Nathan Ge
905597306

HW3

A clear step-by-step description of your method, including pseudocode/algorithms for: (i) force/energy evaluation, (ii) time stepping, (iii) enforcement of Dirichlet constraints, and (iv) the path-planning or control law mapping the tracking error of node j to {xc, yc, θc}.

<u>Time stepping main loop</u>

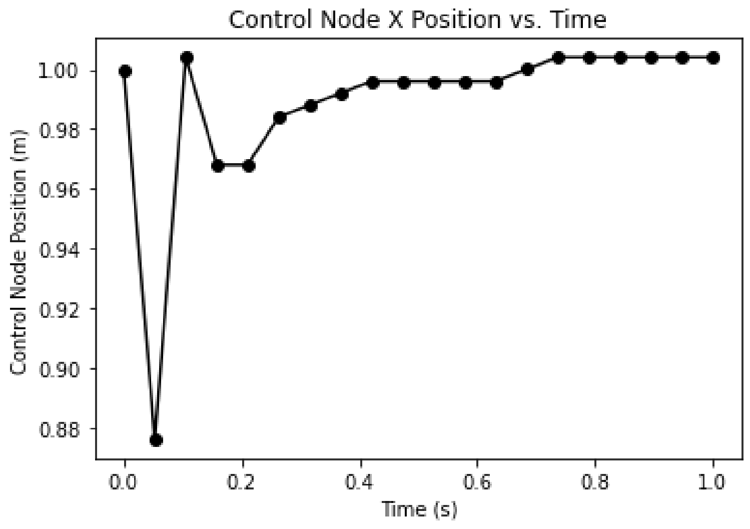
1. Set up initial guess of **{xc, yc, θc}** (from time step)
2. Solve the equilibrium position of the beam at the given time step due to the guess using the objective function (unmodified 2d Euler beam, implicit Euler Newton-Raphson solver from class).
   - Tack onto the objective function a simple equation that puts a Dirichlet constraint on the n-1 (node before the control node), taking **{xc, yc, θc}** to constrain the node based on the constraint equation provided in the homework. This recomputation of n-1 is done before running the objective function as an initial positional guess of the n-1 node
3. Measure midpoint error (the current midpoint versus the desired midpoint based on the provided trajectory)
4. Find Jacobian (J) mapping **{xc, yc, θc}** to the midpoint node. (This is done by "perturbing" the inputs to figure out how much the midpoint changes due to small changes in input, by a small, arbitrary quantity)
5. Solve dU = -pinv(J) given the midpoint error (pseudo inverse jacobian)
6. new guess of control node = old guess + du (this solves the inverse kinematics of the control node)
7. Repeat until the system finds a beam that matches the midpoint within a given tolerance, or it reaches the maximum iterations (set at 30)
8. Repeat for all points in time

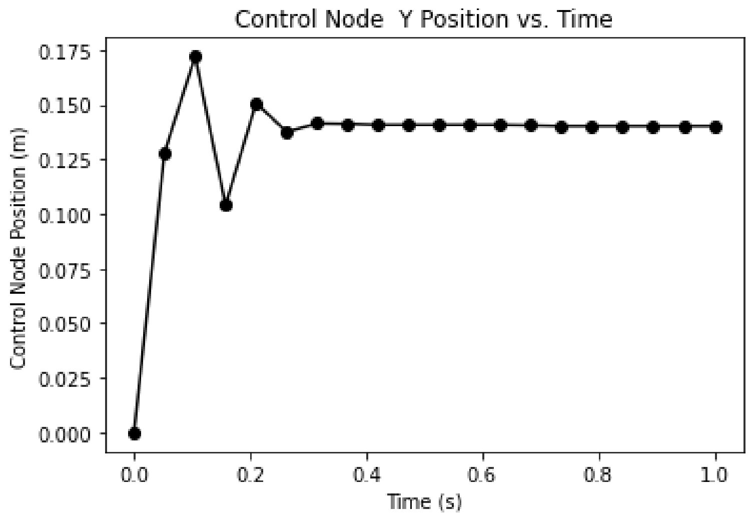
<u>Force/energy evaluation</u>

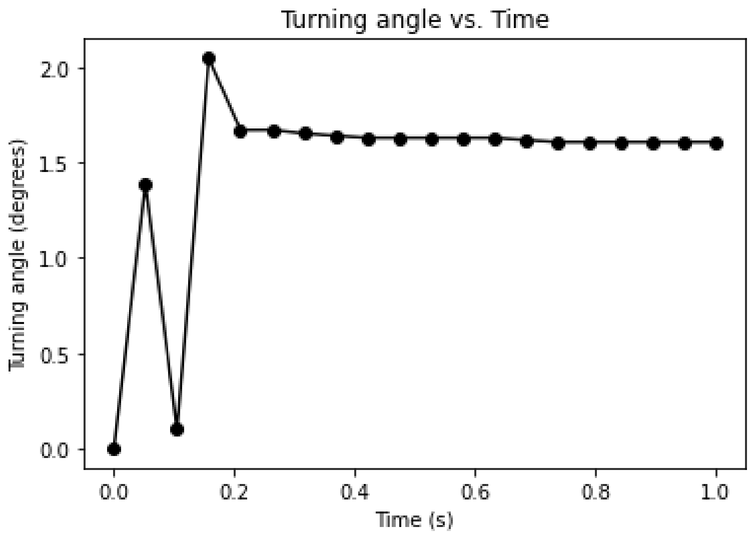
Unmodified 2d Euler beam, implicit Euler Newton-Raphson solver from class

Nathan Ge
905597306

Plots of the control inputs xc(t), yc(t), θc(t) over time.

