Gauntlet Challenge

Berwin Lan and Zoie Leo QEA 2 Spring 2021, Olin College of Engineering

The purpose of this challenge was to guide a virtual Neato robot through the Gauntlet to the Barrel of Benevolence (BoB) with gradient descent methods using MATLAB, ROS, and Gazebo. The Gauntlet landscape was generated with the use of sources and sinks, with the BoB, obstacles, and walls represented as points. This landscape was represented with the use of a potential field. Then, a gradient descent algorithm was implemented to drive the Neato across the potential field to the BoB represented by a circle sink. Our Neato successfully navigated the Gauntlet using these methods.

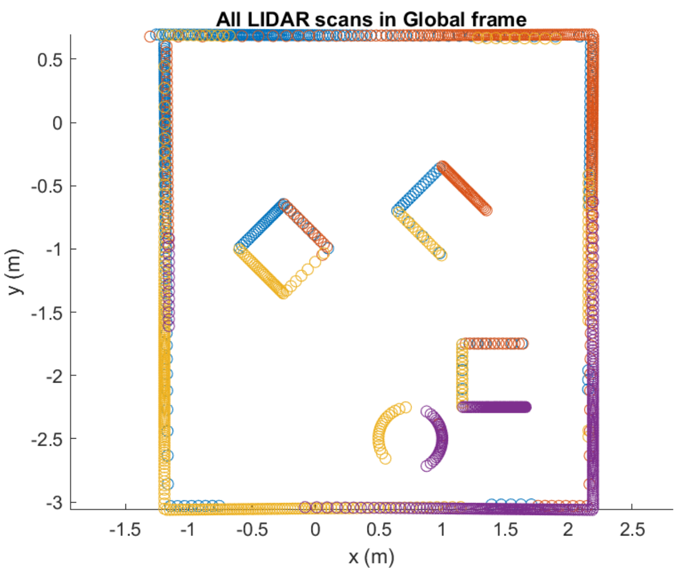

Figure 1: A map of the Gauntlet using LIDAR scan data.

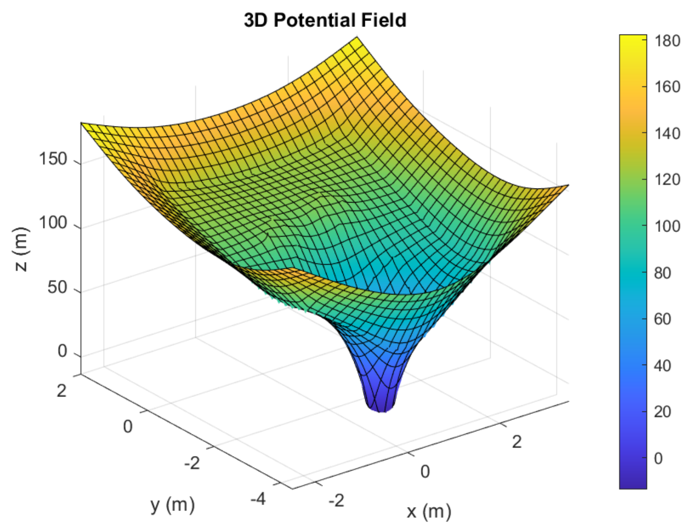

Figure 2: A 3-D plot of the potential field described using the equation in Appendix C.

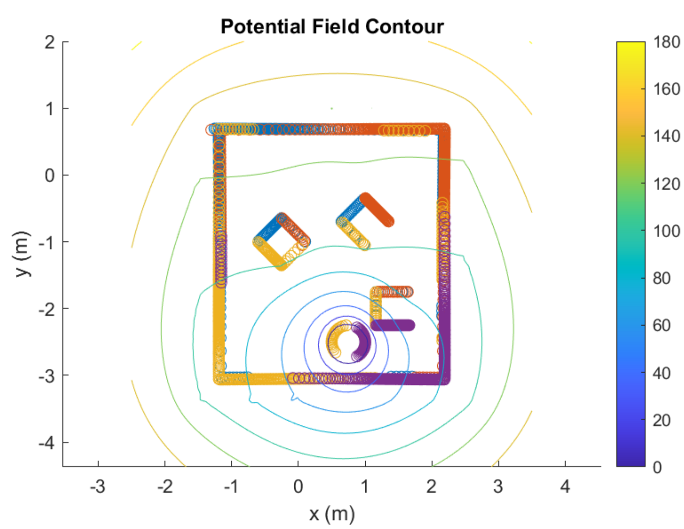

Figure 3: A contour plot of the potential field described using the equation in Appendix C.

1 Methods

First, a map of the Gauntlet was created by reconstructing data from LIDAR scans taken by the Neato, with the final points shown in Figure 1.

