Manipulator Jacobian based on the elementary transform sequence

Peter Corke

January 2015 (updated December 2018)

1 Introduction

The Denavit and Hartenberg notation for describing a serial-link mechanism geometry is a fundamental tool of the roboticist. Once a manipulator's kinematics is parameterized in this form a large body of standard algorithms and code implementations for kinematics, dynamics, motion planning and simulation are available[1].

The fundamentals of serial-link robot kinematics and the Denavit-Hartenberg[2] notation are well covered in standard texts[3, 4]. Each link is represented by two parameters: the *link length*, a_i, and *link twist*, α_i, which define the relative location of the two attached joint axes in space. The classical method of assigning Denavit-Hartenberg parameters is to systematically assign a coordinate frame to each link, but there are significant constraints on each frame, and also the coordinate system of the base and the end-effector. An interesting complication with the Denavit-Hartenberg notation is the zero-angle configuration, that is, the pose when all joint angles are zero. For the Puma robot this is a non-obvious 'L-shaped' pose with the upper arm horizontal and the lower arm vertically upward.

Corke[5] introduced an alternative representation of the kinematics for a serial-link manipulator: the elementary transform sequence (ETS). This intuitive but systematic approach allows the forward kinematics to be written down by inspection using a simple "walk through" procedure. The result is a string of elementary translations and rotations, from the user defined base coordinate to the end-effector. For example, the ETS for a Puma 560 robot is

$$T_z(L_1)R_z(q_1)R_y(q_2)T_y(L_2)T_z(L_3)R_y(q_3)T_x(L_6)T_y(L_4)T_z(L_5)R_z(q_4)R_y(q_5)R_z(q_6)T_z(L_7)$$

This readable string describes motion along the joint and link chain in terms of translations and rotations about the canonic axes. The arguments of the transformations are constants or generalized joint coordinates. Unlike the Denavit-Hartenberg approach a joint can involve a rotation or translation about *any* axis – note in this example the term $R_y(q_2)$ is a rotation about the y-axis.

The paper[5] goes on to describe an algebraic procedure that can be automatically applied to an ETS sequence to factorize it into the standard or modified Denavit-Hartenberg form. Joint angle offsets, if required by the chosen zero-angle configuration chosen, are generated automatically, as are base and tool transformations. This algorithm is provided in the Robotics Toolbox for MATLAB® by the function `DHFactor` which is implemented in Java.

Denavit-Hartenberg parameters are still required to determine the manipulator Jacobian and the inverse dynamics. This article extends [5] by describing an algorithm to determine the manipulator Jacobian matrix directly from the ETS representation.

2 ETS-based forward kinematics

We will first recap on the forward kinematics, the pose of the robot's end-effector given a set of joint coordinates.

With ETS represent the forward kinematics of a serial-link robot manipulator by a sequence of M elementary transformations $\boldsymbol{E}_i \in SE(3)$

$$\boldsymbol{T}_E = \boldsymbol{E}_1 \boldsymbol{E}_2 \boldsymbol{E}_3 \cdots \boldsymbol{E}_M \tag{1}$$

Each factor is either a translation along, or a rotation about, one of either the x-, y- or z-axis. That is, it is one of six elementary transformations

$$\boldsymbol{E}_i = \begin{cases} \boldsymbol{T}_x(\eta_i) \\ \boldsymbol{T}_y(\eta_i) \\ \boldsymbol{T}_z(\eta_i) \\ \boldsymbol{R}_x(\eta_i) \\ \boldsymbol{R}_y(\eta_i) \\ \boldsymbol{R}_z(\eta_i) \end{cases}$$

where the parameter is either a constant (a translational offset or a rotation) or a generalized joint coordinate

$$\eta_i = \begin{cases} c_i \\ q_j \end{cases} .$$

The forward kinematics is simply the cumulative product of the elementary transforms with the joint coordinates plugged in as appropriate. There is no need to factorize it first into Denavit-Hartenberg parameters.

Using MATLAB® we can take a conventional toolbox Denavit-Hartenberg-based model and convert it to ETS format[1]

```
>> mdl_puma560
>> s = p560.trchain
s =
    'Rz(q1)Rx(90)Rz(q2)Tx(0.4318)Rz(q3)Tz(0.15005)Tx(0.0203)Rx(-90)Rz(q4)
    Tz(0.4318)Rx(90)Rz(q5)Rx(-90)Rz(q6)'
```

where translations are in units of metres and angles are in degrees. We can convert this human-readable string into a sequence of ETS object

```
>> Te = ETS(s)
Te =
Rz(q1)Rx(90)Rz(q2)Tx(0.4318)Rz(q3)Tz(0.15005)Tx(0.0203)Rx(-90)Rz(q4)Tz(0.4318)
  Rx(90)Rz(q5)Rx(-90)Rz(q6)

>> about Te
Te [ETS] : 1x14 (112 bytes)
```

which is a 14-element row vector of `ETS` objects. We can look at individual elements in this sequence, for example

```
>> Te(3)
ans =
Rz(q2)
>> Te(1:3)
ans =
Rz(q1)Rx(90)Rz(q2)
```

and the objects support a number of methods to determine the parameters, joint angle if any and so on.

