Trabajo de Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

Análisis cinemático y dinámico de la plataforma Hexaglide

ANEXOS

Autor:	Rita Roca Taxonera
Director:	Federico Thomas
Codirector:	Lluís Ros
Convocatòria:	Juny 2016

ETSEIB

Escuela Técnica Superior
De Ingenieria Industrial de Barcelona

Sumario

ETSEIB

1. MainHexaglide.m

```
clc;
clear all;
close all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Initialize global variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global lenaxis;
global baseradious;
global heightstructure;
global heightguides;
global Xguide;
global Zguide;
global h1;
global alpha1;
global k;
global beta;
global alpha2;
global side1;
global side2;
global lengthleg;
global platformmass;
global g_world;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Start Index Options
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

INDEX = strcat('INDEX\n',...
    '1.Draw Hexaglide using inverse kinematics\n',...
    '2.Forward Kinematics: calculate the platform position knowing the
position of the actuators\n',...
    '3.Video Hexaglide\n',...
    '4.Jacobian Kinematic Matrix\n',...
    '5.Workspace for a fixed orientation\n',...
    '6.Forward Dynamics: calculate the movement of the platform knowing the
forces of the actuators\n',...
    '7.Forward Dynamics: path of a point of the platform\n',...
    '8.Inverse Dynamics: calculate forces of the actuators knowing the
movement of the platform\n');

fprintf (INDEX);
op = input ('\nChoose an option: ');

if (op == 1) || (op == 4)
    fprintf ('Now, select the platform transformation:\n');
    disp = input ('Enter the platform displacement as follows [dispx, dispy,
dispz]: ');
    rotx_plat = input('Enter the platform x rotation in radians: ');
    roty_plat = input('Enter the platform y rotation in radians: ');
    rotz_plat = input('Enter the platform z rotation in radians: ');
    TT = transl(disp)*trotx(rotx_plat)*troty(roty_plat)*trotz(rotz_plat);
end

if (op == 2)
    y = 1;
    n = 0;
    ini_cond = input ('Do you want to see an example with a prefixed value?
(y,n): ');
    if ini_cond == y
        fprintf('z=[1410, 1540.9, 1510.3, 1019.1, 931, 1297.9]\n');
        zfinal = [1410, 1540.9, 1510.3, 1019.1, 931, 1297.9];
    else
    zfinal = input('Enter z of the actuators as follows
[z(1),z(2),z(3),z(4),z(5),z(6)]: ');
    end
end

if (op == 5)
    rotx_plat = input('Enter the platform x rotation in radians: ');
    roty_plat = input('Enter the platform y rotation in radians: ');
    rotz_plat = input('Enter the platform z rotation in radians: ');
end

if (op == 6) || (op == 7)
    y = 1;
    n = 0;
    ini_cond = input('Do you want to use prefixed initial conditions? (y,n):
');
    if ini_cond == y
        p = [0;0;200];
        v = [0;0;0];
        rotx_plat = 0;
        roty_plat = 0;
        rotz_plat = 0;
        w_body = [0;0;0];
    else
        p = input('Enter the platform position in mm [x; y; z]: ');
        v = input('Enter the platform initial velocity in mm/s [v_x; v_y;
v_z]: ');
        rotx_plat = input('Enter the platform x rotation in radians: ');
        roty_plat = input('Enter the platform y rotation in radians: ');
        rotz_plat = input('Enter the platform z rotation in radians: ');
        w_body = input('Enter initial angular velocity in the body frame in
rad/s [w_x; w_y; w_z]: ');
    end
end

if (op == 8)
    fprintf('If you want to see an example, use next movement:
perfil_movimiento.txt\n');

    file_mov_path = input('Enter the file path with the movement you want:
');
    file_dest_data = input('Enter the file path to save the results: ');
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Set elements necessaries for the diferent functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Robot parameters in mm, rad and kg.

lenaxis = 300;
baseradious = 959;
heightstructure = 1482.52;
heightguides = 2320;
Xguide = 354.7;
Zguide = 300.0;
h1 = (2/3)*sqrt(170755.2);
alpha1 = atan((60*sin(5*pi/6))/(h1+(60*cos((5*pi/6)))));
k = sqrt(5391.8289);
beta = asin((sin(pi/2)*42.33)/k);
alpha2 = atan((k*sin((5*pi/6)-beta))/(h1+(k*cos((5*pi/6)-beta))));
side1 = ((60*sin(pi/6)/(sin(alpha1))));
side2 = ((k*sin((pi/6)+beta)/(sin(alpha2))));
lengthleg = 1000;
platformmass = 1.7;
g_world = [ 0; 0; -9806.65 ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% We generate a reference frame associated with each vertex of the
% hexagonal base.

for i=1:6
    TB(:,:,i) = trotz(2*pi*i/6)*transl(0,-baseradious,0);
%   trplot(TB(:,:,i), 'length', lenaxis);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% We generate a reference frame for each guide

for i=1:2:5
    TG(:,:,i) = TB(:,:,i)*trotz(-pi/6)*transl(-Xguide,0,Zguide);
    TG(:,:,i+1) = TB(:,:,i+1)*trotz(pi/6)*transl(Xguide,0,Zguide);
%   trplot(TG(:,:,i),'frame', int2str(i), 'arrow', 'color', 'b', 'length',
0.75*lenaxis);
%   trplot(TG(:,:,i+1),'frame', int2str(i+1), 'arrow', 'color', 'b',
'length', 0.75*lenaxis);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% We generate a reference frame for each attaching point in the moving
% platform (TP1) and the bars fixed to it (TP2) and we obtain
% their coordinates

for i = 1:2:5
    TP1(:,:,i) = trotz(2*pi*(i-1)/6 - alpha1 +pi/2)*transl(0,-side1,0);
    TP1(:,:,i+1) = trotz(2*pi*(i-1)/6 + alpha1 +pi/2)*transl(0,-side1,0);
    TP2(:,:,i) = trotz(2*pi*(i-1)/6 - alpha2 +pi/2)*transl(0,-side2,0);
    TP2(:,:,i+1) = trotz(2*pi*(i-1)/6 + alpha2 +pi/2)*transl(0,-side2,0);
%   trplot(TP(:,:,i),'color', 'k', 'length', lenaxis);
%   trplot(TP(:,:,i+1),'color', 'k', 'length', lenaxis);
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Switch to option selected by user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
booleanExit = false;
firstAsk = true;
y = 1;
n = 0;
while booleanExit == false
    switch op
        case 1 %DrawHexaglide
            if exist('TT','var') && firstAsk == false
                newt = input ('Do you want to set a new transformation?
(y,n): ');
                if newt == y
                    disp = input ('Enter the platform displacement in mm as
follows [dispx, dispy, dispz]: ');
                    rotx_plat = input('Enter the platform x rotation in
radians: ');
                    roty_plat = input('Enter the platform y rotation in
radians: ');
                    rotz_plat = input('Enter the platform z rotation in
radians: ');
                    TT =
transl(disp)*trotx(rotx_plat)*troty(roty_plat)*trotz(rotz_plat);
                end
            end
            DrawHexaglide(TT, TP1, TP2, TB, TG);
            firstAsk = false;
            exit = input ('Do you want to exit the program? (y/n): ');
            if exit == y
                booleanExit = true;
