The Gauntlet:
Threading the NEA(TO) - dle
Quantitative Engineering Analysis

Sam Cabrera Valencia
Sam Kaplan
Daniel Arnott

May 6, 2020

1 Introduction

Our objective for this project was to write a program using MATLAB and its ROS toolbox to autonomously pilot a simulated robot from a defined starting position and heading to the goal. To achieve this, we implemented gradient ascent to have the robot traverse "The Gauntlet" (pictured below) and gently tap the "Barrel of Benevolence" (the cylindrical object)

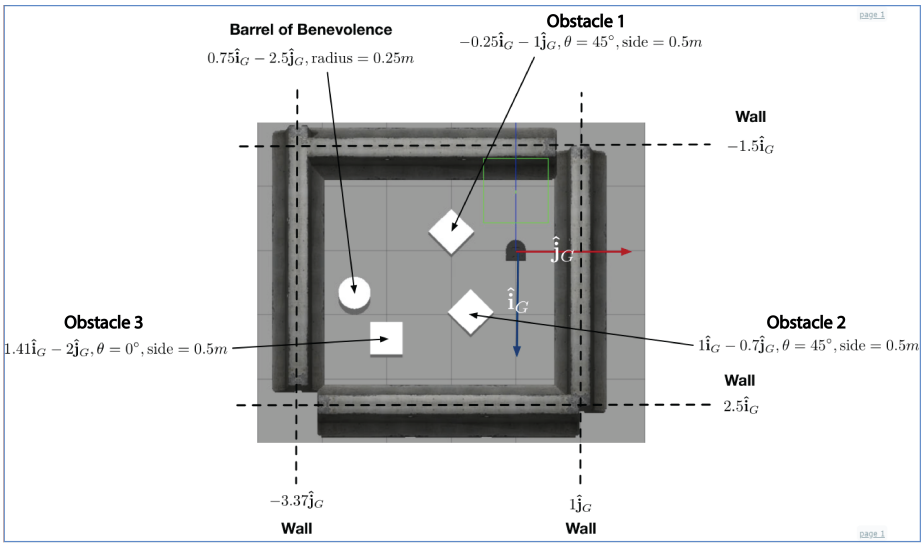

Figure 1: Layout for the "Gauntlet" simulation with respective feature labels and equations that define them.

2 Methodology and Data

We started by constructing a single function that packages all these features of the Gauntlet and created a potential field from it.

