Lab 1: Time and Motion

Exercises

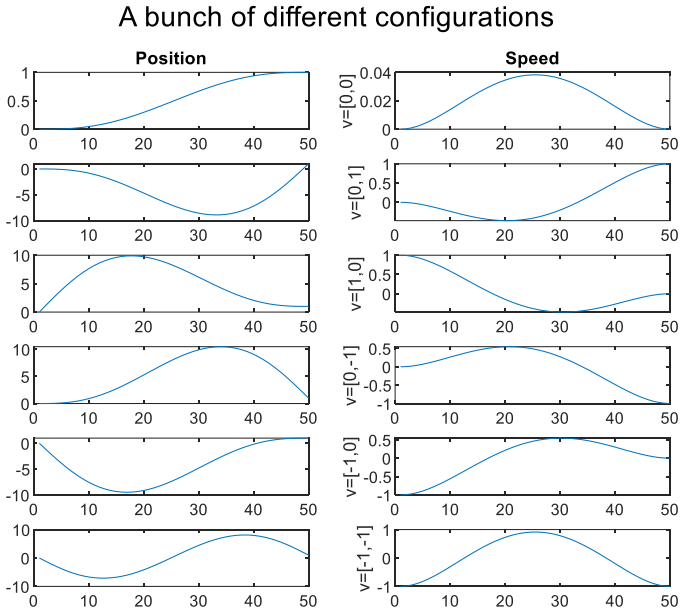
Tasks

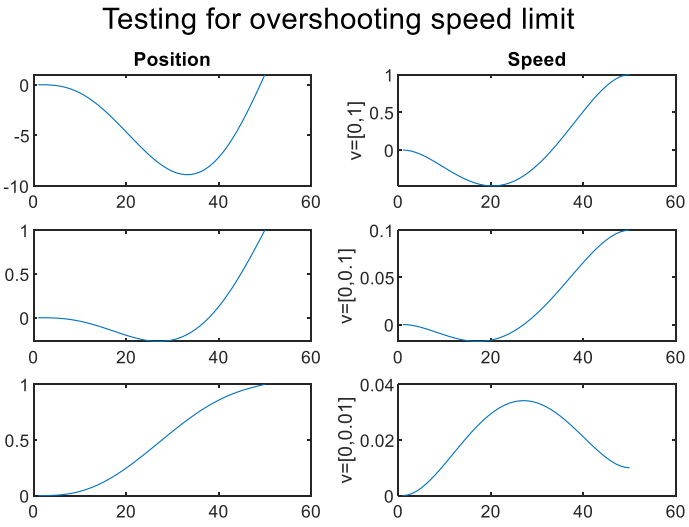
Exercise 7

Position overshooting happens:

- When either initial or final speeds are negative: because the robot starts/finishes its trajectory going backwards.

- When either initial or final speeds are too high: if the initial speed is to high the robot will overpass the destiny and will have to come back. If the final speed is too high the robot may need to ”take a step back” at the beginning so it has enough space to gain the required speed.





```
fprintf('$ Running exercise 7')


figure(); sgtitle('A bunch of different configurations')
[s, sd, sdd] = tpoly(0, 1, 50, 0, 0);
subplot(6, 2, 1); plot(s); title('Position')
subplot(6, 2, 2); plot(sd); ylabel('v=[0,0]'); title('Speed')

[s, sd, sdd] = tpoly(0, 1, 50, 0, 1);
subplot(6, 2, 3); plot(s)
subplot(6, 2, 4); plot(sd); ylabel('v=[0,1]')

[s, sd, sdd] = tpoly(0, 1, 50, 1, 0);
subplot(6, 2, 5); plot(s)
subplot(6, 2, 6); plot(sd); ylabel('v=[1,0]')

[s, sd, sdd] = tpoly(0, 1, 50, 0, -1);
subplot(6, 2, 7); plot(s)
subplot(6, 2, 8); plot(sd); ylabel('v=[0,-1]')

[s, sd, sdd] = tpoly(0, 1, 50, -1, 0);
subplot(6, 2, 9); plot(s)
subplot(6, 2, 10); plot(sd);ylabel('v=[-1,0]')

[s, sd, sdd] = tpoly(0, 1, 50, -1, -1);
subplot(6, 2, 11); plot(s)
subplot(6, 2, 12); plot(sd); ylabel('v=[-1,-1]')



figure(); sgtitle('Testing for overshooting speed limit')
[s, sd, sdd] = tpoly(0, 1, 50, 0, 1);
subplot(3, 2, 1); plot(s);  title('Position')
subplot(3, 2, 2); plot(sd); ylabel('v=[0,1]'); title('Speed')

[s, sd, sdd] = tpoly(0, 1, 50, 0, 0.1);
subplot(3, 2, 3); plot(s)
subplot(3, 2, 4); plot(sd); ylabel('v=[0,0.1]')

[s, sd, sdd] = tpoly(0, 1, 50, 0, 0.01);
subplot(3, 2, 5); plot(s)
subplot(3, 2, 6); plot(sd); ylabel('v=[0,0.01]')
```

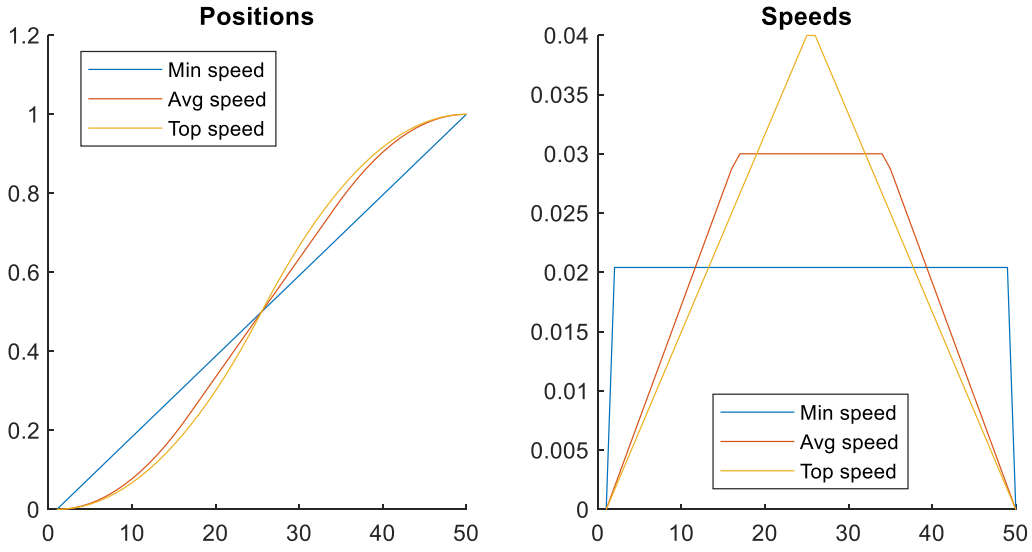
Exercise 8

LSPB starts working at a maximum speed of 2.041000e-02 units/sec and stops working at a maximum speed of 4.081000e-02 units/sec.

