Design of a motion planning algorithm for robot manipulators
Project Report: MEC 529

Sravana Sumanth Koneru
ID:112526331

Stony Brook University
May 17, 2020

Abstract

This report presents a general motion planning algorithm for a robot manipulator to move it's end effector from one pose to another. Then, we implement this algorithm on a **Baxter® robot arm** (by Think Robotics).
Motion planning is a term used in robotics is to find a sequence of valid configurations that moves the robot from the source to destination. In this project, we will study **inverse kinematics-based motion planning**. We first define the terminology used in Motion planning, such as Joint space, Task space etc. Then we define a problem statement and we produce a solution approach for the given problem. Also, the interpolation in SE(3) and Jacobian based motion planning are discussed in this report. The main motion planning is then discussed and is implemented and results are shown.
Keywords: *Motion Planning; Joint Space; Task Space; End effector Configuration, Unit Dual Quaternions.*

1 Introduction

This document presents **a inverse kinematics based motion planning algorithm** and applies it to a Baxter® robot arm. The Baxter Robot System is a human-sized humanoid robot with dual 7-degree-of-freedom (dof) arms with stationary pedestal, torso, and 2-dof head, a vision system, a robot control system, a safety system, and an optional gravity-offload controller and collision detection routine.
Motion planning problems are formulated to move the robots to perform tasks. For this, we define the following terms:

- **Task Space:** Set of all the poses(positions and orientations) the end effector can achieve. This is denoted by τ and $\tau \subset$ SE(3).
- **Joint Space:** Set of all joint angles(or displacements) of the manipulator. This is denoted by **J** and J $\subset \mathbb{R}^n$.
- **Environment:** Set of all points in three dimensional world that a robot and the other objects can occupy. We denote this by **W** and W $\subset \mathbb{R}^3$.

For a manipulator, tasks are defined in the end effector space of the robot. A **task** is a sequence of configurations in SE(3) that the robot must go through. The sequence of configurations may be time parameterized. Motion Planning problems can be formulated either in the end-effector space (aka task space, operational space), which is a subspace of SE(3), or the configuration space (aka joint space) of the manipulator. Both formulations and solutions has its own advantages and disadvantages. The output of a motion planning algorithm is a path (or a trajectory) in a joint space and consequently in task space. A path in the end effector space is a sequence of configurations in SE(3).

2 Problem Statement

Let the initial configuration of the end effector frame of the robot be g(0) = $g_o \in$ SE(3), and g_d be the final desired configuration.Let n the the degrees of freedom of the robot manipulator and the initial configuration of the manipulator be $\Theta(0) \in \mathbb{R}^n$ and θ_k be the angle of the kth joint.
Goal: Through the motion planning algorithm, we would like to obtain a sequence of joint angles Θ(i), i=0,...,m, such that FK(Θ(m))=g_d, where FK is the forward kinematics map for the manipulator.

3 Solution Approach

We know that, SE(3) is the Cartesian product of SO(3) and R3. We can use the individual parameterizations of SO(3) and $\mathbb{R}^3$ to form a parameterization of SE(3). However, for motion planning purposes, for a given initial and final configuration of the end effector in SE(3), we will need to use interpolation in SE(3) to form a path (trajectory) in SE(3). So, we use the **"Unit Dual Quaternion"** approach i.e., Dual Quaternion parameterization of SE(3) for the motion planning. For this we transform Transformation matrix to Unit dual Quaternion representation.

3.1 Interpolation in SE(3) using unit dual quaternion representation

For the interpolation between any two configuration we use **"Screw Linear Interpolation(ScLERP)"** scheme which is analogous to the straight line interpolation in euclidean space.
Let $A = A_r + \epsilon A_d$ and $B = B_r + \epsilon B_d$ be two unit dual quaternions corresponding to two rigid body configurations. To compute a curve in SE(3) that interpolates between the two configurations, let $\tau \in [0, 1]$ be the interpolation parameter. Therefore any point on a path between A and B is given by

$$C(\tau) = A \otimes (A^* \otimes B)^\tau \tag{1}$$

This scheme makes a simple way to plan motion between two configurations in SE(3). But, in manipulators, we have several links and a direct control over the end effector is not possible. The motion of end effector is determined by moving the joints and we need a motion plan in the joint space. We discuss the process of converting the motion plan in end effector space to motion plan in joint space in the next subsection.