Figures 2 and 3 represent the potential field of the Gauntlet, with the BoB as a sink and obstacles and walls as sources. The equation used was developed iteratively: starting with a potential field of 0, each feature (BoB, obstacles, walls) was described as a series of points using either the source or sink equations. By adding all the features cumulatively to a flat xy-plane, the full equation accounted for all the features of the

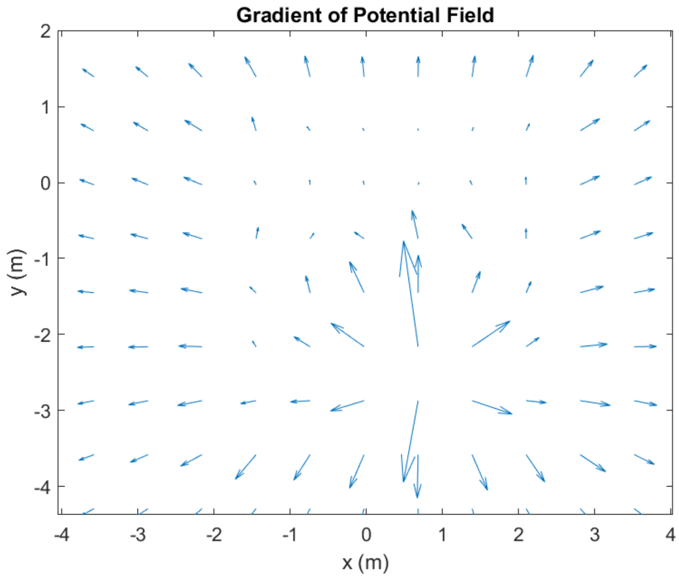

Figure 4: A quiver plot of the gradient of the potential field.

gauntlet. Furthermore, this method allowed certain features to be amplified or damped by a scalar factor to ensure that the Neato would successfully use gradient descent to reach the BoB. Please see Appendix C for the full equation used; the MATLAB code used in this portion of the exercise is included in Appendix B.

The quiver plot shown in Figure 4 shows the gradient of the potential field described by the equation in Appendix C. The lengths of the vectors are proportional to the rate of change of the gradient, and the arrows point in the direction of most rapid increase. In order to implement gradient ascent, the Neato needs to drive in the direction of most rapid decrease, or directly opposite to the vectors in Figure 4.

Figure 5 shows the planned path of gradient descent, calculated by using the modified gradient ascent algorithm included in Appendix B.

2 Results

The Neato successfully traversed the Gauntlet and reached the BoB, as seen in Figure 5. The robot took **36.8 seconds** to get to the BoB, driving at 0.1 m/s and rotating at 0.2 rad/s. This time was calculated by finding the difference between the first and final times collected by the encoder. The robot traveled **2.4674 m**, as calculated by averaging the total distance traveled by each of the two wheels.

A video of the Neato traversing the Gauntlet is available here. (Long link: `https://youtu.be/lfqMyJkfFQg`)

3 Discussion

Figure 5 shows the actual path of gradient descent that the Neato took plotted against the theoretical path that was calculated. The actual path was reconstructed by processing the data outputs from the wheel encoder into location coordinates over time. A source of variation in the two paths may be the Neatos wheel base not being accounted for during path planning; additionally, the simulator accounts for natural error that would occur in the servo motors.

The experimental path is very close to a straight line, which disagrees with the predicted path. This is likely due to the Neato not stopping and re-evaluating its path at different positions; however, the wheel encoder data shows different distances traveled for the left and right wheels, meaning that the Neato was turning as it traversed its path through the Gauntlet.