The specified velocity sets how high the acceleration is allowed to be:

- Going at the same speed the whole time sets the minimum possible max speed for the algorithm (~infinite acceleration at start and finish).

- Reaching top speed just for a moment then having to decrease sets the maximum possible max speed for the algorithm (~minimum possible acceleration).

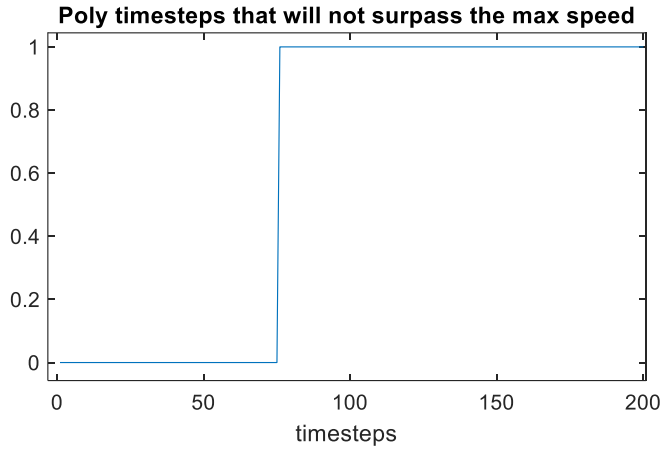


```
fprintf('\n$ Running exercise 8\n')

% THESE BOUNDS ONLY WORK FOR THIS SPECIFIC TRAJECTORY 0 1 50
for i = 0.02:0.00001:0.5
    try
        [s,sd,sdd] = lspb(0, 1, 50, i);
        fprintf('lspb starts working at topspeed %d \n', i)
        break
    catch
    end
end

% THESE BOUNDS ONLY WORK FOR THIS SPECIFIC TRAJECTORY 0 1 50
for i = 0.03:0.00001:0.5
    try
        [s,sd,sdd] = lspb(0, 1, 50, i);
    catch
        fprintf('lspb stops working at topspeed %d \n', i-0.00001)
        break
    end
end


% Speed limit configurations. Hardcoded for readability + code economy
[s_low,sd_low,sdd] = lspb(0, 1, 50, 2.041000e-02);
[s_top,sd_top,sdd] = lspb(0, 1, 50, 4.081000e-02);

% some intermediate configuration
[s_avg,sd_avg,sdd] = lspb(0, 1, 50, 3e-02);


figure()
subplot(1,2,1); hold on ; title('Positions')
plot(s_low); plot(s_avg); plot(s_top)
legend('Min speed','Avg speed','Top speed')


subplot(1,2,2); hold on; title('Speeds')
plot(sd_low); plot(sd_avg); plot(sd_top)
legend('Min speed','Avg speed','Top speed')
```

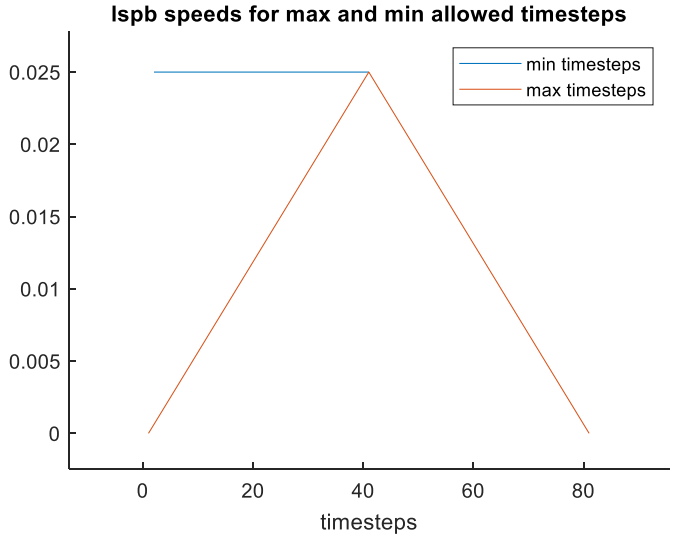
Exercise 9

- Poly works for more than 76 steps.
- LSPB starts working at 41 steps and stops working at 81 steps.

Poly trajectory needs a minimum of 2 points or a warning will pop. A trajectory of 1 point only doesn't make sense anyway. To reach the destination without surpassing max speed it also needs a minimum of time steps. The higher the max speed the less timesteps it allows, because higher speeds require less time to reach the target.



LSPB has both upper and lower bounds for timesteps. The minimum allowed number of timesteps matches the required time for reaching the destination at constant max speed. The maximum allowed number of timesteps happens when there is so much time available the robot cannot (doesn't need to) reach the maximum speed at all.

