Describing the Motion of a Carving Snowboarder

Justin Bak

Fall 2024

Description of Chosen System

If you have experienced skiing or snowboarding before, it is likely you are familiar with the term ‘carving’. If you are not familiar with this term, ‘carving’ refers to a curvilinear path caused by the dynamics of snowboarding. When a snowboarder travels in a straight line, motion can be modeled as a pure transformation of potential energy into kinetic energy, with a dissipative friction force added between the snowboard and the snow. However, snowboarders typically prefer a curvilinear trajectory down the mountain to mitigate speed. To achieve this, the snowboarder will typically shift their weight between the lateral ends of the board (i.e. the ’heel’ and ’toe’ of the board). This essentially creates a moment around the roll axis of the board, causing a rotation of the board itself. At a certain angle of this rotation, the snowboard ‘bites’, in which case the snowboarder is bound on a carving radius, r_c.

Figure 1: Carving Snowboarder

In this analysis, we are concerned with modeling and animating the motion of a snowboarder carving down a hill. To achieve this, we will first define Free Body Diagrams, modeling the snowboarder as an inverted pendulum attached to the snowboard at an oscillatory angular velocity. We expect that the snowboarder will follow a curvilinear trajectory down the hill with a constant inclination angle.

Setup of Analytical Models

Before we begin our detailed analysis of the system, we must clarify some assumptions to simplify the motion and create a feasible system.

Assumptions

- The snowboard is assumed to be a rigid body, meaning it does not flex or deform under forces.
- The snowboarder's body is modeled as an inverted pendulum, neglecting detailed limb motion or joint flexibility.
- The snowboarder's center of mass is treated as a point mass located at a fixed position relative to the snowboard (or varying only with ψ, if the body tilts).
- The mass of the snowboarder and snowboard is constant, with no loss of material or energy due to friction, snow spray, etc.
- The slope angle (β) is uniform and well-defined across the region of motion, meaning there are no sudden changes in incline or terrain irregularities.
- Frictional forces depend only on the tilt angle (ϕ) and the normal force, simplifying the interaction between the snowboard edge and snow.
- Lateral slipping of the snowboard edge is negligible during carving (no skidding).
- The centripetal force required for carving is assumed to act perfectly along the j-direction in the k-j plane.

Before developing the free body diagrams and equations of motion, it is crucial to understand the coordinate system from which the motion is defined. As the motion is three-dimensional, a physical body coordinate system can be defined below:

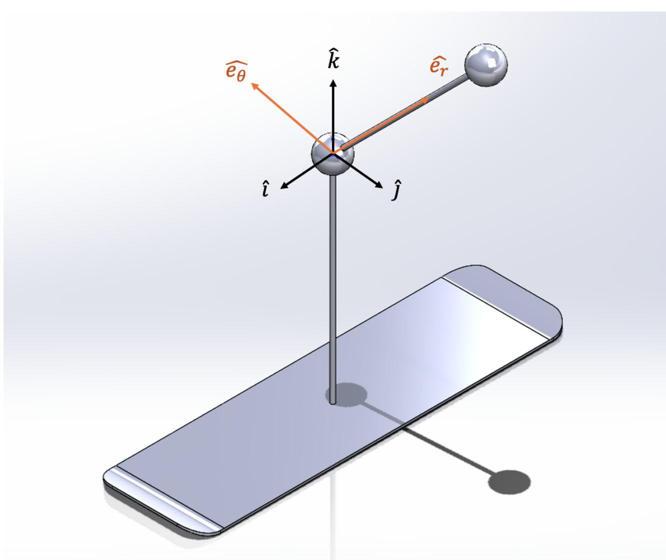

Figure 2: 3-D Body Coordinate System

Now that the physical coordinate system has been defined, we can develop a Free Body Diagram for each of the three-dimensional planes:

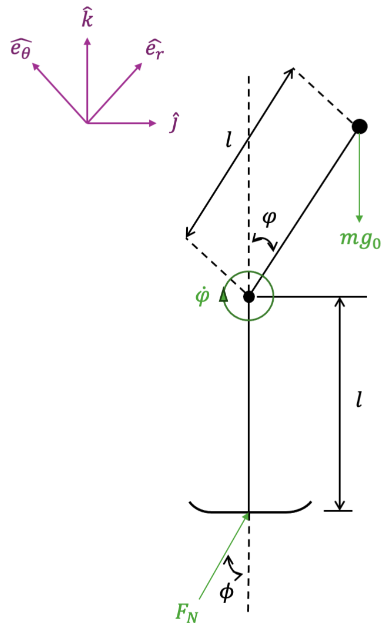

Figure 3: Free Body Diagram in k-j Plane

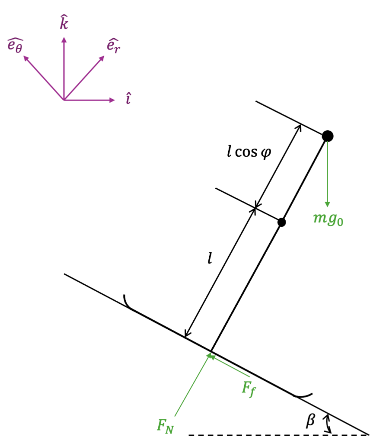

Figure 4: Free Body Diagram in i-k Plane

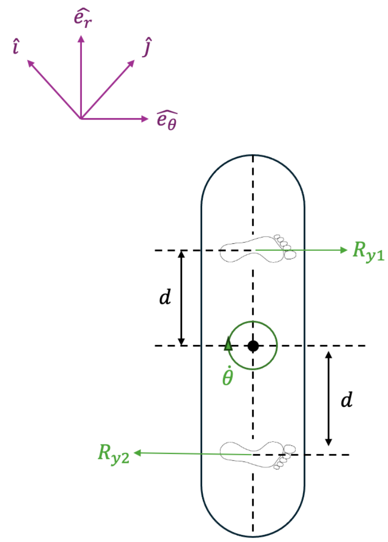

Figure 5: Free Body Diagram in i-j Plane

It is important to denote that $\hat{e}_r$ in Figure 5 is not the previously denoted $\hat{e}_r$ in the system, but rather the direction the snowboard is facing with respect to the $\hat{i}$ and $\hat{j}$ directions. This way, we can still decouple the reaction forces using trigonometry.

Equations of Motion

DAE Method

The Differential Algebraic Equations (DAE) Method aims to govern the motion through constraints in conjunction with Newton-Euler equations of motion to effectively describe the system. In general, the DAE method aims to solve the matrix equation $\mathbf{u} = \mathbf{A}^{-1}\mathbf{F}$, where $\mathbf{u}$ is a column matrix comprised of the parameters that describe the motion. The system of interest contains five accelerations and two reaction forces that fully define the motion:

$$\{\ddot{x}, \ddot{y}, \ddot{z}, \ddot{\psi}, \ddot{\theta}, R_{y_1}, R_{y_2}\}$$

Therefore, we can express the system through five Newton-Euler equations and two constraint equations. The full analysis is conducted below:

Force Balance (ΣF)

$$\Sigma F = m_s\left(\ddot{x}\hat{i} + \ddot{y}\hat{j} + \ddot{z}\hat{k}\right) = -mg\hat{k} + F_n\hat{e}_r - F_f\hat{e}_0 + R_{y_1}\cos\theta\hat{j} + R_{y_1}\sin\theta\hat{i} - R_{y_2}\cos\theta\hat{j} - R_{y_2}\sin\theta\hat{i}$$

Expanding forces in their respective directions:

- In the $\hat{k}$ direction:
$$m_s\ddot{z} = -mg + F_n\cos\phi\cos\beta + F_f\sin\beta \tag{1}$$
- In the $\hat{j}$ direction:
$$m_s\ddot{y} = F_n\sin\phi + R_{y_1}\cos\theta - R_{y_2}\cos\theta \tag{2}$$
- In the $\hat{i}$ direction:
$$m_s\ddot{x} = F_n\sin\beta + F_f\cos\beta + R_{y_1}\sin\theta - R_{y_2}\sin\theta \tag{3}$$

