Design and Analysis of Dynamical Systems
Final Project Report
Alex Klein, Max Muss, Daniel Leongomez
May 2021

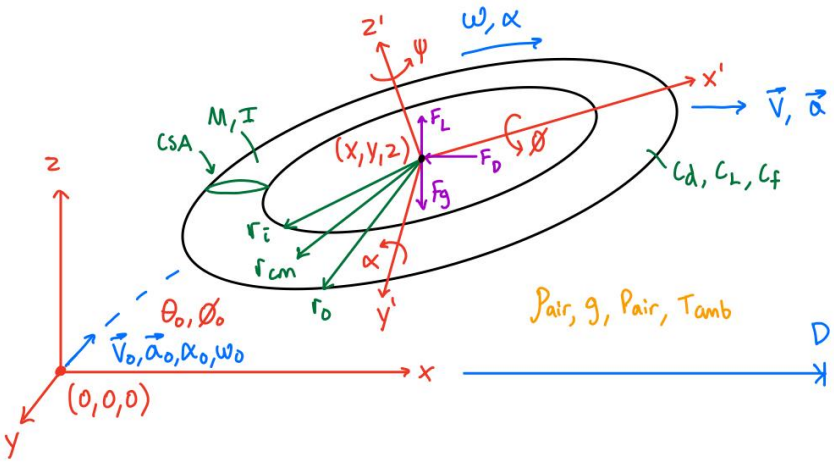
Executive Summary

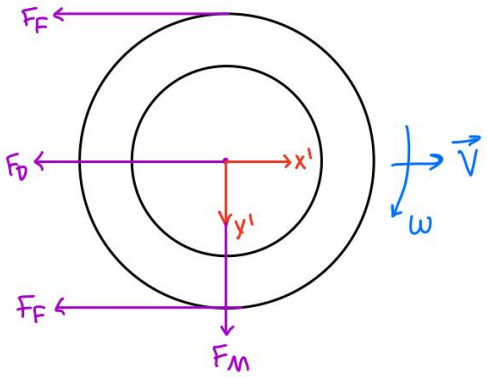
We focused on the steadiness of a flying ring in order to maximize the distance it will travel. After researching the governing aerodynamic properties and relationships, we created equations of motions for the flight. Our MATLAB model incorporates the initial and iterated angle of attack of the disk into the distance calculation. We optimized for distance by varying radius and mass of the disk, averaging many throws across initial angle-of-attack angular position and velocity. We found that the radius of the disk does not have a significant effect on the flight and distance is maximized when the disk is 0.5kg.

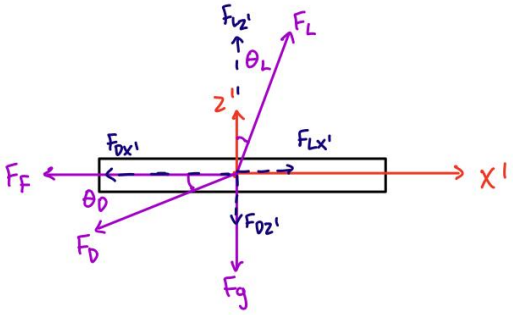
Table of Contents

Forces, Equations of Motion, and FBD

Coordinate system defined as follows: A global x, y, and z coordinate system details the location of the disk center of mass in respect to the origin, while an x', y' and z' centered at the center of mass describes the frisbee orientation relative to this center of mass. There are essentially two different equations of motion that must interact: one which determines the flight path based on forces acting over the entire disc (which depend on the disc orientation) and one which determines the disc orientation based on moments acting on the disc due to unbalanced forces (which are dependent on the rest of the current flight path).







Assume:

→ no body roll ∅

→ F_D/F_L account for rear airfoil too

→ no wind, uniform air

→ Magnus effect will have to be averaged for airfoil shape

$$\Sigma F_{x'} = F_L \sin\theta_L - F_D \cos\theta_D - F_F$$

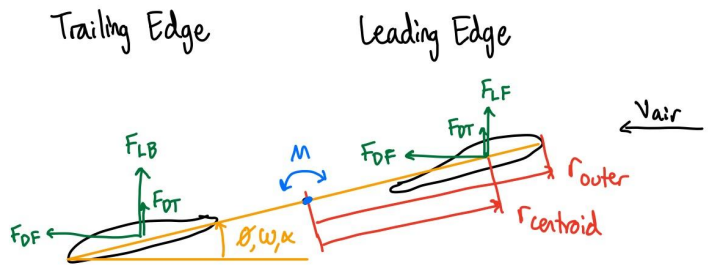
$$\Sigma F_{y'} = F_M = L_{cyl} \cdot \rho_{air} \|\vec{V}\| (2\pi r_o^2 \omega)$$

