

Clear Workspace

```
clear;  
close all;
```

Import dependencies

```
addpath(genpath('Functions'));  
addpath(genpath('Classes'));
```

Create the world

```
% Origin  
O = [0 0 0];  
T_O = TranslationMatrix(O);  
  
% Floor  
T_O = T_O*TranslationMatrix(O);
```

Populate the objects

```
% Create sample objects  
% Space_Object(name, pose, mass, shape, inertia, partNumber)  
  
% Chassis  
chassis_width = 1400;  
chassis_depth = 1640;  
chassis_heigh = 50;  
chassis_mass = 10;  
  
% Main Wheels  
wheel_radius = 250;  
wheel_amplitude = 230;  
wheel_mass = 4;  
  
chassis_pose = TranslationMatrix([0 (chassis_depth*1/3) wheel_radius]);  
chassis = Space_Object('Chassis', chassis_pose, ...  
    chassis_mass, 'triangle', [chassis_width chassis_depth  
chassis_heigh], ...  
    [0.1 0.1 0.9], 0.6 , ... %bluish  
    RingedTriangularPrism(chassis_mass,chassis_width, chassis_depth,  
chassis_heigh, 50), 1);
```

```

right_wheel_pose = TranslationMatrix([-chassis_width/2)+(-
wheel_amplitude/2) 0 (wheel_radius)])*RotationMatrix4(90,'y','deg');
right_wheel = Space_Object('Right Wheel', right_wheel_pose, ...
    wheel_mass, 'cylinder', [wheel_radius wheel_amplitude 32], ...
    [0.1 0.1 0.1], 0.8, ... %blackish
    EmptyCylinderInertia(wheel_mass, wheel_radius, wheel_amplitude),
2);

left_wheel_pose = TranslationMatrix([(chassis_width/2)+(wheel_amplitude/2) 0
(wheel_radius)])*RotationMatrix4(90,'y','deg');
left_wheel = Space_Object('Left Wheel', left_wheel_pose, ...
    wheel_mass, 'cylinder', [wheel_radius wheel_amplitude 32], ...
    [0.1 0.1 0.1], 0.8, ... %blackish
    EmptyCylinderInertia(wheel_mass, wheel_radius, wheel_amplitude),
3);

% Caster Wheels
caster_wheel_radius = 125;
caster_wheel_amplitude = 115;
caster_wheel_mass = 1;

center_wheel_pose = TranslationMatrix([0 (chassis_depth)
(caster_wheel_radius)])*RotationMatrix4(90,'y','deg');
center_wheel = Space_Object('Center Wheel', center_wheel_pose, ...
    caster_wheel_mass, 'cylinder', [caster_wheel_radius
caster_wheel_amplitude 32], ...
    [0.1 0.1 0.1], 0.8, ... %blackish

EmptyCylinderInertia(caster_wheel_mass, caster_wheel_radius, caster_wheel_ampli
tude), 4);

% Astronauts body
astronaut_body_mass = 243;
astronaut_body_heigh = 400;
astronaut_body_width = 1000;
astronaut_body_depth = 1000;
astronaut_body_com_heigh = astronaut_body_heigh * 0.5;

astronaut_body_pose = TranslationMatrix([0 astronaut_body_depth/2-400
(astronaut_body_heigh/2+wheel_radius)]);
astronaut_body = Space_Object('Injured Astronaut', astronaut_body_pose,
...
    astronaut_body_mass, 'box', [astronaut_body_width/2
astronaut_body_width/2 astronaut_body_depth/2 astronaut_body_depth/2
astronaut_body_heigh-astronaut_body_com_heigh astronaut_body_com_heigh], ...
    [0.1 0.9 0.1], 0.8, ... %greenish
    CuboidInertia(astronaut_body_mass, astronaut_body_heigh,
astronaut_body_width, astronaut_body_depth), 6);

% Astronaut legs

