CUSTOM PD FLIGHT STACK CONTROLLER FOR A QUADROTOR

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE OF

BACHELORS IN TECHNOLOGY

IN

ENGINEERING PHYSICS

SUBMITTED BY:

HARSH MITTAL (2K19/EP/036)

&

ANISH KALSI (2K19/EP/014)

UNDER THE SUPERVISION OF

PROF. AJEET KUMAR

ENGINEERING PHYSICS
DELHI TECHNOLOGICAL UNIVERSITY
(FORMERLY DELHI COLLEGE OF ENGINEERING)
BAWANA ROAD - 110042

MARCH-2021

DEPARTMENT OF APPLIED PHYSICS

INDEX

S. NO.	TOPIC	PAGE NO.
1.	. ……Candidate's Declaration ……	3
2.	. ……Certificate…… ……	4
3.	. ……Acknowledgement ….. ……	5
4.	. ……Abstract……..……..	6
5.	. ……Theory…… ……..	7
6.	……..Code Files…….……	13
7.	……..Simulation Results……	19
8.	. ……Conclusion……	21
9.	. ……References……	22

Delhi Technological University
(Formerly Delhi College Of Engineering)
Bawana Road, 110042

CANDIDATE'S DECLARATION

We, Harsh Mittal 2K19/EP/036, and Anish Kalsi 2K19/EP/014 of the Engineering Physics Department hereby declare that the project dissertation "CUSTOM PD FLIGHT STACK CONTROLLER OF A QUADROTOR" which is submitted by us to the department of Applied Physics, Delhi Technological University in the partial fulfilment of the award of the degree of Bachelors of Technology, is our original and not copied from any source without citation. The work has not been previously formed for the basis of award of any degree, diploma, associateship, fellowship or any other recognition.

Place : New Delhi
Date : 15 March 2021

Harsh Mittal
Anish Kalsi

ENGINEERING PHYSICS

Delhi Technological University
(Formerly Delhi College Of Engineering)
Bawana Road, 110042

CERTIFICATE

We hereby certify that the project dissertation titled “CUSTOM PD FLIGHT STACK CONTROLLER OF A QUADROTOR”, which is submitted by Harsh Mittal 2K19/EP/036 and Anish Kalsi 2K19/EP/014, Delhi technological University, Delhi, in partial fulfilment of the requirement of the award of the Bachelors of Technology degree, is a record of the project carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any degree or diploma to this university or elsewhere.

Harsh Mittal 2K19/EP/036
Anish Kalsi 2K19/EP/014

AKNOWLEDGEMENT

We would like to express our special thanks and gratitude to our Project supervisor Prof. Ajeet Kumar, who gave us the golden opportunity to work on this wonderful project on “CUSTOM PD FLIGHT STACK CONTROLLER OF A QUADROTOR”, which also helped us in doing a lot of research and we came to learn many new things.

Harsh Mittal
(2K19/EP/036)

Anish Kalsi
(2K19/EP/014)

ABSTRACT

Aerial Robotics is an emerging field of robotics, with a special emphasis on quad-rotors, are rapidly growing in popularity. Over the past few years the drone technology have become a central function of various business and government organisations. The drone technology has emerged in the last few years and grew its domain from military usage to hobbyists, photographers, precision agriculture, medical to delivery. This project demonstrates a model that demonstrates a vertical flight stack for an aerial robot in two conditions, first where a drone starts from ground and reaches a height of one metre on earth surface and the second where the drone starts on the surface of mars and tries to reach one metre height, in the extreme conditions as per told by Jet Propulsion Laboratory, NASA. In this project we implement and simulate a Proportional Derivative controller using Matlab.

