# Term Project Report

Introduction to Robot Mechanics

ME-GY 6913

Tandon School of Engineering-NYU

Shivam Joshi (sj3104)

Prof. Dr. Joo H. Kim

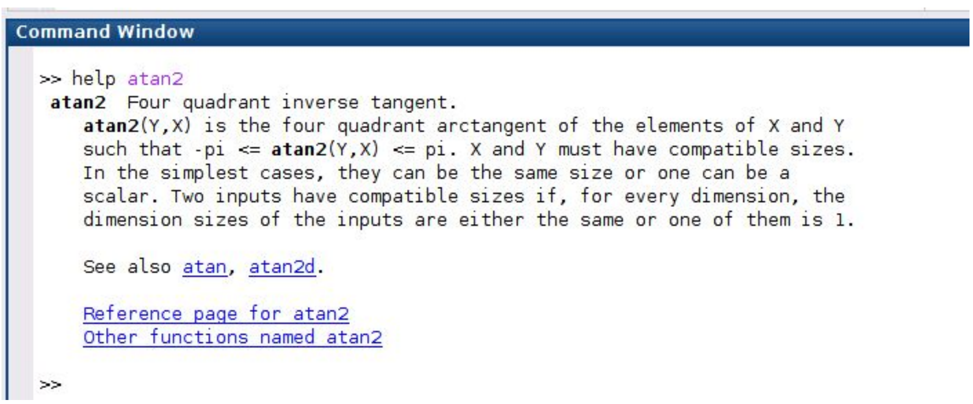
02 December 2019

**Matlab Programs for Robot Kinematics:**

Following are the MATLAB Programs are solutions for programming exercises from the book Introduction to Robot Mechanics by J.J Craig. - 4th edition.

CHAPTER 2:

**Problem 1**: If your function library does not include an Atan2 function subroutine, write one.
**Solution 1**: Though matlab contains a atan2 function:


Command Window

```
>> help atan2
 atan2  Four quadrant inverse tangent.
    atan2(Y,X) is the four quadrant arctangent of the elements of X and Y
    such that -pi <= atan2(Y,X) <= pi. X and Y must have compatible sizes.
    In the simplest cases, they can be the same size or one can be a
    scalar. Two inputs have compatible sizes if, for every dimension, the
    dimension sizes of the inputs are either the same or one of them is 1.

    See also atan, atan2d.

    Reference page for atan2
    Other functions named atan2

>>
```

We can still create our own simple Atan2 function as:

Editor - /home/shivam/Desktop/RobotMechanics/Atan2.m

UTOI.m | ITOU.m | Atan2.m

```matlab
function [theta] = Atan2(y,x)

if (x>0)
    theta = atand(y/x);
elseif (x<0 && y>=0)
    theta = atand(y/x)+180;
elseif (x<0 && y<0)
    theta = atand(y/x)-180;
elseif (x==0 && y>0)
    theta = 90;
elseif (x==0 && y<0)
    theta = -90;
else
    theta = 0;
end

```

**Problem 2**: To make a friendly user interface, we wish to describe orientations in the planar world by a single angle, 9, instead of by a 2 x 2 rotation matrix. The user wifi always communicate in terms of angle 9, but internally we will need the rotation-matrix form. For the position-vector part of a frame, the user will specify an x and a y value. So, we want to allow the user to specify a frame as a 3-tuple: (x, y, 9). Internally, we wish to use a 2 x 1 position vector and a 2 x 2 rotation matrix, so we need conversion routines. Write a subroutine whose Pascal definition would begin

```
Procedure UTOI (VAR uforni: vec3; VAR if orm: frame);
```

where "UTOI" stands for "User form TO Internal form." The first argument is the 3-tuple (x, y, 0), and the second argument is of type "frame," consists of a (2 x 1) position vector and a (2 x 2) rotation matrix. If you wish, you may represent the frame with a (3 x 3) homogeneous transform in which the third row is [0 0 1]. The inverse routine will also be necessary:

```
Procedure ITOU (VAR if orm: frame; VAR uform: vec3);
```

**Solution 2**: Here wee can create a function which takes in a 3x1 vector (User Format) and returns required 3x3 matrix (Homogenous or Internal Format) as:

