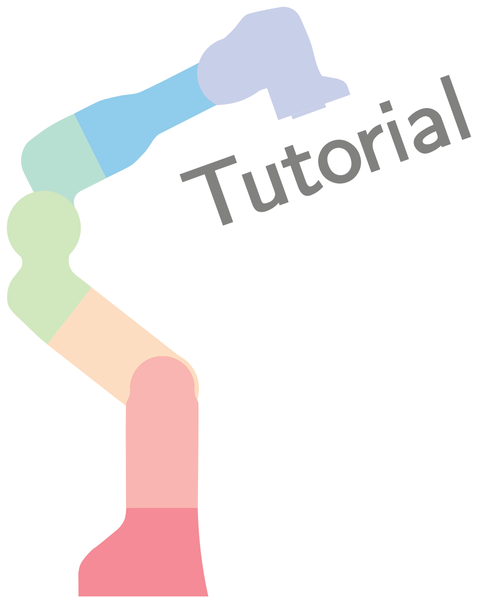
arXiv:2207.01794v2 [cs.RO] 16 May 2023



Manipulator Differential Kinematics

Part 2: Acceleration and Advanced Applications

By Jesse Haviland and Peter Corke

Kinematics, in the context of robotic manipulators is concerned with the relationship between the position of the robot's joints and the pose of its end effector, as well as the relationships between various derivatives of those quantities. In this second part of our two-part tutorial, we focus on second-order differential kinematics and subsequent applications. These applications demonstrate advanced techniques which are highly relevant to topics including sensor-based control, constrained control, and motion planning.

We start by introducing the second-order differential kinematics and the manipulator Hessian. The second-order differential kinematics expose a relationship between the robot's joint velocities and the end-effector acceleration. We then describe the differential kinematics' analytical forms, which are essential to dynamics applications. Subsequently, we provide a general formula for higher-order derivatives before detailing and experimenting with three advanced applications.

The first application we consider is advanced velocity control – an important topic for reactive and sensor-based control tasks. We demonstrate this by extending resolved-rate motion control (RRMC) to perform additional sub-tasks while still achieving its goal and then reformulate the problem as a quadratic program to enable greater flexibility and additional constraints. We then take another look at numerical inverse kinematics (IK) with an emphasis on adding constraints to improve robustness and solvability. We subsequently present a comprehensive experiment that compares the performance and characteristics of each IK method on three different manipulators. Finally, we analyse how incorporating the manipulator Hessian into a motion controller can help to escape singularities.

In Part 1 [1], we described a method of modelling kinematics using the elementary transform sequence (ETS), before formulating forward kinematics and the manipulator Jacobian. We then described some introductory but fundamental applications of the manipulator Jacobian including RRMC, numerical IK, and some manipulator performance measures. Part 1 provides many important definitions, functions and conventions, and we recommend that readers review Part 1 before reading this Part.

Once again, we have provided Jupyter Notebooks to accompany each section within this tutorial. The Notebooks are written in Python and use the Robotics Toolbox for Python, and the Swift Simulator [2] to provide full implementations of each concept, equation, and algorithm presented in this tutorial. The Notebooks use rich Markdown text and LaTeX equations to document and communicate key concepts. While not absolutely essential, for the most engaging and informative experience, we recommend working through the Jupyter Notebooks while reading this article. The Notebooks and setup instructions can be accessed at github.com/jhavl/dkt.

Deriving the Manipulator Hessian

The Manipulator Hessian

We begin with the forward kinematics of a manipulator as described in Part 1

$$
\begin{aligned}
{}^0\mathbf{T}_e(t) &= \mathcal{K}(\boldsymbol{q}(t)) \\
&= \prod_{i=1}^{M} \mathbf{E}_i(\eta_i). \qquad (1)
\end{aligned}
$$

where $\boldsymbol{q}(t) \in \mathbb{R}^n$ is the vector of joint generalised coordinates, n is the number of joints, and M is the number of elementary transforms $\mathbf{E}_i \in \mathbf{SE}(3)$. From the derivative of (1) we can express the spatial translational and angular velocity as a function of the joint coordinates and velocities

$$
\boldsymbol{\nu} = \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{pmatrix} = \mathbf{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \qquad (2)
$$

where $\mathbf{J}(\boldsymbol{q})$ is the manipulator Jacobian, $\boldsymbol{v} = (v_x,\ v_y,\ v_z)$, and $\boldsymbol{\omega} = (\omega_x,\ \omega_y,\ \omega_z)$. Taking the temporal derivative gives

$$
\begin{pmatrix} \dot{\boldsymbol{v}} \\ \dot{\boldsymbol{\omega}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{\alpha} \end{pmatrix} = \dot{\mathbf{J}}\dot{\boldsymbol{q}} + \mathbf{J}\ddot{\boldsymbol{q}} \qquad (3)
$$

where $\boldsymbol{a} \in \mathbb{R}^3$ is the end-effector translational acceleration, $\boldsymbol{\alpha} \in \mathbb{R}^3$ is the end-effector angular acceleration, and

$$
\begin{aligned}
\dot{\mathbf{J}} &= \frac{\mathrm{d}\mathbf{J}(\boldsymbol{q})}{\mathrm{d}t} \\
&= \frac{\partial \mathbf{J}(\boldsymbol{q})}{\partial q_1}\dot{q}_1 + \frac{\partial \mathbf{J}(\boldsymbol{q})}{\partial q_2}\dot{q}_2 + \cdots + \frac{\partial \mathbf{J}}{\partial q_n}\dot{q}_n \\
&= \begin{pmatrix} \frac{\partial \mathbf{J}(\boldsymbol{q})}{\partial q_1} & \frac{\partial \mathbf{J}(\boldsymbol{q})}{\partial q_2} & \cdots & \frac{\partial \mathbf{J}(\boldsymbol{q})}{\partial q_n} \end{pmatrix} \dot{\boldsymbol{q}} \\
&= \mathbf{H}(\boldsymbol{q})\dot{\boldsymbol{q}} \\
&= \begin{pmatrix} \mathbf{H}_a(\boldsymbol{q}) \\ \mathbf{H}_\alpha(\boldsymbol{q}) \end{pmatrix} \dot{\boldsymbol{q}} \qquad (4)
\end{aligned}
$$

where $\mathbf{H}(\boldsymbol{q}) \in \mathbb{R}^{n \times 6 \times n}$ is the manipulator Hessian tensor, which is the partial derivative of the manipulator Jacobian with respect to the joint coordinates, $\mathbf{H}_a(\boldsymbol{q}) \in \mathbb{R}^{n \times 3 \times n}$ forms the translational component of the Hessian, and $\mathbf{H}_\alpha(\boldsymbol{q}) \in \mathbb{R}^{n \times 3 \times n}$ forms the angular component of the Hessian.

The second partial derivative of a pose with respect to the joint variables q_j and q_k, can be obtained by taking

$$
\begin{aligned}
\frac{\partial^2 \mathbf{T}}{\partial q_k q_j} &= \begin{pmatrix} \mathbf{H}_{R_{jk}} & \mathbf{H}_{t_{jk}} \\ 0 & 0 \end{pmatrix} \\
&= \frac{\partial}{\partial q_k} \left(\prod_{i=1}^{\mu(j)-1} \mathbf{E}_i(\eta_i) \frac{\mathrm{d}\mathbf{E}_{\mu(j)}(q_j)}{dq_j} \prod_{i=\mu(j)+1}^{M} \mathbf{E}_i(\eta_i) \right) \\
&= \begin{cases}
\prod\limits_{i=1}^{\mu(k)-1} \mathbf{E}_i(\eta_i) \frac{\mathrm{d}\mathbf{E}_{\mu(k)}(q_k)}{dq_k} \prod\limits_{i=\mu(k)+1}^{\mu(j)-1} \mathbf{E}_i(\eta_i) \frac{\mathrm{d}\mathbf{E}_{\mu(j)}(q_j)}{dq_j} \prod\limits_{i=\mu(j)+1}^{M} \mathbf{E}_i(\eta_i) & \text{if } k < j \\
\prod\limits_{i=1}^{\mu(k)-1} \mathbf{E}_i(\eta_i) \frac{\mathrm{d}^2\mathbf{E}_{\mu(k)}(q_k)}{dq_k^2} \prod\limits_{i=\mu(k)+1}^{M} \mathbf{E}_i(\eta_i) & \text{if } k = j \\
\prod\limits_{i=1}^{\mu(j)-1} \mathbf{E}_i(\eta_i) \frac{\mathrm{d}\mathbf{E}_{\mu(j)}(q_j)}{dq_j} \prod\limits_{i=\mu(j)+1}^{\mu(k)-1} \mathbf{E}_i(\eta_i) \frac{\mathrm{d}\mathbf{E}_{\mu(k)}(q_k)}{dq_k} \prod\limits_{i=\mu(k)+1}^{M} \mathbf{E}_i(\eta_i) & \text{if } k > j
\end{cases} \qquad (6)
\end{aligned}
$$

the derivative of

$$\frac{\partial \mathbf{T}(\boldsymbol{q})}{\partial q_j} = \frac{\partial}{\partial q_j}\left(\mathbf{E}_1(\eta_1)\mathbf{E}_2(\eta_2)\cdots\mathbf{E}_M(\eta_M)\right)$$
$$= \prod_{i=1}^{\mu(j)-1} \mathbf{E}_i(\eta_i)\frac{\mathrm{d}\mathbf{E}_{\mu(j)}(q_j)}{\mathrm{d}q_j}\prod_{i=\mu(j)+1}^{M}\mathbf{E}_i(\eta_i). \quad (5)$$

with respect to q_k. This results in (6), where the function $\mu(j)$ returns the index in (1) where q_j appears as a variable.

