```
function [nextChasis,nextJoint,nextWheel] = NextState(current,speeds, dt, limits)
%Next State Summary of this function goes here
% inputs:  Current: 12-vector representing the current config of the robot.
%          in the order: PHI, X, Y, θ1, θ2, θ3, θ4, θ5, W1, W2, W3, W4
%          Speeds: a 9-vector indicating 4 wheel speeds & 5 joint speeds in
%          the order: U1, U2, U3, U4, (derivative of: θ1, θ2, θ3, θ4, θ5)
%          dt : timestep
%          limit: positive value indicating the maximum angular speed
% outputs: nextJoint: the new joint angles
%          nextWheel: the new wheel angles
%          nextChasis: the new chassis config from odometry
%   This function takes the current positions and speeds, and advances them
%   by one timestep, outputing the new positions, under the max speed
%% check for speeds over the limit, reduce if neccassary, but keep the sign
for i = 1:9
    if abs(speeds(i)) > limits
        speeds(i) = limits*sign(speeds(i));
    end
end
%% update the simple states via Euler step
nextJoint = current(4:8) + speeds(5:9)*dt;
nextWheel = current(9:12) + speeds(1:4)*dt;

%% odometry
% set the constants for this robot:
r = .0475; l=.47/2; w=.3/2;
H0 = [-l-w, 1 -1; l+w 1 1; l+w 1 -1; -l-w 1 1]./r ;
F = pinv(H0);
%calculate change in wheel angles
dtheta= (speeds(1:4).*dt)';
%find twist change
Vb= F*dtheta;
%find dQb, the change in q in the body frame
if Vb(1) == 0
    dQb = [0; Vb(2); Vb(3)];
else
    dQb = [Vb(1); (Vb(2).*sin(Vb(1))+ Vb(3).*(cos(Vb(1))-1))./Vb(1);(Vb(3).*sin(Vb ↙
(1))+ Vb(2).*(1-cos(Vb(1))))./Vb(1)];
end
%delta Q is a rotation matrix times dQb and QnexTtime= Qnow +delta Q
nextChasis = transpose([1,0,0;0,cos(current(1)),-sin(current(1));0,sin(current(1)), ↙
cos(current(1))]*dQb) +current(1:3);




end
```