Editor - /home/shivam/Desktop/RobotMechanics/UTOI.m

UTOI.m

```
function frame = UTOI (vec3)

%Insert angle in degrees
theta = (vec3(3)*pi)/180;

frame = [cos(theta) , -sin(theta), vec3(1);
          sin(theta), cos(theta), vec3(2);
          0 , 0 , 1];
end
```

Command Window

```
>> v = [1 2 30]

v =

     1     2    30

>> UTOI(v)

ans =

    0.8660   -0.5000    1.0000
    0.5000    0.8660    2.0000
         0         0    1.0000

>> 
```

And similarly for converting from Internal to user format wee can write a ITOU function which takes in a 3x3 matrix as input and calculates theta using elements from the first column and position using third column of the input matrix to output 3x1 vector as:

Editor - /home/shivam/Desktop/RobotMechanics/ITOU.m

UTOI.m | ITOU.m

```
function vec3 = ITOU(frame)

%output angle is in degrees
theta = (atan2(frame(2,1),frame(1,1)))*180/pi;

%output vector is 1x3
vec3 = [frame(1,3),frame(2,3),theta];
end
```

Command Window

```
>> v = [1 2 30]

v =

     1     2    30

>> UTOI(v)

ans =

    0.8660   -0.5000    1.0000
    0.5000    0.8660    2.0000
         0         0    1.0000

>> ITOU(ans)

ans =

    1.0000    2.0000   30.0000

>>
```

**Problem 3**. Write a subroutine to multiply two transforms together. Use the following procedure heading:

```
Procedure TMULT (VAR brela, creib, crela: frame);
```

The first two arguments are inputs, and the third is an output. Note that the names of the arguments document what the program does.

**Solution 3:** Here we can simply convert two 3x1 input vectors in 3x3 internal format, do matrix multiplication and then convert the result back to 3x1 user format as follows:

Editor - /home/shivam/Desktop/RobotMechanics/TmulT.m

TmulT.m

```
function crela = TmulT(brela,crelb)
% frame is crela
% user inputs are in 1x3 format

% output is also converted to 1x3 format (user format) for uniformity
crela = ITOU( (UTOI(brela)) * (UTOI(crelb)) );

% InInternalFormat = UTOI(crela)
end
```

Command Window

```
>> v
v =
     1     2    30
>> v2 = [ 1 3 45]
v2 =
     1     3    45
>> TmulT(v,v2)
ans =
    0.3660    5.0981   75.0000
>>
```

**Problem 4:** Write a subroutine to invert a transform. Use the following procedure heading:

```
Procedure TINVERT (VAR brela, areib: frame);
```

The first argument is the input, the second the output. Note that the names of the arguments document what the program does.

**Solution 4:** To calculate the inverse of the 3x1 input vector we can simply convert it to internal form using UTOI, then after extracting and transposing rotation matrix out of it we can calculate position vector by matrix-vector multiplication. And eventually return the required inverted transform in both user or internal format.

Editor - /home/shivam/Desktop/RobotMechanics/TINVERT.m

TINVERT.m

```
function frame = TINVERT(brela)
% frame is arelb

% homogenous transformation of user input using UTOI function
HT = UTOI(brela);

% extracting rotation matrix from HT and transposing it
R = HT(1:2,1:2);
Rtranspose = R';

% multiplying position vector with transpose of rotation matrix
trans = Rtranspose*HT(1:2,3);

% final frame arelb is returned in user format for uniformity
frame = ITOU([Rtranspose,-trans;0,0,1]);

InInternalFormat = UTOI(frame);
end
```

Command Window

```
>> TINVERT(v)
InInternalFormat =
    0.8660    0.5000   -1.8660
   -0.5000    0.8660   -1.2321
         0         0    1.0000
ans =
   -1.8660   -1.2321  -30.0000
>>
```