$$\Sigma F_{z'} = F_L \cos\theta_L - F_D \sin\theta_D - F_g$$

$$C_D = C_{D0} + C_{D\alpha} (\alpha - \alpha_0)^2$$

$$C_L = C_{L0} + C_{L\alpha}\alpha$$

- Gravity is a constant force acting in the negative Z direction, and does not depend on velocity or angle of attack
- Lift acts from the center of pressure of the disc (not the center of mass, instead at a point closer to the leading edge of the disc) and is always oriented perpendicular to the disc's velocity vector. However, the location of this center of pressure is not constant throughout the frisbee's flight path, as it varies with velocity and angle of attack. In order to simplify the calculation, an entirely equivalent system is created in which the lift does act from the center of mass, but is also accompanied by a moment acting around the y' axis (pitching moment) that changes throughout the flight. The lift force is described by the following equation: $L = 0.5\rho V^2 A * C_L$, where 'V' is the magnitude of the velocity vector, A is the planform area of the disc, and C_L is the coefficient of lift. This coefficient varies with angle of attack α, and is described as follows: $C_L = C_{L_0} + C_{L_\alpha} * \alpha$ (Morrison, V. R), where C_{L_0} is the coefficient of lift at zero angle of attack and C_{L_α} is a constant over the range of α describing the slope of C_L as α is varied.
- Similarly, drag also acts from the center of pressure and has been converted to act from the center of mass with an accompanying pitching moment. In contrast to lift, the drag force acts parallel to the disc's velocity vector. The drag force is given by: $D = 0.5\rho V^2 A * C_D$, in which the coefficient of drag varies with the angle of attack as follows: $C_D = C_{D_0} + C_{D_\alpha} * (\alpha - \alpha_0)^2$(Morrison, V. R). The constant C_{D_0} is the minimum drag coefficient possible, at the angle of attack that generates zero lift (α_0). C_{D_α} is a constant that describes how $C_D$varies with α.
- The pitching moment is calculated by separating the disc into a leading and trailing edge with their own lift and drag relationships given by the above equations, with coefficients for the leading and trailing airfoil shapes given by computational analysis in Ansys. These coefficients are used to calculate forces acting at the leading and trailing airfoils that can be summed to find the total pitching moment about the center of mass. This pitching moment directly affects the current angle of attack of theentire disc, thus affecting the flight path.
- Gyroscopic forces: the spinning disc has an angular momentum given by $H = I\omega$, where I is the disc's moment of inertia and $\omega$is the angular velocity. This gyroscopic resistance acts against the pitching moment forces. Additionally, the effect of skin friction drag on the spinning disc causes a moment acting against this spin (in the z' direction) which results in gyroscopic precession (a roll about the x' axis), although this effect was not quantified in our analysis.



$$F_{Dx} = \frac{1}{2} \rho_{air} V_{disk} A_{disk} \left(C_{Dx0} + C_{Dx} \phi^2 \right)$$

$$F_{Lx} = \frac{1}{2} \rho_{air} V_{disk} A_{disk} \left(C_{Lx0} + C_{Lx} \phi \right)$$

$$M = r_{centroid} \cos(\phi) \left(F_{LF} - F_{LB} \right) + r_{centroid} \sin(\phi) \left(F_{DF} - F_{DB} \right)$$

$$M_{Dampen} = 2 r_{centroid} \sin(\phi) F_{DT}$$

Matlab Model

Run the file "Disk_Flight(m_disk, r_moment, phi_initial, omega_initial)" with the given inputs: mass of the disk, radius of the centroid of the airfoil,and initial angular displacement and velocity at throw. This simulation assumes several constant initial conditions, while allowing for the user to input/change the initial angle-of-attack angular displacement and velocity. Through research, we set the thrown velocity to v(1) = [20,0,4] m/s, position to x(1) = [0,0,1.5] m, and angular velocity to omegaR = 50 rad/s. The forces on the body are summed, divided by mass/moment of inertia, and iterated through timesteps to determine the new velocities and positions.

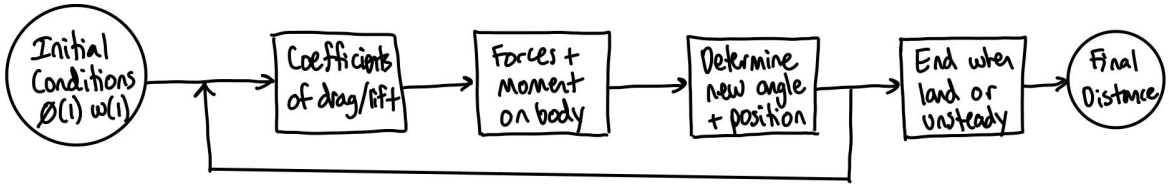
Simulation Setup:

1. Define the physical properties of the disk and environment
2. Define aerodynamic properties: drag / lift
3. Set the initial overall velocity, acceleration, and rotational velocity