```

```

astronaut_legs_mass = 100;
astronaut_legs_heigh = 250;
astronaut_legs_width = 500;
astronaut_legs_depth = 1000;
astronaut_legs_com_heigh = astronaut_legs_heigh * 0.5;

astronaut_legs_pose = TranslationMatrix([0
astronaut_body_pose(2,4)+astronaut_body_depth/2+astronaut_legs_depth/2
(astronaut_legs_heigh/2+wheel_radius)]);
astronaut_legs = Space_Object('Healthy Astronaut', astronaut_legs_pose,
...
    astronaut_legs_mass, 'box', [astronaut_legs_width/2
astronaut_legs_width/2 astronaut_legs_depth/2 astronaut_legs_depth/2
astronaut_legs_heigh-astronaut_legs_com_heigh astronaut_legs_com_heigh], ...
    [0.1 0.9 0.1], 0.8, ... %greenish
    CuboidInertia(astronaut_legs_mass, astronaut_legs_heigh,
astronaut_legs_width, astronaut_legs_depth), 5);

% Store objects in array
objects_array = [astronaut_body, astronaut_legs, right_wheel, left_wheel,
center_wheel, chasis];

figure;
DrawObjects(objects_array)

% Access properties example
disp(['First object name: ', objects_array(1).name])

```

First object name: Injured Astronaut

```
disp(['Second object mass: ', num2str(objects_array(2).mass)])
```

Second object mass: 100

Add Ground plane

```

ground_inclination = 20; % degrees

% Find maximum distance from origin
max_dist = 0;
max_idx = 1;

for i = 1:length(objects_array)
    dist = sqrt(objects_array(i).pose(1,4)^2 + objects_array(i).pose(2,4)^2);
    if dist > max_dist
        max_dist = dist;
        max_idx = i;
    end
end

```

```

end

% Calculate orientation angle to farthest object
theta = atan2(objects_array(max_idx).pose(2,4),
objects_array(max_idx).pose(1,4));

% Add wheel_radius + 100 in the same direction
additional_dist = wheel_radius + 100;
max_dist = max_dist + additional_dist;

% Calculate final point coordinates
max_x = max_dist * cos(theta);
max_y = max_dist * sin(theta);

% Plot the point
hold on
plot3(max_x, max_y, 0, 'r*', 'MarkerSize', 10)
text(max_x, max_y, 0, 'Max Point', 'FontSize', 10)

% Create structure array for all points and their data

plane_size = 3000;

plane_angle_disc = 5;

angle_disc_size = length(0:plane_angle_disc:360)

```

```

angle_disc_size =
73

```

```

% Create structure array for all points and their data
circle_data = struct( ...
    'angle', cell(1,angle_disc_size), ...
    'point', cell(1,angle_disc_size), ...
    'radial_vector', cell(1,angle_disc_size), ...
    'tangential_vector', cell(1,angle_disc_size), ...
    'plane_points', cell(1,angle_disc_size), ...
    'transformed_objects', cell(1,angle_disc_size), ...
    'wheel_contact_points', cell(1,angle_disc_size), ...
    'normal_direction', cell(1,angle_disc_size), ...
    'reactions', cell(1,angle_disc_size)...
);

```

```

idx = 0

```

```

idx =
0

```

```

for angle = 0:plane_angle_disc:360
    % disp(['Angle: ', num2str(angle)]);

