RESEARCH ARTICLE

A new implementation for online calculation of manipulator Jacobian

Pramod Chembrammel1*, Thenkurussi Kesavadas1,2

1 Health Care Engineering Systems Center, University of Illinois at Urbana-Champaign, Urbana, Illinois, United States of America, 2 Industrial and Enterprise Systems Engineering Dept., University of Illinois at Urbana-Champaign, Urbana, Illinois, United States of America

* pramodch@illinois.edu

Abstract

This paper describes a new implementation for calculating Jacobian and its time derivative for robot manipulators in real-time. The estimation of Jacobian is the key in the real-time implementation of kinematics and dynamics of complex planar or spatial robots with fixed as well as floating axes in which the Jacobian form changes with the structure. The proposed method is suitable for such implementations. The new method is based on matrix differential calculus. Unlike the conventional methods, which are based on screw theory, the Jacobian calculation in the proposed approach has been reduced to the inner product of two matrices. Use of the new method to derive linear and angular velocity parts of Jacobian and its time derivative is described in detail. We have demonstrated the method using a two-DOF spatial robot and a hyper-redundant spatial robot.

1 Introduction

The agility of a robot depends on how fast it can adapt to an environment. The dynamics of a robot required for such control primarily involves calculation of Jacobian. It relates joint rates to end-effector velocities and relates end-effector forces to joint torques. Also, the columns of the Jacobian are the instantaneous directions in which a desired point on a robot can move [1]. Thus, Jacobian is the key to the analysis and control of robots. Various methods of calculating the Jacobian with a fixed number of links have been explored and successfully demonstrated in [2–4] and in numerous other publications on robot kinematics and dynamics.

Current methods [2, 5] deal with fixed-configuration of the robot in which the Jacobian is calculated a priori with respect to a fixed set of points on the robot. In some cases, Jacobian needs to be calculated with respect to points that vary within a local coordinate system. Such points arise in the case of fixed as well as reconfigurable robots. In the case of reconfigurable robots, these points are generated because of the change in configuration (e.g. change in the configuration of a walking robot or addition of intermediate links to a serial manipulator), whereas in the case of fixed-configuration robots, the variation is either due to floating axes [6] or due to external influences. Such variations require that dynamic control be exercised with respect to the varying points, which demand recursive and real-time calculations. Conventional methods of finding Jacobian fail in such cases, these methods involve manual...
reformulation of the Jacobian when it changes form for the above reasons. In this paper, we propose a new method for the real-time calculation and reformulation of Jacobian. This method can be applied to fixed as well as reconfigurable robots. The scope of this paper is limited to fixed-configuration. The number of matrix operations is $2n + n/(2\delta)$ (see Section 6) as compared to $(3n - 6)$ of the widely used Renaud’s method [5]. The performance of the proposed method is comparable to that of Renaud’s method for low DOF ($n < 8$) robots and is superior for high-DOF robots (see Section 6). In terms of reformulation, Renaud’s method cannot be compared to the proposed method, since Renaud’s method involves manual reformulation if the Jacobian changes form.

Our method also permits the autonomous online computation of the time derivative, $\dot{J}$, of Jacobian, which is required for the robot dynamics as well as to resolve kinematic redundancy using optimal control techniques [7]. $\dot{J}$ is also required for the estimation of link positions from joint angle measurements (proprioception) using optimal estimation techniques like unscented Kalman filters and particle filters [8–10]. Symbolic computation of Jacobian and its derivative is detailed in [11], but it requires extensive manual intervention and hence is not an autonomous approach. In addition, we are not interested in a symbolic derivation, but in a numerical implementation that can autonomously estimate the Jacobian and its time derivative. Authors could not find any available literature on systematic and autonomous derivation of $\dot{J}$.

2 Related work on Jacobian estimation

Conventional methods use either the loop closure method [1, 2, 4, 12] or screw theory [2, 5] to calculate Jacobian. The main disadvantage of the loop-closure method is that although it is very useful for planar and spatial mechanisms with a few degrees of freedom (DOF), it is not suitable for complex spatial mechanisms with high DOF and linkages with lower pair joints [13]. Screw-theory-based methods are useful in such cases, but fail to address situations with floating axes of the joints, as demonstrated in [6, 14]. All the reported works efforts have dwelt on finding Jacobian with respect to only one point, i.e., the end-effector. However, there are many situations in which the Jacobian has to be calculated with respect to many points on the robot, which may also vary with time. These methods do not perform automated calculation of the Jacobian matrix and require manual intervention before final solution can be found. That is the main obstacle to real-time implementation.