Simulation (looped with small time steps until disk lands or becomes unsteady)

4. Angle of attack calculations (shown below)
5. Position of disk calculations (shown below)
6. Check if landed and output final distance

Block Diagram:



Angle of attack calculations

```
% find magnitude of disk velocity
v_disk = norm(v(i,:));

% calculate lift/drag on front/back
Fdf = rho_air*v_disk^2*A_disk*0.5*(Cdfo + Cdfx*phi(i)^2);
Fdb = rho_air*v_disk^2*A_disk*0.5*(Cdbo + Cdbx*phi(i)^2);
Flf = rho_air*v_disk^2*A_disk*0.5*(Clfo + Clfx*phi(i));
Flb = rho_air*v_disk^2*A_disk*0.5*(Clbo + Clbx*phi(i));

% forward lift/drag moment
M = r_moment*cos(phi(i))*(Flf-Flb) + r_moment*sin(phi(i))*(Fdf-Fdb);

% drag dampening moment
FDf = rho_air*(omega(i)*r_moment)^2*10*A_disk*0.5*(Cdfb);
damp = 2*r_moment*sin(phi(i))*FDf;

% determine kinematics
alpha(i+1) = (M+damp)/I_disk;
omega(i+1) = (alpha(i)*deltaT+omega(i));
phi(i+1) = omega(i)*deltaT+phi(i);
```

Position of disk calculations

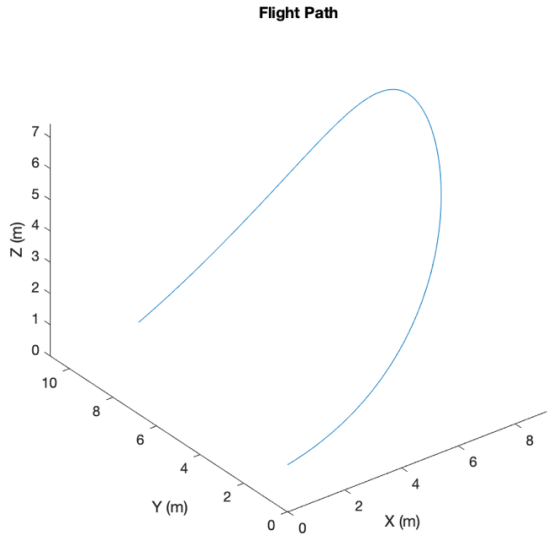
```
% calculate forces
F_l = Flf + Flb; % forward lift
F_d = Fdf + Fdb; % forward drag
F_dy = rho_air*v(i,2)^2*A_disk*0.5*(Cdfo); % drag in y
F_m = L_m*rho_air*v(i,1)*2*pi*r_outer^2*omegaR; % magnus
F_g = m_disk * g; % gravity

% sum forces in x/y/z
F_xp = F_l*sin(theta_L) - F_d*cos(theta_D);
F_yp = F_m - F_dy;
F_zp = F_l*cos(theta_L) - F_d*sin(theta_D) - F_g;
F(i,:) = [F_xp,F_yp,F_zp]; % concatenate

% determine new kinematics
a(i,:) = F(i,:)/m_disk;
v(i+1,:) = a(i,:)*deltaT+v(i,:);
x(i+1,:) = v(i,:)*deltaT+x(i,:);
```

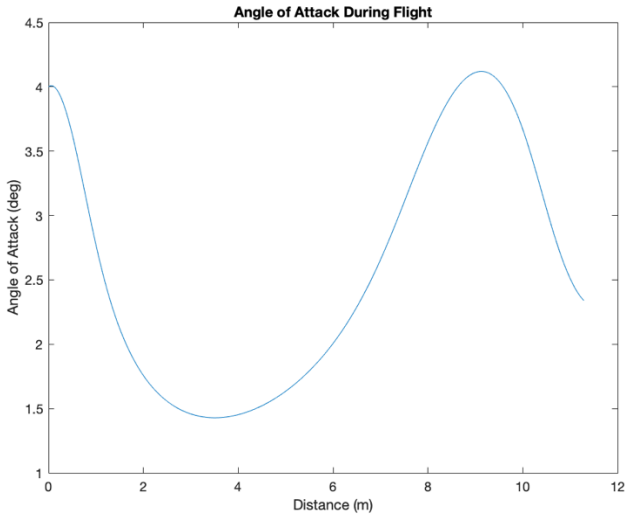
The simulation can be optimized by running the function "Disk_Optimization", which loops through variations of mass and radius combinations, finding the average distance for each disk dimension across many varied initial angles of attack and angle-of-attack angular velocity. A three-dimensional graph is plotted with distance on the Z axis, affected by radius and mass of the disk on the X and Y axis.

Simulation Results

All throws are from the starting location (x=0m, y=0m, z=1.5m) at hand height.