We can *evaluate* this string for a particular set of joint coordinates

```
>> Te.eval(qz)
ans =
    1.0000         0         0    0.4521
         0    1.0000         0   -0.1500
         0         0    1.0000    0.4318
         0         0         0    1.0000
```

which agrees with the classical Denavit-Hartenberg-based forward kinematics

[1]This string is different to the one shown above because the link coordinate frames have been assigned using Denavit-Hartenberg constraints.


```
>> p560.fkine(qz)
ans =
         1         0         0    0.4521
         0         1         0     -0.15
         0         0         1    0.4318
         0         0         0         1
```

3 ETS-based manipulator Jacobian

The time derivative of the end-effector pose is

$$\dot{\boldsymbol{T}}_E = \frac{d\boldsymbol{T}_E}{dt} \in \mathbb{R}^{4\times 4}$$

We are interested in the change of the end-effector pose due to a change in the j'th joint so we introduce the joint velocity

$$\frac{d\boldsymbol{T}_E}{dt} = \frac{d\boldsymbol{T}_E}{dq_j}\frac{dq_j}{dt} = \frac{d\boldsymbol{T}_E}{dq_j}\dot{q}_j \tag{2}$$

The matrix derivative on the left-hand-side of (2) has 16 elements – it is a non-compact representation of the end-effector spatial velocity which is commonly represented by $(\boldsymbol{v}, \omega) \in \mathbb{R}^6$ where $\boldsymbol{v}$ is the end-effector translational velocity and ω is the end-effector rotational velocity. If we write the end-effector pose as

$$\boldsymbol{T}_E = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \in \mathbf{SE}(3) \\ 0\,0\,0 & 1 \end{pmatrix}$$

where $\boldsymbol{R} \in \mathbf{SO}(3)$ is an orthonormal rotation matrix representing the orientation of the end-effector and $\boldsymbol{t} \in \mathbb{R}^3$ is the translation of the end-effector then the time derivative is

$$\dot{\boldsymbol{T}} = \begin{pmatrix} \dot{\boldsymbol{R}} & \dot{\boldsymbol{t}} \\ 0\,0\,0 & 0 \end{pmatrix} \tag{3}$$

The translational velocity is simply

$$\boldsymbol{v} = \dot{\boldsymbol{t}}$$

Angular velocity is related to the rate of change of a rotation matrix by the well known identity[1]

$$\dot{\boldsymbol{R}} = [\omega]_\times \boldsymbol{R}$$

where $[\cdot]_\times : \mathbb{R}^3 \mapsto \mathbf{so}(3)$ is a skew-symmetric matrix and we can express the angular velocity as

$$\omega = \vee_\times \left(\dot{\boldsymbol{R}}\boldsymbol{R}^T\right)$$

$\frac{d\boldsymbol{T}_x(q_j)}{dq_j} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\frac{d\boldsymbol{R}_x(q_j)}{dq_j} = \begin{pmatrix} & & 0 \\ [1,0,0]_\times\, \boldsymbol{R}_x(q_j) & & 0 \\ & & 0 \\ 0\,0\,0 & & 0 \end{pmatrix}$
$\frac{d\boldsymbol{T}_y(q_j)}{dq_j} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\frac{d\boldsymbol{R}_y(q_j)}{dq_j} = \begin{pmatrix} & & 0 \\ [0,1,0]_\times\, \boldsymbol{R}_y(q_j) & & 0 \\ & & 0 \\ 0\,0\,0 & & 0 \end{pmatrix}$
$\frac{d\boldsymbol{T}_z(q_j)}{dq_j} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\frac{d\boldsymbol{R}_z(q_j)}{dq_j} = \begin{pmatrix} & & 0 \\ [0,0,1]_\times\, \boldsymbol{R}_z(q_j) & & 0 \\ & & 0 \\ 0\,0\,0 & & 0 \end{pmatrix}$

Table 1: Derivative of primitive transforms with respect to joint coordinate q_j.

where $\vee_\times(\cdot) : \mathbf{so}(3) \mapsto \mathbb{R}^3$ is the inverse operator to $[\cdot]_\times$. Now we can write

$$\begin{pmatrix} \boldsymbol{v} \\ \omega \end{pmatrix} = \begin{pmatrix} \dot{\boldsymbol{t}} \\ \vee_\times\left(\dot{\boldsymbol{R}}\boldsymbol{R}^T\right) \end{pmatrix} = \vee(\dot{\boldsymbol{T}}_E) = \vee(\frac{d\boldsymbol{T}_E}{dq_j})\dot{q}_j$$

where $\vee(\cdot) : \mathbf{se}(3) \mapsto \mathbb{R}^6$.