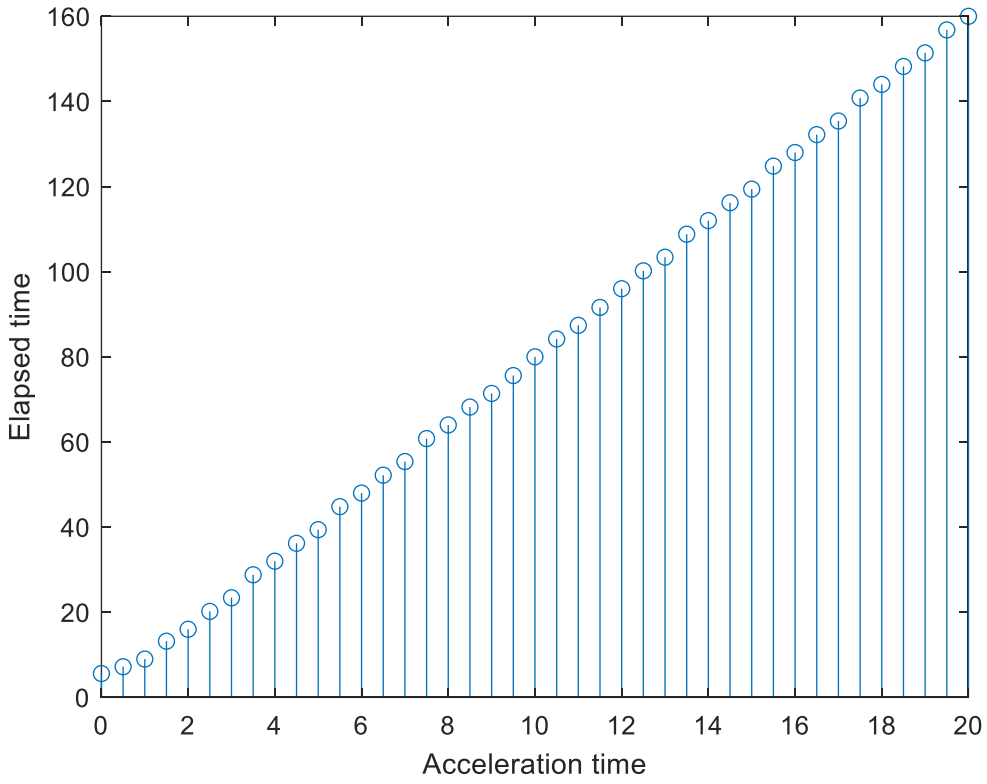


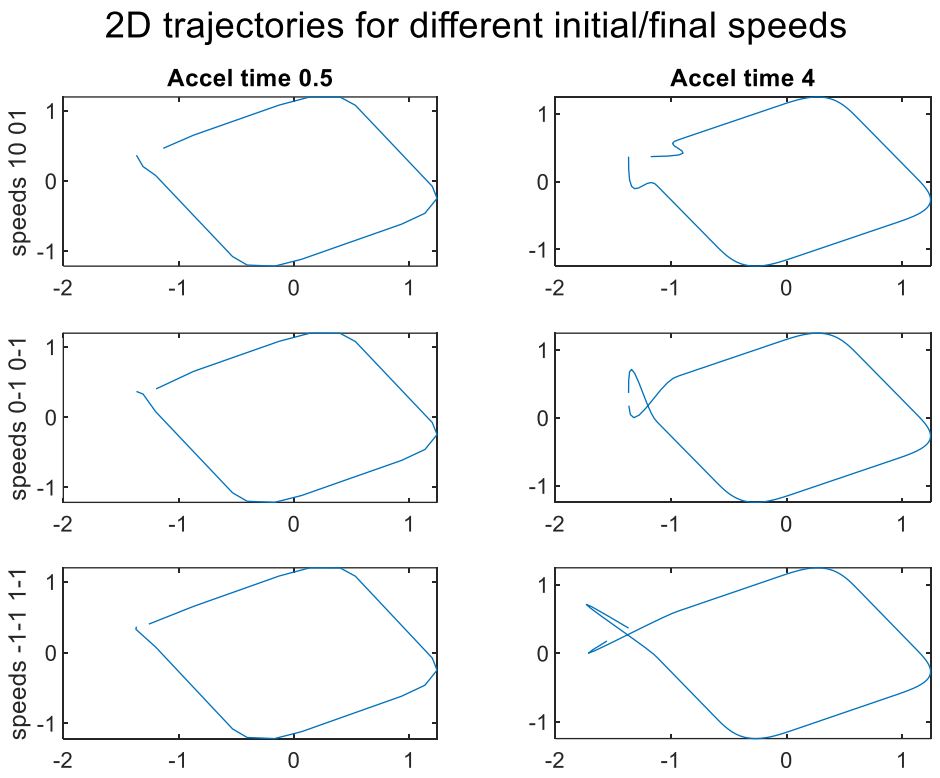
```
fprintf('\n$ Running exercise 9\n')

speed = 0.025

% TPOLY: run for multiple steps and check which ones broke max speed
% skiping 1&2 because singularities were found
i0 = 3
godspeed = [99 99]
for i = i0:1:200
    [~, sd, ~] = tpoly(0, 1, i, 0, 0);
    godspeed = [godspeed max(sd)];
end

working_steps = godspeed <= speed;
a = find(working_steps)
fprintf('1: Poly works for more than %d\n', a(1))

figure(); plot(working_steps)
title('Poly timesteps that will not surpass the max speed')
xlabel('timesteps')



% LSPB: try at which numbers it starts and stops working
for i = 1:1:100
    try
        [~,sd_low,~] = lspb(0, 1, i, speed);
        fprintf('2: lspb starts working at %d steps \n', i)
        break
    catch
    end
end

for i = i:1:100
    try
        [~,sd_top,~] = lspb(0, 1, i, speed);
    catch
        fprintf('3: lspb stops working at %d steps \n', i-1)
        break
    end
end

figure(); hold on
plot(sd_low); plot(sd_top);
legend('min timesteps','max timesteps')
title('lspb speeds for max and min allowed timesteps')
xlabel('timesteps')
```

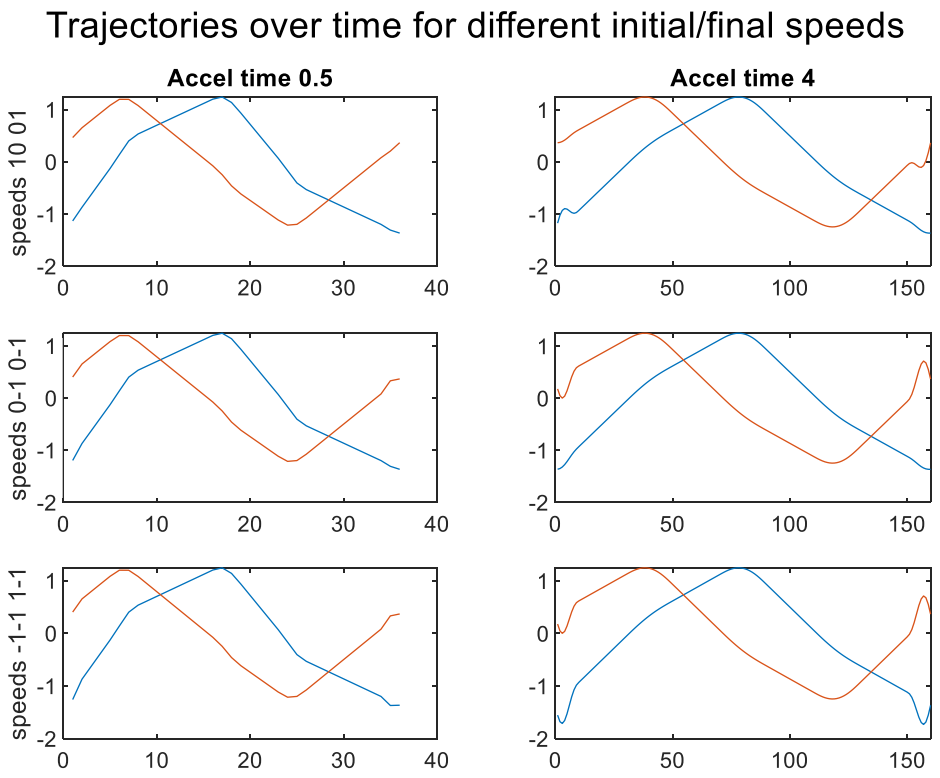
Exercise 10

Changing the initial speeds may force the robot to start moving in an incorrect direction, thus creating the need for a path correction and consuming extra time. Same goes for the final speed, if its direction doesn't match. If it does, the trajectory may be faster because the robot does not have to slow down.