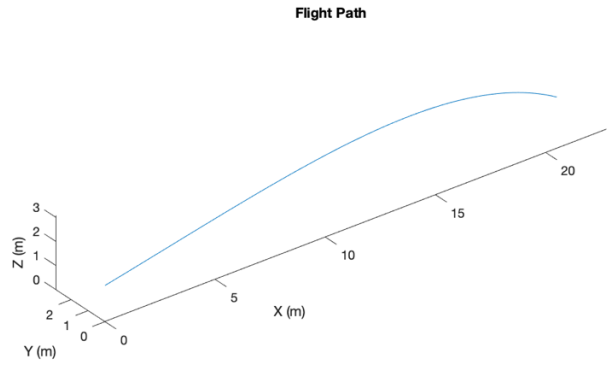
Example 1:
r_moment = 0.18; %m
m_disk = 0.1; %kg
phi(1) = 4; %deg
omega(1) = 1; %deg/s

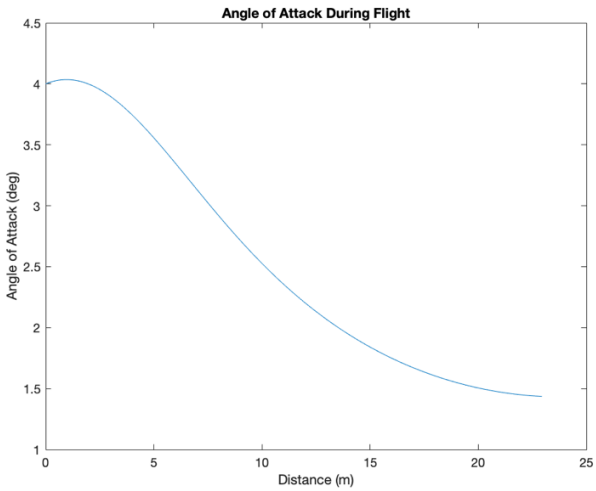




Distance traveled: 11.28m

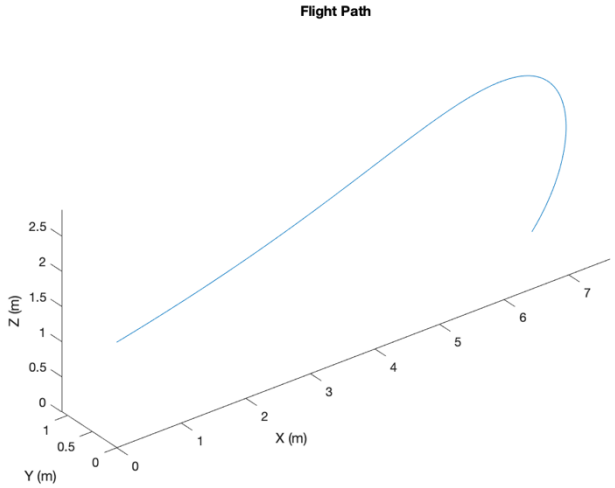
Example 2: (increasing mass)
r_moment = 0.18; %m
m_disk = 0.5; %kg
phi(1) = 4; %deg
omega(1) = 1; %deg/s

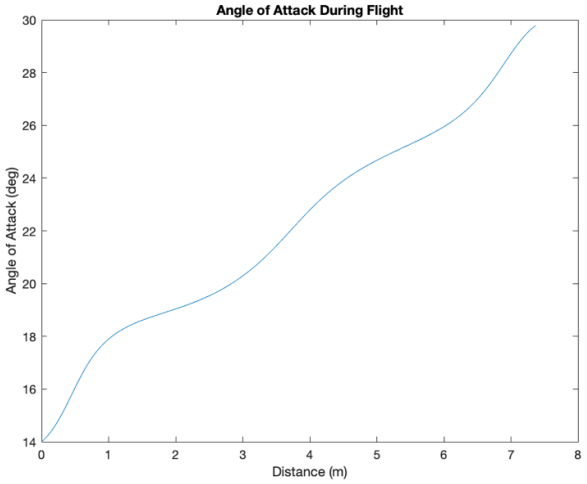




Distance traveled: 22.9m

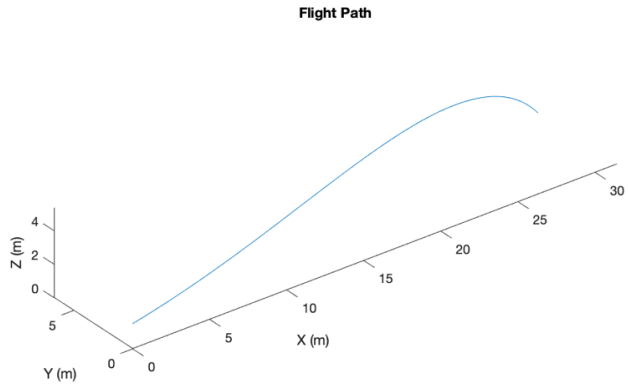
Example 3: (unsteady behavior)
r_moment = 0.18; %m
m_disk = 0.5; %kg
phi(1) = 14; %deg
omega(1) = 12; %deg/s