2.1 Comparison of existing methods

This section compares the computational efficiencies of different existing methods based on screw theory. There are six such methods, as discussed in [5]. Although the methods use the same basic concepts, they yield different forms of the Jacobian matrix. The methods work on the premise of finding the angular ($\omega$) and linear ($v$) velocities of the desired point on the manipulator by summing the joint rates.

$$\omega = \sum_{i=1}^{n} \dot{q}_i \mathbf{z}_{i-1}$$

$$v = -\sum_{i=1}^{n} (\dot{q}_i \mathbf{z}_{i-1}) \times \mathbf{p}_{i-1}$$

where $(\cdot) \times (\cdot)$ is a cross product, $\mathbf{p}_{i-1}$ is the coordinate of the desired point in the local coordinate system, and all other $\mathbf{p}$s represent the coordinates of the origins of the link coordinate...
systems with respect to the preceding coordinate system. These methods differ only in the way
the coefficients are extracted out of Eqs (1) and (2) to form the Jacobian. Among the methods
discussed in [5], the first three approaches differ from the fourth and fifth in that their computa-
tions begin from opposite ends of a multi-link robot. The first three methods begin from the
base, while the others begin from the end-effector. Among them, the fourth method, developed
by Renaud [15], is computationally superior to the other methods with \((30n - 87)\) multiplica-
tions, \((15n - 66)\) additions and subtractions, and \((2n - 2)\) sines and cosines. Here, \(n\) is
the number of degrees of freedom. The number of matrix operations \((3n - 6)\) is also the least for
this method [5]. Although these methods are very efficient in calculating the Jacobian, they
require manual intervention to select the axis of rotation or translation \((z_{i-1} \text{ in Eq (1)})\),
depending on the type of joint variable. That renders the existing methods incapable of finding
the form of Jacobian in real-time, both for re-configurable and for fixed-configuration robots
in which the Jacobian with respect to a new point has to be found.

Automatic estimation of Jacobian via zeroing dynamics [16] while the kinematic structure
(DH parameters) of a robot is not known, is explained in [17]. In this method, the Jacobian is
iteratively estimated by tracking the kinematics of the end-effector using feedback from exter-
nal sensors like cameras or inertial measurement units (IMUs). However, external feedback
makes this approach less attractive for most of the applications for which external tracking is
either not required or not possible. In addition, this method is not applicable for the regular
(nonredundant) robots; it is applicable only for redundant robots.

In this paper, we propose a new method that facilitates estimation of Jacobian in real-time.
Like the loop-closure method, the proposed method is founded on matrix differential calculus;
however, the procedure involved in finding the Jacobian is different from those of the loop-
closure and screw-theory-based methods. Also, it does not distinguish between joints, either
prismatic or revolute. It is therefore possible to use numerical methods for the real-time esti-
mation of Jacobian.

This paper is laid out as following. In section 3, proposed method is described. In section 4,
we discuss how we demonstrated it using a simple two-link planar manipulator. The method
is summarized as an algorithm in section 5. In section 6 we explain how we demonstrated the
real-time application of the method with the help of a redundant spatial robotic arm.

3 The new method
A point, \(p\), in a coordinate system, \(j\), attached to a link of a robot is represented in another
coordinate frame, \(i\), or the base coordinate frame, through the use of homogeneous transfor-
mation.

\[
x^i(q) = A(q)p^j
\]  

(3)

where \(A\) is the homogeneous transformation matrix, and \(x^i\) is the point after transformation.
The superscripts represent the respective coordinate systems. The matrix \(A\) is a function of the
generalized coordinates, \(q\), of the robot thus making \(x^i\) also a function of \(q\).

The velocity of the transformed point in the \(i^{th}\) coordinate system is given by

\[
\dot{x}^i(q) = J(q, p)\dot{q}
\]  

(4)

where \(J\) is the Jacobian of the robot with respect to the point \(p^j\). \(\dot{x}^i\) has both global linear veloc-
ity, \((\dot{v} = \dot{x}^i)\), and angular velocity, \((\dot{\omega})\). As is evident from [Eq (4)], the Jacobian relates the joint
rates to the velocities of the desired point \(p^j\) represented in frame \(i\). In the transformation of
Eqs (3) to (4), the coordinates of the point \(p\) in the coordinate system \(j\) have been absorbed
into the \(J\). Thus, if the Jacobian is proved to be a function of independent entities, i.e. as a
function of the derivative of the transformation matrix $A(q)$ and the constant vector $p$, then it is only sufficient to perform simple matrix multiplications to find out the Jacobian. This is the key to real-time estimation of $J(q)$.