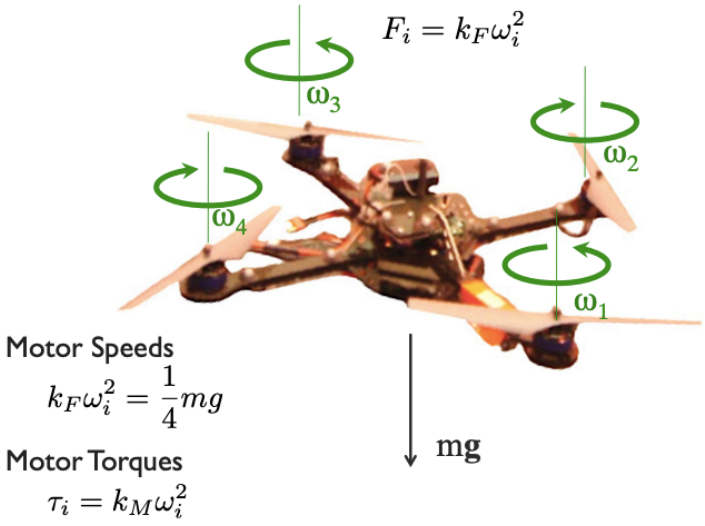
THEORY

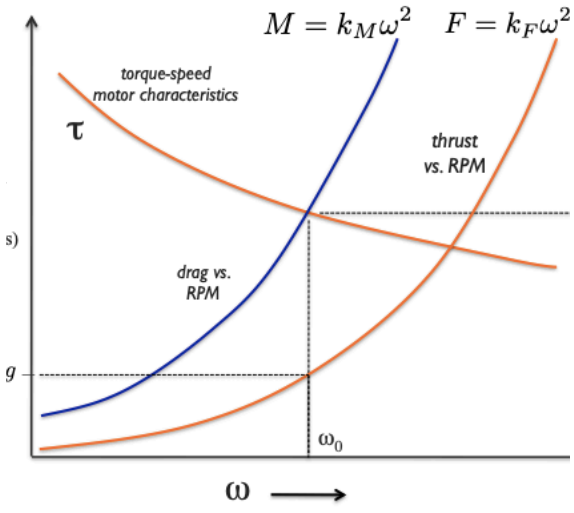
Quadrotor Simulator

We utilise one of the Matlab's ODE solver called ode45, to simulate the behaviour of the quadrotor. We then utilise plot3 to visualise the current state of quadrotor at each time at each step.

Basic Mechanics

The thrust versus rotor speed relation is nearly quadratic in nature. By geometry, every motor has to support roughly one fourth of the total weight of the quadrotor in the equilibrium and overcome the drag in each motor.
The motor size should be such that they provide the torque necessary to overcome drag.





Quadrotor Equations of Motion :

Let the force experienced by each rotor be Fi = F1, F2, F3, F4

Let the weight of the quadrotor be = m

Let the Momentum of each of the rotor be = Mi = M1, M2, M3, M4

The equations become

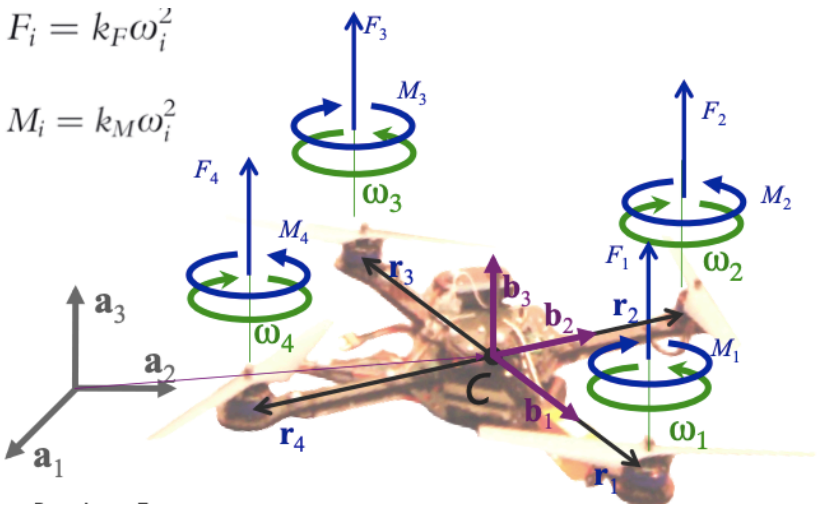
$$F = F1 + F2 + F3 + F4 - mg$$

$$M = r1XF1 + r2XF2 + r3XF3 + r4XF4 + M1 + M2 + M3 + M4$$

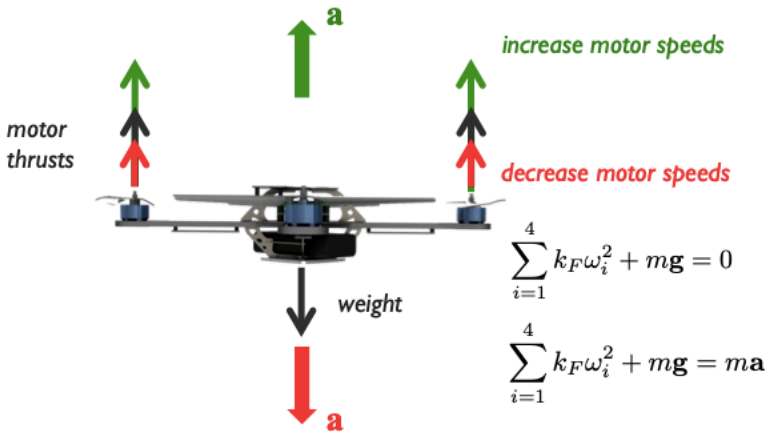
The generalised equations are

$$Fi = kf(\omega i)^2$$

$$Mi = km(\omega i)^2$$



Acceleration in Vertical Direction

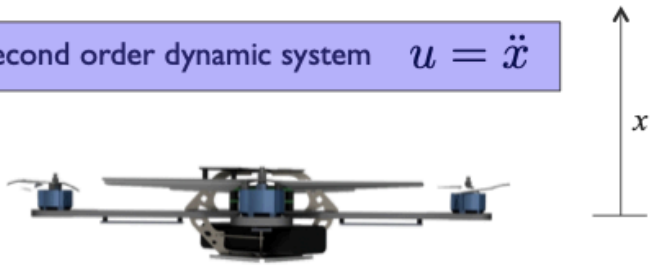


Control of Height

$$\sum_{i=1}^{4} k_F \omega_i^2 + m\mathbf{g} = m\mathbf{a} \rightarrow a = \frac{d^2x}{dt^2} = \ddot{x}$$