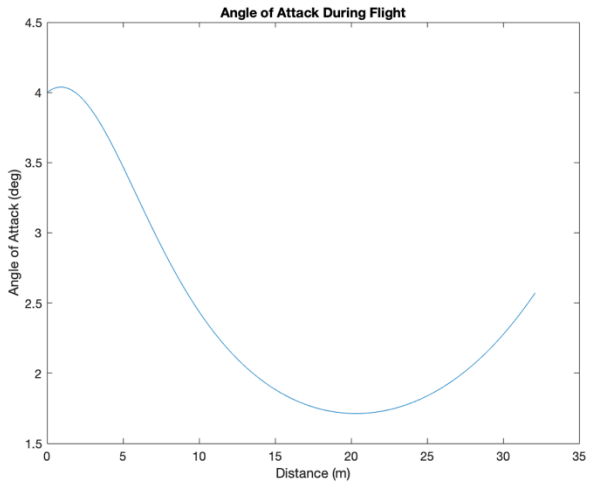




Distance traveled: 22.9m

Example 4: (lift increased by 2 times)
r_moment = 0.18; %m
m_disk = 0.5; %kg
phi(1) = 4; %deg
omega(1) = 1; %deg/s





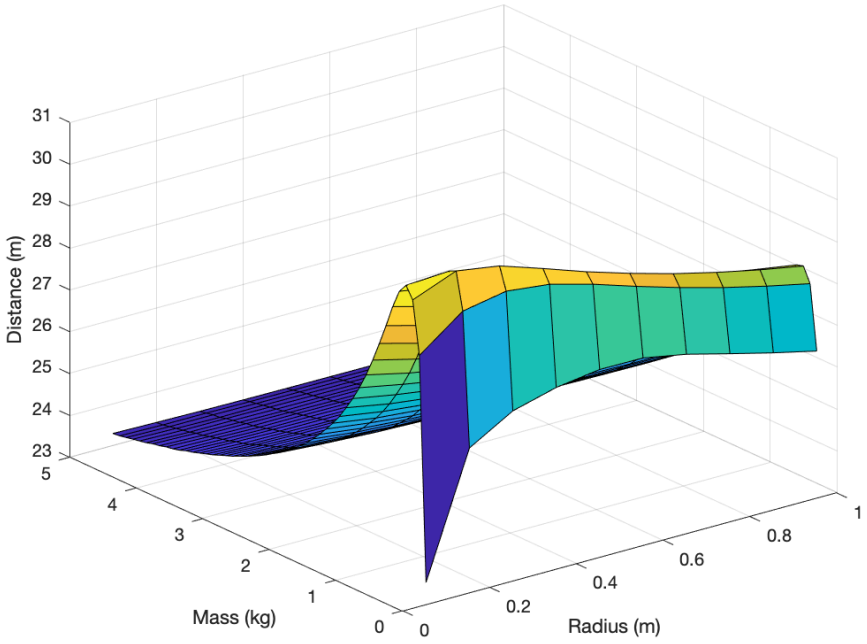
Distance traveled: 32.1m

Optimization Results

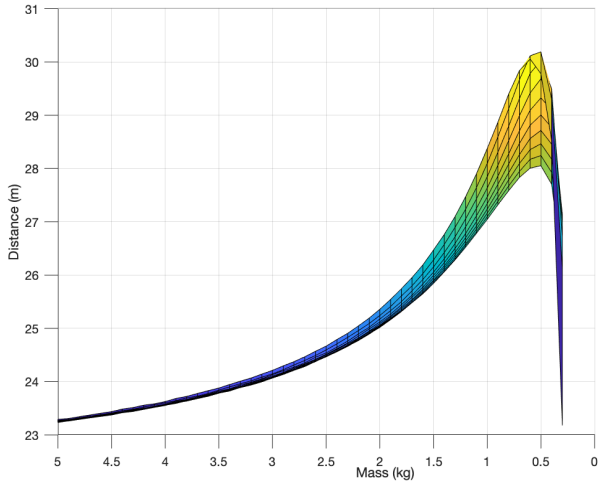
With the simulation describing the flight path and yielding how far the disk flies, we are able to better optimize the controllable properties of the disk to maximize distance traveled, specifically focusing on mass and radius, which can easily be physically altered. Other properties are either directly/loosely based on mass and radius (such as moment of inertia and other dimensions) or are not easily "selected" by a designer (such as drag and lift coefficients).