Angular Momentum Balance

For $\ddot{\psi}$:

$$\Sigma M_{O_1O} + \vec{r}_{O_1O}\times m_s\vec{a}_{O_1O} = \dot{\vec{h}}_{O_1O}$$
$$\vec{h}_{O_1O} = \psi_{\max}\sin(\omega_0 t)$$
$$\frac{d^2}{dt^2}\left[h_{O_1O}\right] = -\psi_{\max}\omega_0^2\sin(\omega_0 t)\hat{i}$$

$$\vec{r}_{G_1O}\times\vec{F}_g = \vec{r}_{G_1O}\times m_s\vec{a}_{O_1O} + I_{O_1}\ddot{\psi}\hat{k}$$
$$l\sin(\psi)\hat{k}\times(-mg\hat{j}) = l\sin(\psi)\hat{k}\times m_s\left(-\pi\omega_0^2\sin(\omega_0 t)\hat{j}\right) + \left(I_G + m_s l^2\right)\ddot{\psi}\hat{k}$$

Solving:

$$\ddot{\psi} = \frac{l m_s\left(g_0 - \pi\omega_0^2\sin(\omega_0 t)\right)\sin(\psi)}{I_G + m_s l^2} \tag{4}$$

For $\ddot{\theta}$:

$$\Sigma M_G = \dot{\vec{h}}_G$$

$$d\left(R_{y_1} + R_{y_2}\right) = I_G \ddot{\theta} \tag{5}$$

Constraint Equations

Centripetal Force:

$$\frac{\dot{s}^2}{g\cos\psi\sin\phi} - r_c = 0$$

$$\frac{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}{g\cos\psi\sin\phi} - r_c = 0$$

Taking time derivative:

$$\frac{g\cos\psi\sin\phi \cdot 2(\dot{x}\ddot{x} + \dot{y}\ddot{y} + \dot{z}\ddot{z}) - (\dot{x}^2 + \dot{y}^2 + \dot{z}^2)\cdot(-g\dot{\psi}\sin\psi\sin\phi + g\dot{\phi}\cos\psi\cos\phi)}{(g\cos\psi\sin\phi)^2} - \dot{r}_c = \varnothing \tag{6}$$

Circular Path:

$$\ddot{x}^2 + \ddot{y}^2 - \ddot{z}^2 = r_c\ddot{\theta} \tag{7}$$

Should the DAE Method be implemented?

As seen above, the differential algebraic equations approach requires the equations of motion to be explicitly defined. When translating this method to MATLAB, the equation $\mathbf{u} = \mathbf{A}^{-1}\mathbf{F}$ must be solved at each time step. This equation becomes exceedingly complicated when the matrix coefficients themselves are complicated. One glaring example of this can be seen in equation (6), as all coefficients of translational acceleration require extensive calculation. This can cause stiffness in the MATLAB solution. Furthermore, our system includes oscillatory dynamics that repeatedly shift between positive and negative values. With regard to the DAE Method, this oscillatory behavior is undesirable, as the coupling of algebraic constraints with differential equations is sensitive to small errors. Additionally, the system contains both fast and slow dynamics, as the oscillatory motion of the snowboarder shifting their weight occurs much quicker in comparison to the snowboard progression along the slope. This disparity can also contribute to the stiffness of the solution, which is not ideal for MATLAB's `ode45` solver. Due to these reasons, we prefer to define the system using a Lagrangian approach. By formulating the equations of motion directly in terms of generalized coordinates, Lagrangian mechanics avoids the algebraic constraints altogether. This leads to simpler and more numerically stable systems of ordinary differential equations.