```

```

idx = idx +1;

circle_data(idx).angle = angle;

theta = deg2rad(angle);

% Store point coordinates
point = [max_dist * cos(theta); max_dist * sin(theta); 0];
plot3(point(1), point(2), point(3), 'r.', 'MarkerSize', 5)
circle_data(idx).point = point;

% Store vectors
radial = [cos(theta); sin(theta); 0];
tangential = [-sin(theta); cos(theta); 0];
circle_data(idx).radial_vector = radial;
circle_data(idx).tangential_vector = tangential;

% Store plane points
T_point =
TranslationMatrix(circle_data(idx).point)*RotationMatrix4(angle, 'z', 'deg');
T_v1 = T_point*TranslationMatrix([0, +plane_size/2, 0]);
v1 = T_v1(1:3,4);

T_v2 = T_point*TranslationMatrix([0, -plane_size/2, 0]);
v2 = T_v2(1:3,4);

T_v3 = T_v2*RotationMatrix4(ground_inclination, 'y',
'deg')*TranslationMatrix([-plane_size,0,0]);
v3 = T_v3(1:3,4);

T_v4 = T_v1*RotationMatrix4(ground_inclination, 'y',
'deg')*TranslationMatrix([-plane_size,0,0]);
v4 = T_v4(1:3,4);

circle_data(idx).plane_points = [v1 v2 v3 v4];

transformed_objects = objects_array;

% Transform each object's pose
for i = 1:length(transformed_objects)
    % Create transformations
    T_point = TranslationMatrix(circle_data(idx).point);
    R_z = RotationMatrix4(angle, 'z', 'deg');
    R_y = RotationMatrix4(ground_inclination, 'y', 'deg');
    R_z_inv = RotationMatrix4(-angle, 'z', 'deg');
    T_point_inv = TranslationMatrix(-circle_data(idx).point);
    % Apply transformations in correct order
    transformed_objects(i).pose =
T_point*R_z*R_y*R_z_inv*T_point_inv*transformed_objects(i).pose;
end

```

```

circle_data(idx).transformed_objects = transformed_objects;

% Normal direction to the ground
plane_vec1 = v2 - v1;
plane_vec2 = v3 - v1;
ground_normal = cross(plane_vec2, plane_vec1);
circle_data(idx).normal_direction = ground_normal / norm(ground_normal);

% Wheel contact points
caster_wheel_pose = ...

TranslationMatrix(transformed_objects(contains({transformed_objects.name},
'Center Wheel'))).pose(1:3,4)')*...
    TranslationMatrix(-
circle_data(idx).normal_direction*caster_wheel_radius);

right_wheel_pose = ...

TranslationMatrix(transformed_objects(contains({transformed_objects.name},
'Right Wheel'))).pose(1:3,4)')*...
    TranslationMatrix(-circle_data(idx).normal_direction*wheel_radius);

left_wheel_pose = ...

TranslationMatrix(transformed_objects(contains({transformed_objects.name},
'Left Wheel'))).pose(1:3,4)')*...
    TranslationMatrix(-circle_data(idx).normal_direction*wheel_radius);

circle_data(idx).wheel_contact_points.caster_wheel =
caster_wheel_pose(1:3,4);
circle_data(idx).wheel_contact_points.right_wheel =
right_wheel_pose(1:3,4);
circle_data(idx).wheel_contact_points.left_wheel =
left_wheel_pose(1:3,4);

end

```



```

%         circle_data(idx).radial_vector(1)*scale,
circle_data(idx).radial_vector(2)*scale,
circle_data(idx).radial_vector(3)*scale, 0, 'r-', 'LineWidth', 2);
%     quiver3(circle_data(idx).point(1), circle_data(idx).point(2),
circle_data(idx).point(3), ...
%         circle_data(idx).tangential_vector(1)*scale,
circle_data(idx).tangential_vector(2)*scale,
circle_data(idx).tangential_vector(3)*scale, 0, 'b-', 'LineWidth', 2);
%
%     % Plot plane with more visible properties
%     patch('Vertices', circle_data(idx).plane_points', 'Faces', [1 2 3
4], ...
%         'FaceColor', [0.8 0.8 0.8], 'FaceAlpha', 0.2, 'EdgeColor', 'k',
'LineWidth', 1);
%
%     DrawObjects(circle_data(idx).transformed_objects);
% end

```

```

%     'wheel_contact_points', cell(1,angle_disc_size), ...
%     'normal_direction', cell(1,angle_disc_size) ...