In (6), the derivative of an elementary transform with respect to a joint coordinate is obtained using one of

$$\frac{\mathrm{d}\mathbf{T}_{\mathbf{R}_x}(\theta)}{\mathrm{d}\theta} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\mathbf{T}_{\mathbf{R}_x}(\theta) = \left[\hat{\boldsymbol{R}}_x\right]\mathbf{T}_{\mathbf{R}_x}(\theta), \quad (7)$$

$$\frac{\mathrm{d}\mathbf{T}_{\mathbf{R}_y}(\theta)}{\mathrm{d}\theta} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\mathbf{T}_{\mathbf{R}_y}(\theta) = \left[\hat{\boldsymbol{R}}_y\right]\mathbf{T}_{\mathbf{R}_y}(\theta), \quad (8)$$

$$\frac{\mathrm{d}\mathbf{T}_{\mathbf{R}_z}(\theta)}{\mathrm{d}\theta} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\mathbf{T}_{\mathbf{R}_z}(\theta) = \left[\hat{\boldsymbol{R}}_z\right]\mathbf{T}_{\mathbf{R}_z}(\theta), \quad (9)$$

for a revolute joint, or one of

$$\frac{\mathrm{d}\mathbf{T}_{\boldsymbol{t}_x}(d)}{\mathrm{d}d} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \left[\hat{\boldsymbol{t}}_x\right], \quad (10)$$

$$\frac{\mathrm{d}\mathbf{T}_{\boldsymbol{t}_y}(d)}{\mathrm{d}d} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \left[\hat{\boldsymbol{t}}_y\right], \quad (11)$$

$$\frac{\mathrm{d}\mathbf{T}_{\boldsymbol{t}_z}(d)}{\mathrm{d}d} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \left[\hat{\boldsymbol{t}}_z\right], \quad (12)$$

for a prismatic joint. For the second derivative of an elementary transform with respect to the same joint variable, as is the case for $k = j$ in (6), the result is

$$\frac{\mathrm{d}^2\mathbf{T}_{\mathbf{R}_x}(\theta)}{\mathrm{d}\theta^2} = \left[\hat{\boldsymbol{R}}_x\right]^2\mathbf{T}_{\mathbf{R}_x}(\theta), \quad (13)$$
$$\frac{\mathrm{d}^2\mathbf{T}_{\mathbf{R}_y}(\theta)}{\mathrm{d}\theta^2} = \left[\hat{\boldsymbol{R}}_y\right]^2\mathbf{T}_{\mathbf{R}_y}(\theta), \quad (14)$$
$$\frac{\mathrm{d}^2\mathbf{T}_{\mathbf{R}_z}(\theta)}{\mathrm{d}\theta^2} = \left[\hat{\boldsymbol{R}}_z\right]^2\mathbf{T}_{\mathbf{R}_z}(\theta), \quad (15)$$

for a revolute joint or a zero matrix for a prismatic joint.

As for the manipulator Jacobian in Part 1, to form the manipulator Hessian, we partition it into translational and rotational components as expressed in (4).

To form $\mathbf{H}_{\alpha_{jk}}$, the angular component of the manipulator Hessian of a joint variable j with respect to another joint variable k, we take the partial derivative of the j^{th} column of the manipulator Jacobian in

$$\mathbf{J}_{\omega_j}(\boldsymbol{q}) = \vee_\times\left(\rho\left(\frac{\partial\mathbf{T}(\boldsymbol{q})}{\partial q_j}\right)\rho\left(\mathbf{T}(\boldsymbol{q})\right)^\top\right) \quad (16)$$

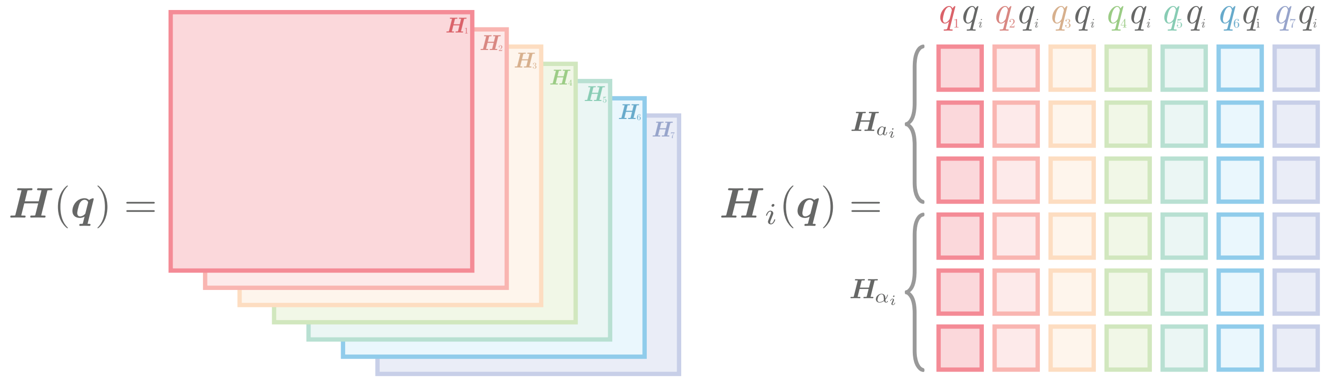


Figure 1. Visualisation of the Hessian $\mathbf{H}(\boldsymbol{q})$ representing a 7-joint manipulator. Each slice of the Hessian $\mathbf{H}_i(\boldsymbol{q})$ represents the acceleration of the end-effector caused by the velocities of each joint $\boldsymbol{q}$ with respect to the velocity of joint q_i. Within a slice, the top three rows $\mathbf{H}_{\alpha_i}$ correspond to the linear acceleration, while the bottom three rows $\mathbf{H}_{a_i}$ correspond to the angular acceleration, of the end-effector $\boldsymbol{\alpha}$ caused by the velocities of the joints.

using the product rule

$$\begin{aligned}\mathbf{H}_{\alpha_{jk}} &= \frac{\partial \mathbf{J}_{\omega_j}(\boldsymbol{q})}{\partial q_k} \\ &= \vee_{\times}\left(\rho\left(\frac{\partial^2 \mathbf{T}(\boldsymbol{q})}{\partial q_j \partial q_k}\right)\rho\left(\mathbf{T}(\boldsymbol{q})\right)^{\top} + \right. \\ &\qquad \left. \rho\left(\frac{\partial \mathbf{T}(\boldsymbol{q})}{\partial q_j}\right)\rho\left(\frac{\partial \mathbf{T}(\boldsymbol{q})}{\partial q_k}\right)^{\top}\right) \\ &= \vee_{\times}\left(\mathbf{H}_{Rjk}\ \rho\left(\mathbf{T}(\boldsymbol{q})\right)^{\top} + \mathbf{J}_{Rj}\mathbf{J}_{Rk}^{\top}\right) \end{aligned} \tag{17}$$

where $\mathbf{H}_{\alpha_{jk}} \in \mathbb{R}^3$, and $\mathbf{H}_{Rjk}$ is obtained from (6).

To form $\mathbf{H}_{a_{jk}}$, the translational component of the manipulator Hessian for joint variable j with respect to another joint variable k, we take the partial derivative of the j^{th} translational component of the manipulator Jacobian in

$$\mathbf{J}_{\nu_j}(\boldsymbol{q}) = \tau\left(\frac{\partial \mathbf{T}(\boldsymbol{q})}{\partial q_j}\right) \tag{18}$$

which provides

$$\begin{aligned}\mathbf{H}_{a_{jk}} &= \frac{\partial \mathbf{J}_{\nu j}(\boldsymbol{q})}{\partial q_k} \\ &= \tau\left(\frac{\partial^2 \mathbf{T}(\boldsymbol{q})}{\partial q_j \partial q_k}\right) \\ &= \mathbf{H}_{tjk}\end{aligned} \tag{19}$$

where $\mathbf{H}_{a_{jk}} \in \mathbb{R}^3$, and $\mathbf{H}_{tjk}$ is obtained from (6).

Stacking (17) and (19) we form the component of the manipulator Hessian for joint variable j with respect to another joint variable k

$$\mathbf{H}_{jk} = \begin{pmatrix}\mathbf{H}_{a_{jk}} \\ \mathbf{H}_{\alpha_{jk}}\end{pmatrix} \tag{20}$$

where $\mathbf{H}_{jk} \in \mathbb{R}^6$.

The component of the manipulator Hessian for joint variable j is formed by arranging (20) into columns of a matrix

$$\mathbf{H}_j = \begin{pmatrix}\mathbf{H}_{j1} & \cdots & \mathbf{H}_{jn}\end{pmatrix} \tag{21}$$

where $\mathbf{H}_j \in \mathbb{R}^{6\times n}$.