            else
                fprintf (INDEX);
                op = input ('\nChoose an option: ');
            end

        case 2 %Forward Kinematics
            if exist('zfinal','var') && firstAsk == false
                ini_cond = input ('Do you want to see an example with a
prefixed value? (y,n): ');
                if ini_cond == y
                    fprintf('z=[1410, 1540.9, 1510.3, 1019.1, 931,
1297.9]\n');
                    zfinal = [1410, 1540.9, 1510.3, 1019.1, 931, 1297.9];
                else
                    zfinal = input('Enter z of the actuators as follows
[z(1),z(2),z(3),z(4),z(5),z(6)]: ');
                end
            end
            ForwardKinematics(zfinal, TP1, TP2, TB, TG);
            firstAsk = false;
            exit = input ('Do you want to exit the program? (y/n): ');
            if exit == y
                booleanExit = true;
            else
                fprintf (INDEX);
                op = input ('\nChoose an option: ');
            end

        case 3 %Video_Hexaglide
            Video_Hexaglide(TP1, TP2, TB, TG);
            exit = input ('Do you want to exit the program? (y/n): ');
            if exit == y
                booleanExit = true;
            else
                fprintf (INDEX);
                op = input ('\nChoose an option: ');
            end

        case 4 %JacobianMatrix
            if exist('TT','var') && firstAsk == false
                newt = input ('Do you want to set a new transformation?
(y,n): ');
                if newt == y
                    disp = input ('Enter the platform displacement in mm as
follows [dispx, dispy, dispz]: ');
                    rotx_plat = input('Enter the platform x rotation in
radians: ');
                    roty_plat = input('Enter the platform y rotation in
radians: ');
                    rotz_plat = input('Enter the platform z rotation in
radians: ');
                    TT =
transl(disp)*trotx(rotx_plat)*troty(roty_plat)*trotz(rotz_plat);
                end
            end
            z = InverseKinematics(TT, TG, TP2);
            Jac = JacobianMatrix(z, TP2, TT, TG)
            firstAsk = false;
            exit = input ('Do you want to exit the program? (y/n): ');
            if exit == y
                booleanExit = true;
            else
                fprintf (INDEX);
                op = input ('\nChoose an option: ');
            end

        case 5 %Workspace
            if exist('rotx_plat','var') && firstAsk == false
                newt = input ('Do you want to set a new platform orientation?
(y,n): ');
                if newt == y
                    rotx_plat = input('Enter the platform x rotation in
radians: ');
                    roty_plat = input('Enter the platform y rotation in
radians: ');
                    rotz_plat = input('Enter the platform z rotation in
radians: ');
                end
            end
            WorkSpace(TP2, TG, TB, rotx_plat, roty_plat, rotz_plat);
            firstAsk = false;
            exit = input ('Do you want to exit the program? (y/n): ');
            if exit == y
                booleanExit = true;
            else
                fprintf (INDEX);
                op = input ('\nChoose an option: ');
            end

        case 6 %Forward Dynamics
            if exist('p','var') && exist('rotx_plat','var') && ...
                    exist('w_body','var') && firstAsk == false
                newt = input ('Do you want to set new initial conditions?
(y,n): ');
                if newt == y
                    p = input('Enter the platform position in mm [x; y; z]:
');
                    v = input('Enter the platform initial velocity in mm/s
[v_x; v_y; v_z]: ');
                    rotx_plat = input('Enter the platform x rotation in
radians: ');
                    roty_plat = input('Enter the platform y rotation in
radians: ');
                    rotz_plat = input('Enter the platform z rotation in
radians: ');
                    w_body = input('Enter initial angular velocity in the
body frame in rad/s [w_x; w_y; w_z]: ');
                end
            end
            Video_Forward_Dynamics(p, v, rotx_plat, roty_plat, rotz_plat,
w_body, TP1, TP2, TB, TG);
            firstAsk = false;
            exit = input ('Do you want to exit the program? (y/n): ');
            if exit == y
                booleanExit = true;
            else
                fprintf (INDEX);
                op = input ('\nChoose an option: ');
            end

        case 7 %Path of a point when moving the platform
            if exist('p','var') && exist('rotx_plat','var') && ...
                    exist('w_body','var') && firstAsk == false
                newt = input ('Do you want to set new initial conditions?
(y,n): ');
                if newt == y
                    p = input('Enter the platform position in mm [x; y; z]:
');
                    v = input('Enter the platform initial velocity in mm/s
[v_x; v_y; v_z]: ');
                    rotx_plat = input('Enter the platform x rotation in
radians: ');
                    roty_plat = input('Enter the platform y rotation in
radians: ');
```

```
                    rotz_plat = input('Enter the platform z rotation in
radians: ');
                    w_body = input('Enter initial angular velocity in the
body frame in rad/s [w_x; w_y; w_z]: ');
                end
            end
            Video_Path_ForwardDynamics(p, v, rotx_plat, roty_plat, rotz_plat,
w_body, TP2, TB, TG);
            firstAsk = false;
            exit = input ('Do you want to exit the program? (y/n): ');
            if exit == y
                booleanExit = true;
            else
                fprintf (INDEX);
                op = input ('\nChoose an option: ');
            end

        case 8 %Inverse Dynamics
            if exist('file_mov_path','var') && ...
                    exist('file_dest_data','var') && firstAsk == false
                fprintf('If you want to see an example, use next movement:
perfil_movimiento.txt\n');
                newt = input ('Do you want to set new variables? (y,n): ');
                if newt == y
                    file_mov_path = input('Enter the file path with the
movement you want: ');
                    file_dest_data = input('Enter the file path to save the
results: ');
                end
            end
            InverseDynamics(file_mov_path, file_dest_data, TG, TP2);
            firstAsk = false;
            exit = input ('Do you want to exit the program? (y/n): ');
            if exit == y
                booleanExit = true;
            else
                fprintf (INDEX);
                op = input ('\nChoose an option: ');
            end
    end
end
```