$J$ has two parts; a relative velocity part of dimension $(3 \times n)$, and an angular velocity part of dimension $(3 \times n)$ [2]. The following subsections explain the real-time estimation of both parts of $J$. (It is to be noted that $x$ and $\omega$ are represented in $i$ and that $p$ is represented in $j$. Hence, superscripts are ignored in the following sections for conciseness).

### 3.1 Calculation of linear velocity part

The velocity equation in Eq (4) is derived as follows. Taking the time derivative of Eq (3),

$$\dot{x}(q) = A(q)p$$  (5)

The time derivative of $p$ is not present in Eq (5) since it is a constant vector in the local coordinate system, $j$. The time derivative of $A(q)$ is written as

$$\dot{A}(q) = \sum_{i=1}^{n} \frac{\partial A(q_i)}{\partial q_i} \dot{q}_i$$  (6)

Using Eq (6) in Eq (5),

$$\dot{x}(q) = \left[ \sum_{i=1}^{n} \frac{\partial A(q_i)}{\partial q_i} \right] \dot{q}_i$$  (7)

Since the vector $p$ is common and $\dot{q}_i$ is a scalar, Eq (7) is rewritten as

$$\dot{x}(q) = \sum_{i=1}^{n} \left[ \frac{\partial A(q_i)}{\partial q_i} p \right] \dot{q}_i$$  (8)

In matrix notation, Eq (8) is represented by

$$\dot{x}(q) = J_v(q) \dot{q}$$  (9)

where, $J_v(q)$ is the linear velocity part of $J(q)$. What is left to perform is to decompose $J_v(q)$ as a product of independent matrices. From Eq (8), $J_v(q)$ is given by

$$J_v(q) = \left[ \frac{\partial A(q_1)}{\partial q_1} p \quad \ldots \quad \frac{\partial A(q_i)}{\partial q_i} p \quad \ldots \quad \frac{\partial A(q_n)}{\partial q_n} p \right]$$  (10)

Eq (10) is written as the product of two matrices:

$$J_v(q) = LP$$  (11)

where

$$L = \left[ \frac{\partial A(q_1)}{\partial q_1} \ldots \frac{\partial A(q_i)}{\partial q_i} \ldots \frac{\partial A(q_n)}{\partial q_n} \right]$$  (12)

and $P = I_n \otimes p$, where $I_n$ is an $(n \times n)$ identity matrix, $p$ is in homogeneous form, and $(.) \otimes (.)$ is the Kronecker product.

$$p = [x \ y \ z \ 1]^T$$  (13)

The dimensions of $J_v$, $L$, and $P$ are, respectively, $(4 \times n)$, $(4 \times 4n)$, and $(4n \times n)$. Thus, the linear
velocity part of the Jacobian is

\[ J_v(q) = (LP)_{(1,3-1,n)} \]  \hspace{1cm} (14) \]

It is to be noted that \( P \) is a sparse matrix and hence can efficiently store values.

### 3.2 Calculation of angular velocity part

The matrix \( A \) in Eq (3) is also called the transition matrix from the initial value to the final value \([2, 19]\), and is written as

\[ x(q,t) = A(q(t))x(q,0) \]  \hspace{1cm} (15) \]

Taking the time derivative of Eq (15),

\[ \dot{x}(q,t) = \dot{A}(q(t))x(q,0) \]  \hspace{1cm} (16) \]

Since \( A \) is invertible, from Eq (15),

\[ \dot{x}(q,t) = \dot{A}(q(t))A(q(t))^{-1}x(q,t) \]  \hspace{1cm} (17) \]

But,

\[ \dot{A}(q(t))A((q(t))^{-1} = \begin{bmatrix} R & \dot{r}^T \\ 0 & 0 \end{bmatrix} \begin{bmatrix} R^T & -R^T\dot{r}^T \\ 0 & 1 \end{bmatrix} \]  \hspace{1cm} (18) \]

where \( R \) is the \((3 \times 3)\) orientation part of \( A \), and \( \dot{r}^T \) is a \((3 \times 1)\) translation part of \( A \). Eq (18) is written as

\[ \dot{A}(q(t))A(q(t))^{-1} = \begin{bmatrix} RR^T & \dot{r}^T - RR^T\dot{r}^T \\ 0 & 0 \end{bmatrix} \]  \hspace{1cm} (19) \]

The element \( \dot{R}R^T \) of Eq (19) is called the twist matrix \([2]\) which is the skew-symmetric form of the angular velocity vector \([20]\) of the desired link. Similar to Eq (8), the left hand side of Eq (19) is written using (6) as

\[ \dot{A}(q(t))A(q(t))^{-1} = \sum_{i=1}^{n} \frac{\partial A(q(t))}{\partial q_i} A(q(t))^{-1} \dot{q}_i \]  \hspace{1cm} (20) \]