First, we plot distance traveled as a function of mass and radius, as shown for the initial conditions phi(1) = 2 deg and omega(1) = 5 deg/s:

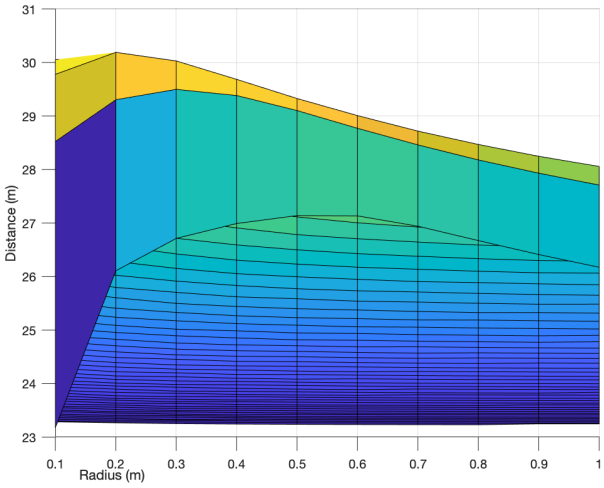
3D View:



Mass View



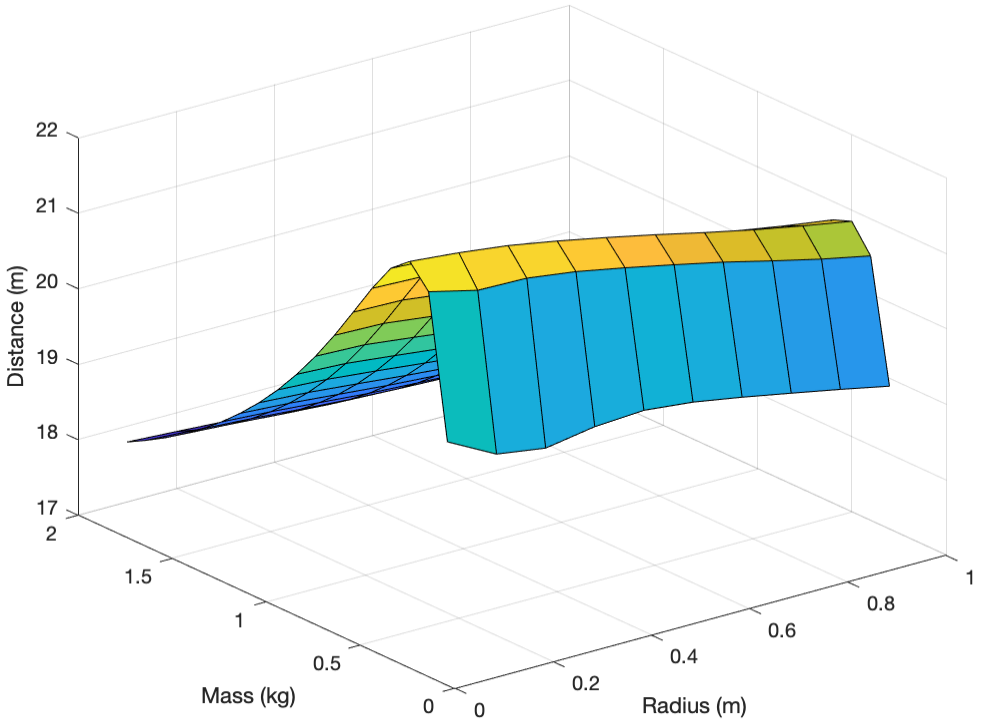
Radius View



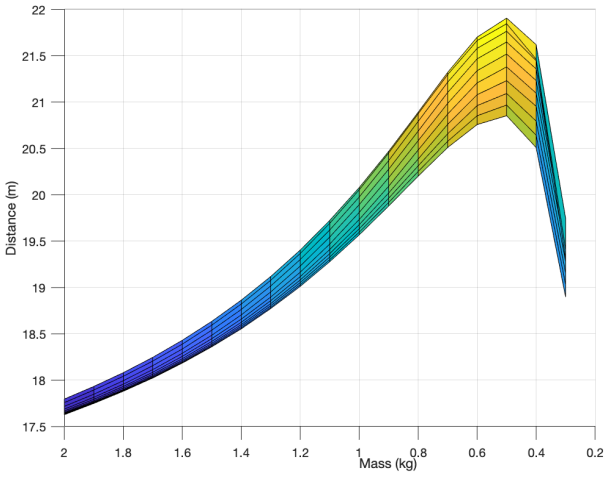
The furthest distance the disk can be thrown (given the initial velocities, position, and aerodynamic properties) is 30.18m when the mass is 0.5kg and radius is 0.2m.