**Problem 5:** The following frame definitions are given as known:

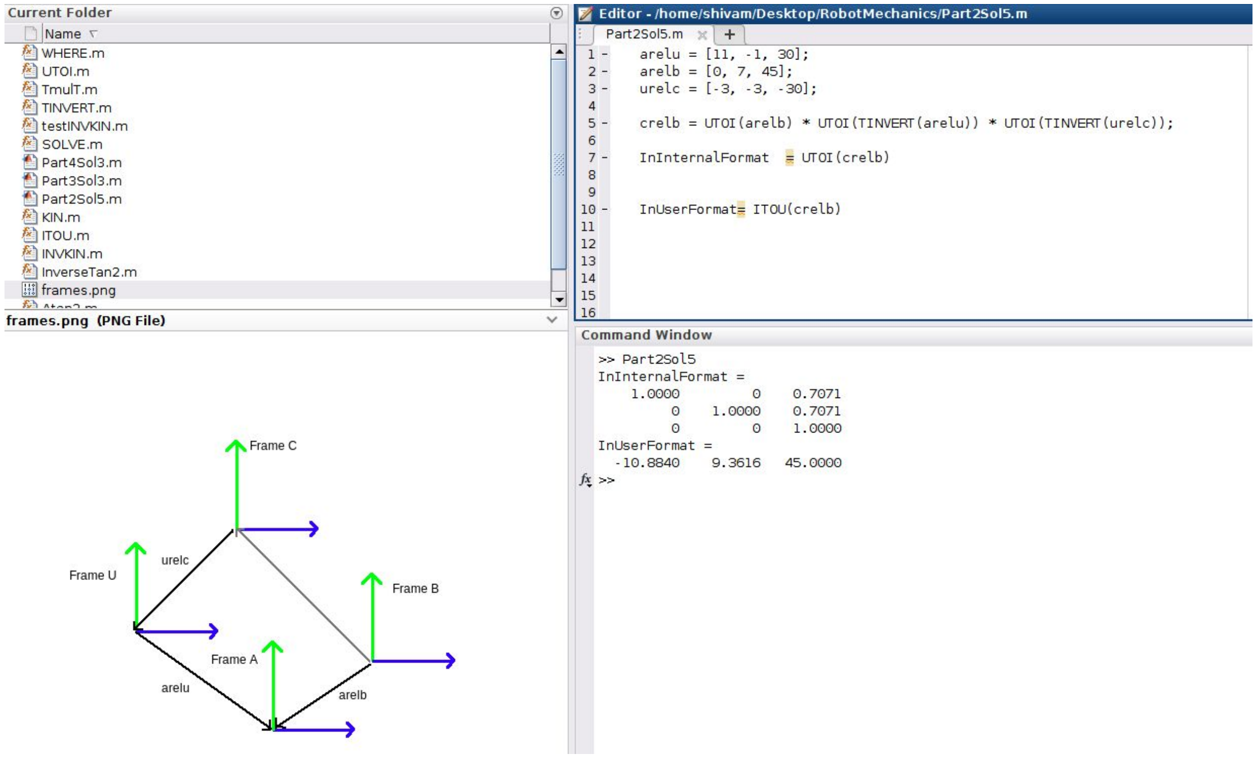
$$^{U}_{A}T = [x\ y\ \theta] = [11.0\ \ -1.0\ 30.0],$$

$$^{B}_{A}T = [x\ y\ \theta] = [0.0\ 7.0\ 45.0],$$

$$^{C}_{U}T = [x\ y\ \theta] = [-3.0\ \ -3.0\ \ -30.0].$$

These frames are input in the user representation [x, y, theta] (where theta is in degrees). Draw a frame diagram (like Fig. 2.15, only in 2-D) that qualitatively shows their arrangement. Write a program that calls TMIJLT and TINVERT (defined in programming exercises 3 and 4) as many times as needed to solve for crelb. Then print out crelb in both internal and user representation.

**Solution 5:** Here after understanding the frame relationships we can easily generate a plot. Then we can use functions created above to calculate the required transformations in both internal and user format.




CHAPTER 3:

**Problem 1:** Write a subroutine to compute the kinematics of the planar 3R robot in Example 3.3—that is, a routine with the joint angles' values as input, and a frame (the wrist frame relative to the base frame) as output. Use the procedure heading (or equivalent in C)

```
Procedure KIN(VAR theta: vec3; VAR wreib: franie);
```

where "wreib" is the wrist frame relative to the base frame, The type "frame" consists of a 2 x 2 rotation matrix and a 2 x 1 position vector. If desired, you may represent the frame with a 3 x 3 homogeneous transform in which the third row is [0 0 1]. (The manipulator data are l1 = l2 = 0.5 meters.)