3.2 Jacobian based Motion Planning Algorithm

Let p denote the position of the end effector and Q be the unit quaternion representation of the orientation of the end effector. We know that

$$\omega^s = 2J_1\dot{Q} \tag{2}$$

$$v^s = \dot{p} + 2\hat{p}J_1\dot{Q} \tag{3}$$

writing these equations in matrix form we obatin:

$$\begin{bmatrix} v^s \\ \omega^s \end{bmatrix} = J_2 \begin{bmatrix} \dot{p} \\ \dot{Q} \end{bmatrix} \tag{4}$$

where $J_2 = \begin{bmatrix} \mathbf{I} & 2\hat{p}J_1 \\ 0 & 2J_1 \end{bmatrix}$ and $J_1 = \begin{bmatrix} -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{bmatrix}$ and q_0, q_1, q_2, q_3 are the elements in the quaternion Q. We know that V $= \begin{bmatrix} v^s \\ \omega^s \end{bmatrix} = J^s\dot{\Theta}$, where $\dot{\Theta}$ is the vector of joint velocities and J^s is the spatial Jacobian of the manipulator. Therefore

$$\dot{\Theta} = (J^s)^T(J^s(J^s)^T)^{-1} \begin{bmatrix} \dot{p} \\ \dot{Q} \end{bmatrix} \tag{5}$$

Let $\dot{\gamma} = \begin{bmatrix} \dot{p} \\ \dot{Q} \end{bmatrix}$, and B $= (J^s)^T(J^s(J^s)^T)^{-1}J_2$, and we have

$$\dot{\Theta} = B\dot{\gamma} \tag{6}$$

Let $\Theta(t)$ and $\Theta(t + h)$ be the joint angles at time t and $t + h$ respectively, where h is a small time step. Using Euler time-step to discretize the Equation we get

$$\boxed{\Theta(t + h) = \Theta(t) + B(\Theta(t))(\gamma(t + h) - \gamma(t))} \tag{7}$$

3.3 The Inverse Kinematics based Motion Planning Algorithm

The algorithm is mainly divided into two parts:

- **Take a small step in the direction of the goal in task space(or SE(3)):** Use Unit Dual Quaternion Interpolation between current pose and the goal pose.
- **Compute the step in joint space that approximately corresponds to the small step in the task space:** Use the time-discretized velocity kinematics equation.

Input: Initial configuration of the manipulator ($\Theta(0)$), initial configuration of the end effector ($gst(0)$), Desired configuration of end effector (g_d).
Output: A sequence of nx1 joint angle vectors, $\Theta(1), \Theta(2), ..., \Theta(m)$

ALGORITHM:

1. Start at $t = 0$, $g(0)$ and $\Theta(0)$ are known.
2. Covert 4x4 homogeneous matrix $g(t)$ to 7x1 vector $\gamma(t)$ of concatenation of position and unit quaternion, and a unit dual quaternion representation $\mathbf{A}(t)$.
3. Compute $A(t+1) = A(t) \otimes (A^*(t) \otimes A_f)^\tau$, τ is the interpolation parameter $0 \leq \tau \leq 1$
4. Convert unit dual quaternion $A(t+1)$ to 7x1 concatenated vector $\gamma(t+1)$
5. **Compute:** $\Theta(t+1) = \Theta(t) + \beta B(\Theta(t))(\gamma(t+1) - \gamma(t))$
   where $\beta \leq 1$ is a step length parameter for how far one is moving from $\Theta(t)$. This parameter is set by trial and error method.
6. Compute $g(t+1) = FK(\Theta(t+1))$, **FK** is the forward kinematics map for the manipulator.
7. **Check:** if $\bar{g}(t+1)$ is **close enough** to g_f,
   if **Yes**: STOP, **else** continue to the next step.
8. **set**
   $t \leftarrow (t+1)$
   $g(t) \leftarrow \bar{g}(t+1)$
   $\Theta(t) \leftarrow \Theta(t+1)$
   Go to **Step-2**.