The right hand side of Eq (20) is thus written as

\[ \dot{A}(q(t))A(q(t))^{-1} = \sum_{i=1}^{n} LP_i \dot{q}_i \]  \hspace{1cm} (21) \]

where \( L \) is given by Eq (12) and \( P \equiv A^{-1} \). The subscript \( i \) on the right hand side represents the \( i^{th} \) \((4 \times 4)\) block of the matrix \( L \). The first \((3 \times 3)\) matrix of \( LP \) represents the twist matrix for each link. Thus, the angular velocity part of the Jacobian is

\[ J_{aw} = [(LP)_{(3,2)} \ (LP)_{(1,3)} \ (LP)_{(2,1)}]^T \]  \hspace{1cm} (22) \]

The subscript \( i \) on the left hand side of Eq (22) represents the column index of the manipulator Jacobian. The subscripts \((3, 2), (1, 3), \text{and} (2, 1)\) of the twist matrix from \( LP \) represent the \( x, \ y, \text{and} \ z \) components of the angular velocity part of the Jacobian. To use the standard notation, the
complete manipulator Jacobian is thus

$$J = \begin{bmatrix} J_v \\ J_o \end{bmatrix} \quad (23)$$

From Eqs (12), (14), and (22), it is observed that the Jacobian is affected by any variation in the point \( p \). With this method, unlike the conventional methods, this variation can be easily incorporated into the calculation of the Jacobian just by recalculating \( P = L_n \otimes p \). This method has applications in fixed-configuration robots with varying points.

### 3.3 Time derivative, \( \dot{J} \), of Jacobian

In order to find out the time derivative of Jacobian, we consider the linear and velocity parts separately because \( P \) is different in the two cases. The total time derivative of the linear part is

$$\dot{J}_v = \sum_{i=1}^{n} \frac{\partial J_v}{\partial q_i} \dot{q}_i \quad (24)$$

Since \( P \) is a constant from Eq (13),

$$\frac{\partial J_v}{\partial q_i} = \frac{\partial L}{\partial q_i} P \quad (25)$$

Similarly, the total time derivative of the angular part is

$$\dot{J}_o = \sum_{i=1}^{n} \frac{\partial J_o}{\partial q_i} \dot{q}_i \quad (26)$$

Using Eq (22),

$$\frac{\partial J_o}{\partial q_i} = \frac{\partial}{\partial q_i} \left( [(L_P)_{(3,2)} (L_P)_{(1,3)} (L_P)_{(2,1)}]^T \right) \quad (27)$$

From Eq (21), \( P \), in this case, is not a constant. Therefore,

$$\frac{\partial (L_P)}{\partial q_j} = \frac{\partial L}{\partial q_j} P + L \frac{\partial P}{\partial q_j} \quad (28)$$

But,

$$\frac{\partial P}{\partial q_j} = -A^{-1} \frac{\partial A}{\partial q_j} A^{-1} \quad (29)$$

where, \( L_i \) is the \( i^{th} \) \((4 \times 4)\) block of \( L \). Eq (29) is a standard matrix relationship [21]. By using Eq (29) in Eq (28), we obtain

$$\frac{\partial (L_P)}{\partial q_j} = \left( \frac{\partial L}{\partial q_j} - L_i A^{-1} L_j \right) A^{-1} \quad (30)$$

### 4 Demonstration

In this section, the concepts discussed in the previous section are demonstrated using a two-DOF spatial manipulator shown in Fig 1.
Considering an arbitrary point \((x_e, y_e, z_e)\) in the local coordinate frame attached to the second link:

\[
x(q) = \begin{bmatrix}
-S\theta_1 C\theta_2 & C\theta_1 & -S\theta_1 S\theta_2 & l_1 C\theta_1 - l_2 S\theta_1 C\theta_2 \\
-C\theta_1 C\theta_2 & S\theta_1 & C\theta_1 S\theta_2 & l_1 S\theta_1 + l_2 C\theta_1 C\theta_2 \\
S\theta_2 & 0 & -C\theta_2 & l_2 S\theta_2 \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x_e \\
y_e \\
z_e
\end{bmatrix}
\]  

(31)

where \(C\) stands for cosine and \(S\) for sine. The transformation matrix in the above equation is recursively calculated using a homogeneous transformation matrix [18]. Taking the time derivative of Eq (31),

\[
\dot{x}(q, \dot{q}) = \dot{J}X_e
\]  

