ROBO REPORT

James Jagielski, Krishna Suresh, and Evan Lockwood

March 29th, 2022

ABSTRACT

The scope of this project was to program a NEATO robot across a "bridge" without falling off. We accomplished this task with an open-loop system and then with a closed-loop system. The closed-loop system utilized odometry, motion profiles, and error corrections. We then attempted the challenge problem of a low friction ice bridge.

1. INTRODUCTION

The NEATO robot is a robotic vacuum cleaner connected to a Raspberry Pi that allows us to program the individual wheel velocities, which means we can accelerate and change the heading of the NEATO. The "bridge" is a wooden path characterized by the parametric equation in Equation 1. The ice bridge problem is a path characterized with the same parametric equation, but in the simulator has a near friction-less surface.

2. OPEN LOOP CONTROL

The open-loop system uses a direct feed-forward control to apply the target linear and angular velocity to the robot through the trajectory.

2.1. Generating Wheel Velocities

The bridge path is represented by the parametric equation,

$$r(u) = 0.3960cos(2.65(u+1.4))\hat{i} - 0.99sin(u+1.4)\hat{j} \quad (1)$$

where u is a point on the curve that can be a function of time. To add further control, the equation becomes:

$$r(t) = 4(0.396cos(2.65(\frac{u}{\beta}+1.4))\hat{i} - 0.99sin(\frac{u}{\beta}+1.4)\hat{j}) \quad (2)$$

where β is a constant that we chose to change the velocity of the robot. We used the parametric equation of the bridge of doom to compute the tangent T, normal N and binormal B parametric equations. We used β as a speed control parameter in the parametric equation.

In order to control the robot, we need to find the target linear speed and angular velocity at each loop step:

$$\mathrm{v}(t) = |r'(t)| \quad (3)$$

$$\omega(t) = |B(t)| = |\hat{T}(t) \times N(t)| \quad (4)$$

Finally, we looped until the time exceeded 3.2β and set the individual wheel velocities using these equations at each time step:

$$V_L = \mathrm{v}(t) - \omega(t)\frac{d}{2} \quad (5)$$

$$V_R = \mathrm{v}(t) + \omega(t)\frac{d}{2} \quad (6)$$

where d is the wheel base track width.

2.2. Reconstructing Path from Encoder Data

The data from the encoder data was the distance traveled by the individual wheels across a length of time. First, we calculated the wheel velocities from the equations,

$$V_L = \frac{\Delta D_L}{\Delta t} \quad (7)$$

$$V_R = \frac{\Delta D_R}{\Delta t}, \quad (8)$$

where V_L is equal to the velocity of the left wheel, V_R is equal to the velocity of the right, ΔD_L is the change in distance of the left wheel in the experimental data, and Δt is the change in time. These velocities were calculated for each time step. Then, we used the velocities to calculate the linear velocity and angular velocity of the NEATO,

$$V_{NEATO} = \frac{V_R + V_L}{2}, \quad (9)$$

$$\omega_{NEATO} = \frac{V_R - V_L}{d}, \quad (10)$$

where V_R is velocity of the right wheel, V_L is the velocity of the left wheel, and d is the robot track width. Finally we iterated through the time steps and tracked the position and heading via these equations:

$$r(t+\Delta t) = (x(t)+V(t)cos(\theta(t))\Delta t)\hat{i} + (y(t)+V(t)sin(\theta(t))\Delta t)\hat{j}, \quad (11)$$

$$\theta(t + \Delta t) = \theta(t) + \omega(t)\Delta t \tag{12}$$

where $x(t)$ is the previous $\hat{i}$ position, $y(t)$ is the previous $\hat{j}$ position, $V(t)$ is the velocity given time, and $\theta(t)$ is the heading. $\omega(t)$ represents the angular velocity in respect to time. We used this equation to calculate the position of the experimental data. From this, the tangential and normal vectors of specific points are derived.

2.3. Position Diagrams

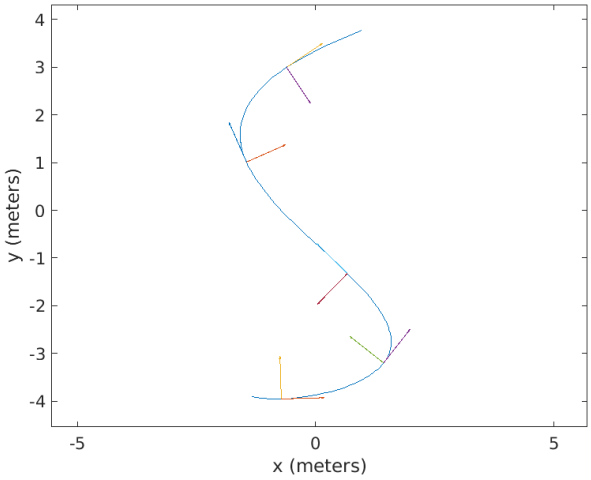

Fig. 1. Parametric curve with the tangential and normal vectors of various points. This path is based on Equation 1.