In **Step-7** we check for the closeness of one configuration to the other configuration and it is reminded that these configurations are points in SE(3). In general, if we have points in $\mathbb{R}^n$, we simply calculate the distance between the two points and check if it is within the specified tolerance to say whether the points are near enough. To do the same for points in SE(3), we should know how to compute distances in SE(3). Unfortunately, for SE(3), one cannot simply give an expression to define a bi-invariant metric. However, one can define metrics on $\mathbb{R}^3$ and SO(3). We will define two configurations to be identical when the distance between them are lower than a tolerance threshold both in the Euclidean distance and metric on SO(3).
To compute the distance between the position part of two configurations the cartesian distance is

used. The Euclidean distance between two points, $P_1 = (x_1; y_1; z_1)$ and $P_2 = (x_2; y_2; z_2)$ is given by:

$$\boxed{d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}} \tag{8}$$

The distance between two rotations is given as the Euclidean distance between two unit quaternions. The distance metric ϕ is a mapping from the unit quaternion space to $\mathbb{R}$ and is given by:

$$\boxed{\phi(q_1, q_2) = min\{\|q_1 - q_2\|, \|q_1 + q_2\|\}} \tag{9}$$

where, q_1, q_2 are unit quaternion representation of the two rotations and $\|.\|$ represents Euclidean norm.
So, to check the closeness of the two configurations, we should consider both the conditions given above.

4 Results

The above mentioned algorithm is now implemented on a Baxter robot arm. Baxter® is a collaborative robot from Rethink. As mentioned earlier, baxter robot arm is a 7 Degree of freedom manipulator with 7 Links.The Baxter® robot arms have seven rotational joints in each arm and these are named S0, S1, E0, E1, W0, W1, W2 starting from the shoulder respectively are shown in Figure 1. The link lengths of the Baxter robot arm are shown in Figure 2.

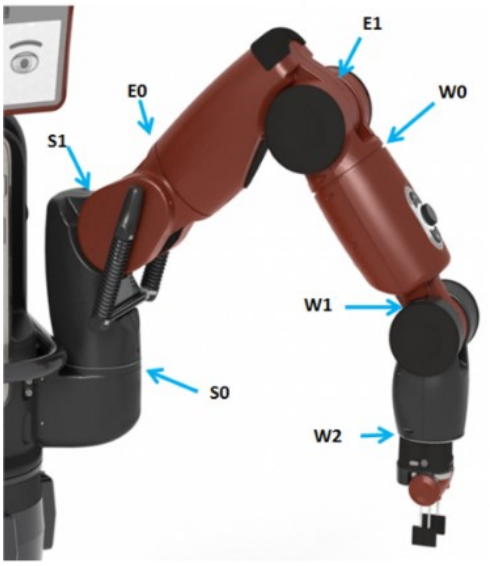

Figure 1: Joints in the Baxter robot Arm

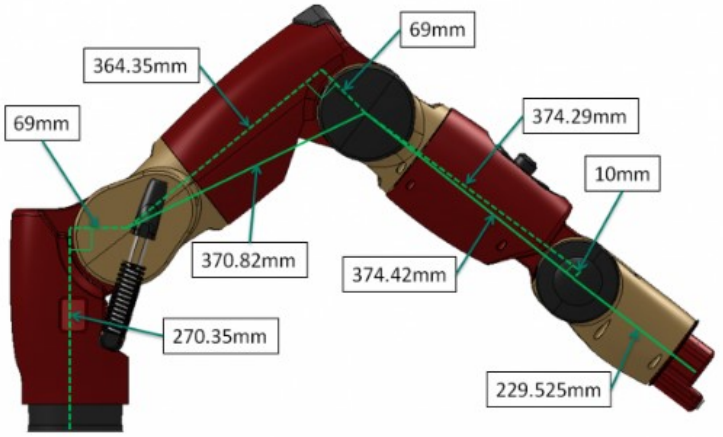

Figure 2: Link Lengths in Baxter Robot Arm

The axes of the baxter arm are given by $\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \\ \omega_5 \\ \omega_6 \\ \omega_7 \end{bmatrix} = \begin{bmatrix} [1,0,0] \\ [\frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0] \\ [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0] \\ \omega_2 \\ \omega_3 \\ \omega_2 \\ \omega_3 \end{bmatrix}$