Increasing the acceleration time the robot makes use of more time for drifting smoothly from one trajectory to the other, thus taking more time to reach its destination. The elapsed time grows linearly with the allowed acceleration time.



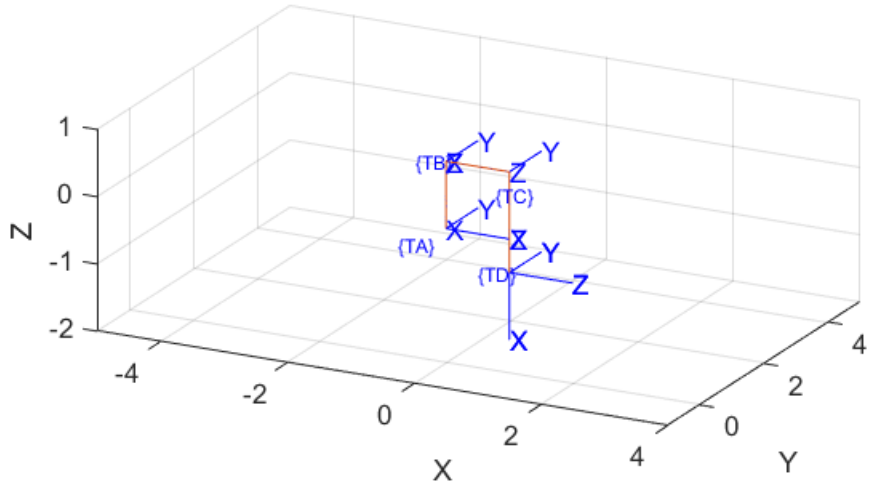

2D trajectories for different initial/final speeds
Accel time 0.5
Accel time 4
speeds 10 01
speeds 0-1 0-1
speeds -1-1 1-1


Trajectories over time for different initial/final speeds
Accel time 0.5
Accel time 4
speeds 10 01
speeds 0-1 0-1
speeds -1-1 1-1

```
fprintf('\n$ Running exercise 13\n')

via = SO2(30, 'deg') * [-1 1; 1 1; 1 -1; -1 -1]';
q0 = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 0);

figure();
% XY plot for different speeds & different accel. times
q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 0.5, [1 0], [0 1]);
subplot(3,2,1); plot(q(:,1), q(:,2)); title('Accel time 0.5'); ylabel('speeds
10 01')
q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 4, [1 0], [0 1]);
subplot(3,2,2); plot(q(:,1), q(:,2)); title('Accel time 4')

q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 0.5, [0 -1], [0 -
1]);
subplot(3,2,3); plot(q(:,1), q(:,2)); ylabel('speeds 0-1 0-1')
q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 4, [0 -1], [0 -1]);
subplot(3,2,4); plot(q(:,1), q(:,2))

q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 0.5, [-1 -1], [1 -
1]);
subplot(3,2,5); plot(q(:,1), q(:,2)); ylabel('speeds -1-1 1-1')
q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 4, [-1 -1], [1 -1]);
subplot(3,2,6); plot(q(:,1), q(:,2))


% tX plot for different speeds & different accel. times
figure(); hold on
q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 0.5, [1 0], [0 1]);
subplot(3,2,1); plot(q); title('Accel time 0.5'); ylabel('speeds 10 01')
q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 4, [1 0], [0 1]);
subplot(3,2,2); plot(q); title('Accel time 4')

q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 0.5, [0 -1], [0 -
1]);
subplot(3,2,3); plot(q); ylabel('speeds 0-1 0-1')
q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 4, [0 -1], [0 -1]);
subplot(3,2,4); plot(q)

q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 0.5, [-1 -1], [1 -
1]);
subplot(3,2,5); plot(q); ylabel('speeds -1-1 1-1')
q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', 0.2, 4, [-1 -1], [1 -1]);
subplot(3,2,6); plot(q)


% total time vs acceleration time
times = []
timestep = 0.2
for i = 0:0.5:20
    q = mstraj(via(:,[2 3 4 1])', [2,1], [], via(:,1)', timestep, i);
    times = [times numrows(q)*timestep];
end
figure();
stem(0:0.5:20, times)
xlabel('Acceleration time')
ylabel('Elapsed time')
```

Task 1

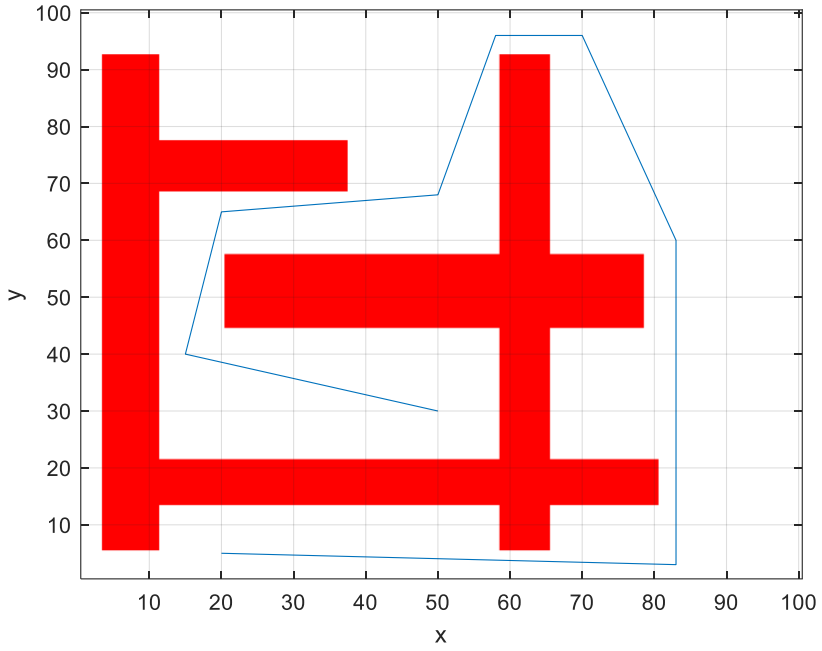
The following picture shows the intermediate coordinate frames and the moving one, now coincident with the coordinate frame D. Please do check the animation by running the program.