The whole manipulator Hessian is formed by arranging (21) into *slices* of a tensor

$$\mathbf{H} = \begin{pmatrix}\mathbf{H}_1 & \cdots & \mathbf{H}_n\end{pmatrix} \tag{22}$$

where $\mathbf{H} \in \mathbb{R}^{n\times 6\times n}$ and last last two dimensions of $\mathbf{H}$ define the dimension of the slices $\mathbf{H}_i$. We show the formation of the manipulator Hessian for a 7-joint manipulator in Figure 1.

Fast Manipulator Hessian

We can calculate the manipulator Hessian using (17) and (19) with (6), however this has $\mathcal{O}(n^3)$ time complexity.

We revisit (17) while substituting in

$$\frac{\mathrm{d}\mathbf{R}(\theta)}{\mathrm{d}\theta} = [\hat{\boldsymbol{\omega}}]_{\times}\,\mathbf{R}(\theta(t)) \tag{23}$$

and simplify

$$\begin{aligned}\mathbf{H}_{\alpha_{jk}} &= \vee_{\times}\Bigg([\hat{\boldsymbol{\omega}}_k]_{\times}[\hat{\boldsymbol{\omega}}_j]_{\times}\mathbf{R}(\boldsymbol{q})\mathbf{R}(\boldsymbol{q})^{\top} + \\ &\qquad [\hat{\boldsymbol{\omega}}_j]_{\times}\mathbf{R}(\boldsymbol{q})\left([\hat{\boldsymbol{\omega}}_k]_{\times}\mathbf{R}(\boldsymbol{q})\right)^{\top}\Bigg) \\ &= \vee_{\times}\left([\hat{\boldsymbol{\omega}}_k]_{\times}[\hat{\boldsymbol{\omega}}_j]_{\times} + [\hat{\boldsymbol{\omega}}_j]_{\times}\mathbf{R}(\boldsymbol{q})\mathbf{R}(\boldsymbol{q})^{\top}[\hat{\boldsymbol{\omega}}_k]_{\times}{}^{\top}\right) \\ &= \vee_{\times}\left([\hat{\boldsymbol{\omega}}_k]_{\times}[\hat{\boldsymbol{\omega}}_j]_{\times} - [\hat{\boldsymbol{\omega}}_j]_{\times}[\hat{\boldsymbol{\omega}}_k]_{\times}\right).\end{aligned} \tag{24}$$

Since we know that $\mathbf{J}_{\omega x} = [\hat{\boldsymbol{\omega}}_x]_{\times}$, and using the identity $[\boldsymbol{a}\times\boldsymbol{b}]_{\times} = [\boldsymbol{a}]_{\times}[\boldsymbol{b}]_{\times} - [\boldsymbol{b}]_{\times}[\boldsymbol{a}]_{\times}$ we show that

$$\begin{aligned}\mathbf{H}_{\alpha_{jk}} &= \vee_{\times}\left([\mathbf{J}_{\omega_k}]_{\times}[\mathbf{J}_{\omega_j}]_{\times} - [\mathbf{J}_{\omega_j}]_{\times}[\mathbf{J}_{\omega_k}]_{\times}\right) \\ &= \mathbf{J}_{\omega_k} \times \mathbf{J}_{\omega_j}\end{aligned} \tag{25}$$

which means that the rotational component of the manipulator Hessian can be calculated from the rotational components of the manipulator Jacobian. A key relationship is that the velocity of joint j, with respect to the velocity of the same, or preceding joint k, does not contribute acceleration to the end-effector from the perspective of joint j. Consequently, $\mathbf{H}_{\alpha_{jk}} = 0$ when $k \geq j$.

For the translational component of the manipulator Hessian $\mathbf{H}_a$, we can see in (6) that two of the conditions will have the same result: when $k < j$, and when $k > j$. Therefore, we have

$$\mathbf{H}_{a_{jk}}(\boldsymbol{q}) = \mathbf{H}_{a_{kj}}(\boldsymbol{q}) \tag{26}$$

and by exploiting this relationship, we can simplify (19) to

$$\begin{aligned}\mathbf{H}_{a_{jk}}(\boldsymbol{q}) &= [\mathbf{J}_{\omega_k}]_{\times}\,\mathbf{J}_{\nu_j} \\ &= \mathbf{J}_{\omega_a} \times \mathbf{J}_{\nu_b}\end{aligned} \tag{27}$$

where $a = \min(j,k)$, and $b = \max(j,k)$. This means that the translational component of the manipulator Hessian can be calculated from components of the manipulator Jacobian.

Through this simplification, computation of the manipulator Hessian reduces to $\mathcal{O}(n^2)$ time complexity.

Deriving Higher Order Derivatives

Obtaining the n^{th} partial derivative of the manipulator kinematics, where $n \geq 3$, can be obtained using the product rule on the $(n-1)^{th}$ partial derivative while considering their partitioned form.

For example, to obtain the 3^{rd} partial derivative, we take the partial derivative of the manipulator Hessian with respect to the joint coordinates, in its partitioned form

$$\begin{aligned}\frac{\partial \mathbf{H}_{jk}(\boldsymbol{q})}{\partial q_l} &= \begin{pmatrix} \dfrac{\partial \mathbf{H}_{a_{jk}}(\boldsymbol{q})}{\partial q_l} \\ \dfrac{\partial \mathbf{H}_{\alpha_{jk}}(\boldsymbol{q})}{\partial q_l} \end{pmatrix} \\ &= \begin{pmatrix} \dfrac{\partial}{\partial q_l}\left(\mathbf{J}_{\omega_k} \times \mathbf{J}_{\omega_j}\right) \\ \dfrac{\partial}{\partial q_l}\left(\mathbf{J}_{\omega_k} \times \mathbf{J}_{\nu_j}\right) \end{pmatrix} \\ &= \begin{pmatrix} \left(\mathbf{H}_{\alpha_{kl}} \times \mathbf{J}_{\omega_j}\right) + \left(\mathbf{J}_{\omega_k} \times \mathbf{H}_{\alpha_{jl}}\right) \\ \left(\mathbf{H}_{\alpha_{kl}} \times \mathbf{J}_{\nu_j}\right) + \left(\mathbf{J}_{\omega_k} \times \mathbf{H}_{a_{jl}}\right) \end{pmatrix} \end{aligned} \tag{28}$$

where $\frac{\partial \mathbf{H}_{jk}(\boldsymbol{q})}{\partial q_l} \in \mathbb{R}^6$. Continuing, we obtain the following

$$\frac{\partial \mathbf{H}_{jk}(\boldsymbol{q})}{\partial \boldsymbol{q}} = \begin{pmatrix} \dfrac{\partial \mathbf{H}_{jk}(\boldsymbol{q})}{\partial q_0} & \cdots & \dfrac{\partial \mathbf{H}_{jk}(\boldsymbol{q})}{\partial q_n} \end{pmatrix} \tag{29}$$

where $\frac{\partial \mathbf{H}_{jk}(\boldsymbol{q})}{\partial q_l} \in \mathbb{R}^{6 \times n}$,

$$\frac{\partial \mathbf{H}_{j}(\boldsymbol{q})}{\partial \boldsymbol{q}} = \begin{pmatrix} \dfrac{\partial \mathbf{H}_{j_0}(\boldsymbol{q})}{\partial \boldsymbol{q}} & \cdots & \dfrac{\partial \mathbf{H}_{j_n}(\boldsymbol{q})}{\partial \boldsymbol{q}} \end{pmatrix} \tag{30}$$

where $\frac{\partial \mathbf{H}_{jk}(\boldsymbol{q})}{\partial \boldsymbol{q}} \in \mathbb{R}^{n \times 6 \times n}$, and finally

$$\frac{\partial \mathbf{H}(\boldsymbol{q})}{\partial \boldsymbol{q}} = \begin{pmatrix} \dfrac{\partial \mathbf{H}_{0}(\boldsymbol{q})}{\partial \boldsymbol{q}} & \cdots & \dfrac{\partial \mathbf{H}_{n}(\boldsymbol{q})}{\partial \boldsymbol{q}} \end{pmatrix} \tag{31}$$

where $\frac{\partial \mathbf{H}(\boldsymbol{q})}{\partial \boldsymbol{q}} \in \mathbb{R}^{n \times n \times 6 \times n}$ is the 4-dimensional tensor representing the 3^{rd} partial derivative of the manipulator kinematics.

We have included a function as part of our open source Robotics Toolbox for Python [2] which can calculate the n^{th} partial derivative of the manipulator kinematics. Note that the function has $\mathcal{O}(n^{order})$ time complexity, where *order* represents the order of the partial derivative being calculated.

Analytical Form

The kinematic derivatives we have presented so far have been in geometric form with translational velocity $\boldsymbol{v}$ and angular velocity $\boldsymbol{\omega}$ vectors and their derivatives. Some applications require the manipulator Jacobian and further derivatives to be expressed with different orientation rate representations such as the rate of change of roll-pitch-yaw angles, Euler angles or exponential coordinates – these are called analytical forms.

One important application that requires this is task space dynamics [3] and operational space control [4]. Operational space control is a dynamics formulation for tasks that require constrained end-effector motion and force control. There are many everyday tasks that can make use of this control formulation such as opening a door or cleaning a surface. While dynamics are outside the scope of this tutorial, these dynamics controllers require the manipulator Jacobian and further derivatives to be represented in analytical form.