Lagrangian Mechanics

To effectively describe the motion, we use a Lagrangian approach, which can be expressed in general as:

$$\sum_{i=1}^{n}\left[\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right)-\frac{\partial \mathcal{L}}{\partial q_i}\right]=\sum_{i=1}^{n}\left[\sum_{j}Q_{ij}^{H}+\sum_{u}Q_{iu}^{NH}+\sum_{l}Q_{il}^{\text{exc}}-\sum_{m}\frac{\partial R_m}{\partial q_i}\right]$$

In the case of the 'free' snowboarder, where biting of the snowboard is not taken into account, we constrain the system from the induced moment caused by the snowboarder shifting their weight and the dissipative force from the friction between the snowboarder and the snow:

Lagrangian for $45° \leq \psi \leq 135°$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right)-\frac{\partial L}{\partial q_i}=M_{\perp}\frac{\partial \psi}{\partial q_i}+\frac{\partial R}{\partial q_i} \quad \text{with:} \quad c\dot{q}_i. \tag{8}$$

In the case of carving, where biting of the snowboard occurs, we must instill a few further constraints to the motion. Firstly, we must define a Pfaffian constraint to linearize the generalized velocity $\dot{s}$ as a function of our other generalized coordinates. A Pfaffian constraint generally follows the form:

$$\sum_{i=1}^{n}A_i(q_1,q_2,\ldots,t)\dot{q}_i=B(q_1,q_2,\ldots,t)$$

To obtain this constraint, we use the definition of centripetal force and our free body diagram in the k-j plane to relate our generalized coordinate, s, to our other generalized coordinates, ψ and θ. We can then take the time derivative of this constraint to also model the equivalent accelerations.

Constraint for $\psi < 45°$ or $\psi > 135°$

$$F_c=\frac{mv^2}{r_c}=F_n\sin\phi \quad \text{where:} \quad F_n=mg\cos\psi \quad \text{and} \quad v=\dot{s}(\hat{i}+\hat{j})=\cos\theta\dot{s}+\sin\theta\dot{s}. \tag{9}$$

$$r_c=\frac{v^2}{g\cos\psi\sin\phi}=\frac{\dot{s}^2}{g\cos\psi\sin\phi}\Rightarrow \dot{s}^2=r_c g\cos\psi\sin\phi. \tag{10}$$

Constraint Time Derivative

$$\frac{d}{dt}\left(\frac{\dot{s}^2}{g\cos\psi\sin\phi}\right)=\frac{d}{dt}(r_c\dot{\theta}-\dot{s}). \tag{11}$$

$$\Rightarrow \frac{d}{dt}\left(\frac{\dot{s}^2}{g\cos\psi\sin\phi}\right)=\frac{(\cos\psi\sin\phi)2\dot{s}\ddot{s}-\dot{s}^2\big(-g\dot{\psi}\sin\psi\sin\phi+g\dot{\phi}\cos\psi\cos\phi\big)}{(g\cos\psi\sin\phi)^2}-\dot{r_c}=\emptyset. \tag{12}$$

Along with our Non-holonomic constraint, we also relate our angular velocity in the i-j plane to the generalized velocity $\dot{s}$ using a holonomic constraint:

$$\dot{s} = r_c\dot{\theta} \quad \Rightarrow \quad \ddot{s} = r_c\ddot{\theta}. \tag{13}$$

Finally, we add a reactionary force from the snowboarder to ensure the orientation of the snowboard always remains parallel to the velocity. Finally our Lagrangian equation for the bounded snowboarder becomes the following:

Lagrangian for $135^\circ < \psi < 45^\circ$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q^H + Q^{NH} + M_\perp \frac{\partial \psi}{\partial q_i} + F_l \frac{\partial R_y}{\partial q_i} + \frac{\partial R}{\partial q_i}. \tag{14}$$

$$f = \lambda \cdot \frac{\partial f}{\partial q_i} \quad \text{where:} \quad f = r_c\ddot{\theta} - \ddot{s} = \emptyset. \tag{15}$$

$$f = \frac{(\cos\psi\sin\phi)2\dot{s}\ddot{s} - \dot{s}^2\big(- g\dot{\psi}\sin\psi\sin\phi + g\dot{\phi}\cos\psi\cos\phi\big)}{(g\cos\psi\sin\phi)^2} - \dot{r_c} = \emptyset. \tag{16}$$

The final step is to define the Lagrangian itself, which is the difference between kinetic energy and potential energy:

Lagrangian $\mathcal{L}$

The Lagrangian is defined as:

$$\mathcal{L} = E_k - E_p$$

Substituting the expressions for E_k and E_p, we have:

$$\mathcal{L} = \frac{1}{2}m\dot{s}^2 + \frac{1}{2}I\dot{\phi}^2 + \frac{1}{2}I\dot{\psi}^2 + \frac{1}{2}mr_c^2\dot{\theta}^2 - mg\left(l\cos\beta + l\cos\beta\cos\psi\right)$$

Where:

- m: Mass of the snowboarder and snowboard.
- s: Position along the slope.
- ψ: Tilt angle of the snowboarder (inverted pendulum).
- r_c: Radius of curvature for the carving trajectory.
- θ: Angular displacement in the carving plane.
- g: Gravitational acceleration.
- l: Length from the center of rotation to the snowboarder's center of mass.
- β: Inclination angle of the slope.

Animation Still Frames

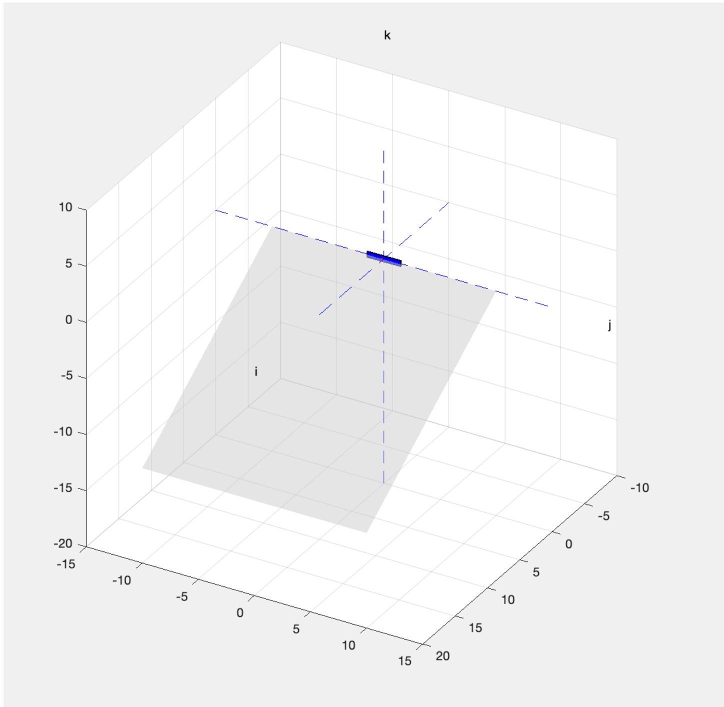

Figure 6: Initial Snowboard Position

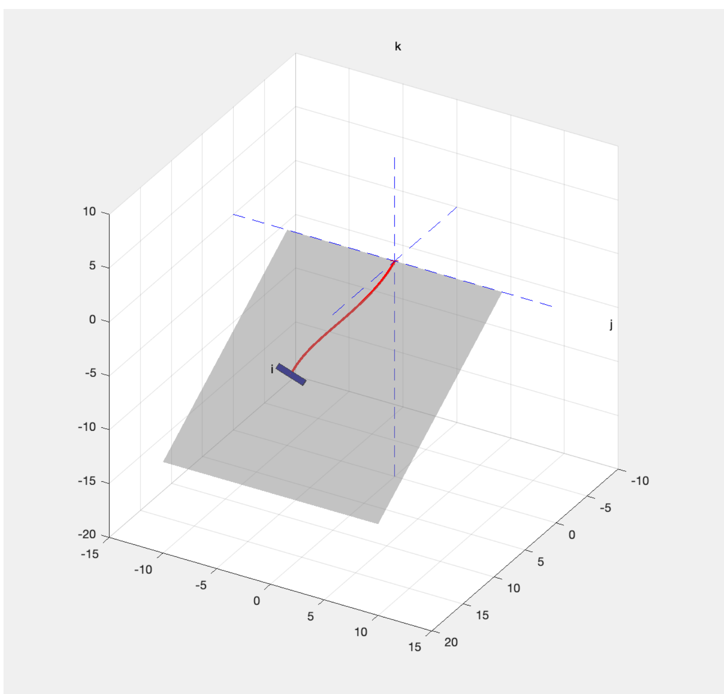

Figure 7: Final Snowboard Position

Discussion

Upon running our MATLAB model using a Lagrangian approach, we find that our solution can explore various behaviors of a snowboarder given different parameters. Altering the characteristics of the system, such as the mass, lever arm, damping coefficient, and inclination display contrasting dynamical behaviors in the animation. Furthermore, we can alter the initial conditions of the system to explore the dynamics of a snowboarder who begins their journey down the mountain with a higher velocity, for example. It should be noted that in Figure 6, the snowboard begins in an orientation perpendicular to the desired motion. This is a sensitivity inherent within the MATLAB code that could not be rectified. Instead, we begin motion by applying an initial rate of rotation in the i-j plane. This behavior is concurrent with physical examples, as usually a snowboarder will begin 'stalled' at the top of the slope, then shift their snowboard to point where they want to go. A further shortcoming of the solution is the use of the Moore-Penrose pseudoinverse, which typically follows the form:

$$\begin{aligned} A^T b &= A^T A \mathbf{x}^* \\ \implies \mathbf{x}^* &= \left(A^T A\right)^{-1} A^T b \end{aligned}$$

The Moore-Penrose pseudoinverse is used to take the inverse of any matrix that is rank-deficient, which arises in our equations of motion as the constraints defined in the system are not uniform. To improve upon this project in the future, one should explore solvers outside of the realm of MATLAB's `ode45`, as the system is hypersensitive to fluctuations in the equations of motion. Ultimately, the project provided an effective comparison between the Differential Algebraic Equations (DAE) approach and the Lagrangian Mechanics approach in the context of three-dimensional constrained motion.