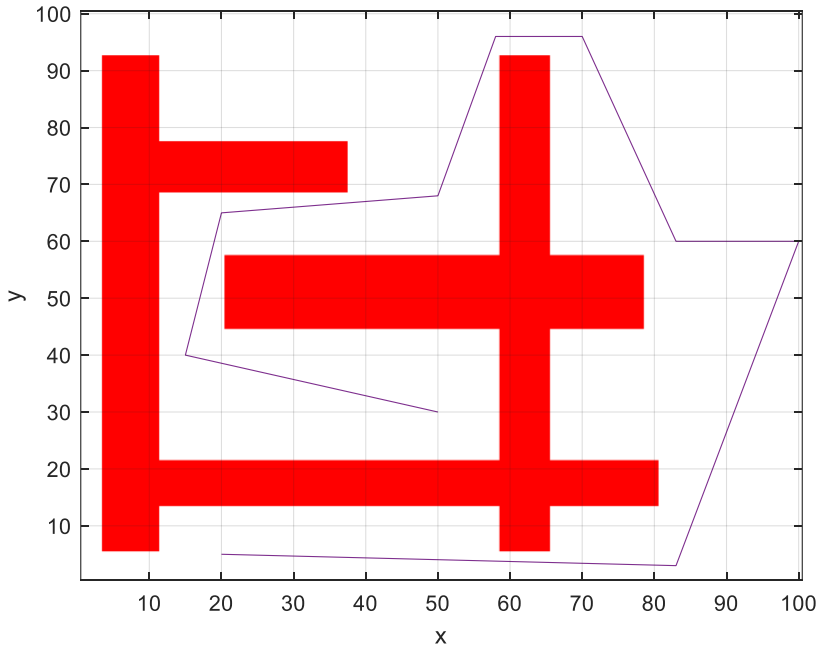



```
% Defining frames
TA = SE3(0, 0, 0)
TB = SE3(0, 0, 1) * SE3.Ry(pi/2)
TC = SE3(1, 0, 1) * SE3.Ry(-pi)
TD = SE3(1,0,-0.5) * SE3.Ry(pi/2)

% Check everything was properly defined
plotvol([-5 4 -1 5])
trplot(TA, 'framelabel', 'TA')
trplot(TB, 'framelabel', 'TB')
trplot(TC, 'framelabel', 'TC')
trplot(TD, 'framelabel', 'TD')

% Compute subtrajectories
Tab = ctraj(TA, TB, 50);
Tbc = ctraj(TB, TC, 50);
Tcd = ctraj(TC, TD, 50);

% Concatenate trajectories
TT = [Tab Tbc Tcd]

% Plot animation
TT.animate

% Extract & plot trajectory
traj = TT.transl
plot3(traj(:,1), traj(:,2), traj(:,3))
```

Task 2

The trajectory was defined by a series of waypoints joined by straight moves. To avoid the door, the robot moves to the right to wait for a few instants and crosses it once it opens (timed approach).

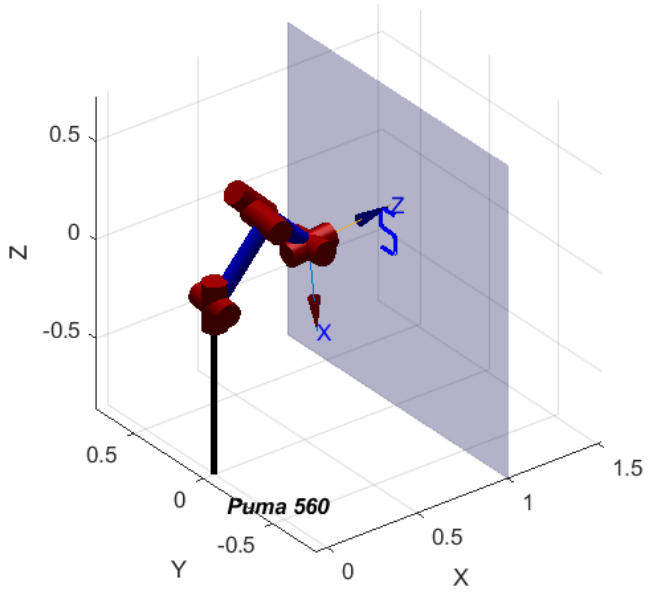




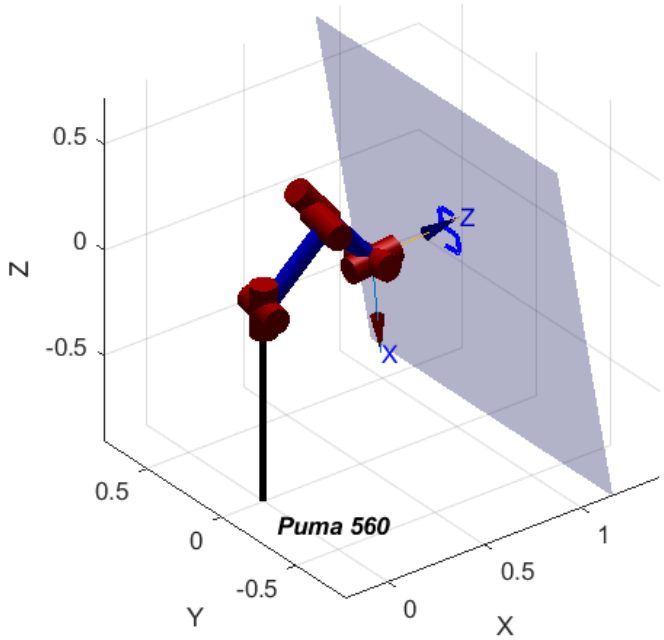
```
%% clear the workspace variables and close all figures
clear all
clc
close all
%% Load the map
load map1
about map
%% plot the map
plot(Bug2(map))
%% Initial position and heading of the robot
p_init=[50 30]; % [x,y]
%% Goal location of the robot
p_goal=[20 5]; % [x,y]

%% Define the desired path
obstacle=1; %--------------------------set obstacle here
if obstacle == 0
    path=[ p_init;...
           15 40 ;...
           20 65 ;...
           50 68 ;...
           58 96 ;...
           70 96 ;...
           83 60 ;...
           83 3 ;...
           p_goal
         ];
else
    path=[ p_init;...
       15 40 ;...
       20 65 ;...
       50 68 ;...
       58 96 ;...
       70 96 ;...
       83 60 ;...
       100 60 ;...
       83 3 ;...
       p_goal
    ];
end


%% Genrate trajectory
QDMAX = [20,20]; %axis speed limits
Q0 = path(1,:); %initial axis coordinates
DT = 0.2;
traj = mstraj(path, QDMAX, [], Q0, DT, 0);
%% Plot the path
obstacle=obstacle; %0 when there is no door to block the way, 1 the door will
be on
animatepath(traj,DT,obstacle)


hold on
plot(path(:,1), path(:,2))
```