$$\text{Input} \quad u = \frac{1}{m}\left[\sum_{i=1}^{4} k_F \omega_i^2 + m\mathbf{g}\right]$$

Second order dynamic system $u = \ddot{x}$



Control of a linear second order system

$$x'' = u$$

We want the trajectory to be as close as xdes(t) as possible.
The error then becomes

$$e(t) = xdes(t) - x(t),$$

Where, we want e(t) to exponentially decrease to zero.

$$e'' + kve' + kpe = 0$$

Kv, Kp > 0

$$u(t) = x''des(t) + kve'(t) + kpe(t)$$

Where the first term is feedforward, second term is derivative and third term is proportional.

PD Controller
Proportional controller acts as a spring (capacitance) response.
Derivative controller is a viscous dashpot (resistance) response.
Large derivative gain makes the system over damped and the system converges slowly.

$$u(t) = \ddot{x}^{\text{des}}(t) + K_v\dot{e}(t) + K_p e(t)$$

The dynamic equation for the motion of the quadrotor in the z direction is given as :

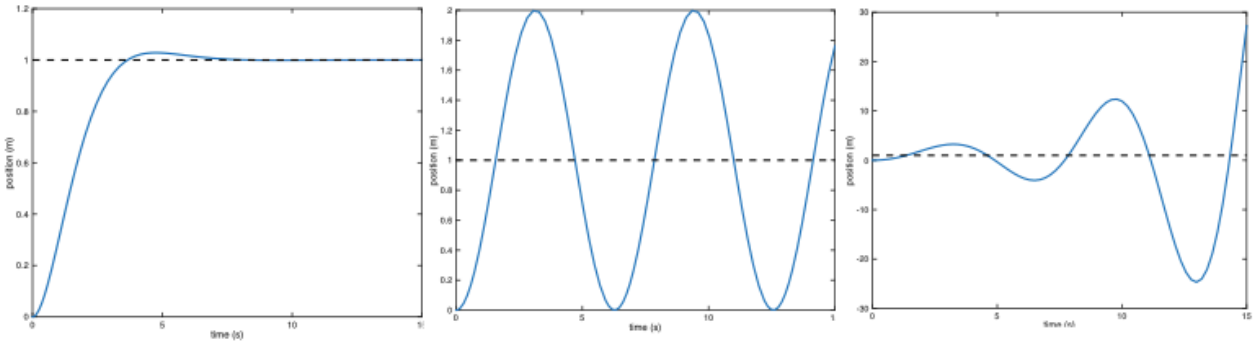
$$z'' = u/m - g$$

Hence the the control input of the PD controller becomes :

$$u = m(z''des + Kpe + Kve' + g)$$

Where e and e' calculated from the current and desired states $z,\ zdes,\ z', z'des.$

Effects of gain for a PD Control System



Stable

$K_p,\ K_v\ >\ 0$

Marginally Stable

$K_p > 0,\quad K_v\ = 0$

Unstable

K_p or $K_v\ < 0$

Euler Angles

Any rotation can be described by three successive rotations about linearly independent axes.
The Euler's angles can be determined by the following equation

$$R = Rot(z, \varphi)\ \text{x}\ Rot(y, \theta)\ \text{x}\ Rot(z, \psi)$$

Euler's Theorem

Any displacement of a rigid body such that a point n the rigid body, say O, remains fixed, is equivalent to the rotation about the fixed axis through the point O.

Quaternions

Quaternion :

$$Q = (Q0, Q1, Q2, Q3)$$

This can be interpreted as a constant + vector

$$Q = (Q0, Q)$$

Quaternions can be used to represent rigid body rotations.

Angle of rotation : ϕ
Axis of rotation : u

The equivalent Quaternion equation becomes :

$$q = (cos(\varphi/2), u1sin(\varphi/2), u2sin(\varphi/2), u3sin(\varphi/2))$$

Quaternion product is more numerically stable than matrix multiplication.

