Lab 5 Exercises

1 Help the Robot Blaster save the Planet from UFOs!

1.1 Ensure you have the latest modified toolbox available on UTSOnline and download *CheckIntersections.m.*

1.2 Create and plot a UFO Fleet of 10 ships

```
ufoFleet = UFOFleet(10);
```

1.3 Create the blaster robot. Note this is the actual "Grit Blasting"[1] robot working on the Sydney Harbor Bridge

```
blasterRobot = SchunkUTSv2_0();
plot3d(blasterRobot,zeros(1,6));
endEffectorTr = blasterRobot.fkine(zeros(1,6));
```

1.4 Now plot a "blast" cone coming out the end effector (assume it is the Z axis of the end effector.

```
[X,Y,Z] = cylinder([0,0.1],6);
Z = Z * 10;
updatedConePoints = [endEffectorTr * [X(:),Y(:),Z(:),ones(numel(X),1)]']';
conePointsSize = size(X);
cone_h = surf(reshape(updatedConePoints(:,1),conePointsSize) ...
             ,reshape(updatedConePoints(:,2),conePointsSize) ...
             ,reshape(updatedConePoints(:,3),conePointsSize));
```

1.5 Now plot a "score board"

```
currentScore = 0;
scoreZ = ufoFleet.workspaceDimensions(end)*1.2;
text_h = text(0, 0, scoreZ,sprintf('Score: 0 after 0 seconds'), 'FontSize', 10, 'Color', [.6 .2 .6]);
```

1.6 Add the following "while loop" to iteratively call your function

```
tic;
% Go through iterations of randomly move UFOs, then move robot. Check for hits and update score and timer
while ~isempty(find(0 < ufoFleet.healthRemaining,1))
    ufoFleet.PlotSingleRandomStep();
    % Get the goal joint state
    goalJointState = GetGoalJointState(blasterRobot,ufoFleet);

    % Fix goal pose back to a small step away from the min/max joint limits
    fixIndexMin = goalJointState' < blasterRobot.qlim(:,1);
    goalJointState(fixIndexMin) = blasterRobot.qlim(fixIndexMin,1) + 10*pi/180;
    fixIndexMax = blasterRobot.qlim(:,2) < goalJointState';
    goalJointState(fixIndexMax) = blasterRobot.qlim(fixIndexMax,2) - 10*pi/180;

    % Get a trajectory
    jointTrajectory = jtraj(blasterRobot.getpos(),goalJointState,8);
    for armMoveIndex = 1:size(jointTrajectory,1)
        animate(blasterRobot,jointTrajectory(armMoveIndex,:));

        endEffectorTr = blasterRobot.fkine(jointTrajectory(armMoveIndex,:));
        updatedConePoints = [endEffectorTr * [X(:),Y(:),Z(:),ones(numel(X),1)]']';
        set(cone_h,'XData',reshape(updatedConePoints(:,1),conePointsSize) ...
                  ,'YData',reshape(updatedConePoints(:,2),conePointsSize) ...
                  ,'ZData',reshape(updatedConePoints(:,3),conePointsSize));

        coneEnds = [cone_h.XData(2,:)', cone_h.YData(2,:)', cone_h.ZData(2,:)'];
        ufoHitIndex = CheckIntersections(endEffectorTr,coneEnds,ufoFleet);
        ufoFleet.SetHit(ufoHitIndex);
        currentScore = currentScore + length(ufoHitIndex);

        text_h.String = sprintf(['Score: ',num2str(currentScore),' after ',num2str(toc),' seconds']);
        axis([-6,6,-6,6,0,10]);
        % Only plot every 3rd to make it faster
        if mod(armMoveIndex,3) == 0
            drawnow();
        end
    end
    drawnow();
end
```

1.7 Create a file called "GetGoalJointState.m" which is called by the highlighted line above. The function must take the parameters "blasterRobot" and "ufoFleet" and use these to determine and return "goalJointState"