Task 3

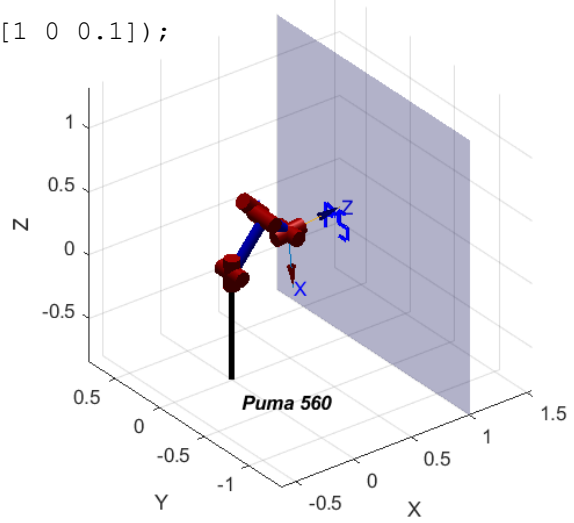
To plot the letter 'S' 6 waypoints are defined and then an interpolated trajectory is calculated. Because the points were first defined according a wrong system of coordinates, the waypoints are rotated by multiplying them by a rotation matrix then loaded into the robot.



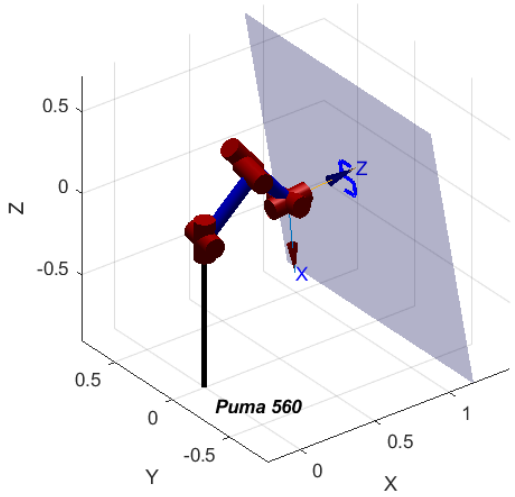
For the inclined wall, the angle between both planes is calculated and then a rotation of this angle about the Y axis is applied to the original trajectory.



```
clear all; close all; clc
%% set up the enviroment
lim = [-2.0, 2.0, -2.0, 2.0, -2.0, 2.0]; % to define the limit of x,y,z
mdl_puma560     % to load the puma560 robot
p560.plot(qz,'workspace',lim)    % to draw the robot

% to define and to plot a wall
a = 1;  b = 0; c = 0; d = -1; % the wall is defined as ax+by+cz+d = 0
wall = Wall(a,b,c,d,lim);  % to define the wall
wall.plotwall();           % to plot the wall

%% place your trajectory generation code here
% Letter S
via = [...
  -.05 -0.1    0;...
    0 -0.1    0;...
    0 -0.1  0.1;...
    0    0  0.1;...
    0    0    0;...
    0   .1    0;...
    0   .1   .1]'

% i drew the coordinates referred to a bad axis so i changed it...
via = (via' * rotx(pi/2))'
traj_s = mstraj(via(:,:)', [0.1,0.1,0.1], [], via(:,1)', 0.2, 5);

% Letter P
via = [...
    0   .25    -0.1;...
    0   .25      0;...
    0   .25   0.1;...
    0   .15    .07;...
    0   .15    .03;...
    0   .25      0;...
  -.05  .25    0;...
  -.05  0    0.1]' % match end-start points to avoid errors

traj_p = mstraj(via(:,:)', [0.5,0.5,0.5], [], via(:,1)', 0.2, 2);

% Combine & run
traj = [traj_p; traj_s]

%% send to robot
Tp = SE3(0.6, 0, 0) * SE3(traj) * SE3.oa([0 1 0], [1 0 0.1]);
q = p560.ikine6s(Tp);
plot_qtraj(q, wall, p560)
```

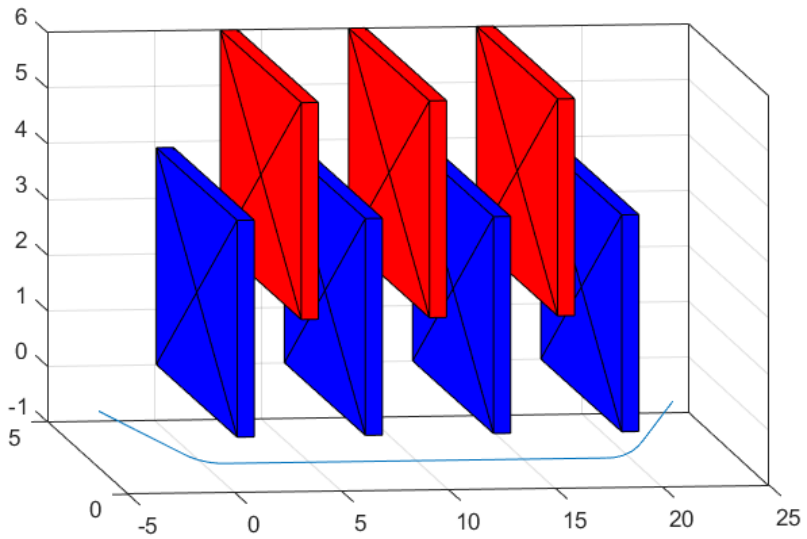


```
clear all; close all; clc
%% set up the enviroment
lim = [-2.0, 2.0, -2.0, 2.0, -2.0, 2.0]; % to define the limit of x,y,z
mdl_puma560     % to load the puma560 robot
p560.plot(qz,'workspace',lim)   % to draw the robot
% to define and to plot a wall
a = cos(deg2rad(10));  % the wall is defined as ax+by+cz+d = 0
b = 0;
c = sin(deg2rad(10));
d = -cos(deg2rad(10));
wall = Wall(a,b,c,d,lim);  % to define the wall
wall.plotwall();           % to plot the wall

%% place your trajectory generation code here
% i drew some stuff on a paper and got the angle i need to rotate along the
% Y axis to match this plane with the one in the previous section as...
theta = atan(c/a)

% so this is exactly from the previous part's S...
via = [...
    0 -0.1    0;...
    0 -0.1  0.1;...
    0    0  0.1;...
    0    0    0;...
    0   .1    0;...
    0   .1   .1]'
via = (via' * rotx(pi/2))'

% and now i fix the plane orientation thing
via = (via' * roty(theta))'
traj = mstraj(via(:,[2 3 4 5 6])', [0.1,0.1,0.1], [], via(:,1)', 0.2, 5);


%% send to robot
Tp =SE3(0.6, 0, 0) *   SE3(traj) * SE3.oa( [0 1 0], [1 0 0.1]);
q = p560.ikine6s(Tp);
plot_qtraj(q, wall, p560)
```