Appendix: MATLAB Code

```

%% MATLAB Code for Snowboarder Carving
% Justin Bak
% 11/05/2024

% Clear previous variables and close figures
close all
clear
clc

syms t m l s s_dot s_doubledot phi phi_dot phi_double_dot psi psi_dot psi_doubledot theta theta_dot theta_doubledot g c beta I r_c eps 'real'


%% Satisfy Lagrangian

% Define kinetic and potential energies
Ek = 0.5 * m * s_dot^2 + 0.5 * I * phi_dot^2 + 0.5 * I * psi_dot^2 + 0.5 * m * r_c^2 * theta_dot^2;
Ep = m * g * (l * cos(beta) + l * cos(beta) * cos(psi));
L = Ek - Ep;

%% Pack Generalized Coordinates

q = [s; phi; psi; theta];
q_dot = [s_dot; phi_dot; psi; theta_dot];
q_doubledot = [s_doubledot; phi_double_dot; psi_doubledot; theta_doubledot];

%% Pack Generalized Forces

% Damping Force
Q_c = [-c * s_dot; 0; 0; 0];

% Non-holonomic Constraint (Centriptetal Force)
Q_nhc = [(1/(g*cos(psi)*sin(phi)+eps)) - r_c; 0; 0; 0];

% Holonomic Constraint (Carving Radius Constraint)
Q_hc = [-1; 0; 0; -r_c];

% Holonomic Constraint (Inclined Plane)
Q_hc2 = [-tan(beta); 0; 0; 0];

% Total Generalized Forces
Q_total = Q_c + Q_nhc + Q_hc + Q_hc2;

%% Compute and Display Jacobian and Equations of Motion

% Define Jacobian function for taking partial derivatives
J = @(f, x) jacobian(f, x);
```

```matlab

% Compute the Jacobian matrices
Jacobian_q_dot = J(J(L, q_dot), q_dot); % Jacobian wrt q_dot
Jacobian_q = J(J(L, q_dot), q);          % Jacobian wrt q
Jacobian_L_q = J(L, q);                  % Jacobian of L wrt q

% Calculate equations of motion (EoM) using the Jacobian approach
EoM = Jacobian_q_dot.' * q_doubledot + ...
      Jacobian_q.' * q_dot - Jacobian_L_q.' == Q_total;

% Display results
disp('Jacobian Matrix wrt q_dot (Jacobian_q_dot):');
disp(Jacobian_q_dot);

disp('Jacobian Matrix wrt q (Jacobian_q):');
disp(Jacobian_q);

disp('Equations of Motion (EoM):');
disp(EoM);

% Analyze rank of the Jacobian matrix
disp('Rank of Jacobian Matrix wrt q_dot:');
disp(rank(Jacobian_q_dot));

disp('Rank of Jacobian Matrix wrt q:');
disp(rank(Jacobian_q));

%% Create System of Linear Equations

% Arrange equations of motion into matrix form
[A, F] = equationsToMatrix(EoM, q_doubledot); % Solve for accelerations
disp('Mass Matrix (A):');
disp(A);

disp('Forcing Vector (F):');
disp(F);

% Analyze rank of the matrices
disp('Rank of Matrix (A):');
disp(rank(A));

disp('Rank of Vector (F):');
disp(rank(F));

%%
fsymbolic = [q_dot; pinv(A)*F];
X = [q; q_dot];
% Check the symbolic variables in fsymbolic
vars_in_fsymbolic = symvar(fsymbolic);

% Display the list of variables
```

```
disp('Symbolic variables in fsymbolic:');
disp(vars_in_fsymbolic);

fdynamic = matlabFunction(fsymbolic, 'Vars', {t, X, g, l, beta, m, c, I, r_c, eps
    });

%% Input Conditions

% Physical Constants
g = 9.81;              % [m/s^2]