(32)

where

\[
\dot{J} = J_1 \dot{q}_1 + J_2 \dot{q}_2
\]  

(33)

\[
X_e = [x_e, y_e, z_e, 1]^T
\]  

(34)

\[
J_1 = \begin{bmatrix}
-C\theta_1 C\theta_2 & -S\theta_1 & -C\theta_1 S\theta_2 & l_1 S\theta_1 - l_2 S\theta_1 C\theta_2 \\
-S\theta_1 C\theta_2 & C\theta_1 & -S\theta_1 S\theta_2 & l_1 C\theta_1 - l_2 S\theta_1 C\theta_2 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]  

(35)
and

\[
J_2 = \begin{bmatrix}
S\theta_1 S\theta_2 & 0 & -S\theta_1 C\theta_2 & L_2 S\theta_1 S\theta_2 \\
-S\theta_1 C\theta_2 & 0 & C\theta_1 C\theta_2 & -L_2 C\theta_1 S\theta_2 \\
C\theta_2 & 0 & S\theta_2 & L_2 C\theta_2 \\
0 & 0 & 0 & 0
\end{bmatrix}
\] (36)

Expanding Eq (32) using Eq (33),

\[
\dot{x}(q, \dot{q}) = [J_1 \dot{q}_1 + J_2 \dot{q}_2]X_e
\] (37)

Since \(X_e\) is common, rearranging Eq (37),

\[
\dot{x}(q, \dot{q}) = J_1 \dot{q}_1 X_e + J_2 \dot{q}_2 X_e
\] (38)

In the matrix form,

\[
\dot{x}(q, \dot{q}) = [J_1, J_2] \begin{bmatrix} I_2 \otimes X_e \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}
\] (39)

In Eq (39), \(X_e\) can be factored out to obtain

\[
\dot{x}(q, \dot{q}) = [J_1, J_2] (I_2 \otimes X_e) \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}
\] (40)

If we compare Eq (40) with the standard form of the velocity equation, \(\dot{x} = J_1 \dot{q}\), the linear velocity part of the Jacobian for the two-link manipulator is,

\[
J_v = [J_1, J_2] (I_2 \otimes X_e)
\] (41)

Eq (41) is the same as Eq (11), in which \(L = [J_1, J_2]\) and \(P = (I_2 \otimes X_e)\). If the end-effector is considered the point of interest, then \(X_e = [0, 0, 0, 1]^T\). Thus, \(J_v\) for the two-DOF spatial manipulator is written as

\[
J_v = \begin{bmatrix}
-S\theta_1 C\theta_2 & C\theta_1 C\theta_2 & S\theta_2 & -L_2 \\
C\theta_2 & S\theta_2 & 0 & -L_1 \\
-S\theta_1 S\theta_2 & C\theta_1 S\theta_2 & -C\theta_2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (42)

As given in Eq (14), the first three rows represent the linear part of the Jacobian. Therefore,

\[
J_v = \begin{bmatrix}
-S\theta_1 C\theta_2 & C\theta_1 C\theta_2 & S\theta_2 & -L_2 \\
C\theta_2 & S\theta_2 & 0 & -L_1 \\
-S\theta_1 S\theta_2 & C\theta_1 S\theta_2 & -C\theta_2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (43)

The angular velocity part is obtained as follows. \(P\) is

\[
P = A^{-1} = \begin{bmatrix}
-S\theta_1 C\theta_2 & C\theta_1 C\theta_2 & S\theta_2 & -L_2 \\
C\theta_2 & S\theta_2 & 0 & -L_1 \\
-S\theta_1 S\theta_2 & C\theta_1 S\theta_2 & -C\theta_2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (44)
Using Eqs (35) & (36), and following the expression in Eq (22), the block matrix multiplication (LP) will yield,

$$\mathbf{J}_o = \begin{bmatrix} 0 & C \theta_1 \\ 0 & S \theta_1 \\ 1 & 0 \end{bmatrix}$$  \hspace{1cm} (45)

From Eqs (43) and (45), the complete Jacobian is

$$\mathbf{J} = \begin{bmatrix} -l_1 S \theta_1 - l_2 C \theta_1 C \theta_2 & l_2 S \theta_1 S \theta_2 \\ l_1 C \theta_1 - l_2 S \theta_1 C \theta_2 & -l_2 C \theta_1 S \theta_2 \\ 0 & l_2 C \theta_2 \\ 0 & C \theta_1 \\ 0 & S \theta_1 \\ 1 & 0 \end{bmatrix}$$  \hspace{1cm} (46)