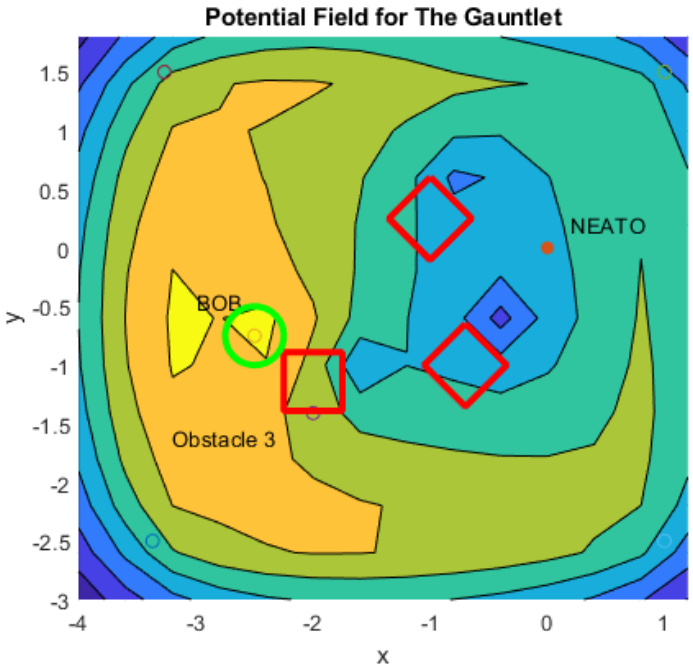

Figure 2: Constructed Potential Field

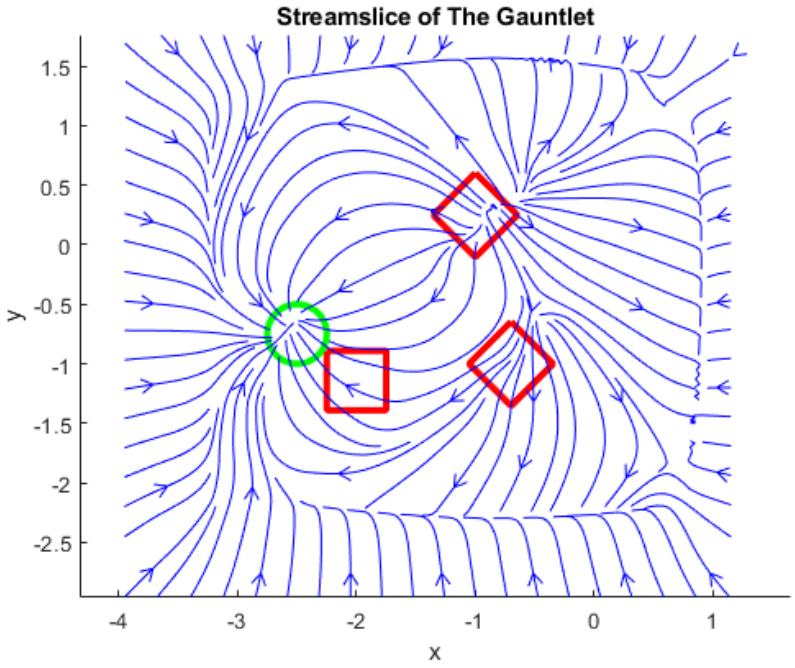

Figure 3: Constructed Potential Field Streamslice

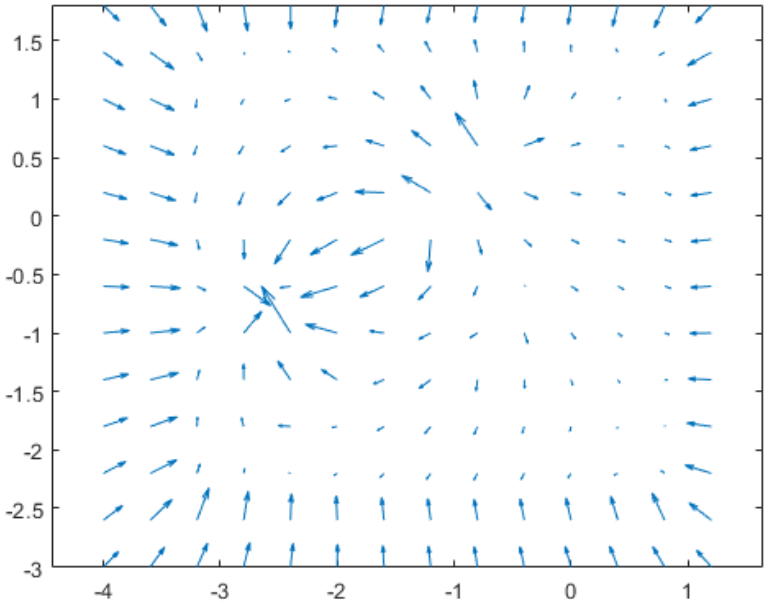

Figure 4: Vector Field of The Gauntlet

We created a findGradient function that would return the gradient vector components at any point on the potential field. With this function we were able to get points for the trajectory that the simulated NEATO would take through The Gauntlet (We then plotted the experimental path that the robot actually took over it).