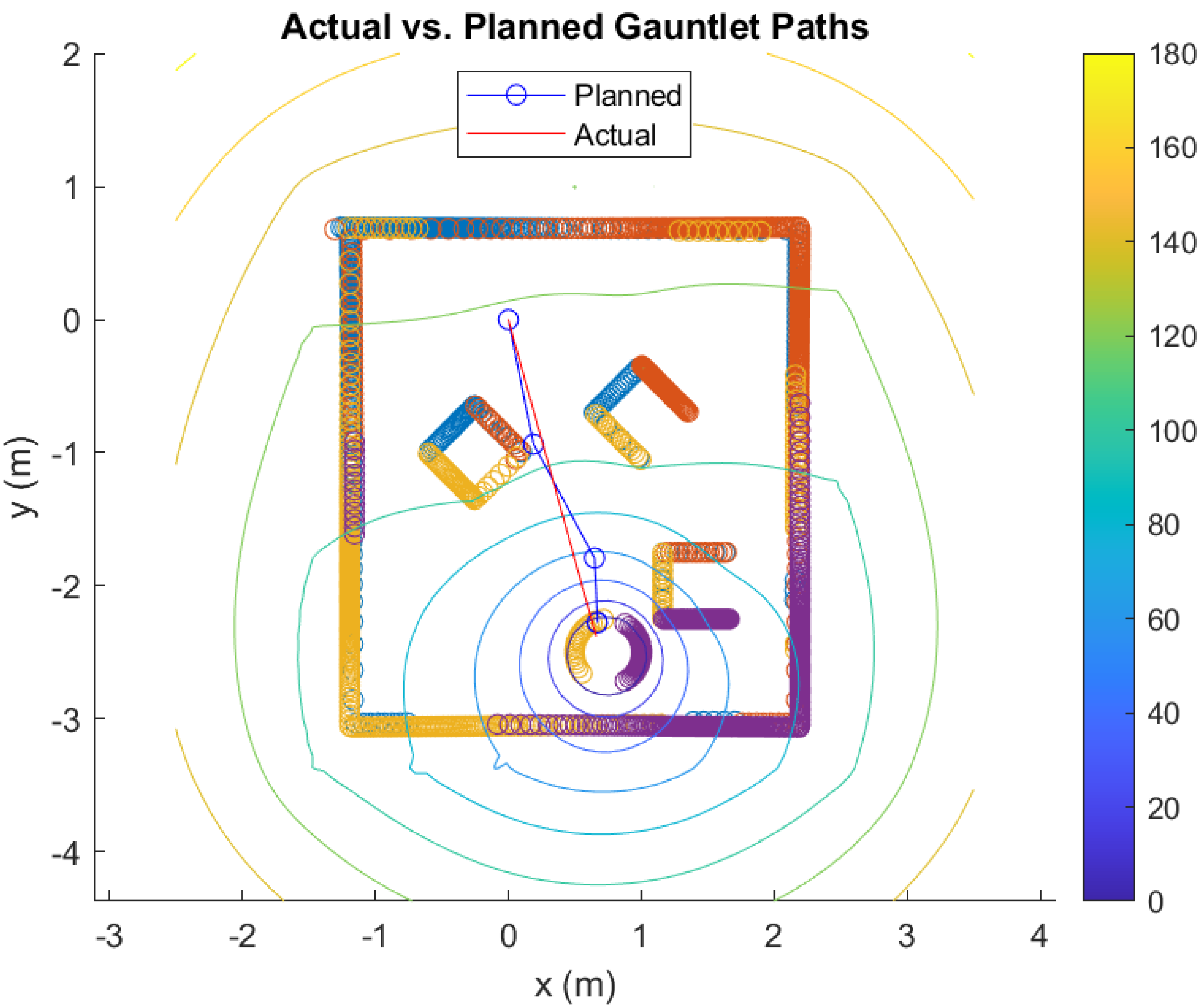

Figure 5: Experimental data of gradient descent.