2. DrawHexaglide.m

```
function z = DrawHexaglide(TT, TP1, TP2, TB, TG)
%Draw the complete robot with given transformation of the platform

global lenaxis;
global heightstructure;
global heightguides;

% Initialization of the the figure

figure(1);
A = [-1000 1000 -1000 1000 -10 2500];
trplot(eye(4),'frame', 'W', 'arrow', 'color', 'r', 'length', lenaxis, 'axis',
A,'text');
%trplot(TT,'frame', 'B', 'arrow', 'color', 'b', 'length', lenaxis);
hold on;

for i=1:6
    PP1(:,i) = TT*TP1(:,4,i);
    PP2(:,i) = TT*TP2(:,4,i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Drawing the hexagonal structure and the guides

fill3(squeeze(TB(1,4,:)), squeeze(TB(2,4,:)), squeeze(TB(3,4,:)), ...
    'k', 'facealpha',.1);
fill3(squeeze(TB(1,4,:)), squeeze(TB(2,4,:)), squeeze(TB(3,4,:)) +
heightstructure, ...
    'k', 'facealpha',.1);

for i=1:6
    plot3([TB(1,4,i) TB(1,4,i)], [TB(2,4,i) TB(2,4,i)], ...
          [TB(3,4,i) (TB(3,4,i) + heightstructure)], 'color', 'r');
    plot3([TG(1,4,i) TG(1,4,i)], [TG(2,4,i) TG(2,4,i)], ...
          [TG(3,4,i) (TG(3,4,i) + heightguides)], 'color', 'b')
end

%%% Drawing the moving platform

fill3(PP1(1,:), PP1(2,:), PP1(3,:), 'g', 'facealpha',.5);

for i=1:6
    text(PP1(1,i),PP1(2,i),PP1(3,i), num2str(i), ...
        'VerticalAlignment','bottom', ...
        'HorizontalAlignment','right', ...
        'color', 'k')
end


z = InverseKinematics(TT, TG, TP2);
```

```
for i=1:6
    plot3([PP1(1,i) PP2(1,i)], [PP1(2,i) PP2(2,i)], ...
        [PP1(3,i) PP2(3,i)], 'color', 'k')
    plot3([PP2(1,i) TG(1,4,i)], [PP2(2,i) TG(2,4,i)], ...
        [PP2(3,i) z(i)], 'color', 'k')
end

hold off;
end
```