However, this surface looks different based on the initial conditions, so we instead loop through the optimization code for many different combinations of initial angle-of-attack angular displacement and velocity (1 deg to 5 deg and 1 deg/s to 5 deg/s). Therefore at each mass and radius combination, there are 100 sample throws taken. The optimization code loops through our simulation 4,700 times. The averaged distances for each mass and radius combination is shown:

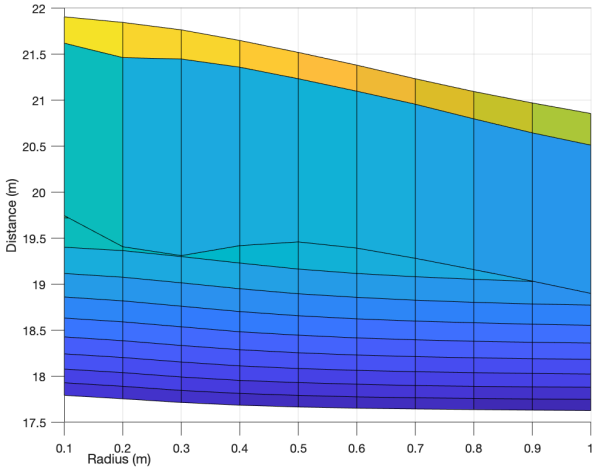
3D View:



Mass View



Radius View



Discussion

As seen in the surface plot above, radius is slightly inversely proportional to distance. While it seems an infinitely small radius would improve distance traveled, the relationship is almost negligible with an order of magnitude change in radius only increasing distance by 4.7%. Mass, on the other hand, has a much more apparent effect on distance traveled and an obvious optimized peak. As clear in the mass view of the final optimized results, distance is maximized in all radius variations when the disk weighs 0.5kgs. Therefore, we conclude that the most optimized throwing disk, at some of our constant initial conditions, should be designed with materials to allow for a 0.5kg object.

Reflection

Contributions of Each Team Member:

- Alex: Instituting forces into matlab to result in flight path, coding optimization calculation, being a supportive team member
- Daniel: Determining governing forces and relationships, getting coefficients of airfoil from Ansys, being a supportive team member
- Max: Research into similar analysis, writing up and submitting intermediate reports, equations of motion, being a supportive team member

Challenges, Changes of Scope, and Lessons Learned:

We set out with the goal of analyzing a flying ring and being able to optimize parameters for flight distance. The complexity of the problem was not immediately apparent until deeper analysis into the actual governing forces and principles. Many of the quantities such as lift and drag turned out to be very difficult to solve for completely analytically, and relationships based on coefficients of airfoil geometry had to be used for these force relationships. These constants could not be solved for by hand and Ansys Fluent needed to be used to determine them. Throughout the project, we were unsure of exactly which parameters we would be able to change. It would have been very difficult to optimize airfoil geometry without having to run numerous Ansys simulations, so we changed the focus to variables more easy to alter (mass and radius) without changing the entire analysis.

Recommendations for the Future:

We think a project based on optimizing an uncontrolled system in flight is a very interesting and engaging project idea, although choosing a flying disc may have over-complicated the problem. A more traditional airfoil, such as a simplified paper airplane, may have made analysis more straightforward.

References

1.) Morrison, V. R. *The Physics of Frisbees*, The Physics of Frisbees.