Appendix A: Calculations & Plotting Code

```
% Initialize symbolic variables
syms x y a b u

% Define equations for point sinks and sources
sinks = log(sqrt((x-a).^2 + (y-b).^2));
sources = -log(sqrt((x-a).^2 + (y-b).^2));

% Independently create walls
a_vals = [u u -1.5 2.5];
b_vals = [-3.37 1 u u];
u_low = [-1.5 -1.5 -3.37 -3.37];
u_high = [2.5 2.5 1 1];

walls = 0;
% Sweep each wall from end to end
% Create a sink at each point, meaning that points outside the walls will
% have a very high potential
for i = 1:length(a_vals)
    for u_actual = u_low(:,i):0.2:u_high(:,i)
        a_actual = subs(a_vals(:,i), u, u_actual);
        b_actual = subs(b_vals(:,i), u, u_actual);
        walls = walls + subs(sinks, [a b], [a_actual b_actual]);
    end
end

% Contour plot
xyinterval = [-1.5-1 2.5+1 -3.37-1 1+1];
figure(); clf; hold on;
fcontour(walls, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
title("Contour Plot of Walls");
colorbar
% 3D plot
figure(); clf;
fsurf(walls, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
title("3D Plot of Walls")
colorbar

% Independently create BoB
% Sink equation
BoB_eq = log10(sqrt((x-a).^2 + (y-b).^2));

BoB = 0; r = 0.25;  % radius = 0.25m
for theta = 0:0.1:2*pi
    a_actual = r*cos(theta) + 0.75;
    b_actual = r*sin(theta) - 2.5;
    BoB = BoB + subs(BoB_eq, [a b], [a_actual b_actual]);
end
```

```
% Contour plot
figure(); clf; hold on;
fcontour(BoB, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar
% 3D plot
figure(); clf;
fsurf(BoB, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar

% Independently create obstacles, iteratively adding each subsequent
% potential field to that of the preceding obstacle

% Obstacle at (1.41, -2)
center = [1.41; -2];
sidelength = 0.5;
offset = sidelength/2;  % normal distance of corners from center

a_vals = [u u 1.41-offset 1.41+offset];
b_vals = [-2-offset -2+offset u u];
u_low = [1.41-offset 1.41-offset -2-offset -2-offset];
u_high = [1.41+offset 1.41+offset -2+offset -2+offset];
theta_actual = 0;       % degrees

syms theta
rot_matrix = [cosd(theta) -sind(theta);
              sind(theta) cosd(theta)];

obstacles = 0;
for i = 1:length(a_vals)
    for u_actual = u_low(:,i):0.2:u_high(:,i)
        a_actual = subs(a_vals(:,i), u, u_actual);
        b_actual = subs(b_vals(:,i), u, u_actual);

        % Similar to walls, except with rotation
        rot_matrix_actual = subs(rot_matrix, theta, theta_actual);
        rotated_point = rot_matrix_actual * [x - center(1,:); y - center(2,:)] ...
            + center;
        obstacles = obstacles + subs(sources, [a b x y], ...
            [a_actual b_actual rotated_point(1,:) rotated_point(2,:)]);
    end
end

% Contour plot
xyinterval = [-1.5-1 2.5+1 -3.37-1 1+1];
figure(); hold on;
fcontour(obstacles, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar
% 3D plot
```

```
figure(); clf;
fsurf(obstacles, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar

% Obstacle at (1, -0.7) rotated 45deg
center = [1; -0.7];
sidelength = 0.5;
offset = sidelength/2;

a_vals = [u u 1-offset 1+offset];
b_vals = [-0.7-offset -0.7+offset u u];
u_low = [1-offset 1-offset -0.7-offset -0.7-offset];
u_high = [1+offset 1+offset -0.7+offset -0.7+offset];
theta_actual = 45;       % degrees

syms theta
rot_matrix = [cosd(theta) -sind(theta);
              sind(theta) cosd(theta)];

for i = 1:length(a_vals)
    for u_actual = u_low(:,i):0.2:u_high(:,i)
        a_actual = subs(a_vals(:,i), u, u_actual);
        b_actual = subs(b_vals(:,i), u, u_actual);

        rot_matrix_actual = subs(rot_matrix, theta, theta_actual);
        rotated_point = rot_matrix_actual * [x - center(1,:); y - center(2,:)] ...
            + center;
        obstacles = obstacles + subs(sources, [a b x y], ...
            [a_actual b_actual rotated_point(1,:) rotated_point(2,:)]);
    end
end

xyinterval = [-1.5-1 2.5+1 -3.37-1 1+1];
figure(); hold on;
fcontour(obstacles, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar
figure(); clf;
fsurf(obstacles, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar

% Obstacle at (-0.25, -1) rotated 45deg
center = [-0.25; -1];
sidelength = 0.5;
offset = sidelength/2;

a_vals = [u u -0.25-offset -0.25+offset];
b_vals = [-1-offset -1+offset u u];
u_low = [-0.25-offset -0.25-offset -1-offset -1-offset];
u_high = [-0.25+offset -0.25+offset -1+offset -1+offset];
```

```
theta_actual = 45;        % degrees

syms theta
rot_matrix = [cosd(theta) -sind(theta);
              sind(theta) cosd(theta)];

for i = 1:length(a_vals)
    for u_actual = u_low(:,i):0.2:u_high(:,i)
        a_actual = subs(a_vals(:,i), u, u_actual);
        b_actual = subs(b_vals(:,i), u, u_actual);

        rot_matrix_actual = subs(rot_matrix, theta, theta_actual);
        rotated_point = rot_matrix_actual * [x - center(1,:); y - center(2,:)] ...
            + center;
        obstacles = obstacles + subs(sources, [a b x y], ...
            [a_actual b_actual rotated_point(1,:) rotated_point(2,:)]);
    end
end

% These plots show all three obstacles
xyinterval = [-1.5-1 2.5+1 -3.37-1 1+1];
figure(); hold on;
fcontour(obstacles, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar
figure(); clf;
fsurf(obstacles, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar

% Overall potential field equation is created by adding all features.
% The BoB is amplified by a factor of 2, and the obstacles are damped by a
% factor of 0.7 after the NEATO was unable to pass through due to the
% gradient being too high
mega_equation = (2*BoB + walls + 0.7*obstacles);

% mydata.mat contains the LIDAR scans
load('mydata.mat','r_all','theta_all');

% The origin of the Neato frame in the Global frame.
% Each row corresponds to the origin for a particular scan
origins = [-0.5 0; 1.5 0; -0.5 -2; 1.5 -2.5];

% the orientation of the Neato relative to the Global frame in radians.
% A positive angle here means the Neato's ihat_N axis was rotated
% counterclockwise from the ihat_G axis.
orientations = [-pi/3 -2*pi/3 0 pi];

% the origin of the Lidar frame in the Neato frame (ihat_N, jhat_N).
origin_of_lidar_frame = [-0.084 0];

allScansFig = figure;
```

```
cartPtsInGFrame = [];

% for each scan
for i = 1 : size(origins,1)
    cartesianPointsInLFrame = [cos(theta_all(:,i)).*r_all(:,i)...
                               sin(theta_all(:,i)).*r_all(:,i)]';

    % add a row of all ones so we can use translation matrices
    cartesianPointsInLFrame(end+1,:) = 1;
    cartesianPointsInNFrame = [1 0 origin_of_lidar_frame(1);...
                               0 1 origin_of_lidar_frame(2);...
                               0 0 1]*cartesianPointsInLFrame;

    rotatedPoints = [cos(orientations(i)) -sin(orientations(i)) 0;...
                     sin(orientations(i)) cos(orientations(i)) 0;...
                     0 0 1]*cartesianPointsInNFrame;

    cartesianPointsInGFrame = [1 0 origins(i,1);...
                               0 1 origins(i,2);...
                               0 0 1]*rotatedPoints;

    figure(allScansFig);
    scatter(cartesianPointsInGFrame(1,:), cartesianPointsInGFrame(2,:), 'HandleVisibility', 'off');
    hold on;
    title('All scans in Global frame');
end
save('cartesianPointsInGFrame', 'cartPtsInGFrame')

% display the figure that contains all of the scans
figure(allScansFig);
hold on;
title('All LIDAR scans in Global frame');
xlabel("x (m)"); ylabel("y (m)"); axis equal;

% Superimpose contour plot on LIDAR scans for context
figure(allScansFig); hold on;
fcontour(mega_equation, xyinterval, 'HandleVisibility', 'off');
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar
% 3D plot of all features
figure(); clf;
fsurf(mega_equation, xyinterval);
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
colorbar

% Calculate gradient of complete potential field
g = gradient(mega_equation, [x y]);

% Substitute values for quiver plotting
[X, Y] = meshgrid(-5:.41:5,-5:.41:5);
G1 = subs(g(1),[x y],{X,Y});
```

```
G2 = subs(g(2),[x y],{X,Y});

% Quiver plot
figure(); clf;
quiver(X,Y,G1,G2);
xlim([-1.5-1 2.5+1]); ylim([-3.37-1 1+1]);
xlabel("x (m)"); ylabel("y (m)"); axis equal;

% Calculating the path of gradient descent
% You can tweak these for fun
delta_0 = 0.19; % 1.1;
lambda_0 = 0.835;
step_length = 0.1;
r_0 = [0, 0];        % [x, y] initial position

syms x y
% Make BoB positive for gradient ascent algorithm
f = mega_equation;

% Since this is a gradient ascent algorithm, we want to flip the signs of
% all the gradient values (now, the BoB is most positive)
f_gradient = -1*gradient(f);

% Initialize data storage
r = r_0;        % [x, y] final points
steps = [];  % change in position between steps [dx, dy]
lambda = lambda_0;
delta = delta_0;
num_gradient = 100;

while norm(num_gradient) >= 0.5 & size(r,1) < 100
    % Calculate gradient at point
    x_pos = r(end, 1);    % get current position to solve
    y_pos = r(end, 2);

    num_gradient = subs(f_gradient, [x y], [x_pos y_pos]);

    % Record data
    lambda = delta * lambda;
    pos_change = num_gradient * lambda;
    x_change = pos_change(1,1);
    y_change = pos_change(2,1);
    steps(end+1,1) = x_change;
    steps(end+1,2) = y_change;
    r(end+1,1) = r(end,1) + x_change;
    r(end,2) = r(end-1,2) + y_change;
end

% Plot planned path
figure(allScansFig); hold on;
fcontour(mega_equation, xyinterval, "HandleVisibility", 'off');
xlabel("x (m)"); ylabel("y (m)"); axis equal;
```

```
colorbar
plot(r(:,1), r(:,2), 'b-o'); %, 'DisplayName', 'Planned');
xlim([-1.5-1 2.5+1]); ylim([-3.37-1 1+1]);

% Data Analysis
encoder_data = readmatrix('encoder_data.csv');  % encoder data from NEATO

timeseconds = encoder_data(:,1);
encoderLeftmeters = encoder_data(:,2);
encoderRightmeters = encoder_data(:,3);

% Calculate drive time by finding the difference
driveTime = timeseconds(end) - timeseconds(1)    % seconds

% Set parameter for distance between wheels
d = 0.235;      % m

% Find wheel velocities
velocityLeft = diff(encoderLeftmeters) ./ diff(timeseconds);
velocityRight = diff(encoderRightmeters) ./ diff(timeseconds);

% Find angular velocity
omega_actual = (velocityRight - velocityLeft) ./ d;  % rad/s

% Find linear speed
speed_actual = (velocityLeft + velocityRight) ./ 2; % m/s

% Integrate the angular velocity to get theta
theta_integrated = cumtrapz(timeseconds(1:end-1,:), omega_actual);

% Get speed components from actual speed
speed_x = speed_actual .* cos(theta_integrated);
speed_y = speed_actual .* sin(theta_integrated);

% Integrate speed components to get location coordinates
pos_x = cumtrapz(timeseconds(1:end-1,:), speed_x);
pos_y = cumtrapz(timeseconds(1:end-1,:), speed_y);
location_actual = [pos_x pos_y]';

% Calculate total distance traveled
(encoderLeftmeters(end)-encoderLeftmeters(1) + encoderRightmeters(end)-encoderRightmeters(1)) / 2

% Plotting the position curve
figure(allScansFig); hold on; legend("Location", "best"); axis equal;
title("Actual vs. Planned Gauntlet Paths"); xlabel("x (m)"); ylabel("y (m)");
plot(location_actual(1,:), location_actual(2,:), "r-"); %, 'DisplayName', "Actual");
legend({"Planned", "Actual"});
```