Newton - Euler Equations

Newton's Equation of Motion for a single particle of mass m is given as

$$F = ma$$

Newton - Euler Equations for a System of Particles

The center of mass for a system of particle, S acceleration in an inertial frame (A), as if it were a single particle of mass M, acted upon by an external force equal to net external force.

$$F = sum(Fi) = m^a d^a/dt(v^c)$$

Rotational Equations of Motion of a Rigid Body

The rate of change of angular momentum of the rigid body B relative to C in A is equal to the resultant momentum of all external forces acting on the body relative to C

$$H^S c = Ic \,.\, \omega^B$$

Where Ic is the inertia tensor with C as the origin

Planar Quadrotor Model

$$my'' = -sin(\varphi)u1$$

$$mz'' = cos(\varphi)u1 - mg$$

$$Ixx\varphi'' = u2$$

The States become :

$$x1 = y$$

$$x2 = z$$

$$x3 = \varphi$$

$$x4 = y'$$

$$x5 = z'$$

$$x6 = \varphi'$$

The State Vector :

$$[x1x2x3x4x5x6]^T = [yz\varphi y'z'\varphi']^T$$

The System of First Order Differential Equations :

$$d/dt(x1) = d/dt(y) = y' = x4$$

$$d/dt(x4) = d/dt(y') = y'' = -(sin(\varphi)u1)/m = -(sin(x3)u1)/m$$

$$d/dt(x2) = d/dt(z) = z' = x5$$

$$d/dt(x5) = d/dt(z') = z'' = (cos(\varphi)u1)/m - g = (cos(x3)u1)/m - g$$

$$d/dt(x3) = d/dt(\varphi) = \varphi' = x6$$

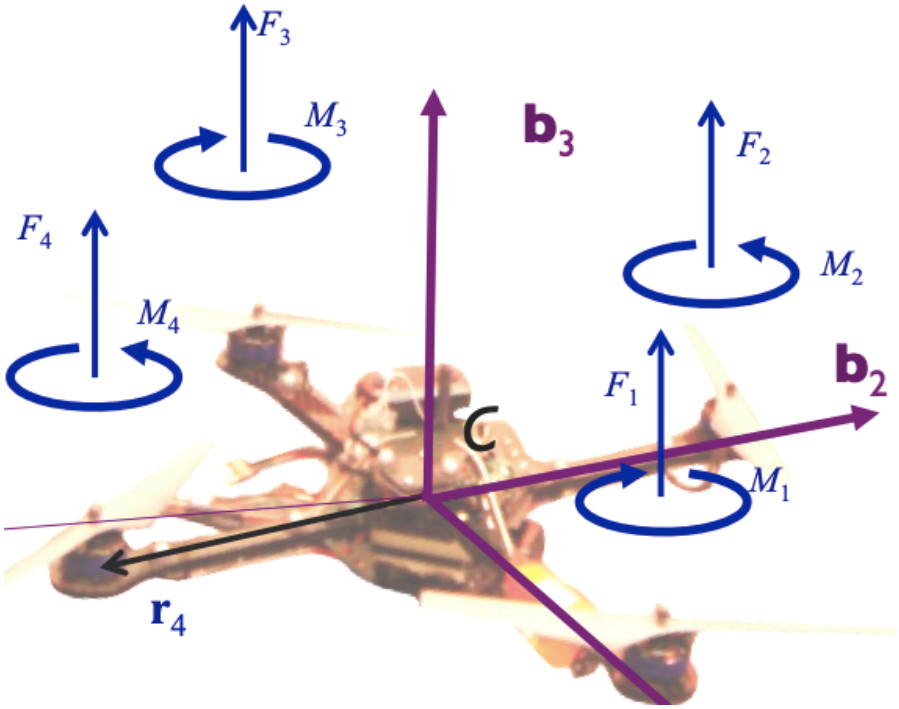
$$d/dt(x6) = d/dt(\varphi') = \varphi'' = u2/Ixx$$

The final system of equations become :

$$my'' = -sin(\varphi)u1$$

$$mz'' = cos(\varphi)u1 - g$$

$$Ixx\varphi'' = u2$$



CODE FILES

1. controller.m

```
function [ u ] = controller(~, s, s_des, params)
u = 0;
Kp = 40;
Kv = 5;
error = s_des - s;
u = params.mass*params.gravity + Kp*error(1) + Kv*error(2);
end
```

2. fixed_set_point.m

```
function [ s_des ] = fixed_set_point(t, z_des)
if t == 0
  s_des = [0;0];
else
  s_des = [z_des;0];
end
end
```

