|  |  |
| --- | --- |
| % This script defines a continuous-time nonlinear quadrotor model and  % generates a state function and its Jacobian function used by the  % nonlinear MPC controller in the quadrotor path following example.    % Copyright 2019 The MathWorks, Inc.    % Create symbolix variables for states, MVs and parameters  syms xt(t) yt(t) zt(t) phit(t) thetat(t) psit(t) xdott(t) ydott(t)...  zdott(t) phidott(t) thetadott(t) psidott(t)  syms u1 u2 u3 u4 Ixx Iyy Izz k l m b g  syms x y z phi theta psi xdot ydot zdot phidot thetadot psidot    % phi: roll angle  % theta: pitch angle  % psi: yaw angle  % ui: squared angular velocity of rotor i  % g: gravity  % b: drag constant  % k: lift constant  % l: distance between rotor and com  % Iii: diagonal elements of inertia matrix    % Set values for dynamics parameters  IxxVal = 1.2;  IyyVal = 1.2;  IzzVal = 2.3;  kVal = 1;  lVal = 0.25;  mVal = 2;  bVal = 0.2;  gVal = 9.81;    paramValues = [IxxVal IyyVal IzzVal kVal lVal mVal bVal gVal];    % Group symbolic variables  statet = {xt(t) yt(t) zt(t) phit(t) thetat(t) psit(t) xdott(t) ...  ydott(t) zdott(t) phidott(t) thetadott(t) psidott(t)};  state = {x y z phi theta psi xdot ydot zdot phidot thetadot psidot};  state\_diff = {diff(xt(t),t), diff(yt(t),t), diff(zt(t),t), ...  diff(phit(t),t), diff(thetat(t),t), diff(psit(t),t)};  state\_dot = {xdot ydot zdot phidot thetadot psidot};    % Transformation matrix for angular velocities from inertial frame to body frame  W = [1, 0, -sin(thetat);  0, cos(phit), cos(thetat)\*sin(phit);  0, -sin(phit), cos(thetat)\*cos(phit)];    %R-ZYX Euler  Rz = [cos(psit), -sin(psit), 0;  sin(psit), cos(psit), 0;  0, 0, 1];  Ry = [cos(thetat), 0, sin(thetat);  0, 1, 0;  -sin(thetat), 0, cos(thetat)];  Rx = [1, 0, 0;  0, cos(phit), -sin(phit);  0, sin(phit), cos(phit)];    % Rotation matrix from body frame to inertial frame  R = Rz\*Ry\*Rx;    % Jacobian (relates body frame to inertial frame velocities)  I = [Ixx, 0, 0; 0, Iyy, 0; 0, 0, Izz];  J = W.'\*I\*W;    % Coriolis forces  dJ\_dt = diff(J);  dJ\_dt = subs(dJ\_dt,[state\_diff statet],[state\_dot state]);  h\_dot\_J = [phidott(t), thetadott(t), psidott(t)]\*J;  h\_dot\_J = subs(h\_dot\_J,[state\_diff statet],[state\_dot state]);  grad\_temp\_h = jacobian(h\_dot\_J,[phi theta psi]);  C = dJ\_dt - 1/2\*grad\_temp\_h;    % Torques in the direction of phi, theta, psi  tau\_beta = [l\*k\*(-u2 + u4);l\*k\*(-u1 + u3);b\*(-u1+u2-u3+u4)];    % Total thrust  T = k\*(u1+u2+u3+u4);    % Dynamics  f(1) = xdott;  f(2) = ydott;  f(3) = zdott;  f(4) = phidott;  f(5) = thetadott;  f(6) = psidott;    % Equations for COM configuration  f(7:9) = -g\*[0;0;1] + R\*[0;0;T]/m;    % Euler Lagrange equations for angular dynamics  f(10:12) = inv(J)\*(tau\_beta - C\*[phidott(t); thetadott(t); psidott(t)]);    % Replace parameters and drop time dependence  f = subs(f, [Ixx Iyy Izz k l m b g], paramValues);  f = subs(f,statet,state);  f = simplify(f);    % Calculate linearization  A = jacobian(f,[state{:}]);  control = [u1, u2, u3, u4];  B = jacobian(f,control);    % Create QuadrotorStateFcn.m  matlabFunction(transpose(f),'File','QuadrotorStateFcn11',...  'Vars',{transpose([state{:}]),transpose(control)})  % Create QuadrotorStateJacobianFcn.m  matlabFunction(A, B,'File','QuadrotorStateJacobianFcn11',...  'Vars',{transpose([state{:}]),transpose(control)})    %Clear symbolic variables  clear    % Confirm the functions are generated successfully  while isempty(which('QuadrotorStateJacobianFcn11'))  pause(0.1);  end | % This script defines a continuous-time nonlinear multirotor model and  % generates a state function and its Jacobian function used by the  % nonlinear MPC controller in the multirotor path following example.    % Copyright 2019 The MathWorks, Inc.    % Create symbolix variables for states, MVs and parameters  syms xt(t) yt(t) zt(t) xdott(t) ydott(t) zdott(t) zerot(t) phit(t) thetat(t) psit(t) ...  