**Solution 1:** We can easily create three vectors in user format using the values for theta1, theta2 and theta3 along with respective link lengths. And then we can use these three vectors to calculate overall transformation by matrix multiplication using UTOI function. Finally, we can return the result in user format using ITOU function.

Editor - /home/shivam/Desktop/RobotMechanics/KIN.m

KIN.m

```
function frame = KIN(vec3)
% vec3 is input vector of three angles
theta1 = vec3(1); theta2 = vec3(2); theta3 = vec3(3);
l1 = 0.5; l2 = 0.5;
% assumption
l3 = 0.0;

% conversion to degrees
t1 = theta1 * pi/180;
t2 = theta2 * pi/180;
t3 = theta3 * pi/180;


In1 = [l1*cos(t1), l1*sin(t1), theta1];
In2 = [l2*cos(t2), l2*sin(t2), theta2];
In3 = [l3*cos(t3), l3*sin(t3), theta3];

%frame is wrelb
frame = ITOU(UTOI(In1) * UTOI(In2) * UTOI(In3));
InInternalFormat = UTOI(frame)
end
```

Command Window

```
vec1 =
    30     0    45
>> KIN(vec1)
InInternalFormat =
    0.2588   -0.9659    0.8660
    0.9659    0.2588    0.5000
         0         0    1.0000
ans =
    0.8660    0.5000   75.0000
>>
```

**Problem 2:** Write a routine that calculates where the tool is, relative to the station frame. The input to the routine is a vector of joint angles:

```
Procedure WHERE(VAR theta: vec3; VAR trels: frame);
```

Obviously, WHERE must make use of descriptions of the tool frame and the robot base frame in order to compute the location of the tool relative to the station frame. The values of trelw and brels should be stored in global memory (or, as a second choice, you may pass them as arguments in WHERE).

**Solution 2:** Using standard frame diagram from the textbook and using frame relationships it is easy to compute position and orientation of tool relative to station frame .ie. trels as follows:

Note: I have passed trelw and brels as input arguments and output is in user format.

Editor - /home/shivam/Desktop/RobotMechanics/WHERE.m

WHERE.m | TINVERT.m | TmulT.m | UTOI.m | SOLVE.m | Part4Sol3.m | Part3

```
function frame = WHERE(Vec3, srelb,trelw)

%calculating wrelb by given vector of joint angles
%assuming l1 = 0.5;l2 = 0.5 in KIN function

%Frame gives trels
frame = ITOU(  UTOI(TINVERT(srelb)) * UTOI(KIN(Vec3)) * UTOI(trelw));

%InInternalFormat = UTOI(frame)
```

Command Window

```
>> srelb
srelb =
   -0.1000    0.3000         0
>> trelw
trelw =
    0.1000    0.2000   30.0000
>> v1
v1 =
     0    30    45
>>
>> WHERE(v1, srelb, trelw)
ans =
    0.8657    0.0984  105.0000
>>
```

**Problem 3:** A tool frame and a station frame for a certain task are defined as follows by the user:

$$ {}^{W}_{T}T = [x\ y\ \theta] = [0.1\ 0.2\ 30.0], $$
$$ {}^{B}_{S}T = [x\ y\ \theta] = [-0.1\ 0.3\ 0.0]. $$

Calculate the position and orientation of the tool relative to the station frame for the following three configurations (in units of degrees) of the arm:

$$ [\theta_1\ \theta_2\ \theta_3] = [0.0\ 90.0\ -90.0], $$
$$ [\theta_1\ \theta_2\ \theta_3] = [-23.6\ -30.3\ 48.0], $$
$$ [\theta_1\ \theta_2\ \theta_3] = [130.0\ 40.0\ 12.0]. $$

**Solution 3:** This problem to determine trels can be solved using previously defined function WHERE we and inputting different configurations of arm:

Configuration 1: [0 , 90 , -90]

