT]);

yT=subs(yt,[sin(theta),cos(theta)],[sinT,cosT]);

Plugging , into equation we obtain an equation in one variable

Defining the left-hand side as we get a rational expression in

The numerator is a degree polynomial in whose roots will provide the solution’s values in the form of a six-element vector - .

Eq1T=subs(Eq1-L^2,[x,y],[xT,yT]); %Eq1-L^2==0

ExprT=lhs(Eq1T);

[polyT,~]=numden(ExprT);

coffs=double(coeffs(polyT,'all'));

TVec=roots(coffs);

Once is found we plug in its values into our substitution equations to find .

sVec = 2\*TVec./(1+TVec.^2);

cVec = (1-TVec.^2)./(1+TVec.^2);

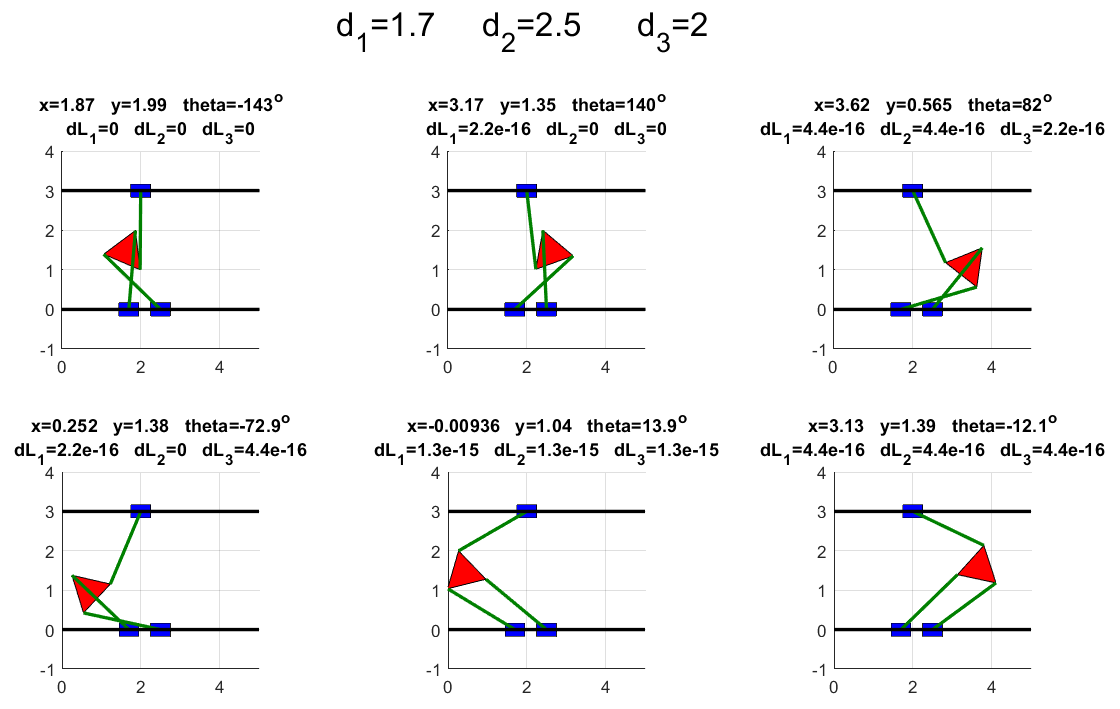
thetaVec = double(atan2(sVec,cVec));

xVec=double(subs(xT,T,TVec));

yVec=double(subs(yT,T,TVec));

## Forward Kinematics for

We applied our forward kinematics solution on and computed the difference between computed and actual link’s length to test it. This test takes the place of applying inverse kinematics on the results - it is simpler and follows the same kinematic constraint equations. Solving the direct kinematics provides 8 solution with one of them being . The difference in links’ length was found tiny in comparison to the actual links’ lengths and so we can acknowledge our forward kinematic solution as correct.



# Computing the Jacobians

We wrote the kinematic constraints equations in an implicit form

Concatenating the implicit equations into one function in vector form will allow us to define the Jacobians.