Appendix B: NEATO Driving Code

```
function position = run_gauntlet()
% to calculate wheel velocities for a given angular speed we need to know
% the wheel base of the robot
wheelBase = 0.235;              % meters
% this is the scaling factor we apply to the gradient when calculating our
% step size
lambda = 0.835;

% setup symbolic expressions for the function and gradient
syms x y;
f = [potential field equation goes here]
grad = -1*gradient(f, [x, y]);

% the problem description tells us to the robot starts at position 0, 0
% with a heading aligned to the x-axis
heading = [1; 0];
position = [0; 0];

angularSpeed = 0.2;  % radians / second
linearSpeed = 0.1;  % meters / second

% get setup with a publisher so we can modulate the velocity
pub = rospublisher('/raw_vel');
msg = rosmessage(pub);
% stop the robot's wheels in case they are running from before
msg.Data = [0, 0];
send(pub, msg);
pause(2);

% put the Neato in the starting location
placeNeato(position(1), position(2), heading(1), heading(2));
% wait a little bit for the robot to land after being positioned
pause(2);

% set a flag to control when we are sufficiently close to the maximum of f
shouldStop = false;

while ~shouldStop
    % get the gradient
    gradValue = double(subs(grad, {x, y}, {position(1), position(2)}));
    % calculate the angle to turn to align the robot to the direction of
    % gradValue. There are lots of ways to do this. One way is to use the
    % fact that the magnitude of the cross product of two vectors is equal
    % to the product of the vectors' magnitudes times the sine of the angle
    % between them. Moreover, the direction of the vector will tell us
    % what axis to turn around to rotate the first vector onto the second.
    % We'll use that approach here, but contact us for more approaches.
    crossProd = cross([heading; 0], [gradValue; 0]);