4.1 Implementation

The implementation of the above algorithm is done in MATLAB®. For the implementation of the above algorithm, we take a random initial configuration and a random Desired configuration (gd). (Collision avoidance constraints and Task constraints are not dealt in this project). To take a random configuration in the beginning, we impose the forward kinematics map on the manipulator. Suppose, we take the joint angles of all the angles as $\pi/3$ for initial configuration. By applying $FK(\pi/3)$ we find the initial configuration as:

$g(t) = \begin{bmatrix} -0.4881 & -0.8727 & -0.0163 & -0.0068 \\ -0.1422 & 0.0979 & -0.9850 & 1.2868 \\ 0.8611 & -0.4784 & -0.1719 & 0.0528 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and say, gd= $\begin{bmatrix} 0.7071 & 0 & 0.7071 & 0.25 \\ 0 & 1 & 0 & 0.8044 \\ -0.7071 & 0 & 0.7071 & 0.7297 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Now, the inverse kinematics based motion planning algorithm is applied on the baxter arm with gt as initial position to reach desired configuration,gd. The output gives us a 7xn matrix with each column being a joint configuration $\Theta(t)$. The path the end effector follows is plotted. It is shown below (Figure 3):

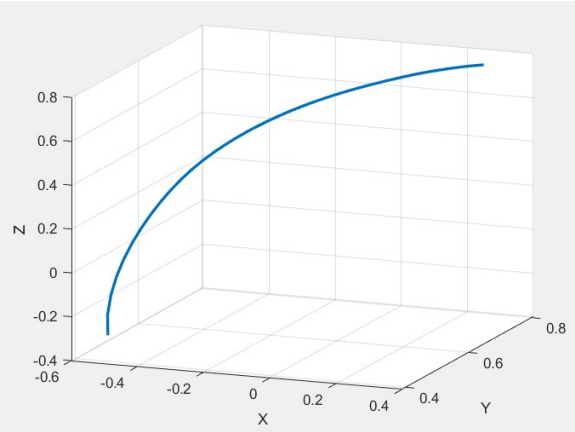

Figure 3: Obtained Path of the end effector - 1

To check the working of the code we verify that for another configuration. So, for this we choose gt and gd as following:

$g(t) = \begin{bmatrix} -0.5403 & -0.8080 & -0.2350 & -0.5977 \\ -0.2791 & 0.4355 & -0.8558 & 0.6037 \\ 0.7939 & -0.3968 & -0.4608 & -0.2519 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and say, $gd = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.2096 \\ 0 & 0 & 1 & 0.2499 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ The path the end effector follows is plotted. It is shown below (Figure 4):

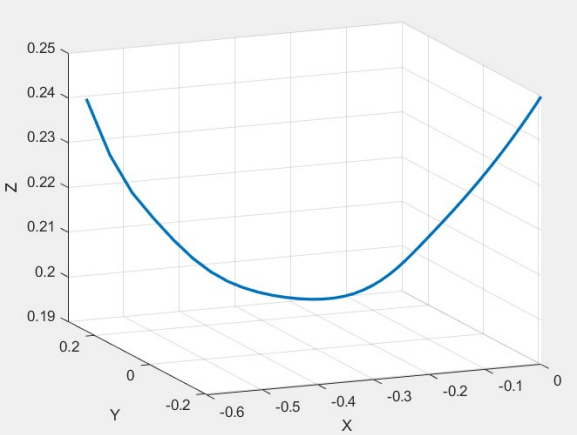

Figure 4: Obtained Path of the end effector - 2

4.2 Validation of the Algorithm

To prove that the code is valid and the output obtained is correct, We take the final set of joint angles, $\Theta(t)$ and apply the forward kinematics mapping ($FK(\Theta(t))$) and check if the obtained configuration is within the tolerance limit of the given desired configuration. The validation results for both the above discussed cases (with tolerance = 0.01m) discussed above are shown below:

```
Desired Config:
    0.7071         0    0.7071    0.2500
         0    1.0000         0    0.8044
   -0.7071         0    0.7071    0.7297
         0         0         0    1.0000
```