```

Physics calculations

```

% Physics calculations for each configuration
g = 1.62; % gravity constant
mu = 0.5; % friction coefficient

debugging_struct.r_2 = struct( ...
    'r_2', [], ...
    'reactions', []);

for idx = 1:length(circle_data)
    % Get the current configuration
    transformed_objects = circle_data(idx).transformed_objects;

    sum_torque_1 = [ 0 ; 0 ; 0 ];
    sum_torque_2 = [ 0 ; 0 ; 0 ];
    sum_torque_3 = [ 0 ; 0 ; 0 ];

    % line around which we will compute the torque
    line_1 = {}; % caster_wheel to right_wheel
    line_1.point = circle_data(idx).wheel_contact_points.caster_wheel;
    line_1.dir = circle_data(idx).wheel_contact_points.right_wheel -
circle_data(idx).wheel_contact_points.caster_wheel;
    line_1.dir = line_1.dir/norm(line_1.dir);

    line_2 = {}; % right_wheel to left_wheel

```



```

    line_2.point = circle_data(idx).wheel_contact_points.right_wheel;
    line_2.dir = circle_data(idx).wheel_contact_points.left_wheel -
circle_data(idx).wheel_contact_points.right_wheel;
    line_2.dir = line_2.dir/norm(line_2.dir);

    line_3 = {}; % left_wheel to caster wheel
    line_3.point = circle_data(idx).wheel_contact_points.left_wheel;
    line_3.dir = circle_data(idx).wheel_contact_points.caster_wheel -
circle_data(idx).wheel_contact_points.left_wheel;
    line_3.dir = line_3.dir/norm(line_3.dir);

%      % =====
%
%      figure;
%      hold on
%      grid on
%      axis equal
%      xlabel('X');
%      ylabel('Y');
%      zlabel('Z');
%      title('Car with Direction Lines');
%
%      % Draw the car objects
%      DrawObjects(circle_data(idx).transformed_objects)
%
%      % Use existing figure from DrawObjects
%      figure(gcf)
%      hold on
%
%      % Scale factor adjusted to match scene size
%      scale = 500;
%
%      % Plot the three lines with better visibility
%      h1 = quiver3(line_1.point(1), line_1.point(2), line_1.point(3), ...
%                  line_1.dir(1)*scale, line_1.dir(2)*scale,
line_1.dir(3)*scale, ...
%                  0, 'r-', 'LineWidth', 3);
%      h2 = quiver3(line_2.point(1), line_2.point(2), line_2.point(3), ...
%                  line_2.dir(1)*scale, line_2.dir(2)*scale,
line_2.dir(3)*scale, ...
%                  0, 'g-', 'LineWidth', 3);
%      h3 = quiver3(line_3.point(1), line_3.point(2), line_3.point(3), ...
%                  line_3.dir(1)*scale, line_3.dir(2)*scale,
line_3.dir(3)*scale, ...
%                  0, 'b-', 'LineWidth', 3);
%
%      legend([h1 h2 h3], {'Line 1', 'Line 2', 'Line 3'})
%
%      % =====

```

```

for i = 1:length(transformed_objects)
    % line 1
    perp_1 = cross(line_1.dir,transformed_objects(i).mass*g*[0;0;-1]);
    r_1 = dot(perp_1,(line_1.point-
transformed_objects(i).pose(1:3,4))) / norm(perp_1) * perp_1/norm(perp_1);
    torque_1 = cross(r_1,transformed_objects(i).mass*g*[0;0;-1]);
    sum_torque_1 = sum_torque_1 + torque_1;

    % line 2
    perp_2 = cross(line_2.dir,transformed_objects(i).mass*g*[0;0;-1]);
    r_2 = dot(perp_2,(line_2.point-
transformed_objects(i).pose(1:3,4))) / norm(perp_2) * perp_2/norm(perp_2);
    torque_2 = cross(r_2,transformed_objects(i).mass*g*[0;0;-1]);
    sum_torque_2 = sum_torque_2 + torque_2;

    % line 3
    perp_3 = cross(line_3.dir,transformed_objects(i).mass*g*[0;0;-1]);
    r_3 = dot(perp_3,(line_3.point-
transformed_objects(i).pose(1:3,4))) / norm(perp_3) * perp_3/norm(perp_3);
    torque_3 = cross(r_3,transformed_objects(i).mass*g*[0;0;-1]);
    sum_torque_3 = sum_torque_3 + torque_3;

end

torque_1_proj = dot(sum_torque_1,line_1.dir)/
(norm(line_1.dir)^2)*line_1.dir;
torque_2_proj = dot(sum_torque_2,line_2.dir)/
(norm(line_2.dir)^2)*line_2.dir;
torque_3_proj = dot(sum_torque_3,line_3.dir)/
(norm(line_3.dir)^2)*line_3.dir;

%  $R = r \times T / \|r\|^2$ 

% left wheel
perp_1 = cross(line_1.dir,circle_data(idx).normal_direction);
r_1 = dot(perp_1,(line_1.point-
circle_data(idx).wheel_contact_points.left_wheel)) / norm(perp_1) * perp_1/
norm(perp_1);
circle_data(idx).reactions.left_wheel = cross(r_1,torque_1_proj)/
norm(r_1)^2;
circle_data(idx).reactions.left_wheel;

% caster wheel
perp_2 = cross(line_2.dir,circle_data(idx).normal_direction);