Roll-Pitch-Yaw Analytical Form

For XYZ roll-pitch-yaw angles in $\boldsymbol{\Gamma} = (\alpha, \beta, \gamma)$, the resulting rotation matrix is

$$\begin{aligned}\mathbf{R} &= \mathbf{R}_x(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_z(\alpha) \\ &= \begin{pmatrix} c\beta c\alpha & -c\beta s\alpha & c\beta \\ c\gamma s\alpha + c\alpha s\beta s\gamma & -s\beta s\gamma s\alpha + c\gamma c\alpha & -c\beta s\gamma \\ s\gamma s\alpha - c\gamma c\alpha s\beta & c\gamma s\beta s\alpha + c\alpha s\gamma & c\beta c\gamma \end{pmatrix}\end{aligned} \tag{32}$$

where $s\theta$ and $c\theta$ are short for $\sin(\theta)$ and $\cos(\theta)$. From Part 1 we have the relationship

$$\begin{aligned}\dot{\mathbf{R}} &= [\boldsymbol{\omega}]_\times \mathbf{R} \\ \dot{\mathbf{R}}\mathbf{R}^\top &= [\boldsymbol{\omega}]_\times \\ \vee_\times\left(\dot{\mathbf{R}}\mathbf{R}^\top\right) &= \boldsymbol{\omega}\end{aligned} \tag{33}$$

which gives us the result

$$\begin{aligned}\boldsymbol{\omega} = \begin{pmatrix}\omega_x \\ \omega_y \\ \omega_z\end{pmatrix} &= \begin{pmatrix} s\beta\dot{\alpha} + \dot{\gamma} \\ -c\beta s\gamma\dot{\alpha} + c\gamma\dot{\beta} \\ c\beta c\gamma\dot{\alpha} + s\gamma\dot{\beta}\end{pmatrix} \\ &= \begin{pmatrix} s\beta & 0 & 1 \\ -c\beta s\gamma & c\gamma & 0 \\ c\beta c\gamma & s\gamma & 0 \end{pmatrix}\begin{pmatrix}\dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma}\end{pmatrix} \\ &= \mathbf{A}(\boldsymbol{\Gamma})\dot{\boldsymbol{\Gamma}}\end{aligned} \tag{34}$$

where $\mathbf{A}(\boldsymbol{\Gamma})$ is a Jacobian that maps XYZ roll-pitch-yaw angle rates to angular velocity.

The analytical Jacobian represented with roll-pitch-yaw angle rates is

$$\begin{aligned}\mathbf{J}_\Lambda(\boldsymbol{q})(\boldsymbol{q}) &= \mathbf{J}_\Gamma(\boldsymbol{\Gamma})\ \mathbf{J}(\boldsymbol{q}) \\ &= \begin{pmatrix} \mathbf{1}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{A}^{-1}(\boldsymbol{\Gamma})\end{pmatrix}\mathbf{J}(\boldsymbol{q}).\end{aligned} \tag{35}$$

In the case where $\beta = \pm 90°$, $\mathbf{A}$ will be singular and its inverse does not exist. Therefore, for applications involving the analytical differential kinematics, it is important to choose an angular representation where the singularity lies outside of the normal operating range of the robot [3].

The derivative of $\mathbf{J}_\Lambda(\boldsymbol{q})$ is typically used in applications that have a task-space acceleration term. We can obtain $\dot{\mathbf{J}}_\Lambda(\boldsymbol{q})$ from (35) using the product rule

$$\begin{aligned}\dot{\mathbf{J}}_\Lambda(\boldsymbol{q}) &= \frac{\mathrm{d}\mathbf{J}_\Gamma(\boldsymbol{\Gamma})}{\mathrm{d}t}\ \mathbf{J}(\boldsymbol{q}) + \mathbf{J}_\Gamma(\boldsymbol{\Gamma})\ \frac{\mathrm{d}\mathbf{J}(\boldsymbol{q})}{\mathrm{d}t} \\ &= \frac{\mathrm{d}\mathbf{J}_\Gamma(\boldsymbol{\Gamma})}{\mathrm{d}t}\ \mathbf{J}(\boldsymbol{q}) + \mathbf{J}_\Gamma(\boldsymbol{\Gamma})\ \left(\mathbf{H}(\boldsymbol{q})\dot{\boldsymbol{q}}\right)\end{aligned} \tag{36}$$

where the derivative of the augmented Jacobian $\mathbf{J}_\Gamma(\boldsymbol{\Gamma})$ is

$$\frac{\mathrm{d}\mathbf{J}_\Gamma(\boldsymbol{\Gamma})}{\mathrm{d}t} = \begin{pmatrix}\mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \dot{\mathbf{A}}^{-1}(\boldsymbol{\Gamma},\ \dot{\boldsymbol{\Gamma}})\end{pmatrix}. \tag{37}$$

As previously mentioned, the analytical Jacobian can be derived for different orientation parameterisations including Euler angles, exponential coordinates or ZYX roll-pitch-yaw angles. To achieve this, the rotation matrix in (32) is replaced with the appropriate elements for the different parameterisation and following through the methodology presented in this section to produce the appropriate analytical Jacobian and derivative.

Advanced Velocity Control

Many modern manipulators are redundant – they have more than six degrees of freedom. We can exploit this redundancy by having the robot optimise some performance measure while still achieving the original goal. In this section, we start with the resolved-rate motion control (RRMC) algorithm explained in Part 1 of this tutorial

$$\dot{\boldsymbol{q}} = \mathbf{J}\left(\boldsymbol{q}\right)^{+}\ \boldsymbol{\nu}. \tag{38}$$

Null-space Projection

The Jacobian of a redundant manipulator has a null space. Any joint velocity vector which is a linear combination of the manipulator Jacobian's null-space basis vectors will result in zero end-effector motion ($\boldsymbol{\nu} = 0$). We can augment (38) to add a joint velocity vector $\dot{\boldsymbol{q}}_{null}$ which can be projected into the null space resulting in zero end-effector spatial velocity

$$\dot{\boldsymbol{q}} = \underbrace{\mathbf{J}\left(\boldsymbol{q}\right)^{+}\ \boldsymbol{\nu}}_{\text{end−effector motion}} + \underbrace{\left(\mathbf{1}_n - \mathbf{J}(\boldsymbol{q})^{+}\mathbf{J}(\boldsymbol{q})\right)\dot{\boldsymbol{q}}_{\text{null}}}_{\text{null−space motion}} \tag{39}$$

where $\dot{\boldsymbol{q}}_{\text{null}}$ is the desired joint velocities for the null-space motion.

We can set $\dot{\boldsymbol{q}}_{\text{null}}$ to be the gradient of any scalar performance measure $\gamma(\boldsymbol{q})$ where the performance measure is a differentiable function of the joint coordinates $\boldsymbol{q}$.

Park [5] proposed using the gradient of the Yoshikawa manipulability index as $\dot{\boldsymbol{q}}_{\text{null}}$ in (39). As detailed in Part 1 of this tutorial, the manipulability index [6] is calculated as

$$m(\boldsymbol{q}) = \sqrt{\det\left(\hat{\mathbf{J}}(\boldsymbol{q})\hat{\mathbf{J}}(\boldsymbol{q})^\top\right)} \tag{40}$$

where $\hat{\mathbf{J}}(\boldsymbol{q}) \in \mathbb{R}^{3\times n}$ is either the translational or rotational rows of $\mathbf{J}(\boldsymbol{q})$ causing $m(\boldsymbol{q})$ to describe the corresponding component of manipulability. Excurse 1 highlights considerations for manipulators with mixed joint types.

Taking the time derivative of (40), using the chain rule

$$\frac{\mathrm{d}\ m(t)}{\mathrm{d}t} = \frac{1}{2m(t)}\frac{\mathrm{d}}{\mathrm{d}t}\det\left(\hat{\mathbf{J}}(\boldsymbol{q})\hat{\mathbf{J}}(\boldsymbol{q})^\top\right) \tag{41}$$

we can write this as

$$\dot{m} = \boldsymbol{J}_m^\top(\boldsymbol{q})\ \dot{\boldsymbol{q}} \tag{42}$$

where

$$\boldsymbol{J}_m^\top(\boldsymbol{q}) = m\begin{pmatrix}\operatorname{vec}\left(\hat{\mathbf{J}}(\boldsymbol{q})\hat{\mathbf{H}}_1(\boldsymbol{q})^\top\right)^\top \operatorname{vec}\left((\hat{\mathbf{J}}(\boldsymbol{q})\hat{\mathbf{J}}(\boldsymbol{q})^\top)^{-1}\right) \\ \operatorname{vec}\left(\hat{\mathbf{J}}(\boldsymbol{q})\hat{\mathbf{H}}_2(\boldsymbol{q})^\top\right)^\top \operatorname{vec}\left((\hat{\mathbf{J}}(\boldsymbol{q})\hat{\mathbf{J}}(\boldsymbol{q})^\top)^{-1}\right) \\ \vdots \\ \operatorname{vec}\left(\hat{\mathbf{J}}(\boldsymbol{q})\hat{\mathbf{H}}_n(\boldsymbol{q})^\top\right)^\top \operatorname{vec}\left((\hat{\mathbf{J}}(\boldsymbol{q})\hat{\mathbf{J}}(\boldsymbol{q})^\top)^{-1}\right)\end{pmatrix} \tag{43}$$

is the manipulability Jacobian $\boldsymbol{J}_m^\top \in \mathbb{R}^n$ and where the vector operation $\operatorname{vec}(\cdot) : \mathbb{R}^{a\times b} \to \mathbb{R}^{ab}$ converts a matrix column-wise into a column vector, and $\hat{\mathbf{H}}_i \in \mathbb{R}^{3\times n}$ is the translational or rotational component (matching the choice of $\hat{\mathbf{J}}(\boldsymbol{q})$) of $\mathbf{H}_i \in \mathbb{R}^{6\times n}$ which is the i^{th} component of the manipulator Hessian tensor $\mathbf{H} \in \mathbb{R}^{n\times6\times n}$.