3. InverseKinematics.m

```
function z = InverseKinematics(TT, TG, TP2)
%Calculate inverse kinematics to obtain the actuators position in their
%respective guides.

global lengthleg;

for i=1:6
    PP2(:,i) = TT*TP2(:,4,i);
end

for i=1:6
    z(i) = PP2(3,i) + sqrt(lengthleg^2 ...
        - (TG(1,4,i)-PP2(1,i))^2 ...
        - (TG(2,4,i)-PP2(2,i))^2);
end

end
```

4. ForwardKinematics.m

```
function TT_ini = ForwardKinematics(zfinal, TP1, TP2, TB, TG)
%Calculate the platform position knowing the position of the actuators.

TT_ini = transl(0,0,600);

scale= 1.5;

for i=1:20
    DrawHexaglide(TT_ini, TP1, TP2, TB, TG);
    zini = InverseKinematics(TT_ini, TG, TP2);
    error(i)= norm(zini-zfinal)
    J= InverseAnalyticJacobian(TT_ini, TG, TP2);
    increase = J*(zini-zfinal)'/scale;
    TT_ini = TT_ini*transl(increase(1), increase(2), increase(3))*...
            trotx(increase(4))*...
            troty(increase(5))*...
            trotz(increase(6));
end

figure(2);
plot(error);
grid on;
end
```

5. InverseAnalyticJacobian.m

```
function J = InverseAnalyticJacobian(TT, TG, TP2)

delta = 0.01;

z = InverseKinematics(TT, TG, TP2);

dz_dx = (z-InverseKinematics(TT*transl(delta, 0, 0), TG, TP2))/delta;
dz_dy = (z-InverseKinematics(TT*transl(0, delta, 0), TG, TP2))/delta;
dz_dz = (z-InverseKinematics(TT*transl(0, 0, delta), TG, TP2))/delta;
dz_rx = (z-InverseKinematics(TT*trotx(delta), TG, TP2))/delta;
dz_ry = (z-InverseKinematics(TT*troty(delta), TG, TP2))/delta;
dz_rz = (z-InverseKinematics(TT*trotz(delta), TG, TP2))/delta;

K = [dz_dx' dz_dy' dz_dz' dz_rx' dz_ry' dz_rz'];

J = inv(K);
end
```

6. VideoHexaglide.m

```
function Video_Hexaglide(TP1, TP2, TB, TG)
samples=100;
turns = 5;

opengl('software');

writerObj = VideoWriter('D:\Hexaglide\Video_Hexaglide.avi','Uncompressed
AVI');
open(writerObj);
set(gca,'nextplot','replacechildren');
set(gcf,'Renderer','opengl'); %'zbuffer');
set(gcf, 'units','normalized','outerposition',[0 0 1 1]); %Full screen

for j=1:turns
    for i=1:samples
        DrawHexaglide(transl(400*cos(2*pi*i/samples),...
                             400*sin(2*pi*i/samples),...
                             600)*...
                      trotz(-6*pi*i/samples)*...
                      trotx(pi/8)*...
                      trotz(6*pi*i/samples),TP1, TP2, TB, TG);
        frame = getframe;
        writeVideo(writerObj,frame);
    end
end

close(writerObj);
```