Equation Eq (46) is a standard result. The time derivative \( \dot{\mathbf{J}} \) is found as follows.

$$\dot{\mathbf{J}} = \left( \frac{\partial \mathbf{L}}{\partial \theta_1} \right) \dot{\theta}_1 + \left( \frac{\partial \mathbf{L}}{\partial \theta_2} \right) \dot{\theta}_2$$ \hspace{1cm} (47)

where

$$\frac{\partial \mathbf{L}}{\partial \theta_1} \mathbf{P} = \begin{bmatrix} -l_1 C \theta_1 + l_2 S \theta_1 C \theta_2 & l_2 S \theta_1 S \theta_2 \\ -l_1 S \theta_1 - l_2 C \theta_1 C \theta_2 & l_2 S \theta_1 S \theta_2 \\ 0 & 0 \end{bmatrix}$$ \hspace{1cm} (48)

$$\frac{\partial \mathbf{L}}{\partial \theta_2} \mathbf{P} = \begin{bmatrix} l_2 C \theta_1 S \theta_2 & l_2 S \theta_1 C \theta_2 \\ l_2 S \theta_1 S \theta_2 & -l_2 C \theta_1 C \theta_2 \\ l_2 C \theta_2 & -l_2 S \theta_2 \end{bmatrix}$$ \hspace{1cm} (49)

Using Eq (30),

$$\frac{\partial \mathbf{L}}{\partial \theta_1} \mathbf{P} = \frac{\partial \mathbf{L}}{\partial \theta_2} \mathbf{P} = \frac{\partial \mathbf{L}}{\partial \theta_3} \mathbf{P} = \mathbf{0}_{4 \times 4}$$ \hspace{1cm} (50)

And,

$$\frac{\partial \mathbf{L}}{\partial \theta_1} \mathbf{P} = \begin{bmatrix} 0 & 0 & C \theta_1 & 0 \\ 0 & 0 & S \theta_1 & 0 \\ -C \theta_1 & -S \theta_1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$ \hspace{1cm} (51)

Following Eq (27),

$$\frac{\partial \mathbf{J}_o}{\partial \theta_1} = \begin{bmatrix} 0 & -S \theta_1 \\ 0 & C \theta_1 \\ 0 & 0 \end{bmatrix}, \frac{\partial \mathbf{J}_o}{\partial \theta_2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$ \hspace{1cm} (52)
Using Eq (26),

\[
J_\omega = \begin{bmatrix}
0 & -\Theta_1 \\
0 & C\Theta_1 \\
0 & 0
\end{bmatrix} \dot{\Theta}_1
\]  

(53)

The time derivative of the complete Jacobian is,

\[
\dot{J} = \begin{bmatrix}
\dot{J}_L \\
\dot{J}_\omega
\end{bmatrix}
\]  

(54)

For a two-DOF robot the formulation may appear to involve more steps than a conventional method. Computationally, there is no clear advantage in using this method for two- or three-link robots (see section 6). However, the expressions in Eqs (35), (36) and (41) can be autonomously calculated using differential calculus. That procedure is also suitable for the real-time calculation of Jacobian with respect to a point other than the chosen point at the end-effector. Also, it is a very useful method for mechanisms with floating axes and robots with high DOF. In prior work, we have very effectively used this method for floating axes to model the dynamics of lower limbs \( n = 21 \) in human walking [14], as well as for modelling flexible guide-wire dynamics [22]. The application for high-DOF robots is explained in section 6 with respect to the real-time implementation of a real 7-DOF spatial robot.

5 Algorithm to find Jacobian

The method discussed in the previous sections can be summarized in the form of an algorithm.

1. Find the final transformation matrix, \( A \).
2. Find the \((4 \times 4)\) block matrix,

\[
L_i = \frac{\partial A(q)}{\partial q_i}
\]

3. Form the \((4 \times 4n)\) matrix \( L \) as given in Eq (12).
4. Find matrix \( P \) as follows:
   a. for the linear part, use \( P = I_n \otimes p \).
   b. for the angular velocity part, use \( P = A^{-1} \).
5. Calculate the Jacobian matrices \( J_{v/\omega} = LP \) for both the parts.
6. Find the derivative of \( L \) as \( \frac{\partial L}{\partial q_i} \).
7. Calculate the time derivative of the linear part of the Jacobian using Eqs (24) and (25) with \( P = I_n \otimes p \).
8. Calculate the time derivative of the angular part of the Jacobian using Eqs (30), (27) and (26) with \( P = A^{-1} \).