(a) Given desired configuration

```
Obtained Config from motion planning:
    0.7077   -0.0016    0.7065    0.2497
    0.0017    1.0000    0.0007    0.8043
   -0.7065    0.0007    0.7077    0.7297
         0         0         0    1.0000
```

(b) Obtained Configuration

Figure 5: Implementation of Motion planning - 1

```
Desired Config:
    1.0000         0         0         0
         0    1.0000         0   -0.2100
         0         0    1.0000    0.2500
         0         0         0    1.0000
```

(a) Given desired configuration

```
Obtained Config from motion planning:
    1.0000   -0.0001   -0.0008   -0.0004
    0.0001    1.0000   -0.0018   -0.2096
    0.0008    0.0018    1.0000    0.2499
         0         0         0    1.0000
```

(b) Obtained Configuration

Figure 6: Implementation of Motion planning - 2

From the above obtained and given configurations of the end effector of the arm, we can verify that the method and the procedure followed is true and is valid.

5 Conclusion

This report discusses about a design of a motion planning algorithm for a manipulator and then implements it on a Baxter® robot arm. The motion planning algorithm given in this report is **inverse kinematics based motion planning algorithm**. The provided algorithm is then verified and validated against various configurations. This algorithm successfully provides a motion planning path in joint space. As a future improvement over this project, the Collision avoidance constraints and Task constraints can be dealt by making necessary changes and additions to the existing algorithm. Much interesting work can be done using these constraints on the arm for making the arm do different types of tasks in the environment.

6 Appendix

The MATLAB® function for the mentioned motion planning algorithm.

```
%This function implements the motion planning algorithm for a manipulator
%Input: gst0 = reference configuration,
%       gt   = First configuration,
%       gf   = Desired configuration,
%       tol  = tolerance for the closeness check,
%       w    = axes,
%       q    = point on the axis,
% Joint_Type = type of joint,
%      Theta = Joint Angles,.
%OUTPUT: M = a matrix with each column as a set of joint angles

function M = Motion_Planning(gst0,gt,gf,tol,w,q_axis,Joint_Type,Theta)
tau = 0.5;
counter = 1;

while 1

for i = 1:3
    for j = 1:3
        R(i,j) = gt(i,j);
    end
end
for i = 1:3
    p(i) = gt(i,4);
end

Phat = [0     -p(3)  p(2);...
```

```
        p(3)   0    -p(1);...
       -p(2)   p(1)   0];
% unit quaternion from the rotation matrix
Q = Rot_to_Quat(R);
q0 = Q(1);
q1 = Q(2);
q2 = Q(3);
q3 = Q(4);
J1 = [-q1   q0   q3 -q2;...
      -q2 -q3   q0   q1;...
      -q3   q2 -q1   q0];
%coverting transformation matrix g(t) to Unit Dual Quaternion A(t)
At = Transform_to_DualQuat(gt);
%coverting Unit dual Quaternion A(t) to Gamma gamma(t)
gammat = DualQuat_to_gamma(At);
%coverting transformation matrix of final pose g(f) to Unit Dual Quaternion A(f)
Af = Transform_to_DualQuat(gf);

Atcon = DualQuat_Conj(At);
Prodtemp1 = DualQuat_Prod(Atcon,Af);
Prodtemp = DualQuat_Power(Prodtemp1,tau);
%interpolating it to the next step
Atplusone = DualQuat_Prod(At,Prodtemp);
gammatplusone = DualQuat_to_gamma(Atplusone);
%spatial Jacobian
Js = SpatialmanipJac(w, q_axis ,Joint_Type,Theta);
J2 = [eye(3) 2*Phat*J1; zeros(3,3) 2*J1];
B = Js'*inv(Js*Js')*J2;
beta = 0.1;
%compute theta(t+1)
theta1 = Theta + ((beta*B)*(gammatplusone-gammat));
%applying the forward kinematics map to the new joint angles
gst1 = manipdkin(gst0, w, q_axis, Joint_Type, theta1);

gt = gst1;
Theta = theta1;
M(:,counter) = Theta;
check = closeness_check(gst1,gf,tol);
%checking for the closeness for the desired and the obtained configuration
if check == 1
   break
end
counter = counter+1;
end
end
```