Code and results follow:

syms d1 d2 d3 x y theta L r H real

f1=(x-d1)^2+y^2-L^2;

f2=(x+r\*cos(theta)-d2)^2+(y+r\*sin(theta))^2-L^2;

f3=(x+r\*cos(theta+pi/3)-d3)^2+(y+r\*sin(theta+pi/3)-H)^2-L^2;

F=[f1;f2;f3];

Jx=jacobian(F,[x,y,theta]);

Jq=-jacobian(F,[d1,d2,d3]);

These Jacobian terms are a function of both and .   
We used the Inverse Kinematics developed in Section ‎1 to create terms that are a function of and **,** where is an elbow vector, correlating to the joints in . **Note: while is an elbow argument, , is here notated as the ,.**  
Code and results follow:

syms s1 s2 s3 real

[d1s,d2s,d3s]=Inverse\_kinematics(x,y,theta,[s1 s2 s3],'method','symbolic');

Jq\_xs=subs(Jq,[d1,d2,d3],[d1s,d2s,d3s]);

Jx\_xs=subs(Jx,[d1,d2,d3],[d1s,d2s,d3s]);

# Drawing and Analyzing Singular Poses with Free Directions of the Mechanism where , and .

We added another constraint to our singular poses search space, having , which will have no effect on the poses shape as all prisms move in the direction.

To search for singular poses with free directions, we ran a greedy algorithm on all possible elbow positions. In each elbow position we equated the determinant of to zero to find a value such that . This kind of singularity is called the Forward Kinematics singularity. Each solution was tested against three conditions:

1. Link’s lengths are valid
2. is real or the imaginary pert is close to zero (
3. determinant is zero or close (

We then took the null space of for each of the validated solutions to find the free direction such that .  
**Pseudo-code** and pose results appear below.

theta0=0

x0=2

for each s in ElbowCombination %(8 in total)

Det(y)=det(Jx(theta0,x0,s))

y0Vec=solve(Det(y)==0)

for each y0 in y0Vec

[d1,d2,d3]=Inverse\_Kinematics(theta0,x0,y0,s)

%Check solution validity

ValidLinksLength(x0,y0,theta0,d1,d2,d3)

DeterminantIsReal(Jx(theta0,x0,y0,s))

DeterminantIsCloseToZero(Jx(theta0,x0,y0,s))

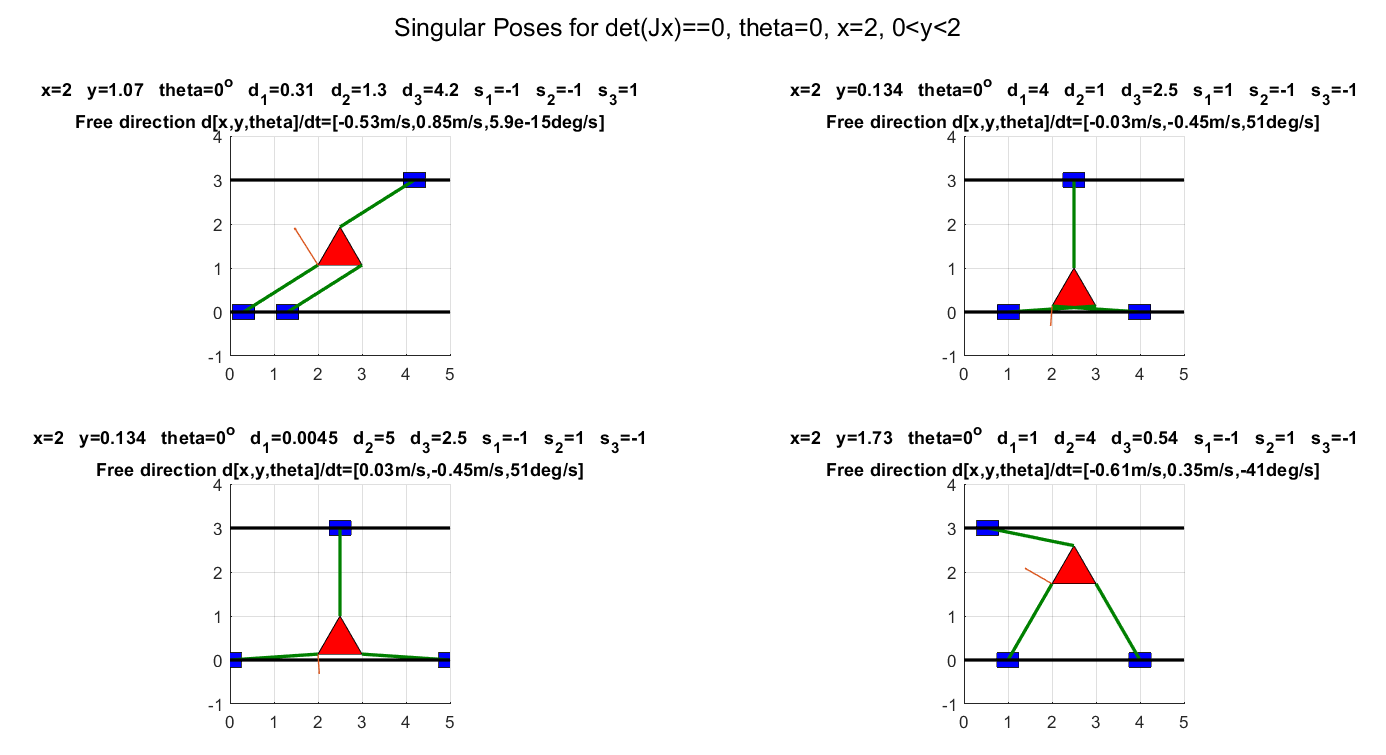
if ValidLinksLength & DeterminantIsReal & DeterminantIsClosetoZero