To determine how motion of the j'th joint affects the end-effector pose we can apply the chain rule to (1)

$$\begin{aligned} \frac{d\boldsymbol{T}}{dq_j} &= \frac{d\boldsymbol{E}_1}{dq_j}\boldsymbol{E}_2\boldsymbol{E}_3\cdots\boldsymbol{E}_M \\ &+ \boldsymbol{E}_1\frac{d\boldsymbol{E}_2}{dq_j}\boldsymbol{E}_3\cdots\boldsymbol{E}_M \\ &+ \boldsymbol{E}_1\boldsymbol{E}_2\frac{d\boldsymbol{E}_3}{dq_j}\cdots\boldsymbol{E}_M \\ &\vdots \\ &+ \boldsymbol{E}_1\boldsymbol{E}_2\cdots\frac{d\boldsymbol{E}_M}{dq_j} \end{aligned}$$

where all the product terms will be zero except for the one where $\boldsymbol{E}_i$ is a function of the joint coordinate q_j. If this is the k'th term in (1) then we can rewrite (1) as

$$\boldsymbol{T} = \left(\prod_{i=1}^{k-1}\boldsymbol{E}_i\right)\boldsymbol{E}_k\left(\prod_{i=k+1}^{M}\boldsymbol{E}_i\right)$$

and the derivative with respect to q_j is

$$\frac{d\boldsymbol{T}}{dq_j} = \left(\prod_{i=1}^{k-1} \boldsymbol{E}_i \right) \frac{d\boldsymbol{E}_k}{dq_j} \left(\prod_{i=k+1}^{M} \boldsymbol{E}_i \right) \tag{4}$$

The middle derivative term is easily computed and the value for all possible cases of $\boldsymbol{E}_k$ are summarised in Table 1.

The end-effector motion is the sum of the components due to each of the joints

$$\begin{pmatrix} \boldsymbol{v} \\ \omega \end{pmatrix} = \vee(\frac{d\boldsymbol{T}_E}{dq_1})\dot{q}_1 + \cdots + \vee(\frac{d\boldsymbol{T}_E}{dq_N})\dot{q}_N \tag{5}$$

which can be written in matrix form as

$$= \left(\vee(\tfrac{d\boldsymbol{T}_E}{dq_1}) \cdots \vee(\tfrac{d\boldsymbol{T}_E}{dq_N}) \right) \begin{pmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_N \end{pmatrix} \tag{6}$$

$$= \boldsymbol{J}\dot{\boldsymbol{q}} \tag{7}$$

where $\boldsymbol{J}$ is the manipulator Jacobian – the end-effector spatial velocity in the world coordinate frame. The algorithm is shown succinctly in pseudocode in Figure 1.

Continuing the MATLAB example from above

```
>> Te.jacobian(qz)
ans =
    0.1500   -0.4318   -0.4318         0         0         0
    0.4521         0         0         0         0         0
         0    0.4521    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000         0         0    1.0000         0    1.0000
```

which agrees with the classical Denavit-Hartenberg-based algorithm[6]

```
>> p560.jacob0(qz)
ans =
    0.1500   -0.4318   -0.4318         0         0         0
    0.4521    0.0000    0.0000         0         0         0
         0    0.4521    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000
```


for j = 1 to N **do**
$k \leftarrow i$ such that $\eta_i = q_j$
$\dot{\boldsymbol{T}} = \prod_{i=1}^{k-1} \boldsymbol{E}_i \cdot D\left\{\boldsymbol{E}_k\right\} \cdot \prod_{i=k+1}^{M} \boldsymbol{E}_i$
$\boldsymbol{v} = \dot{\boldsymbol{T}}[1 \rightarrow 3,\, 4]$;
$\dot{\boldsymbol{R}} = \dot{\boldsymbol{T}}[1 \rightarrow 3,\, 1 \rightarrow 3]$;
$\omega = \mathrm{vex}\left(\dot{\boldsymbol{R}}\boldsymbol{R}^T\right)$
$\boldsymbol{J}\,[1 \rightarrow 6\ ,\, \mathrm{j}] = \left(\boldsymbol{v},\ \omega\right)^T$
end for

Figure 1: ETS Jacobian algorithm

4 Conclusion

Given a robot kinematic model expressed as an elementary transform sequence (ETS), we have shown how to derive the Jacobian matrix. This, combined with a standard minimization algorithm, is sufficient to perform numerical inverse kinematics without the nee for Denavit-Hartenberg parameters.

References

[1] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, second ed., 2017. ISBN 978-3-319-54412-0.

[2] R. S. Hartenberg and J. Denavit, "A kinematic notation for lower pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 77, pp. 215–221, June 1955.

[3] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley and Sons, 2006.

[4] R. P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*. Cambridge, Massachusetts: MIT Press, 1981.

[5] P. I. Corke, "A simple and systematic approach to assigning Denavit-Hartenberg parameters," *IEEE Transactions on Robotics*, vol. 23, pp. 590–594, June 2007.

[6] R. P. Paul, B. Shimano, and G. E. Mayer, "Differential kinematic control equations for simple manipulators," *IEEE Trans. Syst. Man Cybern.*, vol. 11, pp. 456–460, June 1981.