Excurse 1. Mixed Joint Manipulators

Manipulators that contain different joint types – such as those that contain both revolute and prismatic joints – have implications on certain algorithms. Scaling issues can be introduced due to the different units which may cause a translation or rotation component to dominate the result. In the case of using the manipulator Jacobian for a performance metric, care must be taken to ensure the performance is acceptable or use a scaling approach [7]. In the case where a gain is applied to a control input on each joint, simply use an appropriately scaled gain value for each different joint type.

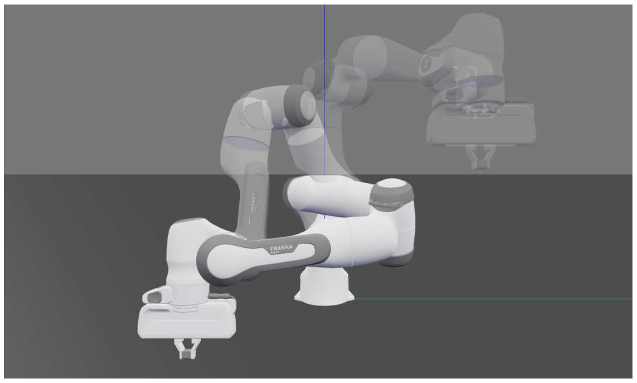


Figure 2. Comparison of a Panda robot which has been controlled by RRMC (semi-transparent robot) and the manipulability maximising controller in (44) (opaque robot). The transparent robot displays the initial pose.

The complete equation proposed by Park [5] is

$$\dot{\boldsymbol{q}} = \mathbf{J}\left(\boldsymbol{q}\right)^{+} \boldsymbol{\nu} + \frac{1}{\lambda}\Big(\left(\mathbf{1}_n - \mathbf{J}(\boldsymbol{q})^{+}\mathbf{J}(\boldsymbol{q})\right)\mathbf{J}_m(\boldsymbol{q})\Big) \tag{44}$$

where λ is a gain that scales the magnitude of the null-space velocities. This equation will choose joint velocities $\dot{\boldsymbol{q}}$ which will achieve the end-effector spatial velocity $\boldsymbol{\nu}$ while also improving the translational and/or rotational manipulability of the robot.

As with RRMC, (44) will provide the joint velocities for a desired end-effector velocity. As we did in Part 1, we can employ (44) in a position-based servoing (PBS) controller to drive the end-effector towards some desired pose. The PBS scheme is

$$\boldsymbol{\nu} = \boldsymbol{ke} \tag{45}$$

where $\boldsymbol{\nu}$ is the desired end-effector spatial velocity to be used in (44), and the choice and calculation of $\boldsymbol{k}$ and $\boldsymbol{e}$ is detailed in (37)-(43) of Part 1. Figure 2 shows the difference in final joint configuration between RRMC and Park's controller (using the rotational manipulability Jacobian) where both controllers are employed with a PBS controller to define the demanded end-effector velocity to achieve the same goal end-effector pose. Figure 3 compares the rotational manipulability of the controllers throughout the trajectory. We can see that, as opposed to the RRMC controller, the Park controller increases the initial rotational manipulability and maintains the higher value throughout the trajectory.

Null-space projection is not limited to manipulability maximisation. Any subtask which can be expressed as a differentiable function of $\boldsymbol{q}$ can be used, and multiple sub-tasks can be individually weighted and added together to be used as $\dot{\boldsymbol{q}}_{\text{null}}$. For example, Baur [8] used the manipulability Jacobian and an additional weighting on joint positions which discouraged joints from getting too close to their physical limits.

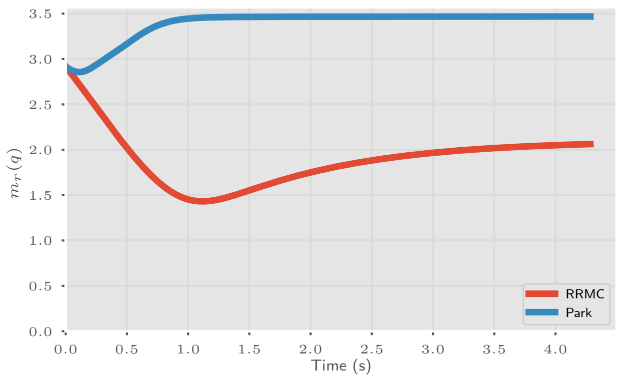


Figure 3. The rotational manipulability of the robot being controlled by the RRMC and Park controllers.

Quadratic Programming

In this section, we are going to redefine our motion controllers as a quadratic programming (QP) optimisation problem rather than a matrix equation. In general, a constrained QP is formulated as [9]

$$\begin{aligned} \min_{x} \quad & f_o(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \mathbf{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x}, \\ \text{subject to} \quad & \mathbf{A}_1\boldsymbol{x} = \boldsymbol{b}_1, \\ & \mathbf{A}_2\boldsymbol{x} \leq \boldsymbol{b}_2, \\ & \boldsymbol{g} \leq \boldsymbol{x} \leq \boldsymbol{h}. \end{aligned} \tag{46}$$

where $f_o(\boldsymbol{x})$ is the objective function which is subject to the equality and inequality constraints, and $\boldsymbol{h}$ and $\boldsymbol{g}$ represent the upper and lower bounds of $\boldsymbol{x}$. A quadratic program is strictly convex when the matrix $\mathbf{Q}$ is positive definite [9]. This framework allows us to solve the same problems as above, but with additional flexibility. In practice, the downside to this QP approach is that it is marginally more complex to implement.

We can rewrite RRMC (38) as a QP

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}} \quad & f_o(\dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^\top \mathbf{1}_n \dot{\boldsymbol{q}}, \\ \text{subject to} \quad & \mathbf{J}\left(\boldsymbol{q}\right)\dot{\boldsymbol{q}} = \boldsymbol{\nu}, \\ & \dot{\boldsymbol{q}}^{-} \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}^{+} \end{aligned} \tag{47}$$

where we have imposed $\dot{\boldsymbol{q}}^{-}$ and $\dot{\boldsymbol{q}}^{+}$ as upper and lower joint-velocity limits. If the manipulator has more degrees of freedom than necessary to reach its entire task space, the QP will achieve the desired end-effector velocity with the minimum joint-velocity norm (the same result as the pseudoinverse solution, (36) in Part 1). If the manipulator has six joints, then the solution will be the same as (35) in Part 1.

We can rewrite Park's controller as

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}} \quad & f_o(\dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^\top \lambda \mathbf{1}_n \dot{\boldsymbol{q}} - \boldsymbol{J}_m^\top(\boldsymbol{q})\dot{\boldsymbol{q}}, \\ \text{subject to} \quad & \mathbf{J}\left(\boldsymbol{q}\right)\dot{\boldsymbol{q}} = \boldsymbol{\nu}. \end{aligned} \tag{48}$$

where the manipulability Jacobian fits into the linear component of the objective function.

This was extended in [10] with velocity dampers to enable joint-position-limit avoidance. Velocity dampers [11] are used to constrain velocities and dampen an input velocity as some position limit is approached. The velocity is only damped if it is within some influence distance of the limit. A joint velocity is constrained to prevent joint limits using a velocity damper constraint

$$\dot{q} \leq \eta \frac{\rho - \rho_s}{\rho_i - \rho_s} \qquad \text{if } \rho < \rho_i \tag{49}$$

where $\rho \in \mathbb{R}^+$ is the distance or angle to the nearest joint limit, $\eta \in \mathbb{R}^+$ is a gain that adjusts the aggressiveness of the damper, ρ_i is the influence distance within which to activate the damper, and ρ_s is the stopping distance in which the distance ρ will never be able to reach or enter. In a robot with mixed joint types see Excurse 1. We can stack velocity dampers to perform joint position limit avoidance for each joint within a robot, and incorporate it into our QP (48) as an inequality constraint

$$\mathbf{1}_n \dot{\boldsymbol{q}} \leq \eta \begin{pmatrix} \frac{\rho_0 - \rho_s}{\rho_i - \rho_s} \\ \vdots \\ \frac{\rho_n - \rho_s}{\rho_i - \rho_s} \end{pmatrix} \tag{50}$$

where the identity $\mathbf{1}_n$ is included to show how the equation fits into the general form $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}$ of an inequality constraint.

It is possible that the robot will fail to reach the goal when the constraints create local minima. In such a scenario, the error $\boldsymbol{e}$ in the PBS scheme is no longer decreasing and the robot can no longer progress due to the constraints in (50).