Editor - /home/shivam/Desktop/RobotMechanics/Part3Sol3.m

Part3Sol3.m

```matlab
%% Configuration 1: [0 , 90 , -90]
trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [0 , 90 , -90];

disp("Position and Orientation in User Frame maintaining uniformity");
WHERE(Vec3, srelb, trelw)


%% Configuration 2: [-23.6 , -30.3 , 48.0]
trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [-23.6 , -30.3 , 48.0];

WHERE(Vec3, srelb, trelw)


%% Configuration 3: [130 , 40 , 12]

trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [130 , 40 , 12];

WHERE(Vec3, srelb, trelw)
```

Command Window

```
Position and Orientation in User Frame maintaining uniformity
ans =
    0.7000    0.4000   30.0000
>>
```

Configuration 2: [-23.6 , -30.3 , 48.0]

Editor - /home/shivam/Desktop/RobotMechanics/Part3Sol3.m

Part3Sol3.m

```
%% Configuration 1: [0 , 90 , -90]
trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [0 , 90 , -90];

disp("Position and Orientation in User Frame maintaining uniformity");
WHERE(Vec3, srelb, trelw)


%% Configuration 2: [-23.6 , -30.3 , 48.0]
trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [-23.6 , -30.3 , 48.0];

WHERE(Vec3, srelb, trelw)


%% Configuration 3: [130 , 40 , 12]

trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [130 , 40 , 12];

WHERE(Vec3, srelb, trelw)
```

Command Window

```
Position and Orientation in User Frame maintaining uniformity
ans =
    0.7000    0.4000   30.0000
ans =
    0.9728   -0.7155   24.1000
fx >>
```

Configuration 3: [130 , 40 , 12]

Part3Sol3.m

```matlab
%% Configuration 1: [0 , 90 , -90]
trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [0 , 90 , -90];

disp("Position and Orientation in User Frame maintaining uniformity");
WHERE(Vec3, srelb, trelw)


%% Configuration 2: [-23.6 , -30.3 , 48.0]
trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [-23.6 , -30.3 , 48.0];

WHERE(Vec3, srelb, trelw)


%% Configuration 3: [130 , 40 , 12]

trelw = [0.1 , 0.2 , 30.0];
srelb = [-0.1 , 0.3 , 0.0];
Vec3 = [130 , 40 , 12];

WHERE(Vec3, srelb, trelw)
```

Command Window

```
Position and Orientation in User Frame maintaining uniformity
ans =
    0.7000    0.4000   30.0000
ans =
    0.9728   -0.7155   24.1000
ans =
   -0.8068   -0.0335 -148.0000
>>
```

CHAPTER 4:

**Problem 1:** Write a subroutine to calculate the inverse kinematics for the three-link manipulator of Section 4.4. The routine should pass arguments in the form

```
Procedure INVKIN(VAR wreib: frame; VAR current, near, far: vec3; VAR
sol: boolean);
```

where "wreib," an input, is the wrist frame specified relative to the base frame; "current," an input, is the current position of the robot (given as a vector of joint angles); "near" is the nearest solution; "far" is the second solution; and "sol" is a flag that indicates whether solutions were found. (sol = FALSE if no solutions were found). The link lengths (meters) are where 11 = 12 = 0.5. The joint ranges of motion are [-170° ,170°] . Test your routine by calling it back-to-back with KIN to demonstrate that they are indeed inverses of one another.