3. height_control.m

```
function [t_out, z_out] = height_control(trajhandle, controlhandle)
addpath('utils');
video = false;
video_filename = 'height_control.avi';
params = sys_params;
% real-time
real_time = true;
% FIGURES
disp('Initializing figures...')
if video
  video_writer = VideoWriter(video_filename, 'Uncompressed AVI');
  open(video_writer);
end
h_fig = figure;
sz = [1000 600]; % figure size
screensize = get(0,'ScreenSize');
xpos = ceil((screensize(3)-sz(1))/2); % center the figure on the screen
horizontally
ypos = ceil((screensize(4)-sz(2))/2); % center the figure on the screen
vertically
set(h_fig, 'Position', [xpos ypos sz])
h_3d = subplot(1,2,1);
axis equal
grid on
view(0,0);
xlabel('x [m]'); ylabel('y [m]'); zlabel('z [m]');
h_2d = subplot(1,2,2);
plot_2d = plot(h_2d, 0, 0);
grid on;
xlabel('t [s]'); ylabel('z [m]');
```

```
quadcolors = lines(1);
set(gcf,'Renderer','OpenGL')
% INITIAL CONDITIONS
max_iter  = 100;       % max iteration
starttime = 0;         % start of simulation in seconds
tstep     = 0.01;      % this determines the time step at which the solution is
given
cstep     = 0.05;      % image capture time interval
nstep     = cstep/tstep;
time      = starttime; % current time
% Get start and stop position
des_start = trajhandle(0);
des_stop  = trajhandle(inf);
stop_pos  = des_stop(1);
x0        = des_start;
xtraj     = nan(max_iter*nstep, length(x0));
ttraj     = nan(max_iter*nstep, 1);
x = x0;        % state
pos_tol   = 0.01;
vel_tol   = 0.01;
% RUN SIMULATION
disp('Simulation Running')
% Main loop
for iter = 1:max_iter
  timeint = time:tstep:time+cstep;
  tic;
  % Initialize quad plot
  if iter == 1
    subplot(1,2,1);
    quad_state = simStateToQuadState(x0);
    QP = QuadPlot(1, quad_state, params.arm_length, 0.05, quadcolors(1,:),
max_iter, h_3d);
    quad_state = simStateToQuadState(x);
    QP.UpdateQuadPlot(quad_state, time);
    h_title = title(h_3d, sprintf('iteration: %d, time: %4.2f', iter, time));
  end
  % Run simulation
  [tsave, xsave] = ode45(@(t,s) sys_eom(t, s, controlhandle, trajhandle,
params), timeint, x);
  x = xsave(end, :)';
  % Save to traj
  xtraj((iter-1)*nstep+1:iter*nstep,:) = xsave(1:end-1,:);
  ttraj((iter-1)*nstep+1:iter*nstep) = tsave(1:end-1);
  % Update quad plot
  subplot(1,2,1)
  quad_state = simStateToQuadState(x);
  QP.UpdateQuadPlot(quad_state, time + cstep);
  set(h_title, 'String', sprintf('iteration: %d, time: %4.2f', iter, time +
cstep))
  time = time + cstep; % Update simulation time
  set(plot_2d, 'XData', ttraj(1:iter*nstep), 'YData', xtraj(1:iter*nstep,1));
  if video
    writeVideo(video_writer, getframe(h_fig));
  end
  t = toc;
  % Check to make sure ode45 is not timing out
  if(t > cstep*50)
    err = 'Ode solver took too long for a step. Maybe the controller is
unstable.';
    disp(err);
    break;
  end
  % Pause to make real-time
  if real_time && (t < cstep)
    pause(cstep - t);
  end
```

```
end
% Check termination criteria
if norm(stop_pos - x(1)) < pos_tol && norm(x(2)) < vel_tol
  err = [];
else
  err = 'Did not converge';
end
disp('Simulation done');
if video
  close(video_writer);
end
if ~isempty(err)
  disp(['Error: ', err]);
end
t_out = ttraj(1:iter*nstep);
z_out = xtraj(1:iter*nstep,1);
end
```

4. runsim.m

```
close all;
clear;
z_des = 1;
trajhandle = @(t) fixed_set_point(t, z_des);
controlhandle = @controller;
[t, z] = height_control(trajhandle, controlhandle);
```

UTILS

1. quad_pos.m

```
function [ quad ] = quad_pos( pos, rot, L, H )
if nargin < 4; H = 0.05; end
wHb   = [rot pos(:); 0 0 0 1];
quadBodyFrame  = [L 0 0 1; 0 L 0 1; -L 0 0 1; 0 -L 0 1; 0 0 0 1; 0 0 H 1]';
quadWorldFrame = wHb * quadBodyFrame;
quad           = quadWorldFrame(1:3, :);
end
```