The methods shown up to this point require a redundant robot, where the number of joints is greater than 6. In [10], extra redundancy was introduced to the QP by relaxing the equality constraint in (50) to allow for intentional deviation, or *slack* from the desired end-effector velocity $\boldsymbol{\nu}$. The slack has the additional benefit of giving the solver extra flexibility to meet constraints and avoid local minima. The augmented QP is defined as

$$\begin{aligned} \min_x \quad & f_o(\boldsymbol{x}) = \frac{1}{2} \boldsymbol{x}^\top \mathcal{Q} \boldsymbol{x} + \mathcal{C}^\top \boldsymbol{x}, \qquad (51) \\ \text{subject to} \quad & \mathcal{J} \boldsymbol{x} = \boldsymbol{\nu}, \\ & \mathcal{A} \boldsymbol{x} \leq \mathcal{B}, \\ & \boldsymbol{x}^- \leq \boldsymbol{x} \leq \boldsymbol{x}^+ \end{aligned}$$

with

$$\boldsymbol{x} = \begin{pmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{\delta} \end{pmatrix} \in \mathbb{R}^{(n+6)} \tag{52}$$

$$\mathcal{Q} = \begin{pmatrix} \lambda_q \mathbf{1}_n & \mathbf{0}_{6 \times 6} \\ \mathbf{0}_{n \times n} & \lambda_\delta \mathbf{1}_6 \end{pmatrix} \in \mathbb{R}^{(n+6) \times (n+6)} \tag{53}$$

$$\mathcal{J} = \begin{pmatrix} \mathbf{J}(\boldsymbol{q}) & \mathbf{1}_6 \end{pmatrix} \in \mathbb{R}^{6 \times (n+6)} \tag{54}$$

$$\mathcal{C} = \begin{pmatrix} \boldsymbol{J}_m \\ \mathbf{0}_{6 \times 1} \end{pmatrix} \in \mathbb{R}^{(n+6)} \tag{55}$$

$$\mathcal{A} = \begin{pmatrix} \mathbf{1}_{n \times n+6} \end{pmatrix} \in \mathbb{R}^{(l+n) \times (n+6)} \tag{56}$$

$$\mathcal{B} = \eta \begin{pmatrix} \frac{\rho_0 - \rho_s}{\rho_i - \rho_s} \\ \vdots \\ \frac{\rho_n - \rho_s}{\rho_i - \rho_s} \end{pmatrix} \in \mathbb{R}^n \tag{57}$$

$$\boldsymbol{x}^{-,+} = \begin{pmatrix} \dot{\boldsymbol{q}}^{-,+} \\ \boldsymbol{\delta}^{-,+} \end{pmatrix} \in \mathbb{R}^{(n+6)}, \tag{58}$$

where $\boldsymbol{\delta} \in \mathbb{R}^6$ is the slack vector, $\lambda_\delta \in \mathbb{R}^+$ is a gain term that adjusts the cost of the norm of the slack vector in the optimiser, $\dot{\boldsymbol{q}}^{-,+}$ are the minimum and maximum joint velocities, and $\boldsymbol{\delta}^{-,+}$ are the minimum and maximum slack velocities. Each of the gains can be adjusted dynamically. For example, in practice λ_δ is typically large when far from the goal, but reduces towards 0 as the goal approaches.

The effect of this augmented optimisation problem is that the equality constraint is equivalent to

$$\boldsymbol{\nu}(t) - \boldsymbol{\delta}(t) = \mathbf{J}(\boldsymbol{q}) \dot{\boldsymbol{q}}(t) \tag{59}$$

which clearly demonstrates that the slack is essentially intentional error, where the optimiser can choose to move components of the desired end-effector motion into the slack vector. For both redundant and non-redundant robots, this means that the robot may stray from the straight-line motion to improve manipulability and avoid a singularity, avoid running into joint position limits or stay bound by the joint velocity limits.

Velocity dampers are further demonstrated in [12] where they are used to incorporate real-time obstacle avoidance into the QP. Furthermore, in [13] the QP framework was extended to allow for holistic differential-kinematic control of a mobile manipulator.

Advanced Inverse Kinematics

In Part 1 of this tutorial, we considered unconstrained numerical inverse kinematics. In this section, we are going to extend this to consider constraints such as joint position limits and introduce some performance measures.

To begin, we will first consider the inverse kinematics solver based on the Newton-Raphson (NR) method which we described in Part 1 as the iteration

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \mathbf{J}(\boldsymbol{q}_k)^+ \boldsymbol{e}_k \tag{60}$$

until the desired end-effector pose is reached, where $\boldsymbol{e}$ is the position and angle-axis error between the current pose and the desired end-effector pose ((37) of Part 1) expressed in the world frame, and $\mathbf{J} = {}^{0}\mathbf{J}$ is the base-frame manipulator Jacobian. The work in [14] redefines (60) as a quadratic program in the form of (46), which is iterated to find a solution. This quadratic programme will find the minimum-norm solution for $\dot{\boldsymbol{q}}$ at each step, which will be the same solution as given by (60).

A naive approach to joint-limit avoidance is to perform a global search (as explained in Part 1) while discarding solutions that exceed iteration limits and joint limits. Alternatively, the popular KDL inverse kinematics solver will not allow joint limits to be exceeded during iteration – they are clamped to the joint limits [15].

We can greatly improve the solvability by using the null-space projection devised by Baur [8]

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \mathbf{J}(\boldsymbol{q}_k)^{+}\boldsymbol{e}_k + \boldsymbol{q}_{null} \tag{61}$$

$$\boldsymbol{q}_{null} = \Big(\big(\mathbf{1}_n - \mathbf{J}(\boldsymbol{q})^{+}\mathbf{J}(\boldsymbol{q})\big) \Big(\frac{1}{\lambda_\Sigma}\boldsymbol{\Sigma}\Big)\Big) \tag{62}$$

$$\boldsymbol{\Sigma} = \begin{cases} \sum_{i=1}^{n} \dfrac{(q_i - \bar{q}_{M_i})^2}{(q_{M_i} - \bar{q}_{M_i})^2} & q_i \geq \bar{q}_{M_i} \\ \sum_{i=1}^{n} \dfrac{(q_i - \bar{q}_{m_i})^2}{(q_{m_i} - \bar{q}_{m_i})^2} & q_i \leq \bar{q}_{m_i} \\ 0 & \text{otherwise} \end{cases} \tag{63}$$

where the maximum and minimum joint angles are specified by q_{M_i} and q_{m_i} respectively, while the maximum and minimum joint angle threshold are specified by $\bar{q}_{M_i}$ and $\bar{q}_{m_i}$. Once this threshold is passed, further progress toward the joint limit is penalised by $\boldsymbol{\Sigma}$. While this addition can help avoid joint limits, it does not guarantee joint-limit avoidance. The term $\lambda_\Sigma \in \mathbb{R}^+$ is a gain that adjusts how aggressively the joint limit is avoided. Furthermore, we can add the manipulability Jacobian to the null-space term as we did in (44)

$$\boldsymbol{q}_{null} = \Big(\big(\mathbf{1}_n - \mathbf{J}(\boldsymbol{q})^{+}\mathbf{J}(\boldsymbol{q})\big) \Big(\frac{1}{\lambda_\Sigma}\boldsymbol{\Sigma} + \frac{1}{\lambda_m}\mathbf{J}_m(\boldsymbol{q})\Big)\Big) \tag{64}$$

where $\lambda_m \in \mathbb{R}^+$ is a gain that adjusts how aggressively the manipulability is to be maximised.

In Part 1, we showed that the Levenberg-Marquardt (LM) method

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + (\mathbf{A}_k)^{-1}\boldsymbol{g}_k \tag{65}$$

$$\mathbf{A}_k = \mathbf{J}(\boldsymbol{q}_k)^{\top}\mathbf{W}_e\,\mathbf{J}(\boldsymbol{q}_k) + \mathbf{W}_n \tag{66}$$

$$\boldsymbol{g}_k = \mathbf{J}(\boldsymbol{q}_k)^{\top}\mathbf{W}_e\boldsymbol{e}_k \tag{67}$$

provided much better results for inverse kinematics than the NR method, where $\mathbf{W}_n = \text{diag}(\boldsymbol{w}_n)\big(\boldsymbol{w}_n \in (\mathbb{R}^+)^n\big)$ is a diagonal damping matrix. In Part 1, we detailed the choice of $\mathbf{W}_n$ based on proposals by Wampler [16], Chan [17], and Sugihara [18].

As with the NR method, we can naively perform joint limit avoidance with the LM method through global search, and discard solutions that exceed iteration limits and joint limits. We can improve this approach by augmenting the vector $\boldsymbol{g}_k$ with the same null-space motion defined in (62)

$$\boldsymbol{g}_k = \mathbf{J}(\boldsymbol{q}_k)^{\top}\mathbf{W}_e\boldsymbol{e}_k + \boldsymbol{q}_{null}. \tag{68}$$

The addition of null-space motion provides much better results for inverse kinematics when trying to avoid joint limits, but it is only available on redundant robots. For improved constrained inverse kinematics, we can use the augmented QP with slack from (51) with

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \dot{\boldsymbol{q}}. \tag{69}$$

To increase the likelihood of finding an IK solution, the TRAC-IK algorithm [19], which is used as the default IK solver for the popular robotics software package Moveit [20], runs two IK solvers in parallel. The first is the NR method shown in (60), and the second is an NR method redefined as a quadratic programme with a custom error metric.

