Dynamic Manipulation of *n*-Revolute-Joint Planar Manipulators

Peiguang Wang, Sichao Zhang, Geng Li

1. Problem Statement

In class project 5, our group pursues the task to develop a planning tool for an *n*-revolute joint planar manipulation systems in OMPL. The planning tool takes a variety of input parameters (joint number, *n*, link lengths and masses) specified by the users to compute the joint torques required to achieve a pose and to generate solutions to motion planning problems.

Our group consists of an electrical engineer and two mechanical engineers, all of whom have strong academic interest in robotic manipulation systems. Precision manipulation is of paramount importance for all robots and has always been a difficult challenge. It adds more complication to the challenge when there is the need to plan motions for the robots. Not only do the motion plans have to satisfy the user queries, but they also have to respect the differential (kinematic and dynamical) constraints. Planning the path while respecting the differential constraints is a two-point boundary value problem (BVP) which is computationally expensive to solve. Our group is intrigued by the challenge and is determined to find a solution to the problem. Over the course of the class, we have gained exposure to motion planners using random sampling. These planners can by-pass the costly 2-point BVP and are highly effective in handling planning with differential constraints. Our group is highly motivated by the prospect of incorporating precision dynamic manipulation into motion planning.

2. Methods

The dynamic manipulation of an *n*-revolute-joint planar robot involves solving ordinary differential equations (ODE) in the general form of:

$$\dot{q} = f(q, u)$$

where in the context of this problem,

$q = [q_1, q_2, \dots q_n, \dot{q}_1, \dot{q}_2, \dots \dot{q}_n]$

$\dot{q} = [\dot{q}_1, \dot{q}_2, \dots \dot{q}_n, \ddot{q}_1, \ddot{q}_2, \dots \ddot{q}_n]$

q_i, $\dot{q}_i$, $\ddot{q}_1$ are the joint angle, angular velocity and angular acceleration of the ith joint respectively

u, the input control to the ODE, in the context of this problem is the applied torque τ_i to each link

$u = [\tau_1, \tau_2 \dots \tau_n]$

More detailed ODEs are set up using the Lagrangian formulation of the dynamics model of the rigid manipulator.

$$\tau = H(q)\ddot{q} + h(q, \dot{q}) + B\dot{q} + g(q)$$

where

H is the inertia matrix

h is the vector of Coriolis, centrifugal forces

B is the viscous friction matrix. In the scope of this project, it is assumed the joints are frictionless, hence $B = 0$

g is the vector of gravity force

The governing ODE can now be written as:

$$\frac{d}{dt}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ H^{-1}(\tau - h - g) \end{bmatrix}$$

The above ODE have render the dynamic manipulation problem solvable by the ompl::control::ODESolver function. The state space of the *n*-revolute-joint planar robot is set up as the Cartesian product of *n* SO2 state space. In OMPL, it is expressed as SO2 + SO2 + … + SO2. However, in our implementation, we use realvectorSpace with bound (-pi – pi) to present the SO^n space. The control space of the robot is set up as the Cartesian product of R^n state space.

The RRT planner is chosen to perform the motion planning tasks. RRT planner is an effective tool to handle motion planning with differential constraints, as it utilizes random sampling to generate probable control set and use the probable control set to propagate tree states rapidly towards goal because of the goal bias. The effectiveness in dealing with differential constraints is due to the fact that the random sampling help by-pass the computationally costly solving of 2-point BVP. The planner is considered *Anytime* for its relative quickness to find the solution.

The comprehension of the underlying mathematics of the dynamics manipulation problem and the knowledge of these key attributes of RRT planner afford us the confidence about our approach to solve the problem.

A visualization tool is designed in MATLAB to provide graphical representation of the motion planning solution. It requires only the input of joint angles and link lengths by the users. The figure below is an example execution (unrelated to the experiments conducted in this study) of the visualization tool to illustrate the effectiveness of the visualization tool.

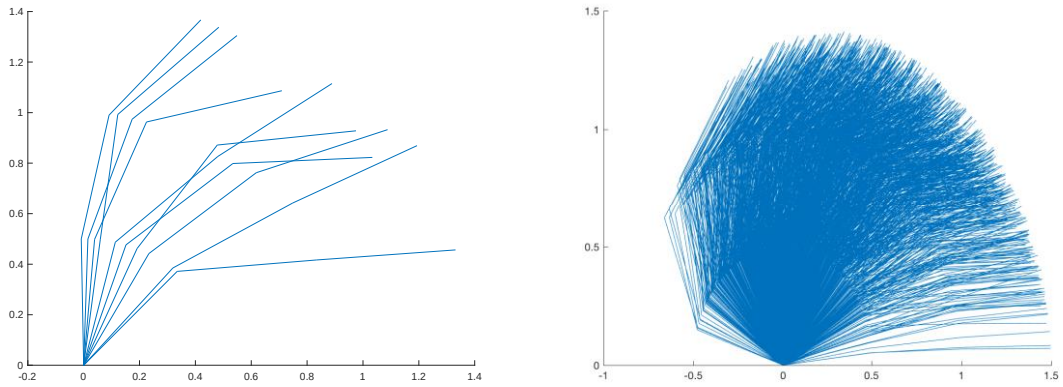

Figure 1. Visualization tool created in MATLAB. 10 random poses (left) and 1000 random poses for a 3-R planar robot with all link length equal to 0.5