2. QuadPlot.m

```
classdef QuadPlot < handle
    %QUADPLOT Visualization class for quad
    properties (SetAccess = public)
        k = 0;
        qn;             % quad number
        time = 0;       % time
        state;          % state
        rot;            % rotation matrix body to world

        color;          % color of quad
```

```
        wingspan;       % wingspan
        height;         % height of quad
        motor;          % motor position

        state_hist;     % position history
        time_hist;      % time history
        max_iter;       % max iteration
    end
    properties (SetAccess = private)
        h_3d
        h_m13;  % motor 1 and 3 handle
        h_m24;  % motor 2 and 4 handle
        h_qz;   % z axis of quad handle
        h_qn;   % quad number handle
        h_pos_hist;     % position history handle
        text_dist;  % distance of quad number to quad
    end
    methods
        % Constructor
        function Q = QuadPlot(qn, state, wingspan, height, color, max_iter,
h_3d)
            Q.qn = qn;
            Q.state = state;
            Q.wingspan = wingspan;
            Q.color = color;
            Q.height = height;
            Q.rot = QuatToRot(Q.state(7:10));
            Q.motor = quad_pos(Q.state(1:3), Q.rot, Q.wingspan, Q.height);
            Q.text_dist = Q.wingspan / 3;

            Q.max_iter = max_iter;
            Q.state_hist = zeros(6, max_iter);
            Q.time_hist = zeros(1, max_iter);

            % Initialize plot handle
            if nargin < 7, h_3d = gca; end
            Q.h_3d = h_3d;
            hold(Q.h_3d, 'on')
            Q.h_pos_hist = plot3(Q.h_3d, Q.state(1), Q.state(2), Q.state(3),
'r.');
            Q.h_m13 = plot3(Q.h_3d, ...
                Q.motor(1,[1 3]), ...
                Q.motor(2,[1 3]), ...
                Q.motor(3,[1 3]), ...
                '-ko', 'MarkerFaceColor', Q.color, 'MarkerSize', 5);
            Q.h_m24 = plot3(Q.h_3d, ...
                Q.motor(1,[2 4]), ...
                Q.motor(2,[2 4]), ...
                Q.motor(3,[2 4]), ...
                '-ko', 'MarkerFaceColor', Q.color, 'MarkerSize', 5);
            Q.h_qz = plot3(Q.h_3d, ...
                Q.motor(1,[5 6]), ...
                Q.motor(2,[5 6]), ...
                Q.motor(3,[5 6]), ...
                'Color', Q.color, 'LineWidth', 2);
            Q.h_qn = text(...
                Q.motor(1,5) + Q.text_dist, ...
                Q.motor(2,5) + Q.text_dist, ...
                Q.motor(3,5) + Q.text_dist, num2str(qn));
            hold(Q.h_3d, 'off')
        end

        % Update quad state
        function UpdateQuadState(Q, state, time)
            Q.state = state;
            Q.time = time;
```

```
            Q.rot = QuatToRot(state(7:10))'; % Q.rot needs to be body-to-world
        end

        % Update quad history
        function UpdateQuadHist(Q)
            Q.k = Q.k + 1;
            Q.time_hist(Q.k) = Q.time;
            Q.state_hist(:,Q.k) = Q.state(1:6);
        end

        % Update motor position
        function UpdateMotorPos(Q)
            Q.motor = quad_pos(Q.state(1:3), Q.rot, Q.wingspan, Q.height);
        end

        % Truncate history
        function TruncateHist(Q)
            Q.time_hist = Q.time_hist(1:Q.k);
            Q.state_hist = Q.state_hist(:, 1:Q.k);
        end
        % Update quad plot
        function UpdateQuadPlot(Q, state, time)
            Q.UpdateQuadState(state, time);
            Q.UpdateQuadHist();
            Q.UpdateMotorPos();
            set(Q.h_m13, ...
                'XData', Q.motor(1,[1 3]), ...
                'YData', Q.motor(2,[1 3]), ...
                'ZData', Q.motor(3,[1 3]));
            set(Q.h_m24, ...
                'XData', Q.motor(1,[2 4]), ...
                'YData', Q.motor(2,[2 4]), ...
                'ZData', Q.motor(3,[2 4]));
            set(Q.h_qz, ...
                'XData', Q.motor(1,[5 6]), ...
                'YData', Q.motor(2,[5 6]), ...
                'ZData', Q.motor(3,[5 6]))
            set(Q.h_qn, 'Position', ...
                [Q.motor(1,5) + Q.text_dist, ...
                Q.motor(2,5) + Q.text_dist, ...
                Q.motor(3,5) + Q.text_dist]);
            set(Q.h_pos_hist, ...
                'XData', Q.state_hist(1,1:Q.k), ...
                'YData', Q.state_hist(2,1:Q.k), ...
                'ZData', Q.state_hist(3,1:Q.k));
            drawnow;
        end
    end

end
```

3. QuadToRot.m

```
function R = QuatToRot(q)
%QuatToRot Converts a Quaternion to Rotation matrix
% normalize q
q = q./sqrt(sum(q.^2));
qahat(1,2) = -q(4);
qahat(1,3) = q(3);
qahat(2,3) = -q(2);
qahat(2,1) = q(4);
```

```
qahat(3,1) = -q(3);
qahat(3,2) = q(2);
R = eye(3) + 2*qahat*qahat + 2*q(1)*qahat;
end
```