The path above is theoretically calculated based upon the parametric equation given. We took the derivative at various points to get the tangent of the curve, which is in the same direction as the velocity. The normal vector is is orthogonal to the curve and thus orthogonal to the tangent line. The normal vector describes which direction the NEATO's heading is going to point next. If there's a right hand curve the normal vector points right and if there's a left hand curve the normal vector points left. The speed and angular speed are based upon these calculations of tangent and normal vectors. The Equations 3 and 4 demonstrate their roles in the calculations.

2.4. Speed and Angular Velocity Diagrams

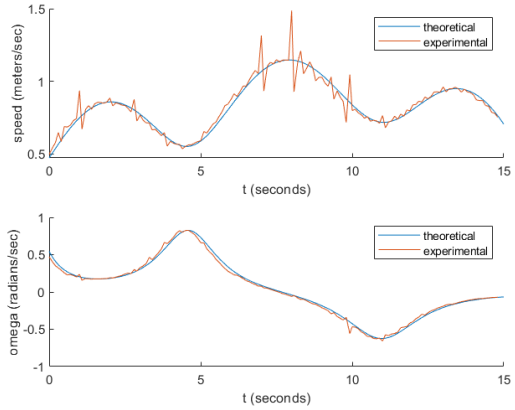

Fig. 2. The graphs compare the theoretical calculations for speed and angular velocity with the experimental calculations. The experimental calculations are from the encoder data run with the simulation for the robot.

The extremes in the angular velocity diagram are describing the curves in the position graph.

2.5. Individual Wheel Velocities

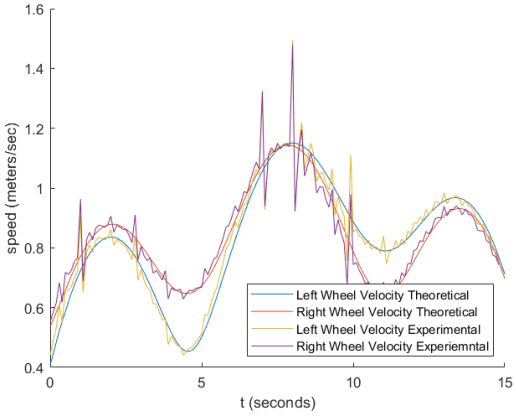

Fig. 3. Illustrated is the comparison between the left and right wheel velocities as well as theoretical versus experimental.

As shown in Fig. 3, our experimental wheel data followed the theoretical wheel data fairly accurately, with the exception of several sharp peaks, which are a result of estimation error.

3. RECONSTRUCTED PATH

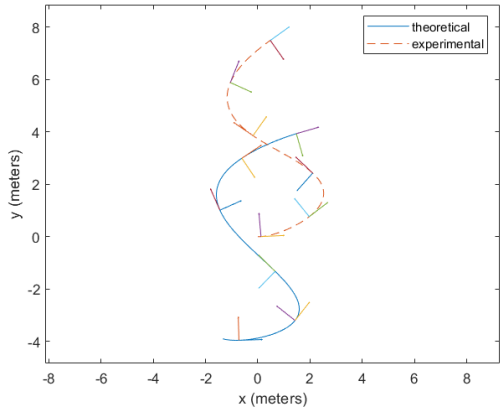

Fig. 4. This diagram has the path of the bridge from the theoretical data and the experimental data. The experimental data is represented by the dashed line.

The reason why the theoretical and experimental data is so off is because we set the starting reference point to the experimental data to the origin $(0, 0)$. The paths are quite close when put to the same reference point with little variance.

4. CLOSED LOOP CONTROL

In order to allow for more consistent path following we aimed to implement a multitude of systems that each help the model to more accurately represent the physical robot and robustly traverse a known trajectory.

4.1. Odometry

Odometry is a system for tracking the current position of the NEATO. Absolute position tracking allows us to apply more advanced control and correction algorithms by giving us a positional error we can aim to minimize. To track the odometry of the NEATO, we start by calculating the robot's position update in relation to the current estimated heading. Using (11) and (12), we iterate through time steps to track $\hat{i}$ and $\hat{j}$ positions. To improve the accuracy of the discrete position tracking, we found that heading accuracy impacted the overall position estimate the most. Therefore, we attempted to improve the estimated heading by reading data from the NEATO's IMU rather than cumulatively tracking θ.