3. Environments and Assumptions

We use two environments to evaluate the efficacy of our computational tool. The fixed end of the first link is placed at the origin of the workspace. The first testing environment is an obstacle-free space. In the second testing environment (Figure x), there is a rectangular obstacle whose vertices are [-0.5, -1.5], [0.0, -1.5], [0.0 -1.0], and [-0.5, -1.0]. Figure 2 shows the environment with one obstacle.
We made one assumption that the mass of each link is proportional to its link length. And we assume the mass of each link is distributed uniformly. Another assumption is that the joints are considered frictionless, the viscous friction matrix B in the governing ODE is set as zero.

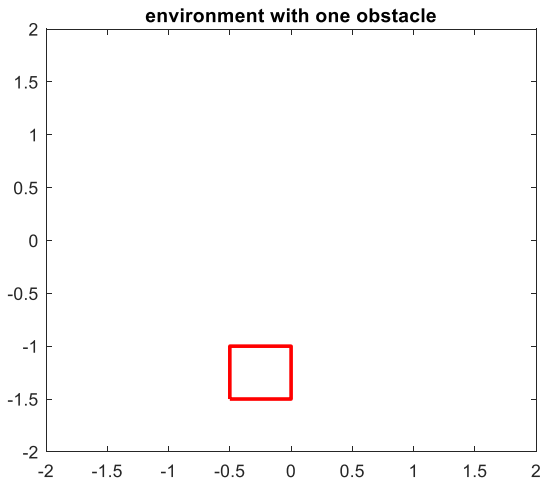

Figure 2. Environment with One Obstacle

4. Collision Checking

Collision Checking of the manipulator has two parts: checking with itself and checking with obstacles.

1) Check with the manipulator its self.

To check whether the manipulator is collide with itself, we need to check every link that whether it intersects with another link.

```
bool checkSelf(std::vector<Point2D> pts){
  int N = pts.size();
  bool result;
  for (int i = 0; i < N-1; ++i)
  {
     for (int j = 0; j < i; ++j)
     {
        result = isLineIntersect(pts[i],pts[i+1],pts[j],pts[j+1]);
        if(result == false){
           return false;
        }
     }
  }
  return true;
}
```

2) Check with the environment.

Checking with the environment is similar with the previous task. We have to check every link that whether it intersects with the obstacle's edge.

5. Experiments and Results

In the first experiment, the number of revolute joint(s), n is equal to 1. The single-link robotic arm effectively behaves as a pendulum. The objective of performing this simple experiment with the most basic form of manipulator is to validate our program on the most fundamental level and test the correctness of the simple pendulum environment. The result of the first experiment is shown in Figure 3 while the length

of the link is 1, the start state of the whole system is $q_1 = 0$, $\dot{q}_1 = 0$and the end state is $q_1 = 1$ and $\dot{q}_1 = 1$, besides, velocity limit and time limit are respectively 10 second and 60 second.

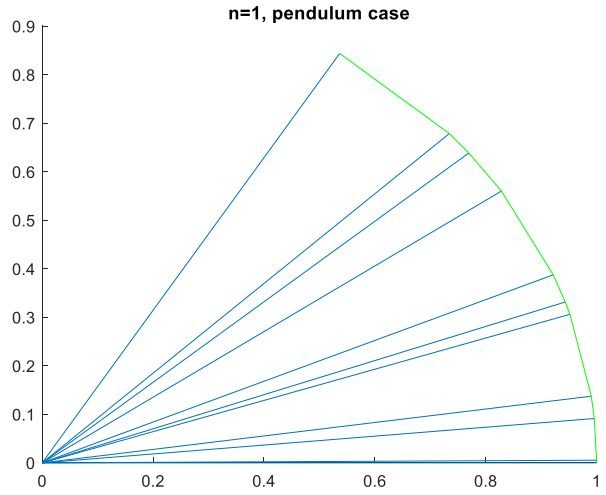

Figure 3. when n=1 in free space

In the second experiment, n is increased to 2, the link length of each link is 1, the starting pose is $q_1 = q_2 = 0$, initial angular velocities are $\dot{q}_1 = \dot{q}_2 = 0$. The angular velocities cannot exceed ±10 rad/s. The applied torque must be within ±100 unit force×length. The goal is for the end effector to reach and rest ($\dot{q}_1 = \dot{q}_2 = 0$) at goal pose [$q_1 = 3$ rad, $q_2 = 0$]. Our program has successfully performed motion planning for this experiment as illustrated in figure 4. The extension to this experiment is to conducting the experiment in testing environment 2 (obstacle present). Our planner returns an approximate solution as the 2-link manipulator has made obvious movement towards the goal state yet unable to reach the goal state when time runs out at t = 100 second. And this result is shown in figure 5.