From the algorithm and the demonstration in section 4, it is evident that no manual intervention is required. The calculations of matrices \( A, P \) and \( L \) and their derivatives are
autonomously carried out. Hence this method is suitable for real-time implementation for any desired point on the robot.

6 Application to high-DOF robots and real-time implementation

The aim of this section is to demonstrate the application to higher-DOF robots for real-time implementation. Following the results given in [5], only the computations that are performed after the computation of the final homogenous transformation matrix, \( A \), are considered. The number of matrix operations required is \( 2n \), which includes matrix-to-vector and matrix-to-matrix multiplications. The matrices and vectors are in the homogenous form \((4 \times 4)\) and \((4 \times 1)\). A comparison of the computational efficiency of the proposed method with that of Renaud’s method is given in Table 1 and Fig 2. We chose to compare our method only to Renaud’s method because Renauad’s performance is superior to that of any other existing methods. Renaud’s method involves multiplication of \((3 \times 3)\) matrices, so our method will involve \(37n\) more scalar multiplications. Thus, the number of operations is modified to \(2n + n/(2\delta)\), where \(\delta = 37/64\) is the ratio between the difference and the number of scalar multiplications in our method. From Fig 2, the performance of our proposed method is comparable to Renaud’s

| Method          | No. of matrix multiplications |
|-----------------|-------------------------------|
| Renaud          | \(3n-6\)                      |
| Proposed method | \(2n + n/(2\delta)\)          |

https://doi.org/10.1371/journal.pone.0212018.t001

Fig 2. Comparison of computational efficiencies in terms of number of matrix operations in Renaud’s and the proposed method. For \(n \geq 6\) the number of matrix operations is lower in the proposed method.

https://doi.org/10.1371/journal.pone.0212018.g002
for \((n \leq 8)\) and superior to it for \((n > 8)\). Note that there is no definite way of comparing the performance for \(n < 3\) since the expression of Renaud's method is valid only for \((n \geq 3)\).

We demonstrated the real-time implementation using a 7-DOF redundant spatial manipulator (Robai Cyton Gamm-300) (Fig 3). We used Jacobian in the redundancy resolution of the manipulator by using an optimal control technique based on Hamiltonian formulation [7] for natural boundary conditions. That involves estimation of \(\dot{\mathbf{J}}\). In our implementation, the end-effector of the manipulator is required to follow a circular trajectory with a 0.05 m radius and a frequency of 0.2 rad/s in the y-z plane at a distance of 0.2314 m from the origin along the x-axis. We implemented the differentiations required to find the Jacobian (as in Eq (10)) by using numerical differential based on the central difference method. The differential equations of the motion of the manipulator are numerically integrated using Runge-Kutta fourth order method. The real-time hardware implementation is done by modifying and appending to the open source C++ libraries for matrix operations, TNT and JAMA [23]. The measurements taken by the joint encoders were simultaneously recorded. We compared the results (i.e., the encoder measurements) (see Fig 4) with the simulation results using a second order semi-implicit integrator (ode23s), in MATLAB; ode23s is an implementation of the Bogacki-Shampine method [24]. An error tolerance of \(10^{-9}\) was used in the integrator. We observed that the
Fig 4. (row-1-(a)) x/y/z components of end-effector trajectory from simulation results using ode23s (Bogacki-Shampine method), (row-2-(b)) x/y/z components of end-effector trajectory from real-time physical implementation using RK-4, and (row-3-(c)) error in x/y/z using Runge-Kutta-4 for integration in real-time implementation.
hardware implementation and the simulation results matched, as is evident from the negligible errors (see third row of Fig 4). Thus, our approach is suitable for real-time implementation and causes negligible delay in the automatic formulation and estimation of the Jacobian.

7 Conclusion
A new implementation for online calculation of manipulator Jacobian has been presented and demonstrated. The method, based on matrix differential calculus, offers a systematic approach to calculating the Jacobian of robotic manipulators. Relative to the conventional methods, the calculation of Jacobian has been reduced to the inner product of two matrices. The matrix differentiations are performed using numerical methods. A real-time implementation of the Jacobian has been demonstrated using a planar two-link robot and a 7-DOF spatial robot. The errors in the hardware implementation of this method have been found to be negligible. Although it is computationally superior only for higher-DOF robots, the method is suitable for autonomous and real-time Jacobian estimations for robots with variable points. Our method is also well-suited for reconfigurable robots, which will be addressed in our future work.