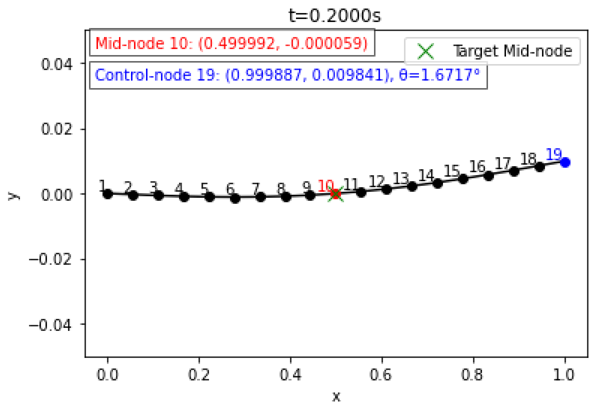


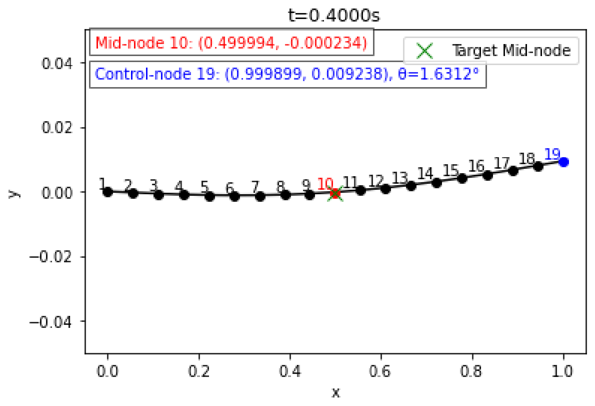


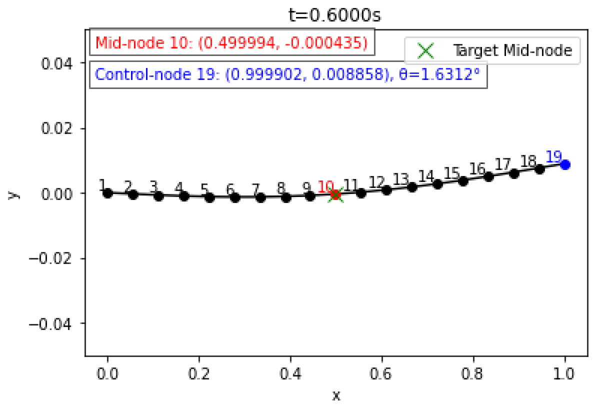


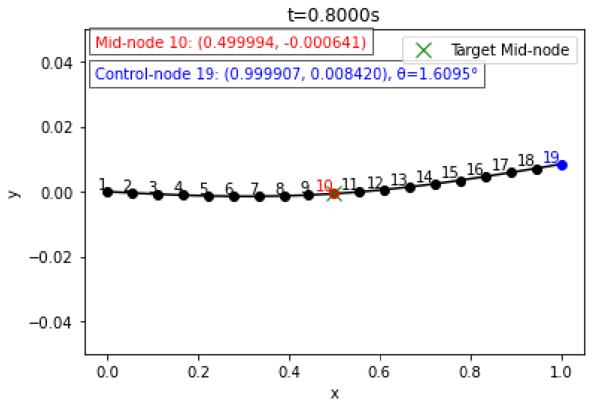
Nathan Ge
905597306

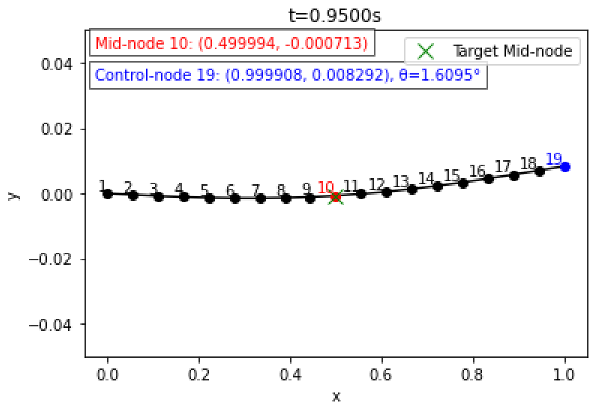
Five snapshots of the beam shape (node positions) during the motion.











Nathan Ge
905597306

A brief discussion of feasibility limits due to the robot's workspace and joint limits; explain how you handle infeasible commands (e.g., saturation or trajectory re-timing).

The algorithm inherently accounts for feasibility limits through the equilibrium solver and iterative control updates. At each time step, the solver computes the physically achievable configuration for the current control input, ensuring that joint and workspace limits are respected. The numerical Jacobian-based control updates provide small, incremental corrections toward the desired mid-node position, preventing commands that exceed actuation limits. In addition, control updates are clipped to further ensure numerical feasibility. If a desired target is outside the feasible workspace, the iterative loop may fail to converge, effectively saturating the control input at the nearest achievable configuration. Thus, feasibility is handled implicitly through the combination of the solver and iterative control strategy.
The algorithm is quasi-static, meaning it computes robot configurations assuming each control input reaches equilibrium before advancing in time. It does not account for dynamic effects or temporal constraints, and it is not phase-based or trajectory-retimed. Desired motions are applied at fixed time steps without adjustment for joint velocity, acceleration, or workspace limits.