7. JacobianMatrix.m

```
function Jacobian = JacobianMatrix(z, TP2, TT, TG)
%Returns the Inverse Kinematic Jacobian Matrix

a = [0;0;-1];
Matrix_an = eye(6);
Matrix_q = eye(6);
for i=1:6
    PP2(:,i) = TT*TP2(:,4,i);
end

%Calculation of "Matrix_an" and "Matrix_q"

for i=1:6
    %Vector of each bar
    vect = [PP2(1,i)-TG(1,4,i); PP2(2,i)-TG(2,4,i); PP2(3,i)-z(i)];
    n = [vect/1000];
    %Vector from platform point B(i) to the center C
    BC = [TT(1,4)-PP2(1,i); TT(2,4)-PP2(2,i); TT(3,4)-PP2(3,i)];
    Matrix_an(i,i) = a'*n;
    q2 = cross(n,BC);
    Matrix_q(i,:) = [n(1),n(2),n(3),q2(1),q2(2),q2(3)];

end

Jacobian = inv(Matrix_an)*Matrix_q;

end
```

8. Workspace.m

```
function Points_to_plot = WorkSpace(TP2, TG, TB, rotx_plat, roty_plat,
rotz_plat)
%Plot the workspace of a determined orientation of the platform.

global lenaxis;
global heightstructure;
global heightguides;
global lengthleg;

% Initialization of the figure
figure(1);
A = [-1000 1000 -1000 1000 -10 2500];
trplot(eye(4),'frame', 'W', 'arrow', 'color', 'r', 'length', lenaxis, 'axis',
A,'text');
hold on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Drawing the hexagonal structure and the guides

fill3(squeeze(TB(1,4,:)), squeeze(TB(2,4,:)), squeeze(TB(3,4,:)), ...
    'k', 'facealpha',.1);
fill3(squeeze(TB(1,4,:)), squeeze(TB(2,4,:)), squeeze(TB(3,4,:)) +
heightstructure, ...
    'k', 'facealpha',.1);

for i=1:6
    plot3([TB(1,4,i) TB(1,4,i)], [TB(2,4,i) TB(2,4,i)], ...
          [TB(3,4,i) (TB(3,4,i) + heightstructure)], 'color', 'r');
    plot3([TG(1,4,i) TG(1,4,i)], [TG(2,4,i) TG(2,4,i)], ...
          [TG(3,4,i) (TG(3,4,i) + heightguides)], 'color', 'b')
end

xy_max = 30; %959;
P = [-1000;-1000;0];
z_max = 17; %1480;
displacement = 100;
Points_to_plot = [];
for z = 0:z_max
    for x = 5:xy_max
        for y = 5:xy_max
            disp = [(P(1)+x*displacement),(P(2)+
y*displacement),(P(3)+z*displacement)]
            TT =
transl(disp)*trotx(rotx_plat)*troty(roty_plat)*trotz(rotz_plat);
            correct_position = 0;
            for i=1:6
                M2 = TP2(:,:,i);
                PP2(:,i) = TT*M2(:,4);
                PG(:,i) = TG(:,4,i);
                h(i) = PP2(3,i) + sqrt(lengthleg^2 ...
                    - (PP2(1,i)-PP2(1,i))^2 ...
                    - (PP2(2,i)-PP2(2,i))^2);
                Pg_x = PG(1,:);
                Pg_y = PG(2,:);
                in = inpolygon(PP2(1,i),PP2(2,i),Pg_x,Pg_y);

                if in == 1 && h(i) <= (heightguides + 300) ...
                        && PP2(3,i) >= 0

                    correct_position = correct_position + 1;
                end
            end
            if correct_position == 6
                for i=1:6
                    Points_to_plot = [Points_to_plot; [PP2(1,i) PP2(2,i)
PP2(3,i)]];
                end
            end
        end
    end
end

k = convhull(Points_to_plot);
trisurf(k,Points_to_plot(:,1),Points_to_plot(:,2),Points_to_plot(:,3),'Faceco
lor','blue','FaceAlpha',.7,'EdgeColor','none')
end
```

9. Video_Forward_Dynamics.m

```
function Video_Forward_Dynamics(p, v, rotx_plat, roty_plat, rotz_plat,
w_body, TP1, TP2, TB, TG)
% p = [x;y;z]
% w_body = angular velocity in the body frame (w_body = [x;y;z])
% M = total torque aplied to the platform (M = [x;y;z])

samples=1000;

opengl('software');

writerObj = VideoWriter('D:\Hexaglide\Video_Orientation.avi','Uncompressed
AVI');
open(writerObj);
set(gca,'nextplot','replacechildren');
set(gcf,'Renderer','opengl'); %'zbuffer');
set(gcf, 'units','normalized','outerposition',[0 0 1 1]); %Full screen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Draw initial position of the moving platform

R = rpy2r(rotx_plat,roty_plat,rotz_plat);
TT = [R p; [0 0 0 1]];

DrawHexaglide(TT, TP1, TP2, TB, TG);
frame = getframe;
writeVideo(writerObj,frame);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%start video orientation
finish = false;
set(gcf,'CurrentCharacter','@');
for t=1:0.2:samples
    if finish == false
        [new_p, new_v, new_R, new_w_body] = DrawDynamics(p, v, R, w_body, t,
TP1, TP2, TB, TG);
        fprintf('tiempo: %i',t)
        p = new_p;
        v = new_v;
        R = new_R;
        w_body = new_w_body;
        frame = getframe;
        writeVideo(writerObj,frame);
        k=get(gcf,'CurrentCharacter');
        if k~='@' % has it changed from the dummy character?
            finish=true;
        end
    else
        break;
    end
end
```

```
close(writerObj);
```