Supporting information
S1 File. Supporting information. This is the data file in .xls format. The file has 5 sheets. The description of each sheet is given in S2 File.
(XLSX)

S2 File. Supporting information. This file is in .txt format. This file has the detailed description of the data in S1 File.
(TXT)

Author Contributions
Conceptualization: Pramod Chembrammel.
Formal analysis: Pramod Chembrammel.
Investigation: Pramod Chembrammel.
Methodology: Pramod Chembrammel.
Resources: Pramod Chembrammel.
Software: Pramod Chembrammel.
Supervision: Thenkurussi Kesavadas.
Writing – review & editing: Thenkurussi Kesavadas.

References
1. Selig J.M. Introductory Robotics. Prentice Hall; 1992.
2. Spong M.W., Hutchinson S., Vidyasagar M. Robot Modeling and Control. John Wiley & Sons, Hoboken, New Jersey; 1970.
3. Craig J.J. Introduction to Robotics: Mechanics and Control. 3rd ed. Pearson Prentice Hall; 2005
4. Tsai L.W. Robot Analysis: The Mechanics of Serial and Parallel Manipulators. John Wiley & Sons, Inc.; 1999
5. Orin D.E., Schrader W.W. Efficient Computation of the Jacobian for Robot Manipulator International Journal of Robotics Research; 1984; 3(4):66–75 https://doi.org/10.1177/027836498400300404
6. Grood E.S., Suntay W. J. A Joint Coordinate System for the Clinical Description of Three-Dimensional Motions: Application to the Knee Transactions of ASME; 1983; 105:136–144
7. Kim S.W., Park K.B., Lee J.J. Redundancy Resolution of Robot Manipulators Using Optimal Kinematic Control. Proceedings of International Conference on Robotics and Automation; 1994; p.683-688
8. Gorner M., Stelzer A. A Leg Proprioception Based 6 DOF Oodometry for Statically Stable Walking Robots. Autonomous Robots; 2013; 34(4); 311–326. https://doi.org/10.1007/s10514-013-9326-3
9. Komsuoglu H., McMordie D., Sarani U., Moore N., Buehler M., Koditschek D.E. Proprioception Based Behavioral Advances in a Hexapod Robot. Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation; 2001; p. 3650-3655 vol.4.
10. Chitta S., Vemaza P., Geykhman R., Lee D.D. Proprioceptive Localization for a Quadrupedal Robot on Known Terrain. Proceedings 2007 IEEE International Conference on Robotics and Automation; 2007; p.4582
11. Bruyninckx H., Schutter J.D. Symbolic differentiation of the velocity mapping for a serial kinematic chain. Mechanism and Machine Theory; 1996; 31(2):135–148. https://doi.org/10.1016/0094-114X(95)00069-B
12. Ghoshal A. Robotics: Fundamental Concepts and Analysis. Oxford University Press, India; 2006.
13. Mallik A.K., Ghosh A., Dittrich G. Kinematic Analysis and Synthesis of Mechanisms. CRC Press; 1994.
14. Chembrahmel P. Forward Dynamics of Lower Limb Based on Constrained Multibody Dynamics [MS Thesis]. State University of New York; 2013.
15. Renaud M. Geometric and Kinematic Models of a Robot Manipulator: Calculation of the Jacobian Matrix and its Inverse. In: Proceedings of 11th International Symposium on Industrial Robots; 1981.
16. Zhang Y., Xiao L., Xiao Z., Mao M. Zeroing Dynamics, Gradient Dynamics, and Newton Iterations. CRC Press; 2015.
17. Chen D., Zhang Y., Li S. Tracking Control of Robot Manipulators with Unknown Models: A Jacobian-Matrix-Adaption Method. IEEE Transactions on Industrial Informatics; 2018; 14(7):3044–3053. https://doi.org/10.1109/TII.2017.2766455
18. Schilling R.J. Fundamentals of Robotics: Analysis and Control. Prentice-Hall, New Jersey; 1990
19. Chen C.T. Linear System Theory and Design. 3rd ed. Oxford University Press, New York; 1999.
20. Greenwood D.T. Principles of Dynamics. Prentice-Hall, Inc., New Jersey; 1965.
21. Petersen K.B., Pedersen M.S. The Matrix Cookbook 2008
22. Chembrahmel P., Younus H.M., Kesavadas T. Modelling and Simulation of Guide-Wire Interaction with Vasculature Using Constrained Multibody Dynamics. In: Proceedings of ASME 2013 International Mechanical Engineering Congress and Exposition. ASME; 2013.
23. Template Numerical Toolkit.; http://math.nist.gov/tnt/overview.html
24. Bogacki P., Shampine L.F. A 3(2) Pair of Runge-Kutta Formulas. Applied Mathematical Letters. 1989; 2(4):321–325. https://doi.org/10.1016/0893-9659(89)90079-7