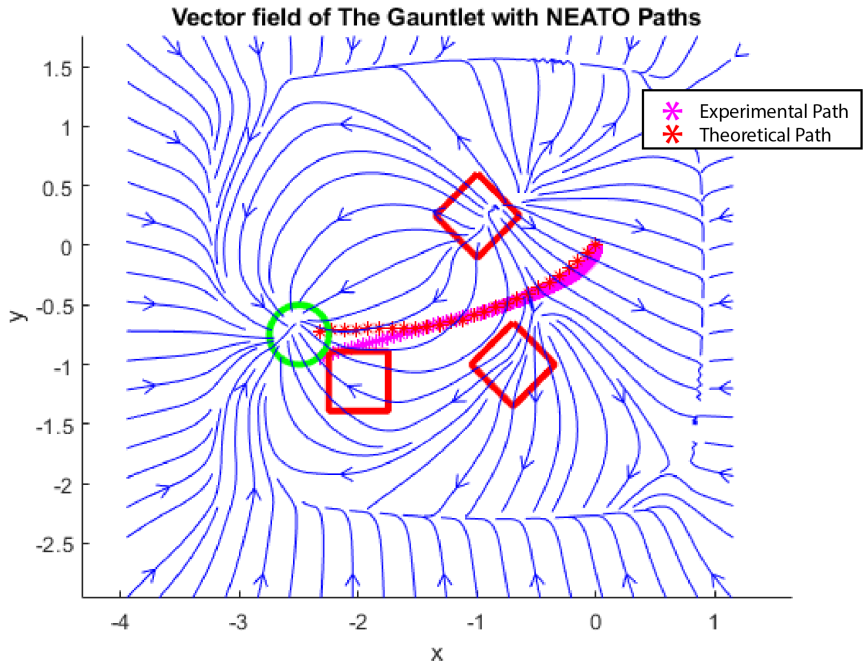

Figure 5: Theoretical and experimental NEATO paths plotted over streamlice of The Gauntlet

We passed these points to an encoder script that would calculate the unit tangent and unit normal vectors for each point we passed using the following formulas:

$$\hat{\mathbf{T}} = \frac{\mathbf{r}'}{|\mathbf{r}'|}$$

$$\hat{\mathbf{N}} = \frac{\hat{\mathbf{T}}'}{|\hat{\mathbf{T}}'|}$$

where r defines the set of positions we passed. We could also calculate the linear speed at each point we passed by calculating magnitude of the same position vectors. With the two unit vectors we could find ω (angular velocity) by using the following formula:

$$\omega = \hat{\mathbf{T}} \times \frac{d\mathbf{T}}{dt}$$

Given the wheel base distance of .235m we could use linear speed and angular speed to find wheel velocities at specified times for both of the robot's wheels with the following equations:

$$V_{left} = V - \omega\frac{d}{2}$$
$$V_{right} = V + \omega\frac{d}{2}$$

where V is the linear speed for a given position. This wheel velocity data can then be sent to the robot using functions in the ROS toolbox.

In addition to sending speeds to each of the wheels, it was important to send them at the right times, which meant adjusting the ROS rate of the program to the frequency where each speed occurred. This mean calculating the time and then frequency of each velocity set by using the formulas:

$$time = distance/speed$$
$$frequency = 1/time$$

This resulted in the final code for sending velocities to look like this

```
times = .1 ./ linSpeeds(2:25); % time = distance/speed
rosrates = [];
time_freq = [];
for j = 1:length(times)
    freq = 1/times(j); % Frequency = 1/time
    time_freq = [time_freq freq];
end
r = rosrate(1);
reset(r);
for i = 1:length(time_freq)
    %sending speeds to the robot
    r = rosrate(time_freq(i)); %Sets frequency for sending a set of speeds at point
    message.Data = [VL(i,:), VR(i,:)];%Send velocities
    send(pub, message);
    waitfor(r);
end
```

Figure 6: Code for sending left and right wheel speeds at the right time for to reach the BOB

Link to our NEATO video: `https://youtu.be/0D3JNF2A-vk`

Link to our Github Repo `https://github.com/samjumjum/QEA_Final_Gauntlet`