```

```
    % if the z-component of the crossProd vector is negative that means we
    % should be turning clockwise and if it is positive we should turn
    % counterclockwise
    turnDirection = sign(crossProd(3));

    % as stated above, we can get the turn angle from the relationship
    % between the magnitude of the cross product and the angle between the
    % vectors
    turnAngle = asin(norm(crossProd)/(norm(heading)*norm(gradValue)));

    % this is how long in seconds to turn for
    turnTime = double(turnAngle) / angularSpeed;
    % note that we use the turnDirection here to negate the wheel speeds
    % when we should be turning clockwise instead of counterclockwise
    msg.Data = [-turnDirection*angularSpeed*wheelBase/2,
                turnDirection*angularSpeed*wheelBase/2];
    send(pub, msg);
    % record the start time and wait until the desired time has elapsed
    startTurn = rostic;
    while rostoc(startTurn) < turnTime
        pause(0.01);
    end
    heading = gradValue;

    % this is how far we are going to move
    forwardDistance = norm(gradValue*lambda);
    % this is how long to take to move there based on desired linear speed
    forwardTime = forwardDistance / linearSpeed;
    % start the robot moving
    msg.Data = [linearSpeed, linearSpeed];
    send(pub, msg);
    % record the start time and wait until the desired time has elapsed
    startForward = rostic;
    while rostoc(startForward) < forwardTime
        pause(0.01)
    end
    % update the position for the next iteration
    position = position + gradValue*lambda;
    % if our step is too short, flag it so we break out of our loop
    shouldStop = forwardDistance < 0.01;
end

% stop the robot before exiting
msg.Data = [0, 0];
send(pub, msg);

end
```