10. DrawDynamics.m

```
function [new_p, new_v, new_R, new_w_body] = DrawDynamics(p, v, R, w_body, t,
TP1, TP2, TB, TG)
% M = total torque aplied to the platform

[new_p, new_v] = Translation(p, v, t, R, TG, TP2);
[new_R, new_w_body] = Orientation(R, w_body, t, p, TG, TP2);

TT = [new_R new_p; [0 0 0 1]];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%PLOT PLATFORM with new TT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DrawHexaglide(TT, TP1, TP2, TB, TG);

end
```

11. Translation.m

```
function [new_p, new_v] = Translation( p, v, t, R, TG, TP2 )

global g_world;
global platformmass;
dt = 0.05;

g_world = [ 0; 0; -9806.65 ];
platformmass = 1.7; %kg
%g_body = R \ g_world;  %g_body = inv(R) * g_world

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function to find the acceleration of the center of mass (mm/s^2)
    function [a_c] = getAc( t, g_world, p, R, TG, TP2 )
        pose = [p; tr2rpy(R)'];
        Actuator = Forces_Actuators(t);
        [F_world,M_world] = GuidesToCenter(Actuator, pose, TG, TP2);

        mass = platformmass * eye(3);

        a_c = inv(mass) * (F_world + platformmass * g_world);
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%R-K4 integrate Acm to find new_vbody
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

k_v(:,1) = dt * getAc( t, g_world, p, R, TG, TP2 );
k_v(:,2) = dt * getAc( t+0.5*dt, g_world, p, R, TG, TP2 );
k_v(:,3) = dt * getAc( t+0.5*dt, g_world, p, R, TG, TP2 );
k_v(:,4) = dt * getAc( t+dt, g_world, p, R, TG, TP2 );

new_v = v + (1/6) * ( k_v * [1;2;2;1] );

k_p(:,1) = dt * v;
k_p(:,4) = dt * new_v;
k_p(:,2) = k_p(:,1) + (k_p(:,4) - k_p(:,1))*0.5;
k_p(:,3) = k_p(:,2);

new_p = p + (1/6) * ( k_p * [1;2;2;1] );


end
```

12. Forces_Actuators.m

```
function F_Act = Forces_Actuators(t)
%Write inside this function the prototype of forces in the actuators
% you want to study. Forces must be expressed in(mN)

%For Example:

if t<7
    F_Act(1)=3000;
    F_Act(2)=3000;
    F_Act(3)=3000;
    F_Act(4)=3000;
    F_Act(5)=3000;
    F_Act(6)=3000;
elseif t>=7 && t<18
    F_Act(1)=2500;
    F_Act(2)=2500;
    F_Act(3)=2500;
    F_Act(4)=2500;
    F_Act(5)=2500;
    F_Act(6)=2500;
elseif t>=18 && t<21
    F_Act(1)=4000;
    F_Act(2)=4000;
    F_Act(3)=4000;
    F_Act(4)=4000;
    F_Act(5)=4000;
    F_Act(6)=4000;
elseif t>=21
    F_Act(1)=2000;
    F_Act(2)=2000;
    F_Act(3)=2000;
    F_Act(4)=2000;
    F_Act(5)=2000;
    F_Act(6)=2000;
end

end
```

13. GuidesToCenter.m

```
function [F_C, M_C] = GuidesToCenter(Actuator, pose, TG, TP2)
%Obtain resultant force and moment applied to the center of mass
%Actuator and pose are 6-component vectors in world frame

global lengthleg;
lengthleg = 1000;

%Points of the platform
TT = transl(pose(1:3))*trotx(pose(4))*troty(pose(5))*trotz(pose(6));
for i=1:6
    PP2(:,i) = TT*TP2(:,4,i);
end
z = InverseKinematics(TT,TG,TP2);
Jac = JacobianMatrix(z,TP2,TT,TG);

Center = transpose(Jac) * (-Actuator)';
F_C = Center(1:3);
M_C = Center(4:6);

end
```