**Solution 1:** following program is written to calculate two solutions for given manipulator:

Editor - /home/shivam/Desktop/RobotMechanics/INVKIN.m

INVKIN.m

```
function [near, far, sol] = INVKIN(wrelb, current)

% given
l1 = 0.5;l2 = 0.5;

% solving for theta 2
C2 = ( (wrelb(1) ^ 2) + (wrelb(2) ^ 2) - (l1^2 + l2^2) ) / (2*l1*l2);
S2 = sqrt(1-C2^2);

S2m = -abs(S2);
S2M = abs(S2);

%two solutions for theta 2
theta2m = atan2( S2m, C2);
theta2M = atan2( S2M, C2);

% now solving for theta 1 using two values for theta 2
    if ((wrelb(1) ^ 2) + (wrelb(2) ^ 2)) == 0
        S1m = 1;
        C1m = 0;
    else
        C1m = (((l1 + l1 * C2)*wrelb(1)) + (l2 * sin(theta2m) * wrelb(2)))  / ((wrelb(1) ^ 2) + (wrelb(2) ^ 2));
        S1m = sqrt(1-C1m^2);
    end

    if ((wrelb(1) ^ 2) + (wrelb(2) ^ 2)) == 0
        S1M = 0;
        C1M = 1;
    else
        S1M = (((l1 + l1 * C2)*wrelb(2)) - (l2 * sin(theta2M) * wrelb(1)))  / ((wrelb(1) ^ 2) + (wrelb(2) ^ 2));
        C1M = (((l1 + l1 * C2)*wrelb(1)) + (l2 * sin(theta2M) * wrelb(2)))  / ((wrelb(1) ^ 2) + (wrelb(2) ^ 2));
        %S1M = sqrt(1-C1M^2);
    end


    theta2m = atan2( S2m, C2) * 180/pi;
    theta2M = atan2( S2M, C2) * 180/pi;

    theta1m = atan2(S1m,C1m) * 180/pi;
    theta1M = atan2(S1M,C1M) * 180/pi;

    theta3m = wrelb(3) - (theta1m + theta2m);
    theta3M = wrelb(3) - (theta1M + theta2M);

    %Solution A
    A = [theta1m - current(1), theta2m - current(2) , theta3m - current(3)];
```

First two solutions for theta2 are calculated and then two solutions are calculated using those two

solutions. Also few conditions are checked such that cosine and sine values for both angles (theta1 and theta2) are defined.
Further theta3 is calculated using values of theta1, theta2 and phi.

Editor - /home/shivam/Desktop/RobotMechanics/INVKIN.m

INVKIN.m

```
    %Solution A
    A = [theta1m - current(1), theta2m - current(2) , theta3m - current(3)];
    for i = (1:3) ...
    %disp('sol for A is '); %for part3
    %disp(A); %for part3

    if ((170 >= A(1) && A(1) >= -170) && (170 >= A(2) && A(2) >= -170) && (170 >= A(3) && A(3)>= -170))
        sola = 1;
       % A = [theta1m - current(1), theta2m - current(2) , theta3m - current(3)];
    else
        sola = 0;
        %disp('Invalid Solution for A')
        A = [0 0 0];
    end

    %Solution B
    B = [theta1M - current(1), theta2M - current(2), theta3M - current(3)];

    for i = (1:3)
        if B(i)>= 180
            B(i) = 360 - B(1);
        elseif B(i) < -180
            B(i) = 360 + B(i);
        end
    end
    %disp('sol for B is '); %for part3
    %disp(B); %for part3

    if ((170 >= B(1) && B(1) >= -170) && (170 >= B(2) && B(2) >= -170) && (170 >= B(3) && B(3) >= -170))
        solb = 1;
       % B = [theta1M - current(1), theta2M - current(2), theta3M - current(3)];
    else
        solb=0;
        disp('Invalid Solution for B');
        B=[0,0,0];
    end
```

```
    %Considering only theta1 to compare for far and near solution
    if (sola == 0 && solb == 0)
        near = [0 0 0];
        far = [0 0 0];
        sol = 0;
    elseif (sola > solb) %sola=1 solb=0
        near = A;
        far = B;
        sol = 1;
    elseif (solb > sola) %sola=0 solb=1
        near = B;
        far = A;
        sol = 1;
    else %sola=1 solb=1
        if (abs(B(1)) > abs(A(1)))
            near = A;
            far = B;
            sol=1;
        else
            near = B;
            far = A;
            sol=1;
        end
    end

end
```

Then two solutions ( Solution A and Solution B) are checked for joint limits and solution not following given constraints is/are discarded.

Check with KIN function:

INVKIN.m

```matlab
function [near, far, sol] = INVKIN(wrelb, current)

% given
l1 = 0.5;l2 = 0.5;

% solving for theta 2
C2 = ( (wrelb(1) ^ 2) + (wrelb(2) ^ 2) - (l1^2 + l2^2) ) / (2*l1*l2);
S2 = sqrt(1-C2^2);

S2m = -abs(S2);
S2M = abs(S2);

%two solutions for theta 2
theta2m = atan2( S2m, C2);
theta2M = atan2( S2M, C2);

% now solving for theta 1 using two values for theta 2
    if ((wrelb(1) ^ 2) + (wrelb(2) ^ 2)) == 0
        S1m = 1;
        C1m = 0;
    else
        C1m = (((l1 + l1 * C2)*wrelb(1)) + (l2 * sin(theta2m) * wrelb(2))) / ((w
        S1m = sqrt(1-C1m^2);
    end

    if ((wrelb(1) ^ 2) + (wrelb(2) ^ 2)) == 0
```