Inverse Kinematics Comparison

We show a comprehensive comparison of various numerical inverse kinematics solvers on three different types of robots. For each robot, the IK algorithm attempts to reach 10 000 randomly generated valid end-effector poses and results are summarised in Tables 1-3. All methods use a global search with a 30 iteration limit within a search, and a maximum of 100 searches. The Infeasible Count column reports how many solutions failed to converge after 100 searches – zero is best. For the LM Wampler method, we use $\lambda = 1e-4$. For the LM Sugihara method, we use $w_n = 0.001$. Methods with a $(\cdot)^+$ indicate that solutions with a joint-limit violation are treated as a failure and another attempt is performed. Methods marked with $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$ have joint limit avoidance projected into the null-space using (62). Methods marked with $\boldsymbol{q}_{null}(\boldsymbol{\Sigma}, \mathbf{J}_m)$ have joint limit avoidance and manipulability maximisation projected into the null-space using (64). For the QP method, the symbols $\boldsymbol{\Sigma}$ and $\mathbf{J}_m$ indicate joint limit avoidance and manipulability maximisation have been incorporated respectively. Note that null-space methods can not be used on the UR5 manipulator as it is not redundant. The time per iteration represents the average of how long one iteration of the corresponding method took relative to the fastest method iteration within the Table. The rightmost column is the relative number of iterations to find a solution

Table 1: Numerical IK Methods Compared over 10 000 Problems on a 6 DoF UR5 Manipulator

Method	Mean Iter.	Median Iter.	Infeasible Count	Mean Searches	Max Searches	Joint Limit Violations	Time per Iter.	Median Time
NR	27.96	16.0	0	1.44	25.0	0	1.68	26.9
LM (Chan)	15.52	8.0	0	1.21	14.0	0	1.04	8.35
LM$^+$ (Wampler)	23.75	13.0	0	1.35	20.0	0	1.0	13.0
LM$^+$ (Chan)	15.52	8.0	0	1.21	14.0	0	1.04	8.29
LM$^+$ (Sugihara)	21.89	13.0	0	1.27	19.0	0	1.07	13.97
QP ($\boldsymbol{J}_m$)	15.93	8.0	0	1.22	13.0	0	3.12	24.99

Table 2: Numerical IK Methods Compared over 10 000 Problems on a 7 DoF Panda Manipulator

Method	Mean Iter.	Median Iter.	Infeasible Count	Mean Searches	Max Searches	Joint Limit Violations	Time per Iter.	Median Time
NR	27.88	16.0	0	1.43	12.0	6705	1.71	27.35
LM (Chan)	11.91	8.0	0	1.12	7.0	5394	1.06	8.46
NR$^+$	139.56	80.0	104	7.5	100.0	0	1.54	123.03
LM$^+$ (Wampler)	127.61	76.0	102	7.11	98.0	0	1.0	76.0
LM$^+$ (Chan)	37.55	18.0	91	3.81	86.0	0	1.03	18.54
LM$^+$ (Sugihara)	50.13	26.0	89	3.64	76.0	0	1.07	27.81
NR$^+$ $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$	347.7	219.0	254	15.88	99.0	0	3.02	661.59
LM$^+$ (Wampler) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$	353.84	196.0	190	14.02	99.0	0	2.67	523.24
LM$^+$ (Chan) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$	37.4	18.0	91	3.79	86.0	0	2.73	49.16
LM$^+$ (Sugihara) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$	44.63	24.0	99	2.85	97.0	0	2.77	66.43
NR$^+$ $\boldsymbol{q}_{null}(\boldsymbol{\Sigma}, \boldsymbol{J}_m)$	232.16	132.0	135	10.19	99.0	0	4.68	618.41
LM$^+$ (Wampler) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma}, \boldsymbol{J}_m)$	178.22	103.0	105	8.58	100.0	0	4.23	435.74
LM$^+$ (Chan) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma}, \boldsymbol{J}_m)$	37.33	18.0	90	3.77	86.0	0	4.25	76.53
LM$^+$ (Sugihara) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma}, \boldsymbol{J}_m)$	49.55	26.0	89	3.6	97.0	0	4.29	111.5
QP ($\boldsymbol{\Sigma}, \boldsymbol{J}_m$)	42.42	14.0	76	2.12	86.0	0	3.95	55.36

Table 3: Numerical IK Methods Compared over 10 000 Problems on a 13 DoF (waist, arm & index finger) Valkyrie Humanoid

Method	Mean Iter.	Median Iter.	Infeasible Count	Mean Searches	Max Searches	Joint Limit Violations	Time per Iter.	Median Time
LM (Chan)	6.31	6.0	0	1.0	1.0	9542	1.28	7.69
NR$^+$	285.88	235.0	2791	34.57	100.0	0	1.38	323.14
LM$^+$ (Chan)	156.13	98.0	1765	25.22	100.0	0	1.0	98.0
NR$^+$ $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$	82.6	37.0	109	6.82	100.0	0	2.97	109.88
LM$^+$ (Wampler) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$	82.43	37.0	109	6.8	100.0	0	2.73	101.01
LM$^+$ (Chan) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$	28.18	15.0	56	2.11	95.0	0	2.87	43.03
LM$^+$ (Sugihara) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma})$	25.59	13.0	50	1.79	100.0	0	2.93	38.15
LM$^+$ (Chan) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma}, \boldsymbol{J}_m)$	28.69	15.0	60	2.14	99.0	0	4.63	69.4
LM$^+$ (Sugihara) $\boldsymbol{q}_{null}(\boldsymbol{\Sigma}, \boldsymbol{J}_m)$	24.86	13.0	56	1.74	91.0	0	4.7	61.16
QP ($\boldsymbol{\Sigma}, \boldsymbol{J}_m$)	15.29	7.0	0	1.27	18.0	0	4.19	29.3

based on the time per iteration and the median number of iterations.

Table 1 displays the results on a six degree-of-freedom UR5 manipulator where each joint can articulate $\pm 180°$ – this means that for each joint, every angle is achievable, which is a larger range than most other manipulators. Table 2 displays the results on a seven degree-of-freedom Panda manipulator where each joint has unique articulation limits, more typical of other manipulators. Table 3 displays the results on a thirteen degree-of-freedom kinematic chain within the Valkyrie humanoid robot involving the waist, right shoulder, right arm, and right index finger. Several joint limits within this kinematic chain are quite small, with some having a total range of $20°$.

This experiment, running on non-redundant, redundant, and hyper-redundant robots with wide, medium, and narrow joint limits respectively, exposes the key differences between each IK algorithm along with some strengths and weaknesses. The first section of Tables 2 and 3, on robots with joints that have unachievable coordinates, shows many solutions with joint limit violations. This shows that, when not constrained otherwise, the IK solvers will converge on a solution that violates the joint limits of the robot the majority of the time. The Chan, Wampler, and Sugihara solvers are consistently the fastest per step. The time cost increases as we add extra functionality, such as active joint-limit avoidance and manipulability maximisation. The QP IK solver is clearly the most robust, uses the least iterations in the median, and will use fewer attempts to find a solution, but is one of the slowest algorithms presented. As more degrees-of-freedom are added and the joint range becomes narrower, the QP solver improves in solution speed relative to the other solvers.

On the UR5, where every joint angle is achievable, Chan's method without any null-space terms is both the fastest and most reliable. Interestingly, the base Chan and Sugihara methods on the Panda slightly outperform the Chan and Sugihara methods with joint limit avoidance, and manipulability maximisation. Although the latter methods will typically arrive at a solution in fewer iterations, the extra computation time makes the median time much worse than the base methods. On the Panda, both the base Chan and the QP solver provide the best overall results. On the Valkyrie, the QP solver clearly outperforms the other solvers, even when accounting for the additional time per step.

Singularity Escapability

A robot can lose degrees of freedom if a joint is at its limit, the arm is fully extended or when two or more joint axes align. In the last failure case, the manipulator is in a singular configuration.

At a singularity, the manipulator Jacobian becomes rank deficient, and when approaching a singularity, the Jacobian becomes ill-conditioned. For Jacobian-based motion controllers, the demanded joint velocities calculated from the Jacobian inverse will approach infinity as the singularity is approached. The Moore-Penrose pseudoinverse is a common approach to avoiding this issue, however the performance is not reliable in all cases. There have been several works that, using the manipulator Jacobian and Hessian, can determine if a singularity is escapable, and if so, which joints should be actuated to do so [21]–[23]. In the remainder of this tutorial, we detail a quadratic-rate motion control that can control a robot when away from, near to, or even at a singularity.