phidott(t) thetadott(t) psidott(t)  syms fax fay faz Ta1 Ta2 Ta3 fc1 fc2 fc3 dF1 dF2 dF3 Pd1 Pd2 Pd3  syms Ixx Iyy Izz k l m b g L dT1 dT2 dT3 rcom1 rcom2 rcom3  syms x y z xdot ydot zdot zero phi theta psi phidot thetadot psidot    % phi: roll angle  % theta: pitch angle  % psi: yaw angle  % ui: squared angular velocity of rotor i  % g: gravity  % b: drag constant  % k: lift constant  % l: distance between rotor and com  % Iii: diagonal elements of inertia matrix    % Set values for dynamics parameters  IxxVal = 1.2;  IyyVal = 1.2;  IzzVal = 2.3;  kVal = 1;  lVal = 0.25;  mVal = 2;  bVal = 0.2;  gVal = 9.81;  LVal = 0;  dTVal\_1=0;dTVal\_2=0;dTVal\_3=0;  rcomVal\_1=0;rcomVal\_2=0;rcomVal\_3=0;  paramValues = [IxxVal IyyVal IzzVal kVal lVal mVal bVal gVal LVal dTVal\_1 dTVal\_2 dTVal\_3 rcomVal\_1 rcomVal\_2 rcomVal\_3];    % Group symbolic variables  statet = {xt(t) yt(t) zt(t) xdott(t) ydott(t) zdott(t) zerot(t)...  phit(t) thetat(t) psit(t) phidott(t) thetadott(t) psidott(t)};  state = {x y z xdot ydot zdot zero phi theta psi phidot thetadot psidot};  state\_diff = {diff(xt(t),t), diff(yt(t),t), diff(zt(t),t), ...  diff(zerot(t),t),diff(phit(t),t), diff(thetat(t),t), diff(psit(t),t)};  state\_dot = {xdot ydot zdot zerodot phidot thetadot psidot};    % Transformation matrix for angular velocities from inertial frame to body frame  % W = [1, 0, -sin(thetat);  % 0, cos(phit), cos(thetat)\*sin(phit);  % 0, -sin(phit), cos(thetat)\*cos(phit)];    %R-ZXY Euler  Rz = [cos(psit), sin(psit), 0;  -sin(psit), cos(psit), 0;  0, 0, 1];  Rx = [1, 0, 0;  0, cos(thetat), -sin(thetat);  0, -sin(thetat), cos(thetat)];  Ry = [cos(phit), 0, -sin(phit);  0, 1, 0;  sin(phit), 0, cos(phit)];    % Rotation matrix from body frame to inertial frame  Rs = Rz\*Rx\*Ry;    % % Jacobian (relates body frame to inertial frame velocities)  J = [Ixx, 0, 0; 0, Iyy, 0; 0, 0, Izz];  % J = W.'\*I\*W;    % % Coriolis forces  % dJ\_dt = diff(J);  % dJ\_dt = subs(dJ\_dt,[state\_diff statet],[state\_dot state]);  % h\_dot\_J = [phidott(t), thetadott(t), psidott(t)]\*J;  % h\_dot\_J = subs(h\_dot\_J,[state\_diff statet],[state\_dot state]);  % grad\_temp\_h = jacobian(h\_dot\_J,[phi theta psi]);  % C = dJ\_dt - 1/2\*grad\_temp\_h;    % Torques in the direction of phi, theta, psi  ta =[Ta1; Ta2; Ta3];    % Total thrust  T = [fax; fay; faz];  % Dynamics  %lambda=L;  Rd=[1 0 0;0 1 0;0 0 1-L];  SL=Rs\*Rd\*transpose(Rs);  %V=[Vx;Vy;Vz];  Rb=Rz;  fa=[fax; fay; faz];  fc=[fc1;fc2;fc3];  dF=[dF1;dF2;dF3];      ab=(m^-1)\*((Rb\*fa)+(L\*fc)+dF)+[0;0;g];  V=[xdott(t);ydott(t);zdott(t)];  P\_dot=SL\*V;  %rT=[rx;ry;rz];  Pd=[Pd1; Pd2; Pd3];  P=[xt(t);yt(t);zt(t)];  rT=P-Pd;  Q=[zerot(t); phit(t);thetat(t);psit(t)];  W=[phidott(t);thetadott(t);psidott(t)];  Q\_dot=(1/2)\*Q.\*[0;W];    J=[Ixx 0 0;0 Iyy 0;0 0 Izz];  %ta=[tx;ty;tz];  h=J\*W;  rcom=[rcom1;rcom2;rcom3];  h1=transpose(Rb)\*[0;0;m\*g];  dT=[dT1;dT2;dT3];  h2=L\*transpose(Rb)\*fc;  W\_dot=(inv(J))\*(ta-cross(W,h)+cross(rcom,h1)+cross(rT,h2)+dT);  V\_dot=SL\*((ab)+Rb\*(cross(W\_dot,rT)+(cross(W,cross(W,rT)))));    f=[P\_dot,V\_dot,Q\_dot,W\_dot];  % Replace parameters and drop time dependence  f = subs(f, [Ixx Iyy Izz k l m b g L dT1 dT2 dT3 rcom1 rcom2 rcom3], paramValues);  f = subs(f,statet,state);  f = simplify(f);    % Calculate linearization  A = jacobian(f,[state{:}]);  control = [fax, fay, faz, Ta1, Ta2, Ta3, fc1 fc2 fc3 dF1 dF2 dF3];  B = jacobian(f,control);    % Create multirotorStateFcn.m  matlabFunction(transpose(f),'File','multirotorStateFcn',...  'Vars',{transpose([state{:}]),transpose(control)})  % Create multirotorStateJacobianFcn.m  matlabFunction(A, B,'File','multirotorStateJacobianFcn',...  'Vars',{transpose([state{:}]),transpose(control)})    %Clear symbolic variables  clear    % Confirm the functions are generated successfully  while isempty(which('multirotorStateJacobianFcn'))  pause(0.1);  end |