

MAE 263F Homework 3

Feng Xu

I. PSEUDOCODE AND CODE STRUCTURE

Algorithm 1 Single step simulation of beam

```

1: procedure INITIALIZATION
2:   Load current nodes info (position  $q_0$  and velocity  $u_0$ )
3:   Define free DOFs
4:   Set Dirichlet constraints
5: end procedure
6: solve  $x \equiv q(t_{k+1})$  using Newton's Method
7: while  $err > \epsilon$  do
8:    $J(x), f(x) \leftarrow$  Inertia,  $\frac{\partial E^{stretch}}{\partial q}$ ,  $\frac{\partial E^{bend}}{\partial q}$ , External
      Force and their Jacobian or Hessian
9:    $\Delta x = J(x) \backslash f(x)$ 
10:   $x = x - \Delta x$  ▷ Only update free DOF
11:   $err = \text{norm}(f(x_{free}))$ 
12: end while
13:  $\dot{q}(t_{k+1}) = \frac{q(t_{k+1}) - q(t_k)}{\Delta t}$ 

```

Algorithm 2 MPC

```

1: procedure INITIALIZATION
2:   Load initial nodes info
3:   Define prediction horizon  $H$ 
4:   Set control input boundaries
5:   Init control input history
6: end procedure
7: procedure MPC OPTIMIZATION LOOP
8:   for  $k$  in steps do
9:     Set initial guess for the control inputs over  $H$ 
10:     $u_{k...k+H} = \text{argmin } J(q_k, u_{k-1}, u_{k-2}, u_{k...k+H})$ 
11:     $q_{k+1}, \dot{q}_{k+1} = \text{single step simulation}(q_k, \dot{q}_k, u_k)$ 
12:   end for
13: end procedure

```

A. Method description

I use the methods in class to do a single step simulation. The single step simulation function takes the current q and \dot{q} and the control input $u = (x_c, y_c, \theta_c)$ then predicts the next q and \dot{q} . This is formulated as

$$q_{k+1}, \dot{q}_{k+1} = S(q_k, \dot{q}_k, u_k) \quad (1)$$

During the single step simulation, bending energy and stretching energy and force are evaluated with the methods discussed in class using the equation $M\ddot{q} + \frac{\partial E^{stretch}}{\partial q} + \frac{\partial E^{bend}}{\partial q} - W = 0$ and use Newton's method to solve. Dirichlet constraints are enforced by making the

first two DOFs and last four DOFs as non-free DOF, so they are not updated in the simulation. $x_1(t_k)$ and $y_1(t_k)$ is always zero. $x_N(t_k) = x_c(t_k)$, $y_N(t_k) = y_c(t_k)$, $x_{N-1}(t_k) = x_c(t_k) - \Delta L \cos(\theta_c(t_k))$, $y_{N-1}(t_k) = y_c(t_k) - \Delta L \sin(\theta_c(t_k))$. The algorithm is shown in Algorithm 1.

For the control method, I use model predictive control(MPC). The problem is discretized with finite time steps for $t = 0 \dots 1000$. My step size $dt = 0.2s$. Within a predefined horizon, I try to minimize the objective function to optimize the control inputs to make the middle point of the beam follow the desired trajectory on each time step. The objective function is defined as

$$J(q_k, u_{k-1}, u_{k-2}, u_{k...k+H}) = \sum_{i=k}^{k+H} ||p_a - p_d|| + \alpha ||u_k - u_{k-1}|| + \beta ||a_k - a_{k-1}|| \quad (2)$$

where u_k is the control input $x_c(t_k), y_c(t_k), \theta_c(t_k)$, p_a is the middle point position from simulation result of q_k from Eq 1. p_d is the desired middle point position. $a_k = u_k - u_{k-1}$. α is the penalization parameter for high velocity of the control input and β is the penalization parameter for high acceleration of the control input.

I use `scipy.optimize.minimize` to solve this optimization problem `argmin J(q_k, u_{k-1}, u_{k-2}, u_{k...k+H})` to find the solution of $u_{k...k+H}$. The solution should be in the defined boundaries $x_c \in (0.2, 1.1)$, $y_c \in (-0.8, 0.1)$ and $\theta_c \in (-\frac{\pi}{2}, \frac{\pi}{2})$. Among $u_{k...k+H}$, the first solution u_k will be used for the actual control input and then do a single step simulation with Eq 1. The Algorithm is shown as Algorithm 2. After I get all the control inputs, I use average smooth to smooth out the inputs.