4. simStateToQuadState.m

```
function quad_state = simStateToQuadState(sim_state)
% simStateToQuadState Convert sim state to the 13 element quad state
% Quad state vector = [x, y, z, xd, yd, zd, qw, qx, qy, qz, p, q, r]
quad_state = zeros(13,1);
quad_state(1) = 0;
quad_state(2) = 0;
quad_state(3) = sim_state(1);
quad_state(4) = 0;
quad_state(5) = 0;
quad_state(6) = sim_state(2);
quad_state(7) = 1;
quad_state(8) = 0;
quad_state(9) = 0;
quad_state(10) = 0;
quad_state(11) = 0;
quad_state(12) = 0;
quad_state(13) = 0;
end
```

5. sys_eom.m

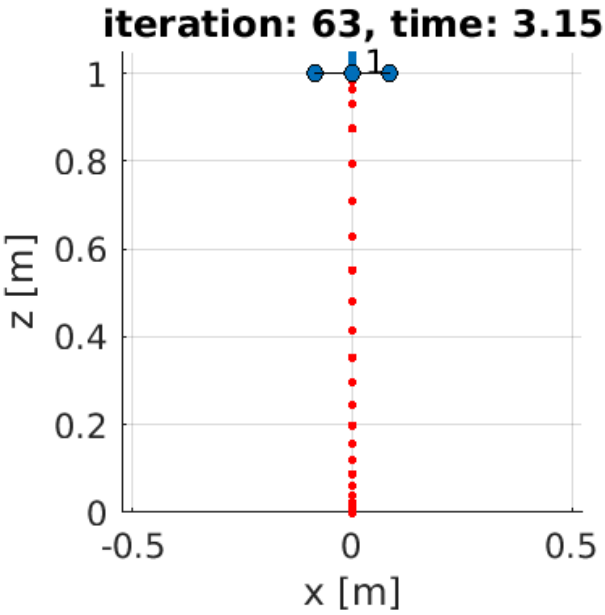
```
function [ sdot ] = sys_eom(t, s, controlhandle, trajhandle, params)
% sys_eom Differential equation for the height control system
s_des = trajhandle(t);
u_des = controlhandle(t, s, s_des, params);
u_clamped = min(max(params.u_min, u_des), params.u_max);
sdot = [s(2);
        u_clamped/params.mass - params.gravity];
end
```

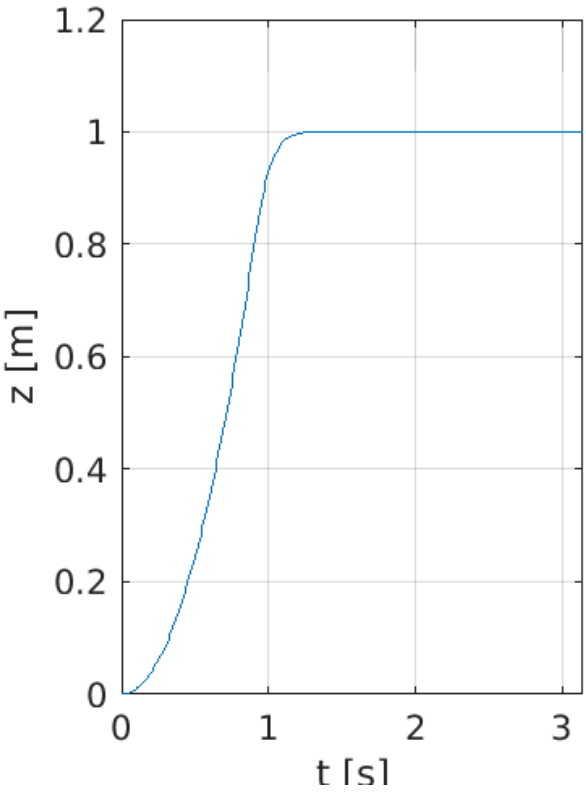
6. sys_params.m

```
function [ params ] = sys_params_limit_thrust()
% Physical properties
params.gravity = 9.81;
params.mass = 0.18;
params.arm_length = 0.086;
% Actuator limits
params.u_min = 0;
params.u_max = 1.2*params.mass*params.gravity;
end
```

