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 that 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 exposes a relationship between the robot’s joint velocities and the end-effector acceleration. We then describe the analytical forms of differential kinematics, 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 the resolved-rate motion control (RRMC) to perform additional subtasks 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 analyze how incorporating the manipulator Hessian into a motion controller can help to escape singularities.

In Part 1 [1], we described a method of modeling kinematics using the elementary transform sequence 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
DERIVING THE MANIPULATOR HESSIAN

THE MANIPULATOR HESSIAN

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

\[ 0T_e(t) = K(q(t)) = \prod_{i=1}^{M} E_i(\eta_i) \quad (1) \]

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

\[ v = \left( \begin{matrix} v \\ \omega \end{matrix} \right) = J(q)q \quad (2) \]

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

\[ \left( \begin{matrix} \dot{v} \\ \dot{\omega} \end{matrix} \right) = \left( \begin{matrix} \dot{J} \\ \dot{\omega} \end{matrix} \right) = \dot{J}q + J\dot{q} \quad (3) \]

where \( \dot{J} \in \mathbb{R}^{6 \times 3} \) is the end-effector translational acceleration; \( \dot{\omega} \in \mathbb{R}^{3} \) is the end-effector angular acceleration; and

\[ \dot{J} = \frac{dJ(q)}{dq}, \quad \dot{\omega} = \frac{d\omega(q)}{dq} \]

where \( J(q) \) is the Jacobian of the manipulator.

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

\[ \frac{dT_i}{dq_j} = \frac{\partial T_i(q)}{\partial q_j} = \frac{\partial}{\partial q_j} \left( E_1(\eta_1)E_2(\eta_2) \cdots E_M(\eta_M) \right) \]

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:

\[ \frac{d^2T_{\mu(j)}(q_{\mu(j)})}{dq_{\mu(k)}dq_{\mu(l)}} = \begin{cases} \frac{\partial^2}{\partial q_{\mu(k)} \partial q_{\mu(l)}} \left( \prod_{i=1}^{\mu(j)-1} E_i(\eta_i) \frac{dE}{dq_{\mu(i)(q_{\mu(i)})}} \frac{dE}{dq_{\mu(i)(q_{\mu(i)})}} \frac{dE}{dq_{\mu(i)(q_{\mu(i)})}} \cdots \frac{dE}{dq_{\mu(i)(q_{\mu(i)})}} \prod_{i=\mu(j)+1}^{M} E_i(\eta_i) \right) \\ if \ k < j \\ \frac{\partial^2}{\partial q_{\mu(k)} \partial q_{\mu(l)}} \frac{dE_{\mu(k)}}{dq_{\mu(k)}} \frac{dE_{\mu(k)}}{dq_{\mu(k)}} \frac{dE_{\mu(k)}}{dq_{\mu(k)}} \cdots \frac{dE_{\mu(k)}}{dq_{\mu(k)}} \prod_{i=\mu(k)+1}^{M} E_i(\eta_i) \right) \\ if \ k = j \\ \frac{\partial^2}{\partial q_{\mu(k)} \partial q_{\mu(l)}} \frac{dE_{\mu(k)}}{dq_{\mu(k)}} \frac{dE_{\mu(k)}}{dq_{\mu(k)}} \frac{dE_{\mu(k)}}{dq_{\mu(k)}} \cdots \frac{dE_{\mu(k)}}{dq_{\mu(k)}} \prod_{i=\mu(k)+1}^{M} E_i(\eta_i) \right) \\ if \ k > j. \end{cases} \]

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

$$\frac{d^2 T_{jk}(\theta)}{d\theta^2} = \hat{R}_j \hat{T}_{jk}(\theta)$$  

(13)

$$\frac{d^2 T_{jk}(\theta)}{d\theta^2} = \hat{R}_j \hat{T}_{jk}(\theta)$$  

(14)

$$\frac{d^2 T_{jk}(\theta)}{d\theta^2} = \hat{R}_j \hat{T}_{jk}(\theta)$$  

(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 $H_{a,j}$, 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

$$J_{ai}(q) = \nabla_x\left(\rho\left(\frac{\partial T(q)}{\partial q_j}\right)\rho(T(q))\right)^T$$  

(16)

using the product rule

$$H_{a,j} = \frac{\partial J_{ai}(q)}{\partial q_k} = \nabla_x\left(\rho\left(\frac{\partial^2 T(q)}{\partial q_j \partial q_k}\right)\rho(T(q)) \rho\left(\frac{\partial T(q)}{\partial q_j}\right)^T + \rho\left(\frac{\partial T(q)}{\partial q_k}\right)\rho\left(\frac{\partial^2 T(q)}{\partial q_j \partial q_k}\right)^T\right)$$  

(17)

where $H_{a,k} \in \mathbb{R}^3$, and $H_{a,k}$ is obtained from (6).

To form $H_{i,j}$, 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

$$J_{ji}(q) = \rho\left(\frac{\partial T(q)}{\partial q_j}\right)^T$$  

(18)

which provides

$$H_{i,j} = \frac{\partial J_{ji}(q)}{\partial q_k} = \nabla_x\left(\rho\left(\frac{\partial^2 T(q)}{\partial q_j \partial q_k}\right)^T\right)$$  

(19)

where $H_{i,j} \in \mathbb{R}^3$, and $H_{i,k}$ 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 $k$:

$$H_{ji} = \begin{pmatrix} H_{a,j} \\ H_{i,j} \end{pmatrix}$$  

(20)

where $H_{ji} \in \mathbb{R}^6$.

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

$$H_j = (H_{ji} \cdots H_{ji})$$  

(21)

where $H_j \in \mathbb{R}^{6 \times n}$.