1.8 Here is one solution to put in GetGoalJointState.m that randomly picks a pose [2]. After running it 200 times the results were as follows: Average time = 463 secs, Average score = 224 points.

```
goalJointState = blasterRobot.getpos() + (rand(1,6)-0.5) * 20*pi/180;
endEffectorTr = blasterRobot.fkine(goalJointState);

% Ensure the Z component of the Z axis is positive (pointing upwards), and the Z component of the point is above 1 (approx mid height)
```

[1] http://www.bbc.co.uk/news/world-asia-23373095
[2] https://www.youtube.com/watch?v=9fL8l193FLY

```
while endEffectorTr(3,3) < 0.1 || endEffectorTr(3,4) < 1
    goalJointState = blasterRobot.getpos() + (rand(1,6)-0.5) * 20*pi/180;
    endEffectorTr = blasterRobot.fkine(goalJointState);
    display('trying again');
end
```

1.9 Write a better solution that uses **<u>ikine</u>** or **<u>ikcon</u>** like in earlier exercises, such that you get consistently faster (and higher scoring) results than the above random method.

2 Simple collision checking for 3-link planar robot

2.1 Create a 3 link planar robot with all 3 links having a = 1m, leave the base at eye(4).

2.2 Put a cube with sides 1.5m in the environment that is centered at [2,0,-0.5].

2.3 Use teach and note when the links of the robot can collide with 4 of the planes:

- Plane 1: point [1.25,0,-0.5] normal [-1,0,0]
- Plane 2: point [2,0.75,-0.5] normal [0,1,0]
- Plane 3: point [2,-0.75,-0.5] normal [0,-1,0]
- Plane 4: point [2.75,0,-0.5] normal [1,0,0]

2.4 Using your understanding of forward kinematics write a function that you can pass in vector of joint angles, q, to represent a joint state of the arm, and it will return a 4x4x4 matrix, TR which contains

TR(:,:,1) = arm.base	TR(:,:,2) = Transform at end of link 1	TR(:,:,2) = Transform at end of link 2	TR(:,:,3) = arm.fkine(q)

2.5 Use *LinePlaneIntersection.m* to check if any of the links (i.e. the link n intersects with any of the 4 planes of the cube when q = [0,0,0]. Note how the link *n* is a line from position TR(1:3,4,*n*-1) to TR(1:3,4,*n*)

2.6 Use *jtraj(q1,q2,steps)* to get a trajectory from q1 = [-pi/4,0,0] to q2 = [pi/4,0,0] and pick a value for *steps* such that the size of each step is less than 1 degree. Hint: look at the step size in degrees using the following

```
diff(rad2deg(jtraj(q1,q2,steps)))
```

2.7 Check each of the joint states in the trajectory to work out which ones are in collision. Return a logical vector of size *steps* which contains 0 = no collision (safe) and 1 = yes collision (Unsafe). You may like to use this structure.

```
result = true(steps,1)
for i = 1: steps
    result(i) = CollisionCheck(robot,q1,q2);
end
```

3 Basic collision avoidance for 3-link planar robot

Determine a path from pose q1 = [-pi/4,0,0] degrees to q2 = [pi/4,0,0] that doesn't collide with the cube from previous question. Use the following 3 methods

3.1 Method 1: Manually determine intermediate joint states that are not in collision with the cube using teach. Then make a path that goes between way points

3.2 Method 2: Manually determine Cartesian points (i.e. [x,y,z] points) that the end effector could follow such that the end effector does not go inside the cube

3.3 Method 3: Now, iteratively, randomly and automatically pick a pose within the joint angle bounds

```
q = (2 * rand(1,3) - 1) * pi
```

3.4 Then interpolate between current pose and this new pose. If all the *results* are equal to 0 then the path is collision free

```
all(~results) == true
```

3.5 At each step try and connect from the current joint state to the final goal state. Keep upon concatenating the joint trajectory until you can reach the goal