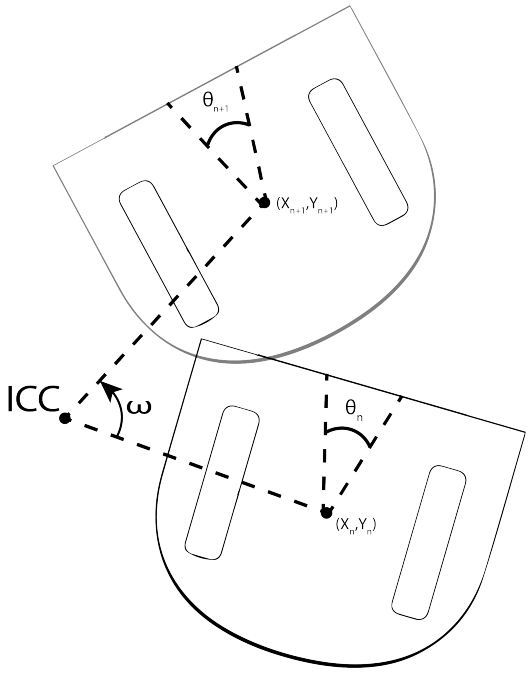

Fig. 5. NEATO between two time steps. ICC is the instantaneous center of curvature [1].

4.2. Motion Profile

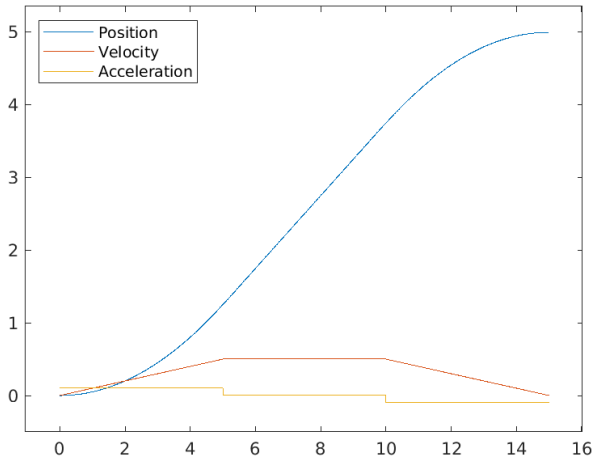

Fig. 6. This is a generated example trapezoidal motion profile with a max acceleration of $0.1\frac{m}{s^2}$, max velocity of $0.5\frac{m}{s}$ and total displacement of $5m$.

First, we generated the acceleration of the NEATO. The NEATO accelerates at a rate of $0.1\frac{m}{s^2}$ to a max velocity of $0.5\frac{m}{s}$, which is constant until deceleration to bring the NEATO to a stop. We recalculated the overall time and parameterized the curved based on the total distanced travelled rather than the total time elapsed. By integrating this acceleration, we create a trapezoidal motion profile, sample the

trapezoid at each time step and multiply the calculated open loop velocities. This allows us to make the NEATO speed up and slow down according to its position.

An object is unable to instantaneously move at any velocity; some form of acceleration is necessary to reach it. A trapezoidal motion profile in the form of a piecewise function was created to allow the NEATO to accelerate and decelerate to a designated velocity. The function represents the NEATO's velocity over time given a certain acceleration and maximum velocity. This allows the NEATO time to accelerate to a given maximum velocity and decelerate back to zero at the end of its path.

The addition of the motion profile to the NEATO's drive path permits the NEATO to move along a more accurate path by providing realistic velocities. We combined the parametric equation used as the path and this generated motion profile to create a trajectory we aim to follow.

4.3. PID Control System

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \tag{13}$$

where K_p is the proportional gain, $e(t)$ is the error value, K_i is the integral gain, de is the change in error value, and dt is the change in time.

A PID (proportional-integral-derivative) controller is a control loop feedback system used for minimizing error in a non-linear system. This is accomplished by generating an actuating signal to produce a desired output. The calculated error is inputted into the controller and the resulting output is fed back to drive our actuator.

4.4. Error Correction

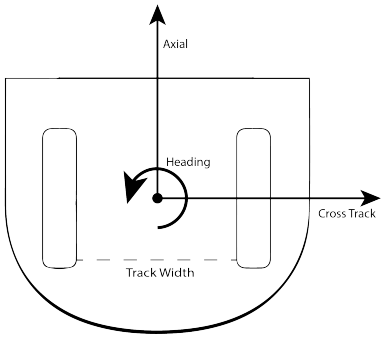

Fig. 7. Diagram of the NEATO, with its cross track, axial, and heading.