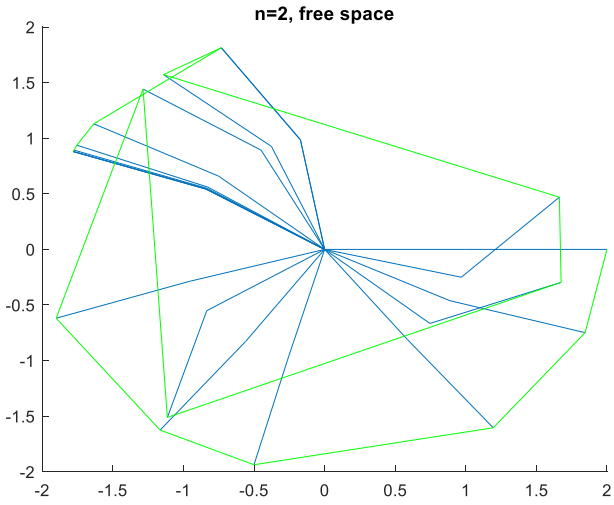

Figure 4. when n=2 in free space

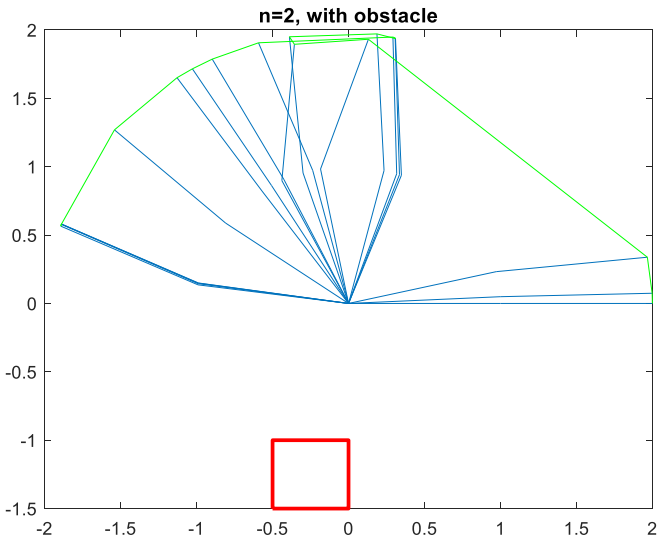

Figure 5. when n=2 in environment with obstacle

In the third experiment, we plan motion for a 3-R manipulator with link length of 0.3, 0.6 and 0.9 each. the starting pose is $q_1 = q_2 = q_3 = 0$, initial angular velocities are $\dot{q}_1 = \dot{q}_2 = \dot{q}_3 = 0$. The angular velocities cannot exceed ±10 rad/s. The applied torque must be within ±100 unit force×length. The goal is for the end effector to reach and rest ($\dot{q}_1 = \dot{q}_2 = \dot{q}_3 = 0$) at goal pose [$q_1 = 3$ rad, $q_2 = 0, q_3 = 0$]. We first test this scenario in free space when time limit is 100 second, and the result is shown in Figure 6. From this result we find that the manipulator is collision free by itself. However, the end effector fails to reach the goal within time limit. So we expand the time to 200 second.

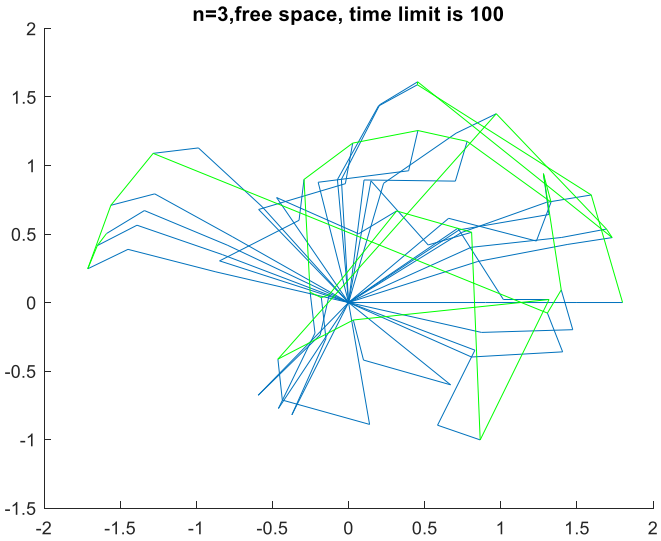

Figure 6. when n=3 in free space with time limit 100

As the time runs out at t = 200 second, the manipulator has made significant progress towards the goal state, but the planner is short of arriving exactly at goal pose, the result is shown in Figure 7.