14. Orientation.m

```
function [new_R, new_w_body] = Orientation(R, w_body, t, p, TG, TP2)
%
% R = rotation matrix
% w_body = angular velocity in the body frame (w_body = [x;y;z])
% Q = quaternion representing orientation

global h1;
global alpha1;
global side1;
global platformmass;

h1 = (2/3)*sqrt(170755.2);
alpha1 = atan((60*sin(5*pi/6))/(h1+(60*cos((5*pi/6)))));
side1 = ((60*sin(pi/6)/(sin(alpha1))));
platformmass = 1.7; %kg
Q = Quaternion(R);

dt=0.05;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% calculation of the moment of inertia of the platform
% Simplification = treat the platform as a thin plate
Ix = (sqrt(3)/96)*(side1^4)*platformmass;
I =[ Ix, Ix, 2*Ix ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function to find angular acceleration
    function [Wdot] = getWdot( w_body, t, I, R, p, TG, TP2 )
        pose = [p; tr2rpy(R)'];
        Actuator = Forces_Actuators(t);
        [F_world,M_world] = GuidesToCenter(Actuator, pose, TG, TP2)
        M_body = inv(R) * M_world;
        Imat = diag(I);
        Imat_inv = diag(1 ./ I);
        Wdot = Imat_inv * (M_body + cross( Imat * w_body, w_body ));
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function to get the derivative of the quaternion
    function [qdot] = getQdot( w_body , Q )
        w_world = Q.R * w_body;
        W = Quaternion(w_world);
        qdot = W * Q;
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%R-K4 integrate Wdot to find new_wbody
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

k_q(:,1) = dt * getQdot( w_body, Q );
k_w(:,1) = dt * getWdot( w_body, t, I, R, p, TG, TP2 );
```

```
k_q(:,2) = dt * getQdot( w_body+0.5*k_w(:,1), Q+0.5*k_q(:,1) );
k_w(:,2) = dt * getWdot( w_body+0.5*k_w(:,1), t+0.5*dt, I, R, p, TG, TP2 );
k_q(:,3) = dt * getQdot( w_body+0.5*k_w(:,2), Q+0.5*k_q(:,2) );
k_w(:,3) = dt * getWdot( w_body+0.5*k_w(:,2), t+0.5*dt, I, R, p, TG, TP2 );
k_q(:,4) = dt * getQdot( w_body+k_w(:,3), Q+k_q(:,3) );
k_w(:,4) = dt * getWdot( w_body+k_w(:,3), t+dt, I, R, p, TG, TP2 );

Q = Q + (1/6) * ( k_q(1)+2*k_q(2)+2*k_q(3)+k_q(4) );
new_w_body = w_body + (1/6) * ( k_w * [1;2;2;1] );

%renormalize q
Q = Q*(1/Q.norm);

%plot new orientation of the platform
new_R = Q.R;

end
```

15. Video_Path_ForwardDynamics.m

```
function Video_Path_ForwardDynamics(p, v, rotx_plat, roty_plat, rotz_plat,
w_body, TP2, TB, TG)

samples=1000;

opengl('software');

writerObj = VideoWriter('D:\Hexaglide\Video_Orientation.avi','Uncompressed
AVI');
open(writerObj);
set(gca,'nextplot','replacechildren');
set(gcf,'Renderer','opengl'); %'zbuffer');
set(gcf, 'units','normalized','outerposition',[0 0 1 1]); %Full screen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization of the figure and global variables
global lenaxis;
global side1;
global platformmass;
global heightstructure;
global heightguides;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Draw initial position of the moving platform

R = rpy2r(rotx_plat,roty_plat,rotz_plat);
TT = [R p; [0 0 0 1]];

figure(1);
A = [-1000 1000 -1000 1000 -10 2500];
trplot(eye(4),'frame', 'W', 'arrow', 'color', 'r', 'length', lenaxis, 'axis',
A,'text');
hold on;




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Draw the hexagonal structure and the guides

fill3(squeeze(TB(1,4,:)), squeeze(TB(2,4,:)), squeeze(TB(3,4,:)), ...
    'k', 'facealpha',.1);
fill3(squeeze(TB(1,4,:)), squeeze(TB(2,4,:)), squeeze(TB(3,4,:)) +
heightstructure, ...
    'k', 'facealpha',.1);

for i=1:6
    plot3([TB(1,4,i) TB(1,4,i)], [TB(2,4,i) TB(2,4,i)], ...
          [TB(3,4,i) (TB(3,4,i) + heightstructure)], 'color', 'r');
    plot3([TG(1,4,i) TG(1,4,i)], [TG(2,4,i) TG(2,4,i)], ...
          [TG(3,4,i) (TG(3,4,i) + heightguides)], 'color', 'b')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% calculate the moment of inertia of the platform
% Simplification = treat the platform as a thin plate

Ix = (sqrt(3)/96)*((side1)^4)*platformmass;
I =[ Ix, Ix, 2*Ix ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%start video orientation

finish = false;
set(gcf,'CurrentCharacter','@');
for t=1:0.2:samples
    if finish == false
        [new_p, new_v] = Translation(p, v, t, R, TG, TP2);
        [new_R, new_w_body] = Orientation(R, w_body, t, p, TG, TP2);
        TT = [new_R new_p; [0 0 0 1]];
        for i = 1:6
            PP2(:,i) = TT*TP2(:,4,i);
        end

        scatter3(PP2(1,1), PP2(2,1), PP2(3,1),'filled','b');
        p = new_p;
        v = new_v;
        R = new_R;
        w_body = new_w_body;
        frame = getframe;
        writeVideo(writerObj,frame);
        k=get(gcf,'CurrentCharacter');
        if k~='@' % has it changed from the dummy character?
            finish=true;
        end
    else
        break;
    end
end

close(writerObj);
close(figure);
end
```