In the case of a differential drive robot following a trajectory, we needed to first apply the open loop control scheme, we discussed earlier, to provide the foundation for our wheel velocities. Next, we estimate our position and find the target position for our current timestamp.

$$\hat{x}_{target} - \hat{x}_{current} = \hat{e} \tag{14}$$

This error vector $\hat{e}$ represents the absolute position vector we aim to minimize. We first transform this vector into the robot's frame of reference by rotating it by $\frac{\pi}{2} - \theta$ where θ is the current robot heading in the global frame of reference. The $\hat{i}$ and $\hat{j}$ components represent the cross-track and axial error respectively. Next, we use two different PID controllers for reducing the axial and cross-track error independently. The output from the axial controller adjusts the speed and the cross-track controller adjusts the heading.

5. ICE BRIDGE

The ice bridge uses the path from the same parametric equation in Equation 1 used for the previous bridge path, except the simulated surface is made of ice. This provides the effect of significantly reduced friction, making any form error correction more difficult to implement. The reduction of friction amplifies the effects of over-correction, resulting in delayed response. This causes the error correction method to fail because the heading correction from the cross track controller fails. Instead, we applied a heading controller to maintain our robot's heading without over-correcting.

6. RESULTS

We applied our final results in three main cases: simulated bridge, small scale bridge on physical robot, and simulated ice bridge. In all trajectory graphs the blue markers represent the target position and the red markers represent the robot position.

6.1. Simulated Bridge

`https://youtu.be/B-t-N9NwGu8`

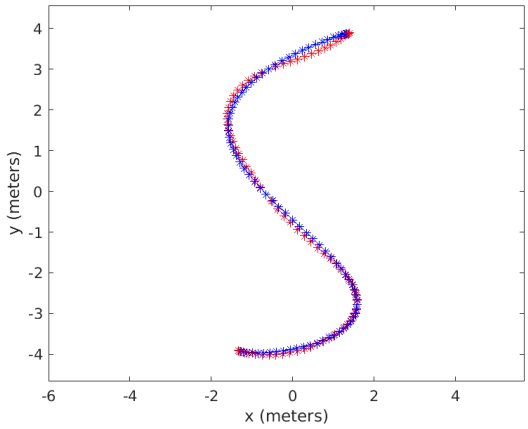

Fig. 8. Simulated robot path.

6.2. Real Bridge

https://youtu.be/2u-klaHMN00

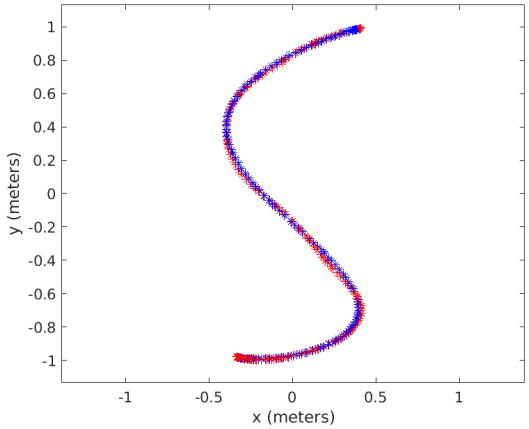

Fig. 9. Real robot path.

6.3. Simulated Ice Bridge

https://youtu.be/MkUIAqCw6v0

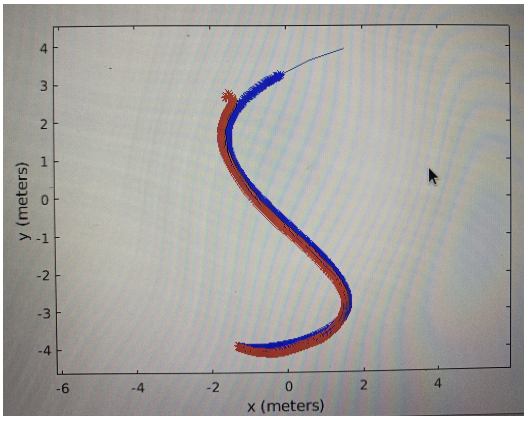

Fig. 10. Path on ice bridge before falling off.

All code written can be found at https://github.com/krish-suresh/qea_2_bridge_of_doom.

7. REFERENCES

[1] Columbia University, *Differential Drive Robots*, CS W4733 NOTES, http://www.cs.columbia.edu/~allen/F17/NOTES/icckinematics.pdf.