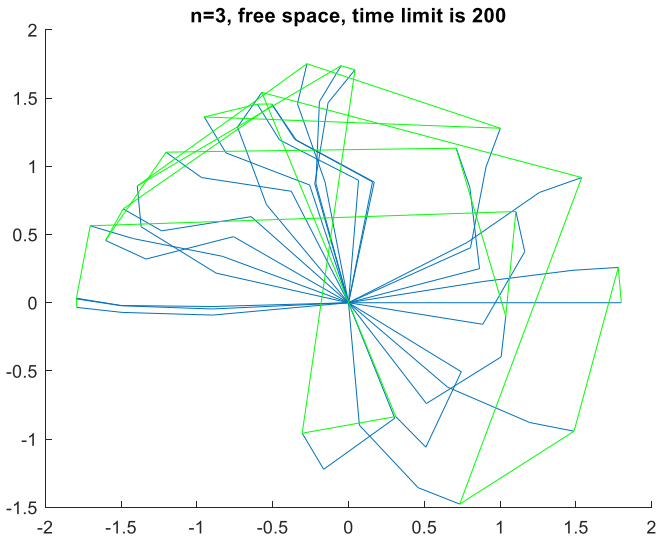

Figure 7. when n=3 in free space with time limit 200

Furthermore, we test the scenario when n=3 in environment with one obstacle, the end state of the experiment becomes $q_1 = 3\,rad, q_2 = q_3 = 0$ and $q_1 = q_2 = q_3 = 0$. The result is shown in Figure 8. As we can see the manipulator is collision free, however, we can see that although the manipulator does not intersect with the obstacle. The green line segment (trajectory of the end effector) does intersects with the obstacle. We think it is because that the torque limit is too large. And applying this huge torque at a small time the manipulator still travels a large distance.

We also did some experiments with smaller torque limit. However, when torque limit is set to 10, the time of the program grows very fast. Even if we set time limit of RRT to 200 seconds, the planner only samples one point.

We can conclude that the torque limit is a very important parameter, when torque limit is small, it takes longer time to find a solution. When torque limit is large, it may directly go over the obstacle.

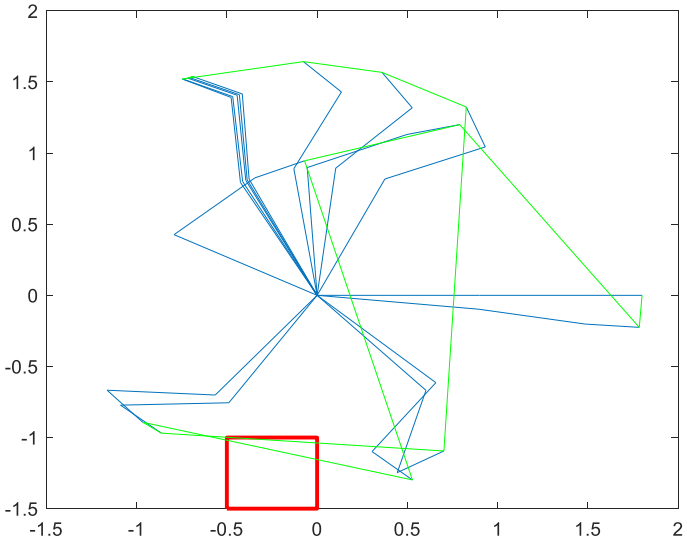

Figure 8. when n=3 in environment with one obstacle

Our observation is that as the number of revolute joints increase, the time required for the planner to comprehensively evaluate all the kinematic and dynamic constraints of all the propagated states increases exponentially. The running time increases rapidly as n becomes larger. When n = 1, i.e. the pendulum scenario, the planner finds solution in less than 1 second. However, when n increases to 2, it needs 60 seconds to find an approximate solution. It even took longer time for manipulator with 3 joints. (Nearly 200 seconds for a approximate solution. And we never get a complete solution.)

6. Analysis

Rate the difficulty of each exercise:
Manipulator ODESolver: 8
Collision Checking: 5
Experiments and Report: 6

Give an estimated time(hours) on each exercise:
Manipulator ODESolver: 15
Collision Checking: 1
Experiments and Report: 4
Hardest part of the assignment

We think the most difficult part of this assignment is transferring the equations of motion. We took several hours to read the paper and sort out the ODE for manipulator with n joints. It even took us longer to implement this ODE into algorithm. The h(q, q_dot) part of the ODE is the most difficult part, we worked together to figure out how to present the mathematical equation into algorithmic language.