l = 1;                 % [m]
beta = pi/6;           % Slope angle (30 degrees)
m = 75;                % [kg]
c = 0.1;               % Damping coefficient [N s/m]
I = m * l^2;           % Moment of inertia for the pendulum
r_c = 1;               % turn radius [m]
eps = 0.0001;          % Non-zero error

% Initial Conditions
s0 = 0.001;                 % Initial position along the slope [m]
s_dot0 = 2;               % Initial velocity along the slope [m/s]
phi0 = 0.01;              % Initial snowboard edge angle [rad]
phi_dot0 = 0.01;          % Initial angular velocity of phi [rad/s]
psi0 = pi/2;              % Initial tilt angle [rad]
psi_dot0 = pi/6;          % Initial angular velocity [rad/s]
theta0 = 0.001;           % Initial carving angle [rad]
theta_dot0 = 0.1;         % Initial angular velocity of carving [rad/s]

% Initial State Vector
ic = [s0; phi0; psi0; theta0; s_dot0; phi_dot0; psi_dot0; theta_dot0];

% Time Span
tspan = [0 10]; % 10 seconds

%% Solve the Equations of Motion

% Numerical ODE Integration

[t, Y] = ode45(@(t, X) fdynamic(t, X, g, l, beta, m, c, I, r_c, eps), tspan, ic);


%% Animate the Motion

% Extract the states
s = Y(:, 1);             % Position along the slope
phi = Y(:, 2);           % Edge angle
psi = Y(:, 3);           % Tilt angle
theta = Y(:, 4);         % Carving angle
s_dot = Y(:, 5);         % Velocity along the slope
phi_dot = Y(:, 6);       % Angular velocity of phi
psi_dot = Y(:, 7);       % Angular velocity of psi
```

```matlab
theta_dot = Y(:, 8);  % Angular velocity of carving

%% Create Empty Figure and View Angle
% Start by drawing a base figure with optimal limits and viewing angle.
h1 = figure;
plot3(0, 0, 0, 'r+');
set(h1, 'WindowStyle', 'docked')
axis equal
xlim([-10, 20])
ylim([-15, 15])
zlim([-20, 10])
xticks(-10:5:30)
yticks(-15:5:15)
zticks(-20:5:10)
view(120, 30)
grid on
hold on

% Draw fixed axes
plot3([-20, 10], [0, 0], [0, 0], 'b--')
text(20, 0, 0, 'i')
plot3([0, 0], [-15, 15], [0, 0], 'b--')
text(0, 20, 0, 'j')
plot3([0, 0], [0, 0], [-20, 10], 'b--')
text(0, 0, 20, 'k')

%% Define the slope
% Slope parameters
slope_length = 40;
slope_width = 20;
slope_angle = 30;

% Create slope as a patch object
[x_slope, y_slope] = meshgrid(linspace(0, slope_length, 2), linspace(-slope_width
    /2, slope_width/2, 2));
z_slope = x_slope * tan(deg2rad(slope_angle)); % Incline along z-axis based on
    x_slope

% Slope transformation
slope_h = hgtransform;
slope_patch = surf(x_slope, y_slope, -z_slope, 'Parent', slope_h, ...
    'FaceColor', [0.8, 0.8, 0.8], 'EdgeColor', 'none', 'FaceAlpha', 0.5); %
        Transparency added with FaceAlpha

% Translate slope to start at the origin
Tx = makehgtform('translate', [0, 0, 0]); % Align slope with the x-axis
Ry = makehgtform('yrotate', 0);           % No rotation needed in y