Quadratic-Rate Motion Control

Quadratic-rate motion control is a method of controlling a manipulator through or near a singularity [24]. By using the second-order differential kinematics, the controller does not break down at or near a singularity, as the resolved-rate motion controller would. We start by considering the end-effector pose $\boldsymbol{x} = f(\boldsymbol{q}) \in \mathbb{R}^6$ given by the forward kinematics. Introducing a small change to the joint coordinates $\Delta \boldsymbol{q}$, we can write

$$\boldsymbol{x} + \Delta \boldsymbol{x} = f(\boldsymbol{q} + \Delta \boldsymbol{q}). \tag{70}$$

and the Taylor series expansion is

$$\begin{aligned} \boldsymbol{x} + \Delta \boldsymbol{x} &= f(\boldsymbol{q}) + \frac{\partial f}{\partial \boldsymbol{q}} \Delta \boldsymbol{q} + \frac{1}{2}\left(\frac{\partial^2 f}{\partial \boldsymbol{q}^2} \Delta \boldsymbol{q}\right) \Delta \boldsymbol{q} + \dots \\ \Delta \boldsymbol{x} &= \mathbf{J}(\boldsymbol{q}) \Delta \boldsymbol{q} + \frac{1}{2}\Big(\mathbf{H}(\boldsymbol{q}) \Delta \boldsymbol{q}\Big) \Delta \boldsymbol{q} + \dots \end{aligned} \tag{71}$$

where for quadratic-rate control we wish to retain both the linear and quadratic terms (the first two terms) of the expansion.

To form our controller, we use an iteration-based Newton-Raphson approach to solve for $\Delta \boldsymbol{q}$ in (71). Firstly, we rearrange (71) as

$$\boldsymbol{g}(\Delta \boldsymbol{q}_k) = \mathbf{J}(\boldsymbol{q}_k) \Delta \boldsymbol{q}_k + \frac{1}{2}\Big(\mathbf{H}(\boldsymbol{q}_k) \Delta \boldsymbol{q}_k\Big) \Delta \boldsymbol{q}_k - \Delta \boldsymbol{x} = 0 \tag{72}$$

where $\boldsymbol{q}_k$ and $\Delta \boldsymbol{q}_k$ are the manipulator's current joint coordinates and change in joint coordinates respectively, and $\Delta \boldsymbol{x}$ is the desired change in end-effector position. Taking the derivative of $\boldsymbol{g}$ with respect to $\Delta \boldsymbol{q}$

$$\begin{aligned} \frac{\partial \boldsymbol{g}(\Delta \boldsymbol{q}_k)}{\partial \Delta \boldsymbol{q}} &= \mathbf{J}(\boldsymbol{q}_k) + \mathbf{H}(\boldsymbol{q}_k) \Delta \boldsymbol{q}_k \\ &= \hat{\mathbf{J}}(\boldsymbol{q}_k, \Delta \boldsymbol{q}_k) \end{aligned} \tag{73}$$

we obtain a new Jacobian $\hat{\mathbf{J}}(\boldsymbol{q}_k, \Delta\boldsymbol{q}_k)$. Using this Jacobian we can create a new linear system

$$\begin{aligned} \hat{\mathbf{J}}(\boldsymbol{q}_k, \Delta\boldsymbol{q}_k)\boldsymbol{\delta}_{\Delta\boldsymbol{q}} &= \boldsymbol{g}(\Delta\boldsymbol{q}_k) \\ \boldsymbol{\delta}_{\Delta\boldsymbol{q}} &= \hat{\mathbf{J}}(\boldsymbol{q}_k, \Delta\boldsymbol{q}_k)^{-1}\boldsymbol{g}(\Delta\boldsymbol{q}_k) \end{aligned} \tag{74}$$

where $\boldsymbol{\delta}_{\Delta\boldsymbol{q}}$ is the update to $\Delta\boldsymbol{q}_k$. The change in joint coordinates at the next step are

$$\Delta\boldsymbol{q}_{k+1} = \Delta\boldsymbol{q}_k - \boldsymbol{\delta}_{\Delta\boldsymbol{q}}. \tag{75}$$

In the case that $\Delta\boldsymbol{q} = \mathbf{0}$, quadratic-rate control reduces to resolved-rate control. This is not suitable if the robot is in or near a singularity. Therefore, a value near 0 may be used to seed the initial value with $\Delta\boldsymbol{q} = (0.1, \cdots, 0.1)$, or the pseudo-inverse approach could be used

$$\Delta\boldsymbol{q} = \mathbf{J}(\boldsymbol{q})^{+}\Delta\boldsymbol{x}. \tag{76}$$

This controller can be used in the same manner as resolved-rate motion control described in Part 1. The manipulator Jacobian and Hessian are calculated in the robot's base frame, and the end-effector velocity $\Delta\boldsymbol{x}$ can be calculated using the angle-axis method described in Part 1.

As quadratic-rate motion control is another form of advanced velocity control, many of the techniques described in the Advanced Velocity Control section can also be applied here. It may also be adapted into an inverse kinematics algorithm using any technique described in the Inverse Kinematics section in Part 1, or the Advanced Inverse Kinematics section in this article.

In the literature [21]–[24], the manipulator Hessian is reported as taking too long to compute to be used in closed-loop control. However, with the approach detailed in this article and the implementation provided by the Robotics Toolbox for Python, the manipulator Hessian can be computed in less than 1 μs [2]. Therefore, in the current day, that argument is no longer valid.

Acknowledgments

We are grateful to the anonymous reviewers whose detailed and insightful comments have improved this article. This research was conducted by the Australian Research Council project number CE140100016, and supported by the QUT Centre for Robotics (QCR). We would also like to thank the members of QCR who provided valuable feedback and insights while testing this tutorial and associated Jupyter Notebooks.

Conclusion

In this tutorial's final instalment, we have covered many advanced aspects of manipulator differential kinematics. We first described a procedure for describing any manipulator's second-and higher-order differential kinematics. We then detailed how differential kinematics can be translated into analytical forms, which is highly advantageous for task-space dynamics applications. We detailed how numerical inverse kinematics solvers can be extended to avoid joint limits and maximise manipulability. Finally, we described how the manipulator Hessian is used to create quadratic-rate motion control, a controller which will work when the robot is in, or near, a singularity.

This tutorial is not exhaustive, and many important and useful techniques have not been visited. For example, resolved-acceleration control can be used for trajectory following, where the required velocities and accelerations are known beforehand. Additionally, task-space control approaches such as operational-space control make strong use of differential kinematics.

References

[1] J. Haviland and P. Corke, “Manipulator differential kinematics: Part 1: Kinematics, velocity, and applications,” *IEEE Robotics & Automation Magazine*, pp. 2–12, 2023.

[2] P. Corke and J. Haviland, “Not your grandmother's toolbox–the robotics toolbox reinvented for Python,” in *2021 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2021.

[3] P. Corke, *Robotics, Vision and Control: fundamental algorithms in Python*, 3rd ed. Springer Nature, 2023.

[4] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation,” *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[5] J. Park, C. Wangkyun, and Y. Youngil, “Computation of gradient of manipulability for kinematically redundant manipulators including dual manipulators system,” *Transactions on Control, Automation and Systems Engineering*, vol. 1, no. 1, pp. 8–15, 1999.

[6] T. Yoshikawa, “Manipulability of Robotic Mechanisms,” *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.

[7] L. J. Stocco, S. E. Salcudean, and F. Sassani, “On the use of scaling matrices for task-specific robot design,” *IEEE Transactions on Robotics and Automation*, vol. 15, no. 5, pp. 958–965, 1999.

[8] J. Baur, J. Pfaff, H. Ulbrich, and T. Villgrattner, “Design and development of a redundant modular multipurpose agricultural manipulator,” in *2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2012, pp. 823–830.

[9] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[10] J. Haviland and P. Corke, “A purely-reactive manipulability-maximising motion controller,” *arXiv preprint arXiv:2002.11901*, 2020.

[11] B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom," in *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4, 1987, pp. 1152–1159.

[12] J. Haviland and P. Corke, "NEO: A novel expeditious optimisation algorithm for reactive motion control of manipulators," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1043–1050, 2021.

[13] J. Haviland, N. Sünderhauf, and P. Corke, "A holistic approach to reactive mobile manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3122–3129, 2022.

[14] S. Kumar, N. Sukavanam, and R. Balasubramanian, "An optimization approach to solve the inverse kinematics of redundant manipulator," *International Journal of Information and System Sciences (Institute for Scientific Computing and Information)*, vol. 6, no. 4, pp. 414–423, 2010.

[15] R. Smits. (2022). "KDL: Kinematics and Dynamics Library," [Online]. Available: `http://www.orocos.org/kdl`.

[16] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 93–101, 1986.

[17] S. K. Chan and P. D. Lawrence, "General inverse kinematics with the error damped pseudoinverse," in *Proceedings. 1988 IEEE international conference on robotics and automation*, IEEE, 1988, pp. 834–839.

[18] T. Sugihara, "Solvability-unconcerned inverse kinematics by the levenberg–marquardt method," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 984–991, 2011.

[19] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2015, pp. 928–935.

[20] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[21] N. S. Bedrossian, "Classification of singular configurations for redundant manipulators," in *Proceedings., IEEE International Conference on Robotics and Automation*, IEEE, 1990, pp. 818–823.

[22] N. Bedrossian and K. Flueckiger, "Characterizing spatial redundant manipulator singularities.," in *ICRA*, 1991, pp. 714–719.

[23] J. Seng, K. A. O'Neil, and Y.-C. Chen, "Escapability of singular configuration for redundant manipulators via self-motion," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, IEEE, vol. 3, 1995, pp. 78–83.

[24] E. D. Pohl and H. Lipkin, "A new method of robotic rate control near singularities," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, IEEE Computer Society, 1991, pp. 1708–1709.