```

```

    r_2 = dot(perp_2,(line_2.point-
circle_data(idx).wheel_contact_points.caster_wheel)) / norm(perp_2) * perp_2/
norm(perp_2);
    circle_data(idx).reactions.caster_wheel = cross(r_2,torque_2_proj)/
norm(r_2)^2;
    circle_data(idx).reactions.caster_wheel;
%
%     debugging_struct(end+1).r_2 = circle_data(idx).reactions.left_wheel;
%     debugging_struct(end).reactions =
circle_data(idx).reactions.caster_wheel;

% right wheel
perp_3 = cross(line_3.dir,circle_data(idx).normal_direction);
r_3 = dot(perp_3,(line_3.point-
circle_data(idx).wheel_contact_points.right_wheel)) / norm(perp_3) * perp_3/
norm(perp_3);
    circle_data(idx).reactions.right_wheel = cross(r_3,torque_3_proj)/
norm(r_3)^2;
    circle_data(idx).reactions.right_wheel;
end

debugging_struct(1) = [];

```

Deb Plot

```

% % Initialize arrays for angles and reactions
% angles = zeros(1, length(circle_data));
% radius = zeros(1, length(circle_data));
% torque = zeros(1, length(circle_data));
%
% % Extract data for plotting
% for idx = 1:length(debugging_struct)
%     radius(idx) = norm(debugging_struct(idx).r_2); % Magnitude of reaction
force
%     torque(idx) = norm(debugging_struct(idx).reactions);
% end
%
% % Create figure with two y-axes
% figure;
% yyaxis left
% plot(radius, '-o', 'DisplayName', 'radius')
% ylabel('Radius')
%
% yyaxis right
% plot(torque, '-x', 'DisplayName', 'torque')
% ylabel('Torque')
%
% % Add common labels and formatting

```

```

% xlabel('Angle (degrees)')
% title('Debugging Graph')
% legend('show')
% grid on