Appendix C: Potential Field Equation

```
log(((x - 1/2)^2 + (y - 1)^2)^(1/2)) - log((((2^(1/2)*(x + 1/4))/2 ...
- (2^(1/2)*(y + 1))/2 + 1/4)^2 + ((2^(1/2)*(x + 1/4))/2 + ...
(2^(1/2)*(y + 1))/2 - 1/4)^2)^(1/2))/2 - log((((2^(1/2)*(x + 1/4))/2 ...
- (2^(1/2)*(y + 1))/2 + 1/4)^2 + ((2^(1/2)*(x + 1/4))/2 + ...
(2^(1/2)*(y + 1))/2 + 1/4)^2)^(1/2)) - log((((2^(1/2)*(y + 1))/2 ...
- (2^(1/2)*(x + 1/4))/2 + 1/4)^2 + ((2^(1/2)*(x + 1/4))/2 ...
+ (2^(1/2)*(y + 1))/2 + 1/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x + 1/4))/2 - (2^(1/2)*(y + 1))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x + 1/4))/2 + (2^(1/2)*(y + 1))/2 + 1/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x + 1/4))/2 + (2^(1/2)*(y + 1))/2 - 1/4)^2 ...
+ ((2^(1/2)*(x + 1/4))/2 - (2^(1/2)*(y + 1))/2 + 1/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x + 1/4))/2 + (2^(1/2)*(y + 1))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x + 1/4))/2 - (2^(1/2)*(y + 1))/2 + 1/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(y + 1))/2 - (2^(1/2)*(x + 1/4))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x + 1/4))/2 + (2^(1/2)*(y + 1))/2 - 3/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x + 1/4))/2 - (2^(1/2)*(y + 1))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x + 1/4))/2 + (2^(1/2)*(y + 1))/2 - 3/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x + 1/4))/2 + (2^(1/2)*(y + 1))/2 - 1/4)^2 ...
+ ((2^(1/2)*(y + 1))/2 - (2^(1/2)*(x + 1/4))/2 + 3/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x + 1/4))/2 + (2^(1/2)*(y + 1))/2 + 1/4)^2 ...
+ ((2^(1/2)*(y + 1))/2 - (2^(1/2)*(x + 1/4))/2 + 3/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(y + 7/10))/2 - (2^(1/2)*(x - 1))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 + 1/4)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x - 1))/2 - (2^(1/2)*(y + 7/10))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 - 1/4)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x - 1))/2 - (2^(1/2)*(y + 7/10))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 + 1/4)^2)^(1/2)) ...
- log((((2^(1/2)*(y + 7/10))/2 - (2^(1/2)*(x - 1))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 + 1/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x - 1))/2 - (2^(1/2)*(y + 7/10))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 + 1/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 - 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 - (2^(1/2)*(y + 7/10))/2 + 1/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 - (2^(1/2)*(y + 7/10))/2 + 1/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(y + 7/10))/2 - (2^(1/2)*(x - 1))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 - 3/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x - 1))/2 - (2^(1/2)*(y + 7/10))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 - 3/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 - 1/4)^2 ...
+ ((2^(1/2)*(y + 7/10))/2 - (2^(1/2)*(x - 1))/2 + 3/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(x - 1))/2 + (2^(1/2)*(y + 7/10))/2 + 1/4)^2 ...
+ ((2^(1/2)*(y + 7/10))/2 - (2^(1/2)*(x - 1))/2 + 3/20)^2)^(1/2))/2 ...
- log((((2^(1/2)*(y + 1))/2 - (2^(1/2)*(x + 1/4))/2 + 1/4)^2 ...
+ ((2^(1/2)*(x + 1/4))/2 + (2^(1/2)*(y + 1))/2 + 1/4)^2)^(1/2))/2 ...
+ log(((x + 1/2)^2 + (y - 1)^2)^(1/2)) + log(((x - 3/2)^2 ...
+ (y - 1)^2)^(1/2)) + log(((x + 3/2)^2 + (y - 1)^2)^(1/2)) ...
+ log(((x - 5/2)^2 + (y - 1)^2)^(1/2)) + log(((x - 1/10)^2 ...
+ (y - 1)^2)^(1/2)) + log(((x + 1/10)^2 + (y - 1)^2)^(1/2)) ...
```

```
+ log(((x - 3/10)^2 + (y - 1)^2)^(1/2)) + log(((x + 3/10)^2 ...
+ (y - 1)^2)^(1/2)) + log(((x - 7/10)^2 + (y - 1)^2)^(1/2)) ...
+ log(((x + 7/10)^2 + (y - 1)^2)^(1/2)) + log(((x - 9/10)^2 ...
+ (y - 1)^2)^(1/2)) + log(((x + 9/10)^2 + (y - 1)^2)^(1/2)) ...
+ log(((x - 11/10)^2 + (y - 1)^2)^(1/2)) + log(((x + 11/10)^2 ...
+ (y - 1)^2)^(1/2)) + log(((x - 13/10)^2 + (y - 1)^2)^(1/2)) ...
+ log(((x + 13/10)^2 + (y - 1)^2)^(1/2)) + log(((x - 17/10)^2 ...
+ (y - 1)^2)^(1/2)) + log(((x - 19/10)^2 + (y - 1)^2)^(1/2)) ...
+ log(((x - 21/10)^2 + (y - 1)^2)^(1/2)) + log(((x - 23/10)^2 ...
+ (y - 1)^2)^(1/2)) - log(((x - 29/25)^2 + (y + 7/4)^2)^(1/2))/2 ...
- log(((x - 29/25)^2 + (y + 9/4)^2)^(1/2)) - log(((x - 34/25)^2 ...
+ (y + 7/4)^2)^(1/2))/2 - log(((x - 34/25)^2 + (y + 9/4)^2)^(1/2))/2 ...
- log(((x - 39/25)^2 + (y + 7/4)^2)^(1/2))/2 - log(((x - 39/25)^2 ...
+ (y + 9/4)^2)^(1/2))/2 + log(((x + 3/2)^2 + (y - 3/100)^2)^(1/2)) ...
+ log(((x - 5/2)^2 + (y - 3/100)^2)^(1/2)) - log(((x - 29/25)^2 ...
+ (y + 37/20)^2)^(1/2))/2 - log(((x - 29/25)^2 ...
+ (y + 41/20)^2)^(1/2))/2 + log(((x + 3/2)^2 + (y + 17/100)^2)^(1/2)) ...
+ log(((x - 5/2)^2 + (y + 17/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y - 23/100)^2)^(1/2)) + log(((x - 5/2)^2 + (y - 23/100)^2)^(1/2)) ...
+ log(((x + 3/2)^2 + (y + 37/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 37/100)^2)^(1/2)) - log(((x - 83/50)^2 + (y + 9/4)^2)^(1/2))/2 ...
+ log(((x + 3/2)^2 + (y - 43/100)^2)^(1/2)) + log(((x - 5/2)^2 + ...
(y - 43/100)^2)^(1/2)) + log(((x + 3/2)^2 + (y + 57/100)^2)^(1/2)) ...
+ log(((x - 5/2)^2 + (y + 57/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y - 63/100)^2)^(1/2)) + log(((x - 5/2)^2 + (y - 63/100)^2)^(1/2)) ...
+ log(((x + 3/2)^2 + (y + 77/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 77/100)^2)^(1/2)) + log(((x + 3/2)^2 + (y - 83/100)^2)^(1/2)) ...
+ log(((x - 5/2)^2 + (y - 83/100)^2)^(1/2)) - log(((x - 83/50)^2 ...
+ (y + 37/20)^2)^(1/2))/2 - log(((x - 83/50)^2 ...
+ (y + 41/20)^2)^(1/2))/2 + log(((x + 3/2)^2 + (y + 97/100)^2)^(1/2)) ...
+ log(((x - 5/2)^2 + (y + 97/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 117/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 117/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 137/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 137/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 157/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 157/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 177/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 177/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 197/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 197/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 217/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 217/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 237/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 237/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 257/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 257/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 277/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 277/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 297/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
+ (y + 297/100)^2)^(1/2)) + log(((x + 3/2)^2 ...
+ (y + 317/100)^2)^(1/2)) + log(((x - 5/2)^2 ...
```

```
+ (y + 317/100)^2)^(1/2)) + log(((x - 1/2)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x + 1/2)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 3/2)^2 ...
+ (y + 337/100)^2)^(1/2)) + 2*log(((x + 3/2)^2 ...
+ (y + 337/100)^2)^(1/2)) + 2*log(((x - 5/2)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 1/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x + 1/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 3/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x + 3/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 7/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x + 7/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 9/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x + 9/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 11/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x + 11/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 13/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x + 13/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 17/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 19/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 21/10)^2 ...