II. PLOTS OF THE CONTROL INPUTS OVER TIME

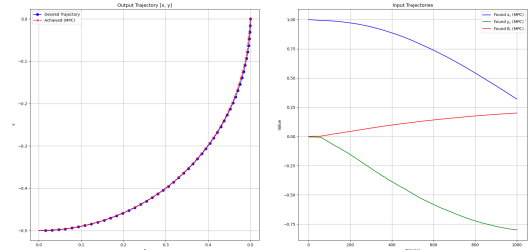


Fig. 1. Left: the trajectory of middle point. Right: the control inputs

III. SNAPSHOTS OF THE BEAM SHAPE DURING THE MOTION

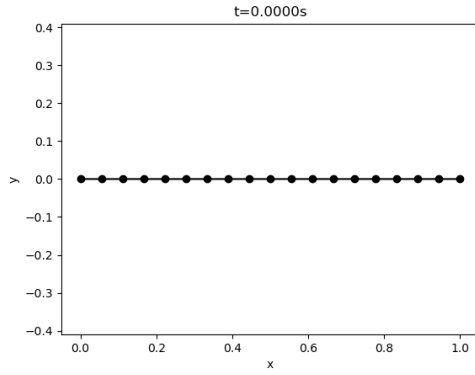


Fig. 2. Shape at t=0s.

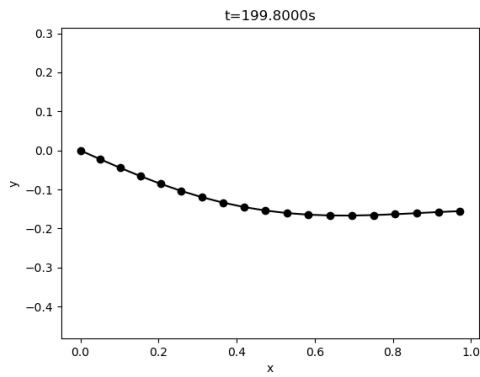


Fig. 3. Shape at t=199.8s.

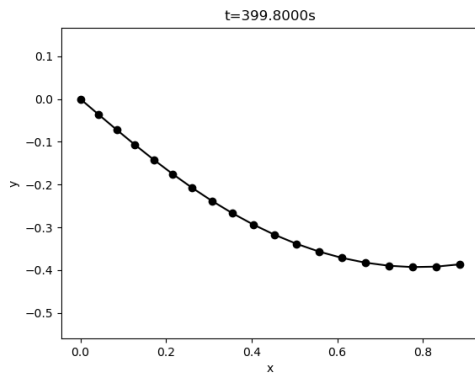


Fig. 4. Shape at t=399.8s.

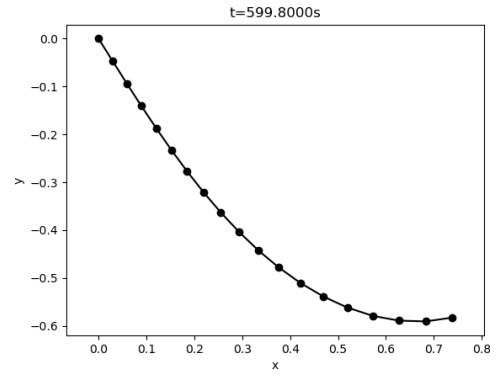


Fig. 5. Shape at t=599.8s.

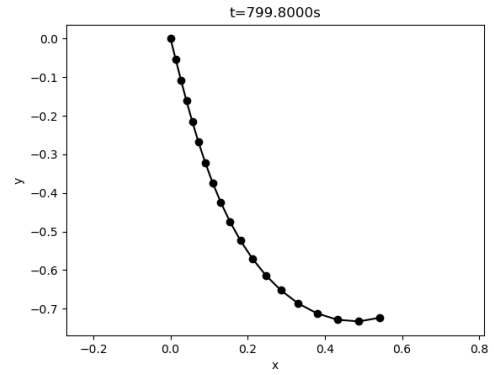


Fig. 6. Shape at t=799.8s.

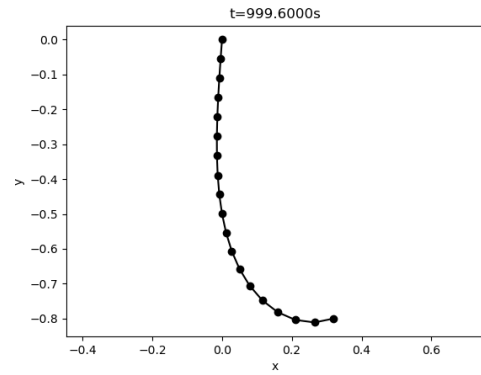


Fig. 7. Shape at t=999.6s.

input. So if the solution gives a high velocity or acceleration that is infeasible for the robot, I can make the penalization parameters higher to prevent it from going beyond the robots limitation.

IV. DISCUSSION OF FEASIBILITY LIMITS

For the workspace limitation, I use boundaries in the solver to make sure the solution(control input) cannot go beyond the robot's workspace. I also have the penalization parameters for high velocity and acceleration of the control