Command Window

```
>> current
current =
     0     0     0
>> wrelb
wrelb =
    0.1000    0.5000   45.0000
>> INVKIN(wrelb, current)
ans =
   19.3474  118.6854  -93.0328
>>
>>
>> KIN(ans)
ans =
    0.1000    0.5000   45.0000
>>
```

Since, output for KIN function (when input is output of INVKIN function) is same as input argument wrelb to INVKIN function when current frame = [0 0 0]. Hence, the function is performing well.

**Problem 2:** tool is attached to link 3 of the manipulator. This tool is described by the tool frame relative to the wrist frame. Also, a user has described his work area, the station frame relative to the base of the robot, as srelb. Write the subroutine

```
Procedure SOLVE(VAR -brels: frame; VAR current, near, far: vec3;
VAR sol: boolean);
```

Where, "trels" is {T} frame specified relative to the {S} frame. Other parameters INVKIN subroutine. The definitions of {T} and {S} should be defined variables or constants. SOLVE should use calls to TMULT, TIN VERT, where "trels" is the are exactly as in the globally and INVKIN.

**Solution 2:** Using previously defined functions, SOLVE function can be easily written as:

Editor - /home/shivam/Desktop/RobotMechanics/SOLVE.m

INVKIN.m | SOLVE.m

```
function [near,far, sol] = SOLVE(trels, current, srelb, trelw)

wrelb = TmulT( (TmulT(srelb, trels)) , TINVERT(trelw));
[near, far, sol] = INVKIN(wrelb, current);
```

Here we are just solving for wrelb using different frame relationships as done in chapter 2 Problem 5. And then we are using INVKIN function to find near, far solutions.

**Problem 3:** Write a main program that accepts a goal frame specified in terms of x, y, and theta. This goal specification is (T} relative to {S}, which is the way the user wants to specify goals. The robot is using the same tool in the same working area as in Programming Exercise (Part 2), so {T} and {S} are defined as

$$ {}^{W}_{T}T = [x\ y\ \theta] = [0.1\ 0.2\ 30.0], $$

$$ {}^{B}_{S}T = [x\ y\ \theta] = [-0.1\ 0.3\ 0.0]. $$

Calculate the joint angles for each of the following three goal frames:

$$ [x_1\ y_1\ \phi_1] = [0.0\ 0.0\ \ -90.0], $$

$$ [x_2\ y_2\ \phi_2] = [0.6\ \ -0.3\ 45.0], $$

$$ [x_3\ y_3\ \phi_3] = [-0.4\ 0.3\ 120.0], $$

$$ [x_4\ y_4\ \phi_4] = [0.8\ 1.4\ 30.0]. $$

Assume that the robot wifi start with all angles equal to 0.0 and move to these three goals in sequence. The program should find the nearest solution with respect to the previous goal point.

You should call SOLVE and WHERE back-to-back to make sure they are truly inverse functions.

**Solution 3:** here we can use modified SOLVE function to get required pose with respect to changing “current” frame. SOLVE is modified to check that required goal configuration does not exceed x, y limits .ie. x or y > max joint length possible.

Modified SOLVE:

Editor - /home/shivam/Desktop/RobotMechanics/SOLVE.m

SOLVE.m | Part4Sol3.m | INVKIN.m

```
function [near,far, sol] = SOLVE(trels, current, srelb, trelw)

if trels(1) >1 || trels(2) > 1
     disp("No Near or Far Solutions");
     disp ("Solution = FALSE");
      near = [0 0 0];
      far = [0 0 0];
      sol = 0;
else
    wrelb = TmulT( (TmulT(srelb, trels)) , TINVERT(trelw));
    [near, far, sol] = INVKIN(wrelb, current);

end
```