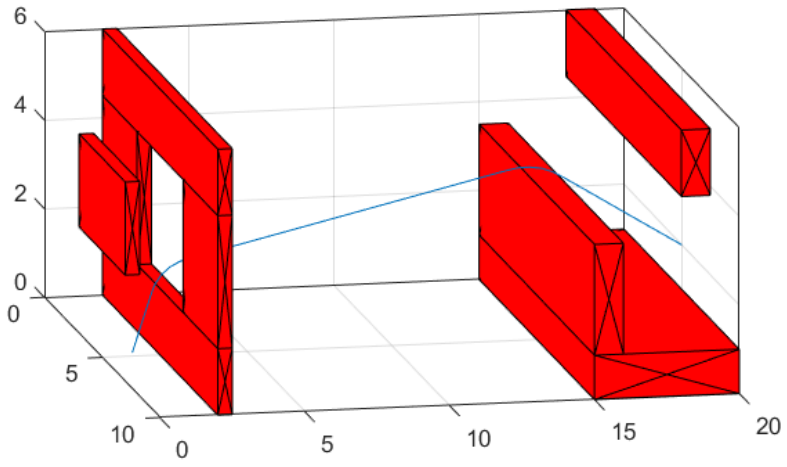


Task 4

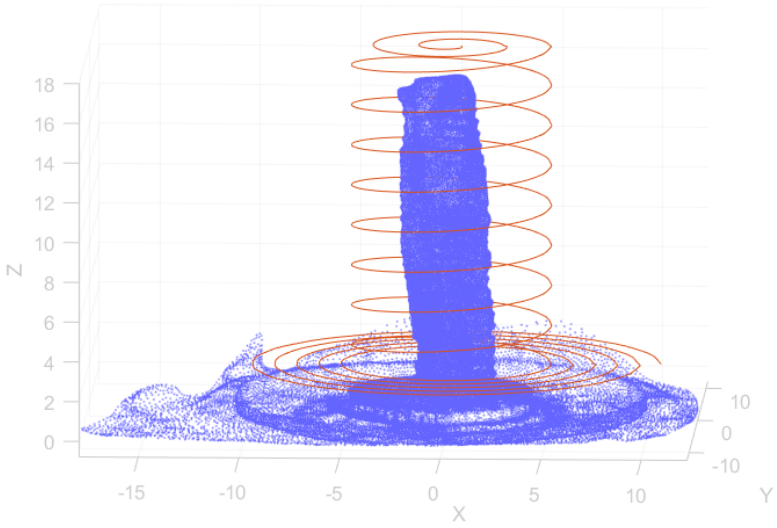
The first trajectory is computed by completely avoiding the slalom and just flying under the obstacles.



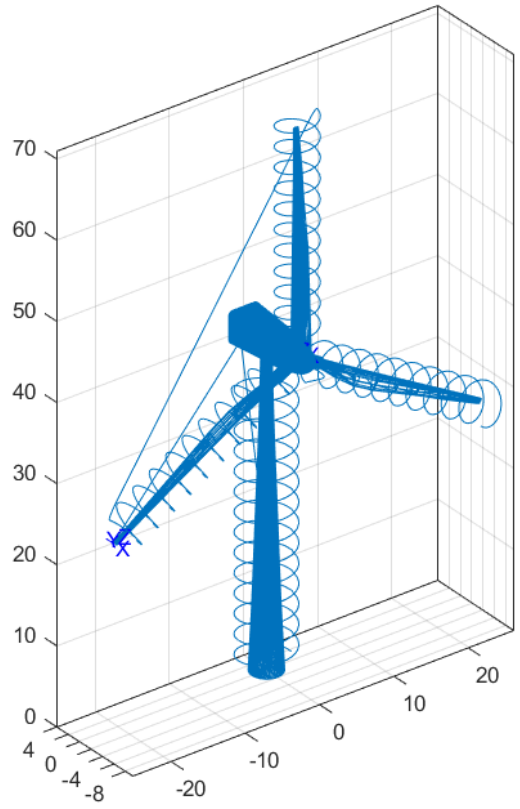
The second trajectory is computed by avoiding the obstacles, this time in a reasonable manner.



For the inspection of the fountain three trajectories are computed: one spiral on the lowest level, an ascending helix and a second spiral for approaching the top part. These shapes were computed by hand-crafted functions.

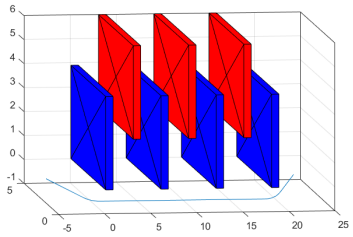


The wind turbine inspection happens in 5 stages: tower, higher blade, left blade, small maneuver to avoid the hub and finally the right blade.