```

Plot Graph

```

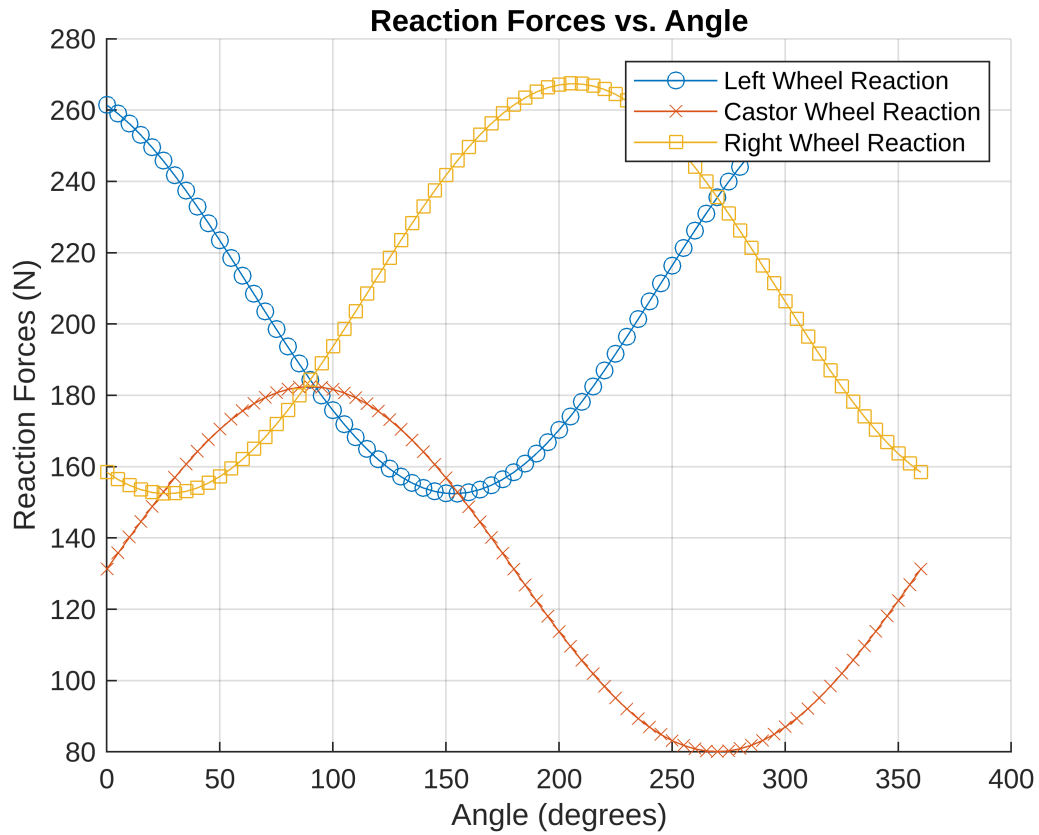
% Initialize arrays for angles and reactions
angles = zeros(1, length(circle_data));
left_reactions = zeros(1, length(circle_data));
caster_reactions = zeros(1, length(circle_data));
right_reactions = zeros(1, length(circle_data));

% Extract data for plotting
for idx = 1:length(circle_data)
    angles(idx) = circle_data(idx).angle;
    left_reactions(idx) =
dot(circle_data(idx).reactions.left_wheel, circle_data(idx).normal_direction);
    % Magnitude of reaction force
    caster_reactions(idx) =
dot(circle_data(idx).reactions.caster_wheel, circle_data(idx).normal_direction
);
    right_reactions(idx) =
dot(circle_data(idx).reactions.right_wheel, circle_data(idx).normal_direction
);
end

% Plot data
figure;
hold on;
plot(angles, left_reactions, '-o', 'DisplayName', 'Left Wheel Reaction');
plot(angles, caster_reactions, '-x', 'DisplayName', 'Castor Wheel Reaction');
plot(angles, right_reactions, '-s', 'DisplayName', 'Right Wheel Reaction');
hold off;

% Add labels and legend
xlabel('Angle (degrees)');
ylabel('Reaction Forces (N)');
title('Reaction Forces vs. Angle');
legend('show');
grid on;

```



Ground Contact points

```
% contact_points = getWheelContactPoints(objects_array, wheel_radius,
wheel_amplitude);
%
% % Plot the contact points
% figure;
% scatter3(contact_points(:,1), contact_points(:,2), contact_points(:,3),
'filled');
% grid on;
% xlabel('X');
% ylabel('Y');
% zlabel('Z');
% title('Wheel Contact Points');
%
```

Clear Workspace

```
clear;  
close all;
```

Import dependencies

```
addpath(genpath('Functions'));  
addpath(genpath('Classes'));
```

Constants and variables

```
% Constants  
TotalMass = 366; %kg  
MoonGravity = 1.625; %m/s  
SlopeAngle = 20; % degrees  
WheelLinearVelocity = 1.389; %m/s (If we assume a maximum speed of 5km/h → 1.389  
m/s)  
CoefficientRolling = 0.05; % Lunar rolling coefficient Josep found it on a paper  
TotalDistance = 2; %km  
Speed = 5; % km/h  
  
% Variables  
WheelR = 0.25; %m  
WheelMass = 2; %kg  
WheelRadiusExternal = WheelR; %m  
WheelRadiusInternal = 0.39; %m  
  
WheelCasterMass = 1.0; % kg  
WheelCasterRadius = 0.12; % m  
WheelCasterRadiusInternal = 0.185; %m  
  
Acceleration = WheelLinearVelocity/10; % m/s^2 we assume 10s can be changed.  
WheelAngularVelocity = WheelLinearVelocity/WheelRadiusExternal
```

```
WheelAngularVelocity = 5.5560
```

```
%WheelForceNormal = 85,98 % for now unused
```