16. InverseDynamics.m

```
function InverseDynamics(file_mov_path, file_dest_data, TG, TP2)
%Inverse Dynamics to obtain resultant force and torque aplied to the center
%of mass of the platform knowing the movement of it.

global platformmass;
global side1;
global g_world;

% calculate the moment of inertia of the platform
% Simplification = treat the platform as a thin plate
Ix = (sqrt(3)/96)*((side1)^4)*platformmass;
I =[ Ix, Ix, 2*Ix ];
Imat = diag(I);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Movement of study (data in world frame)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mov = fopen(file_mov_path);
format = '%f %f %f %f %f %f';
size = [6 Inf];
pose = (fscanf(mov, format, size))'
dt = 1;
fclose(mov);
vx = gradient(pose(:,1),dt);
vy = gradient(pose(:,2),dt);
vz = gradient(pose(:,3),dt);
w1 = gradient(pose(:,4),dt);
w2 = gradient(pose(:,5),dt);
w3 = gradient(pose(:,6),dt);
velocity_cm = [vx,vy,vz,w1,w2,w3]
ax = gradient(velocity_cm(:,1),dt);
ay = gradient(velocity_cm(:,2),dt);
az = gradient(velocity_cm(:,3),dt);
wdot1 = gradient(velocity_cm(:,4),dt);
wdot2 = gradient(velocity_cm(:,5),dt);
wdot3 = gradient(velocity_cm(:,6),dt);
acceleration_cm = [ax,ay,az,wdot1,wdot2,wdot3]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resultant force and torque (inverse dynamics)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rows = length(acceleration_cm(:,1)');
for i=1:rows
    %Total force aplied to the center of mass in world frame
    F_C(i,:) = platformmass * eye(3) * ((acceleration_cm(i,1:3))' - g_world);

    %Total torque aplied to the center of mass in world frame
    R = rpy2r(pose(i,4),pose(i,5),pose(i,6));
    wdot_world = acceleration_cm(i,4:6)';
    w_world = velocity_cm(i,4:6)';
    Imat_world = R * Imat;
    M_C(i,:) =  Imat_world * wdot_world + cross( w_world,
Imat_world*w_world);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Print data of F_C and M_C in a text file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

data_obtained = fopen(file_dest_data,'wt');
fprintf(data_obtained,'%s \n','Position (mm):');
len = length(pose(:,1)');
for i=1:len
    fprintf(data_obtained,'%f\t%f\t%f\t%f\t%f\t%f\n', pose(i,:));
end
fprintf(data_obtained,'\n%s\n','Velocity (mm/s):');
for i=1:len
    fprintf(data_obtained,'%f\t%f\t%f\t%f\t%f\t%f\n', velocity_cm(i,:));
end
fprintf(data_obtained,'\n%s\n','Acceleration (mm/s^2):');
for i=1:len
    fprintf(data_obtained,'%f\t%f\t%f\t%f\t%f\t%f\n', acceleration_cm(i,:));
end
fprintf(data_obtained,'\n%s\n','Force_C (mN):');
for i=1:len
    fprintf(data_obtained,'%f\t%f\t%f\n', F_C(i,:));
end
fprintf(data_obtained,'\n%s\n','Torque_C:');
for i=1:len
    fprintf(data_obtained,'%f\t%f\t%f\n', M_C(i,:));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Find F in each actuator starting from the center of mass
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf(data_obtained, '\nF_Act:\n');
for i=1:len
    Actuator = CenterToGuides(F_C(i,:), M_C(i,:), pose(i,:), TG, TP2)
    fprintf(data_obtained, '%f\t%f\t%f\t%f\t%f\t%f\n',Actuator);
end

fclose(data_obtained);
end
```

17. CenterToGuides.m

```
function [Actuator] = CenterToGuides(F_C, M_C, pose, TG, TP2)
%Obtain forces and moments in each actuator
%F_C, M_C and pose are 6-component vectors in world frame

global lengthleg;
lengthleg = 1000;

%Points of the platform
TT = transl(pose(1:3))*trotx(pose(4))*troty(pose(5))*trotz(pose(6));
for i=1:6
    PP2(:,i) = TT*TP2(:,4,i);
end
z = InverseKinematics(TT,TG,TP2);
Jac = JacobianMatrix(z,TP2,TT,TG);

for i=1:6
    Actuator = -inv(transpose(Jac)) * [F_C'; M_C'];
end
end
```