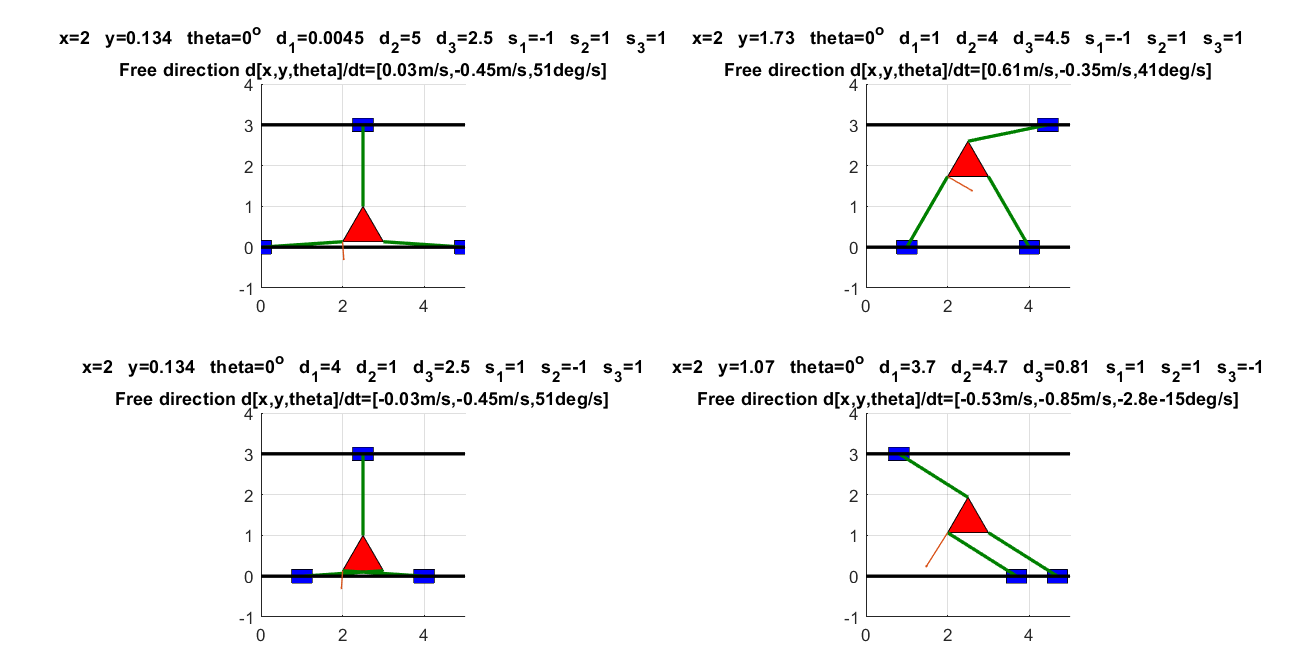
FreeDirection=null(Jx(theta0,x0,y0,s));

SaveSolution();

The algorithm provides eight different solutions for the robot’s pose singularity where four of the poses are mirror image of the others. In all poses, the lines constructed by continuing the coupler links (Links with length ) intersect at a single point inside the triangle, or on its boundary. Rotation around this point we will have no resistance from the robot joints providing an 'extra' DOF at those poses. We call this point the Instantaneous Center. Our analysis is based around the point which may be different from the Instantaneous Center. As such, our free direction vector may include linear velocity correlating to the lever arm between the two points, and angular velocity .

We would note that when the coupler links are parallel to each other (two poses) the Instantaneous Center is at infinity. In those two cases the third element in the free direction vector,, equals zero, as the free direction is a straight line perpendicular to the links.





There is another kind of robot singularity where )=0. This would be the Inverse Kinematics Singularity and it can be computed just as we computed the Forward Kinematics Singularity. Setting as a condition, we were able to find the edge of the workspace under the constraint – where the robot cannot move in the direction.



# Conclusions