For first Goat configuration: **[ 0 0 -90]**

SOLVE.m | Part4Sol3.m

```
%% Goal1: trels  [0 0 -90]
current = [0 0 0];
disp('Current =');
disp(current);
trels = [0 0 -90];
srelb = [-0.1, 0.3 0];
trelw = [0.1 0.2 30];

current = SOLVE(trels, current, srelb, trelw);
disp('new pose =');
disp(current);


%% Goal2: trels [0.6 -0.3 45]
disp('Current =');
disp(current);
trels = [0.6 -0.3 45];
srelb = [-0.1, 0.3 0];
trelw = [0.1 0.2 30];

current = SOLVE(trels, current, srelb, trelw);
disp('new pose =');
disp(current);

```

Command Window

```
Current =
     0     0     0
new pose =
   57.0088   115.2643   67.7269
>>
```

For second Goal configuration: **[ 0.6 -0.3  45 ]**

SOLVE.m    Part4Sol3.m    +

```matlab


%% Goal2: trels [0.6 -0.3 45]
disp('Current =');
disp(current);
trels = [0.6 -0.3 45];
srelb = [-0.1, 0.3 0];
trelw = [0.1 0.2 30];

current = SOLVE(trels, current, srelb, trelw);
disp('new pose =');
disp(current);



%% Goal3: trels = [-0.4 0.3 120]
disp('Current =');
disp(current);
trels = [-0.4 0.3 120];
srelb = [-0.1, 0.3 0];
trelw = [0.1 0.2 30];

current = SOLVE(trels, current, srelb, trelw);
disp('new pose =');
disp(current);



%% Goal4: trels = [0.8 1.4 30]
```

Command Window

```
Current =
     0     0     0
new pose =
   57.0088  115.2643   67.7269
>>
>>
Current =
   57.0088  115.2643   67.7269
new pose =
  -23.0506  125.4176   32.6330
>>
```

For third Goal configuration: **[ -0.4  0.3  30 ]**

Editor - /home/shivam/Desktop/RobotMechanics/Part4Sol3.m

SOLVE.m | Part4Sol3.m | +

```matlab
    disp('new pose =');
    disp(current);



    %% Goal3: trels = [-0.4 0.3 120]
    disp('Current =');
    disp(current);
    trels = [-0.4 0.3 120];
    srelb = [-0.1, 0.3 0];
    trelw = [0.1 0.2 30];

    current = SOLVE(trels, current, srelb, trelw);
    disp('new pose =');
    disp(current);



    %% Goal4: trels = [0.8 1.4 30]
    disp('Current =');
    disp(current);
    trels = [0.8 1.4 30];
    srelb = [-0.1, 0.3 0];
    trelw = [0.1 0.2 30];

    output = SOLVE(trels, current, srelb, trelw);
```

Command Window

```
Current =
     0     0     0
new pose =
   57.0088  115.2643   67.7269
Current =
   57.0088  115.2643   67.7269
new pose =
  -23.0506  125.4176   32.6330
>>
>>
>>
Current =
  -23.0506  125.4176   32.6330
new pose =
   89.6829  -16.7547 -117.9282
>>
>>
```

For Final Goal configuration: **[ 0.8  1.4  30 ] >>> No Solutions found**

Editor - /home/shivam/Desktop/RobotMechanics/Part4Sol3.m

SOLVE.m | Part4Sol3.m | INVKIN.m

```
disp('Current =');
disp(current);
trels = [-0.4 0.3 120];
srelb = [-0.1, 0.3 0];
trelw = [0.1 0.2 30];

current = SOLVE(trels, current, srelb, trelw);
disp('new pose =');
disp(current);



%% Goal4: trels = [0.8 1.4 30]
disp('Current =');
disp(current);
trels = [0.8 1.4 30];

srelb = [-0.1, 0.3 0];
trelw = [0.1 0.2 30];

output = SOLVE(trels, current, srelb, trelw);
disp('new pose =');
disp(output);
```

Command Window

```
  -23.0506  125.4176   32.6330
>>
>>
>>
Current =
  -23.0506  125.4176   32.6330
new pose =
   89.6829  -16.7547 -117.9282
>>
>>
>>
Current =
   89.6829  -16.7547 -117.9282
No Near or Far Solutions
Solution = FALSE
new pose =
     0     0     0
>>
```

******************************** **END OF REPORT** ****************************