Motors power

```
InercyMovingWheel = 1/2*WheelMass*(WheelRadiusInternal^2 + WheelR^2); % kg*m^2  
InercyMovingCasterWheel = 1/2*WheelCasterMass*(WheelCasterRadiusInternal^2 +  
WheelCasterRadius^2); % kg*m^2  
  
AngularAccelerationWheel = Acceleration * WheelR; % rad/s^2
```

```

AngularAccelerationWheelCaster = Acceleration * WheelCasterRadius; % rad/s^2
InertiaTorque= (InercyMovingWheel * AngularAccelerationWheel) * 2 +
InercyMovingCasterWheel* AngularAccelerationWheelCaster; % N*m
Force_Gravity_tan= TotalMass * MoonGravity * sind(SlopeAngle); %N

ForceRolling = TotalMass * MoonGravity * cosd(SlopeAngle) * CoefficientRolling; %N

TotalTorqueMotors = TotalMass* Acceleration *WheelR + InertiaTorque +
Force_Gravity_tan * WheelR + ForceRolling * WheelR; % N*m
TorqueMotor = TotalTorqueMotors/2 % N*m

```

```
TorqueMotor = 35.2824
```

```
PowerMotor = TorqueMotor*(WheelLinearVelocity/WheelR) %Watts
```

```
PowerMotor = 196.0290
```

```
TotalPowerMotors = PowerMotor*2; % Watts
```

Motor Specs Check

```

RealMotorPower = 400;
RealMotorRPM = 3500;
RealMotorAngularVelocity = RealMotorRPM *2*pi/60

```

```
RealMotorAngularVelocity = 366.5191
```

```

RealMotorEficiency = 0.94;
RealMotorPower = RealMotorPower / RealMotorEficiency;
TorqueConstant = 176; %mN.m/A

RealMotorTorque=RealMotorPower/RealMotorAngularVelocity

```

```
RealMotorTorque = 1.1610
```

```
TheoreticalReductor = TorqueMotor/RealMotorTorque
```

```
TheoreticalReductor = 30.3894
```

```

RealReductor = 62/1;
OutputAngularVelocity = RealMotorAngularVelocity / RealReductor

```

```
OutputAngularVelocity = 5.9116
```

```
OutputTorque = RealMotorPower / OutputAngularVelocity
```

```
OutputTorque = 71.9825
```

$$\text{OutputPower} = \text{OutputTorque} * \text{OutputAngularVelocity}$$

$$\text{OutputPower} = 425.5319$$

Batteries

$$\text{PowerConsumption} = \text{RealMotorPower} * 2 + 25$$

$$\text{PowerConsumption} = 876.0638$$

$$\text{Time} = \text{TotalDistance} / \text{Speed}; \%H$$

$$\text{Voltage} = 48; \%V$$

$$\text{EnergyRequired} = \text{PowerConsumption} * \text{Time} \%Watts * h$$

$$\text{EnergyRequired} = 350.4255$$

$$\text{EnergyRequiredAssumingTwentyPercentageLoss} = \text{EnergyRequired} / 0.8 \%Wh$$

$$\text{EnergyRequiredAssumingTwentyPercentageLoss} = 438.0319$$

%If we choose a battery that works at 48 V, the capacity of the battery Ah = Wh/V → 7.795 Ah

%Hence, according to our calculations, choosing batteries of 48V-8Ah is enough for our tricycle.

$$\text{Ah} = (\text{EnergyRequiredAssumingTwentyPercentageLoss} / \text{Voltage})$$

$$\text{Ah} = 9.1257$$