SIMULATION RESULTS

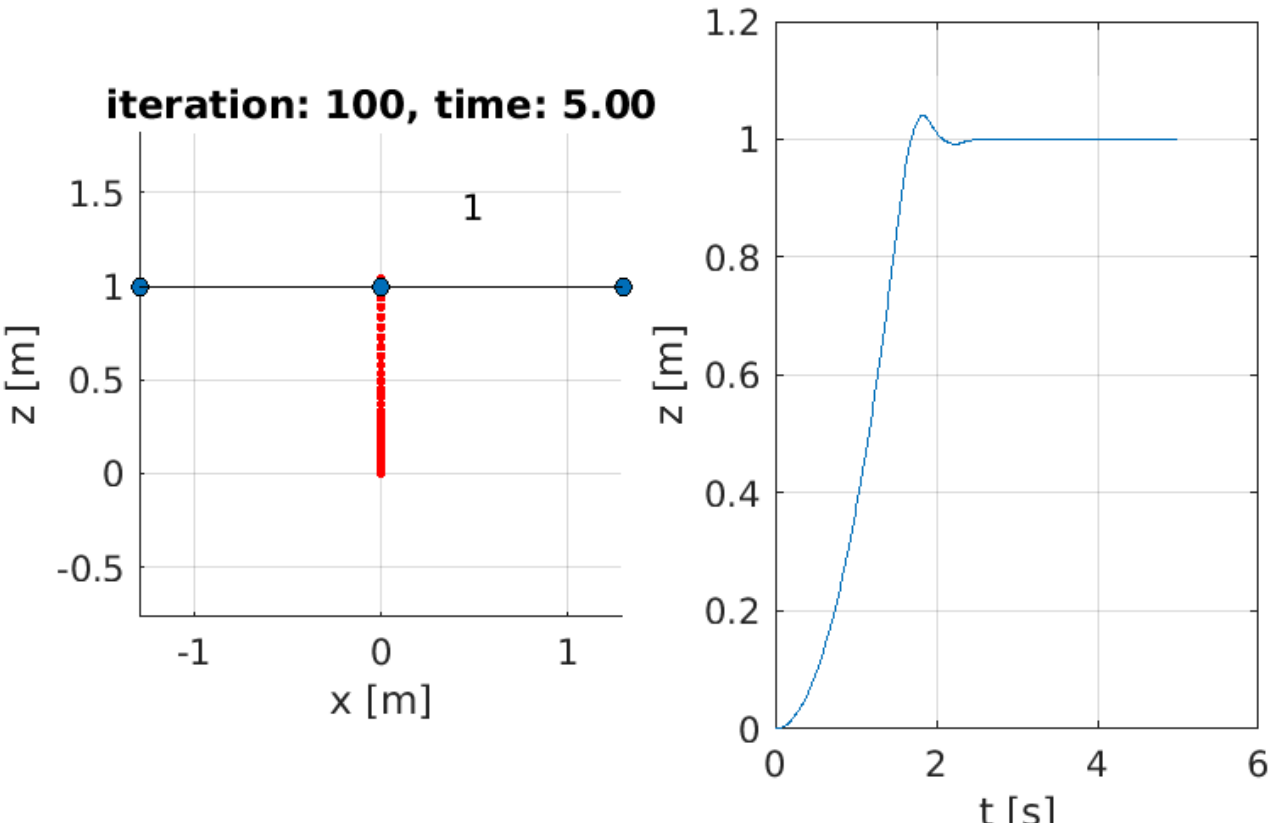
- **For Height = 1m, Earth Like Situation**





- **For Mars Like Situation**

g = ‘3.711’
param.arm_lenght = ’15 * 0.086’



CONCLUSION

In this project we have implemented a PD controller for height control of the quadrotor. Then tune the proportional gain (Kp) and derivative gain (Kv) in the file controller.m until the quadrotor converges quickly and smoothly to a step response input. Firstly the controller was made and tested by flying the drone to a height of one metre on earth like situations and then the controller stack was changed for Mars like situations and the quadrotor was made to fly to one metre, and a comparison was made. The major difference things to consider while making flight stake for Mars is that the stack for Earth when tested on Mars does not converge initially rather it oscillated, indicating an unstable PD controller. Other factors for Mars situations were that the drone rotor blades are longer as well as the PD is changed substantially to account for smaller 'g' values.

The Kp and Kv determine the response as seen in the simulations above.
If the value of Kp is low, then it is called soft response. If Kv is high then it is over-damped system. The values of Kv and Kp chosen for this aerial robot are near perfect for Earth like conditions, while they are less stable for Mars like environment mainly due to change in gravitation, base area and Martian winds. It would be better to develop a PID controller instead for Mars like environment, which will make it more stable and accurate.

$$K_i \int_0^t e(\tau) d\tau$$

Adding the Integral term makes it third order closed loop system and therefore more sensitive and accurate, and thereby stable.

REFRENCES

[1] N.Michael, D. Mellinger, Q. Lindsey and V. Kumar, “The GRASP Multiple Micro-UAV Testbed,” IEEE Robotics and Automation Magazine 2010

[2] Feng Zou, Raymond E. Arvidson, Keith Benet, Brain Trease, Ranson Linderman, Paolo Bellutta, Karl Lagnemma and Carmine Senatore, “ Simulations of Mars Rover Traverses”, Journal of Field Of Robotics 2013