Rz = makehgtform('zrotate', 0);           % No rotation needed in z

% Combine transformations
slope_h.Matrix = Tx * Ry * Rz;
```

```matlab

%% Define the snowboard as a rectangular prism
% Snowboard parameters
snowboard_length = 3;
snowboard_width = 0.5;
snowboard_height = 0.1;

% Define vertices of the prism
vertices = [
    -snowboard_width/2, -snowboard_length/2, -snowboard_height/2;
    snowboard_width/2, -snowboard_length/2, -snowboard_height/2;
    snowboard_width/2, snowboard_length/2, -snowboard_height/2;
    -snowboard_width/2, snowboard_length/2, -snowboard_height/2;
    -snowboard_width/2, -snowboard_length/2, snowboard_height/2;
    snowboard_width/2, -snowboard_length/2, snowboard_height/2;
    snowboard_width/2, snowboard_length/2, snowboard_height/2;
    -snowboard_width/2, snowboard_length/2, snowboard_height/2;
];

% Define faces of the prism
faces = [
    1, 2, 6, 5; % Bottom
    2, 3, 7, 6; % Right
    3, 4, 8, 7; % Top
    4, 1, 5, 8; % Left
    1, 2, 3, 4; % Front
    5, 6, 7, 8; % Back
];

% Create snowboard as a patch object
snowboard_h = hgtransform;
snowboard_patch = patch('Vertices', vertices, 'Faces', faces, ...
    'FaceColor', [0, 0, 1], 'EdgeColor', 'k', 'Parent', snowboard_h);

%% Initialize the Carving Path
% Preallocate arrays to store the trajectory
x_path = [];
y_path = [];
z_path = [];

% Create a carving path line
carve_path = plot3(0, 0, 0, 'r-', 'LineWidth', 2);

%% Animation Loop
for k = 1:length(t)
    % Compute snowboarder's position
    x_pos = s(k) * cos(theta(k));  % Lateral position
    y_pos = -s(k) * sin(theta(k));  % Downhill position
    z_pos = -s(k) * tan(deg2rad(30)); % Height above slope (from slope angle)

    % Ensure finite values for position
```

```
    if ~isfinite(x_pos) || ~isfinite(y_pos) || ~isfinite(z_pos)
        warning('Invalid position at frame %d: x = %f, y = %f, z = %f', k, x_pos, y_pos, z_pos);
        continue; % Skip this frame
    end

    % Compute orientation using rotation matrices
    Rz_theta = makehgtform('zrotate', theta(k)); % Carving angle
    Ry_psi = makehgtform('yrotate', psi(k));     % Body tilt
    Rx_phi = makehgtform('xrotate', phi(k));     % Edge tilt

    % Combine transformations for rigid body motion
    snowboard_h.Matrix = makehgtform('translate', [x_pos, y_pos, z_pos]) * Rx_phi * Ry_psi * Rz_theta;

    % Update carving path points
    x_path = [x_path, x_pos];
    y_path = [y_path, y_pos];
    z_path = [z_path, z_pos];

    % Update the carving path line
    set(carve_path, 'XData', x_path, 'YData', y_path, 'ZData', z_path);

    % Pause for visualization
    pause(0.01); % Adjust for smoother animation
end

% Add a light source for better visualization
light('Position', [0, 10, 10], 'Style', 'infinite');
```