Appendix: Matlab Code (Attached as PDF)

```
function Disk_Optimization ()

D = zeros(17,10,5,5);

% vary initial angular velocity from 1 deg/s to 10 deg/s
for omega0 = 1:5
    % vary initial angle from 1 deg to 10 deg
    for angle0 = 1:5
        % vary mass from 0.3kg to 2kg
        for mass = 3:20
            % vary radius from 0.05m to 0.5m
            for radius = 1:10
                D(mass-2,radius,angle0+1,omega0+1) = Disk_Flight(mass/10, radius/40,↙
angle0, omega0);
            end
        end
    end
end

Davg = sum(D,4)/size(D,4);
Davg = sum(Davg,3)/size(D,3);

xVal = linspace(0.3,2,size(Davg,1));
yVal = linspace(0.1,1,size(Davg,2));

surf(yVal,xVal,Davg);
ylabel('Mass (kg)')
xlabel('Radius (m)')
zlabel('Distance (m)')
set(gcf,'color','w')

end
```

```
function D = Disk_Flight (m_disk, r_moment, phi_initial, omega_initial)

%%% PHYSICAL PROPERTIES %%%

% define environment
rho_air = 1.15; %kg/m^3
g = 9.81; %kg-m/s^2

% define disk
% r_moment = 0.18; %m %commented out for optimization script
% m_disk = 0.5; %kg %commented out for optimization script
I_disk = m_disk * r_moment^2; %kgm^2
A_disk = 0.05; %m^2
r_outer = 0.2; %m
L_m = 0.005; %m

% set up matricies
F = zeros(1,3);
a = zeros(1,3);
v = zeros(1,3);
x = zeros(1,3);

%%% AERODYNAMIC PROPERTIES %%%

% overall aero properties
dragScalingFactor = 1;
liftScalingFactor = 2;

% front side aero properties
Clfo = 0.12 * liftScalingFactor;
Clfx = 0.001 * liftScalingFactor;
Cdfo = 0.2 * dragScalingFactor;
Cdfx = 0.01 * dragScalingFactor;

% back side aero properties
Clbo = 0.1 * liftScalingFactor;
Clbx = 0.01 * liftScalingFactor;
Cdbo = 0.18 * dragScalingFactor;
Cdbx = 0.001 * dragScalingFactor;

% top/bottom aero properties
Cdfb = 0.6;

%%% INITIAL CONDITIONS %%%

% set initial parameters
v(1,:) = [20,0,4]; %m/s
x(1,:) = [0,0,1.5]; %m
omegaR = 50; %rad/s

% set initial angular kinematics
phi(1) = phi_initial; %deg
omega(1) = omega_initial; %deg/s
% phi(1) = 4; %deg %commented out for optimization script
% omega(1) = 1; %deg/s %commented out for optimization script
alpha(1) = 0; %deg/s^2

%%% RUN SIMULATION %%%

% simulation optins
deltaT = 0.001; %s
theta_L = 0; %deg
theta_D = 0; %deg
```

```
flying = 1;
land = 1;
fail = 1;
i = 1;

while flying == 1

    %%% ANGLE OF ATTACK %%%

    % find magnitude of disk velocity
    v_disk = norm(v(i,:));

    % calculate lift/drag on front/back
    Fdf = rho_air*v_disk^2*A_disk*0.5*(Cdfo + Cdfx*phi(i)^2);
    Fdb = rho_air*v_disk^2*A_disk*0.5*(Cdbo + Cdbx*phi(i)^2);
    Flf = rho_air*v_disk^2*A_disk*0.5*(Clfo + Clfx*phi(i));
    Flb = rho_air*v_disk^2*A_disk*0.5*(Clbo + Clbx*phi(i));

    % forward lift/drag moment
    M = r_moment*cos(phi(i))*(Flf-Flb) + r_moment*sin(phi(i))*(Fdf-Fdb);

    % drag dampening moment
    FDf = rho_air*(omega(i)*r_moment)^2*10*A_disk*0.5*(Cdfb);
    damp = 2*r_moment*sin(phi(i))*FDf;

    % determine kinematics
    alpha(i+1) = (M+damp)/I_disk;
    omega(i+1) = (alpha(i)*deltaT+omega(i));
    phi(i+1) = omega(i)*deltaT+phi(i);

    % check if past stall angle / unsteady
    if phi(i) <= -25
        fail = 0;
        failure = "stall";
    elseif phi(i) >= 25
        fail = 0;
        failure = "stall";
    end

    %%% POSITION %%%

    % calculate forces
    F_l = Flf + Flb; % forward lift
    F_d = Fdf + Fdb; % forward drag
    F_dy = rho_air*v(i,2)^2*A_disk*0.5*(Cdfo); % drag in y
    F_m = L_m*rho_air*v(i,1)*2*pi*r_outer^2*omegaR; % magnus
    F_g = m_disk * g; % gravity

    % sum forces in x/y/z
    F_xp = F_l*sin(theta_L) - F_d*cos(theta_D);
    F_yp = F_m - F_dy;
    F_zp = F_l*cos(theta_L) - F_d*sin(theta_D) - F_g;
    F(i,:) = [F_xp,F_yp,F_zp]; % concatenate

    % determine new kinematics
    a(i,:) = F(i,:)/m_disk;
    v(i+1,:) = a(i,:)*deltaT+v(i,:);
    x(i+1,:) = v(i,:)*deltaT+x(i,:);

    % check if landed
    if x(i+1,3) <= 0
        land = 0;
        failure = "land";
```

```
    end

    %%% CONTINUE SIMULATION (?) %%%

    % step time
    i = i+1;

    % notify if landed or stalled
    if land == 0 || fail == 0
        flying = 0;
    end

end

%%% RESULTS %%%

% output results
D = ( (sum(v(:,1))*deltaT)^2 + (sum(v(:,2))*deltaT)^2 )^(1/2);
% finalAngle = phi(end)
% failure

% % plot flight path
% figure(1)
% plot3(x(:,1),x(:,2),x(:,3))
% axis equal;
% title('Flight Path')
% xlabel('X (m)')
% ylabel('Y (m)')
% zlabel('Z (m)')
% set(gcf,'color','w')
%
% % plot angle of attack
% figure (2)
% xval=linspace(0,D,size(phi,2));
% plot(xval,phi)
% title('Angle of Attack During Flight')
% xlabel('Distance (m)')
% ylabel('Angle of Attack (deg)')
% set(gcf,'color','w')

end
```