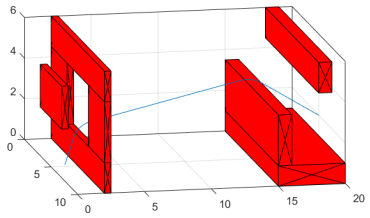


```
clear all
clc
close all
addpath(genpath('./'));

% map1.txt and map2.txt are the complex enviroments
map = load_map('maps/map2.txt', 0.1, 2, 0.25);
plot_path(map);
grid on
box on
hold on
```

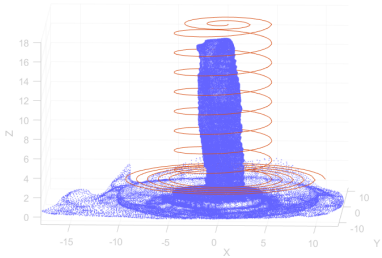
```
%% map1
start  = [-5, 2, 0]
via = [...
    0 2 -1;...
    20 2 -1;...
    22, 2, 0]'
q = mstraj(via(:,[1 2 3])', [2,2,2], [], start, 0.2, 1);
plot3(q(:,1), q(:,2), q(:,3))
```



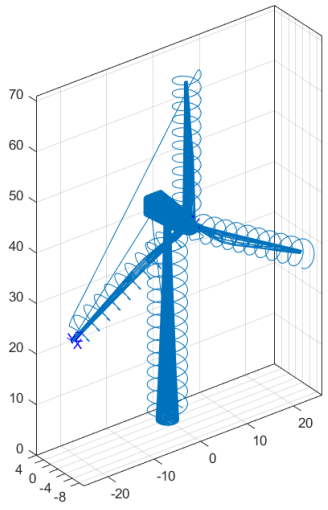
```
%% map2
start = [5, 1, 0]
% finish = [5, 20, 2]
via = [...
    5 2 2;...
    5 15 4;...
    5 20 2]'
q = mstraj(via(:,[1 2 3])', [2,2,2], [], start, 0.2, 1);
plot3(q(:,1), q(:,2), q(:,3))
```



```
close all; clear all; clc
%%
ptCloud = pcread('fountain.ply');
pointscolor = uint8(zeros(ptCloud.Count,3));
pointscolor(:,1) = 100; pointscolor(:,2) = 100; pointscolor(:,3) = 255;
ptCloud.Color = pointscolor; pcshow(ptCloud); hold on
xlabel('X'); ylabel('Y'); zlabel('Z')

% stage 1 constant height
t1 = hor_spiral(21, 10, 2, 5, 20)

% stage 2 upwards inspection
t2 = helix(10, 2, 18, 8, 20)

% stage 3 inspect the top
t3 = hor_spiral(10, 1, 18, 2, 20)

% smoothing & plotting
t = [t1; t2; t3]'
t = mstraj(t(:,2:end)', [5,5,5], [], t(:,1)', 0.2, 0.5);
plot3(t(:,1), t(:,2), t(:,3))


function coords = helix(diameter, min_height, max_height, nloops,
points_per_loop)
    %%% Draws a vertical 3d helix centered at the origin

    theta = []
    for i = 1:nloops
        theta = [theta linspace(0, 2*pi, points_per_loop)]
    end

    height = linspace(min_height, max_height, points_per_loop*nloops)

    coords = []
    coords(:,1) = diameter/2 * cos(theta)
    coords(:,2) = diameter/2 * sin(theta)
    coords(:,3) = height
end

function coords = hor_spiral(diam_start, diam_finish, height, nloops,
points_per_loop)
    %%% Draws a horizontal 3d spiral centered at the origin

    theta = []
    for i = 1:nloops
        theta = [theta linspace(0, 2*pi, points_per_loop)]
    end

    diameter = linspace(diam_start, diam_finish,  points_per_loop*nloops)

    coords = []
    coords(:,1) = diameter/2 .* cos(theta)
    coords(:,2) = diameter/2 .* sin(theta)
    coords(:,3) = height .* ones(size(theta))
end
```



```
close all; clear all; clc
load('Points10.mat') %All
X=X/10; Y=Y/10; Z=Z/10; xyz=[X Y Z]; plot3(X,Y,Z); ylim([-20 20])
grid on; box on; axis equal; hold on

% A few new frames to help with the task..
TH = SE3(0, -5.4, 41); trplot(TH)                                    % at the hub
TLB = SE3(-25, -5.4, 27) * SE3.Ry(deg2rad(60)); trplot(TLB)      % left blade
tip

% inspecting the tower
ttw = helix(7, 1, 36, 14, 20)    % straightforward with helix motion

% inspecting top blade
ttb = helix(5, 2, 30, 11, 20)    % start from base geometry function
ttb = TH * SE3(ttb)              % move trajectory to desired position
ttb = ttb.transl                 % get just the path set of points
ttb = ttb(3:end,:)               % remove risky start/finish

% inspect left blade
tlb = helix(5, 0, 28, 11, 20)
tlb = TLB * SE3(tlb)
tlb = tlb.transl
tlb = tlb(10:end-7,:)

% return to safe point to avoid the hub
tsf1 = TH * SE3(-1, -4, 1)
tsf2 = tsf1 * SE3(2, 0, -2)
tsf = [tsf1.transl; tsf2.transl]

% inspect right blade
trb = helix(5, 0, 30, 11, 20)
trb = TH * SE3.Ry(deg2rad(120)) * SE3(trb)
trb = trb.transl
trb = trb(20:end,:)

% assemble and interpolate
t = [ttw; ttb; tlb; tsf; trb]'
t = mstraj(t(:,2:end)', [5,5,5], [], t(:,1)', 0.2, 0.5);
plot3(t(:,1), t(:,2), t(:,3),'r')
```




```
%% me-defined funcs
function coords = hor_circular(diameter, height, npoints)
    %%% Draws a horizontal 3d circle centered at the origin

    theta = linspace(0, 2*pi, npoints)

    coords = []
    coords(:,1) = diameter/2 * cos(theta)
    coords(:,2) = diameter/2 * sin(theta)
    coords(:,3) = height .* ones(length(theta),1)
end
```

```
function coords = helix(diameter, min_height, max_height, nloops,
points_per_loop)
    %%% Draws a vertical 3d helix centered at the origin

    theta = []
    for i = 1:nloops
        theta = [theta linspace(0, 2*pi, points_per_loop)]
    end

    height = linspace(min_height, max_height, points_per_loop*nloops)

    coords = []
    coords(:,1) = diameter/2 * cos(theta)
    coords(:,2) = diameter/2 * sin(theta)
    coords(:,3) = height
end


function coords = hor_spiral(diam_start, diam_finish, height, nloops,
points_per_loop)
    %%% Draws a horizontal 3d spiral centered at the origin

    theta = []
    for i = 1:nloops
        theta = [theta linspace(0, 2*pi, points_per_loop)]
    end

    diameter = linspace(diam_start, diam_finish,  points_per_loop*nloops)

    coords = []
    coords(:,1) = diameter/2 .* cos(theta)
    coords(:,2) = diameter/2 .* sin(theta)
    coords(:,3) = height .* ones(size(theta))
end
```