+ (y + 337/100)^2)^(1/2)) + log(((x - 23/10)^2 ...
+ (y + 337/100)^2)^(1/2)) ...
+ (2*log(((x - 2253719725459081/4503599627370496)^2 ...
+ (y + 5662361255536685/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5826972935901969/2251799813685248)^2 ...
+ (x - 580835805886253/1125899906842624)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 8969523127543341/9007199254740992)^2 ...
+ (y + 5732047907508205/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5494814135476029/2251799813685248)^2 ...
+ (x - 2284498012837863/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 8501818541785761/9007199254740992)^2 ...
+ (y + 5984871058599771/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 2487149247367907/4503599627370496)^2 ...
+ (y + 2986972452709893/1125899906842624)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 367413524822121/140737488355328)^2 ...
+ (x - 2368039503116917/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 1537671716626571/562949953421312)^2 ...
+ (x - 7606529651674005/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 6192406285483717/2251799813685248)^2 ...
+ (x - 3363751325489099/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 2422823973831897/4503599627370496)^2 ...
+ (y + 1481942691264621/562949953421312)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 1279402647533871/562949953421312)^2 ...
+ (x - 5818320072053183/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 4633703349887981/9007199254740992)^2 ...
+ (y + 2720458985343081/1125899906842624)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 1281948598094451/562949953421312)^2 ...
+ (x - 3888403551385105/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 6063347341698987/9007199254740992)^2 ...
+ (y + 6165203877384847/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 2624623784092019/1125899906842624)^2 ...
+ (x - 2547468204893647/4503599627370496)^2)^(1/2)))/log(10) ...
```

```
+ (2*log(((y + 2731568223819031/1125899906842624)^2 ...
+ (x - 4453312984637751/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 35768464461985/70368744177664)^2 ...
+ (y + 1443339096424089/562949953421312)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5517658643695797/2251799813685248)^2 ...
+ (x - 4481156589218895/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 1297131421731167/562949953421312)^2 ...
+ (x - 8155140651792519/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 562246852190303/562949953421312)^2 ...
+ (y + 2786649158480949/1125899906842624)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 6064527266488407/2251799813685248)^2 ...
+ (x - 8184600740850723/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 4458755356820575/4503599627370496)^2 ...
+ (y + 2893398237954699/1125899906842624)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 4421947599301125/4503599627370496)^2 ...
+ (y + 5839973385264617/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 8324243479356953/9007199254740992)^2 ...
+ (y + 5225663956254739/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 1316709304297369/562949953421312)^2 ...
+ (x - 2119417733970393/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 4317502767218629/4503599627370496)^2 ...
+ (y + 2969753967384125/1125899906842624)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 317171290913567/140737488355328)^2 ...
+ (x - 446133213860695/562949953421312)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 673613256317785/281474976710656)^2 ...
+ (x - 2359804980051851/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 7972052072743331/9007199254740992)^2 ...
+ (y + 5155793482510129/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 1266697405234933/562949953421312)^2 ...
+ (x - 1672411990544501/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 3072627552816209/1125899906842624)^2 ...
+ (x - 2926439970020427/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 8003739467975863/9007199254740992)^2 ...
+ (y + 6098024452047139/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5410276496549061/2251799813685248)^2 ...
+ (x - 4414722207872095/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 323397282583849/140737488355328)^2 ...
+ (x - 1357553183716649/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 2945523654736767/1125899906842624)^2 ...
+ (x - 4374706062159429/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 350380731389567/140737488355328)^2 ...
+ (x - 4505547095823991/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 6169325909969299/2251799813685248)^2 ...
+ (x - 7394149897431387/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5550056032249003/2251799813685248)^2 ...
+ (x - 1131533630415267/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 3785678282536861/4503599627370496)^2 ...
+ (y + 1276202043545257/562949953421312)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 6465267432806711/9007199254740992)^2 ...
+ (y + 5071241875359349/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5081272055770785/2251799813685248)^2 ...
+ (x - 780473225819619/1125899906842624)^2)^(1/2)))/log(10) ...
```

```
+ (2*log(((x - 1206464376467641/2251799813685248)^2 ...
+ (y + 1334824515239401/562949953421312)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 2604852409876103/1125899906842624)^2 ...
+ (x - 5255079220474887/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 193446573573663/70368744177664)^2 ...
+ (x - 1738107408915585/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 1)^2 + (y + 5/2)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 3905202597099183/4503599627370496)^2 ...
+ (y + 6126840291500621/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5067959777966375/2251799813685248)^2 ...
+ (x - 6914685458591693/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 1522537148943139/562949953421312)^2 ...
+ (x - 5461010985350311/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 330220303586157/562949953421312)^2 ...
+ (y + 6055541463695851/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 2182884922603477/2251799813685248)^2 ...
+ (y + 334975434349181/140737488355328)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 4951384396832033/9007199254740992)^2 ...
+ (y + 5292589668565845/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 6016677461490493/2251799813685248)^2 ...
+ (x - 2560372606854883/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5087064498613787/2251799813685248)^2 ...
+ (x - 1839438313378403/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 4175590953384585/4503599627370496)^2 ...
+ (y + 6026683427629829/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 2838137354829517/1125899906842624)^2 ...
+ (x - 2249853187365533/2251799813685248)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 3091285593364317/1125899906842624)^2 ...
+ (x - 3587693979885747/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 1412857553725895/2251799813685248)^2 ...
+ (y + 3060076537348247/1125899906842624)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 2859151232691737/1125899906842624)^2 ...
+ (x - 2265896339578505/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 5096779943926907/2251799813685248)^2 ...
+ (x - 3013708027313061/4503599627370496)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 772475008552577/281474976710656)^2 ...
+ (x - 6280729499171881/9007199254740992)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 8613890024937157/9007199254740992)^2 ...
+ (y + 2655817040057845/1125899906842624)^2)^(1/2)))/log(10) ...
+ (2*log(((x - 1625713600449765/2251799813685248)^2 ...
+ (y + 1547224459605939/562949953421312)^2)^(1/2)))/log(10) ...
+ (2*log(((y + 321472241341581/140737488355328)^2 ...
+ (x - 2809293538389031/4503599627370496)^2)^(1/2)))/log(10);
```