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

$$H = (H_1 \cdots H_n)$$  

(22)

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

**FAST MANIPULATOR HESSIAN**

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

We revisit (17) while substituting in

$$\frac{dR(\theta)}{d\theta} = \hat{\omega}_j \cdot R(\theta(i))$$  

(23)

and simplify:

$$H_{a,j} = \nabla_x\left(\hat{\omega}_j \cdot [\hat{\omega}_j]_x R(q) [\hat{\omega}_j]_x^T + [\hat{\omega}_j]_x R(q) ([\hat{\omega}_j]_x R(q))^T \right)$$  

(24)

where $[\hat{\omega}_j]_x \hat{\omega}_j \hat{\omega}_j \hat{\omega}_j \hat{\omega}_j \hat{\omega}_j \hat{\omega}_j$ corresponds to the linear acceleration, while the bottom three rows $H_i$ correspond to the angular acceleration, of the end-effector $\alpha$ caused by the velocities of the joints.
Since we know that \( \mathbf{J}_{\alpha_i} = [\partial \mathbf{J}]/[\partial \alpha_i] \), and using the identity 
\[ [a \times b]_x = [a]_x [b]_x - [b]_x [a]_x \]
we show that
\[
\mathbf{H}_{\alpha_i} = \nabla \times \left( [\mathbf{J}_{\alpha_i}]_x [\mathbf{J}_{\alpha_i}]_x - [\mathbf{J}_{\alpha_i}]_x [\mathbf{J}_{\alpha_i}]_x \right)
= \mathbf{J}_{\alpha_i} \times \mathbf{J}_{\alpha_i}
\] (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, \( \mathbf{H}_{\alpha_j} = 0 \) when \( k \geq j \).

For the translational component of the manipulator Hessian \( \mathbf{H}_{\alpha_i} \), 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
\[
\mathbf{H}_{\alpha_i}(\mathbf{q}) = \mathbf{H}_{\alpha_i}(\mathbf{q})
\] (26)
and, by exploiting this relationship, we can simplify (19) to
\[
\mathbf{H}_{\alpha_i}(\mathbf{q}) = [\mathbf{J}_{\alpha_i}]_x [\mathbf{J}_{\alpha_i}]
= \mathbf{J}_{\alpha_i} \times \mathbf{J}_{\alpha_i}
\] (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**

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 the partitioned form.

For example, to obtain the third partial derivative, we take the partial derivative of the manipulator Hessian with respect to the joint coordinates in its partitioned form
\[
\frac{\partial \mathbf{H}_{\alpha_k}(\mathbf{q})}{\partial q_l} = \left( \frac{\partial \mathbf{H}_{\alpha_k}(\mathbf{q})}{\partial q_i} \right)_l = \left( \frac{\partial \mathbf{H}_{\alpha_k}(\mathbf{q})}{\partial q_i} \right)_l = \frac{\partial (\mathbf{J}_{\alpha_k} \times \mathbf{J}_{\alpha_k})}{\partial q_i}
\]
\[
= \left( \frac{\partial \mathbf{J}_{\alpha_k}}{\partial q_i} \right)_l \times \mathbf{J}_{\alpha_k} + \mathbf{J}_{\alpha_k} \times \left( \frac{\partial \mathbf{J}_{\alpha_k}}{\partial q_i} \right)_l
\]
\[
= \left( \mathbf{J}_{\alpha_k} \times \mathbf{J}_{\alpha_k} \right) + \left( \mathbf{J}_{\alpha_k} \times \mathbf{J}_{\alpha_k} \right)
\] (28)
where
\[
\frac{\partial \mathbf{H}_{\alpha_k}(\mathbf{q})}{\partial q_l} \in \mathbb{R}^6.
\]
Continuing, we obtain the following:
\[
\frac{\partial \mathbf{H}_{\alpha_k}(\mathbf{q})}{\partial \mathbf{q}} = \left( \frac{\partial \mathbf{H}_{\alpha_k}(\mathbf{q})}{\partial q_0} \ldots \frac{\partial \mathbf{H}_{\alpha_k}(\mathbf{q})}{\partial q_n} \right)
\] (29)
where
\[
\frac{\partial \mathbf{H}_{\alpha_k}(\mathbf{q})}{\partial q_l} \in \mathbb{R}^{6 \times n}
\]
and, finally,
\[
\frac{\partial \mathbf{H}(\mathbf{q})}{\partial q} = \left( \frac{\partial \mathbf{H}_{\alpha_0}(\mathbf{q})}{\partial q} \ldots \frac{\partial \mathbf{H}_{\alpha_n}(\mathbf{q})}{\partial q} \right)
\] (30)
where
\[
\frac{\partial \mathbf{H}(\mathbf{q})}{\partial q} \in \mathbb{R}^{6 \times n \times 6 \times n}
\]
is the 4D tensor representing the third partial derivative of the manipulator kinematics.

We have included a function as part of our open source Robotics Toolbox for Python [2] that can calculate the \( n \)th partial derivative of the manipulator kinematics. Note that the function has \( \mathcal{O}(n^{\text{order}}) \) time complexity, where \( \text{order} \) 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 \( \mathbf{v} \) and angular velocity \( \omega \) 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 \( \mathbf{\Gamma} = (\alpha, \beta, \gamma) \), the resulting rotation matrix is
\[
\mathbf{R} = \mathbf{R}_z(\gamma) \mathbf{R}_x(\beta) \mathbf{R}_z(\alpha)
\]
\[
= \begin{bmatrix}
    c\beta c\alpha & c\beta s\alpha & c\beta \\
    s\gamma c\alpha + c\beta s\gamma & -s\beta s\gamma + c\gamma c\alpha & -c\beta s\gamma \\
    s\gamma s\alpha - c\gamma c\beta & c\gamma s\beta & c\beta c\gamma
\end{bmatrix}
\] (32)
where $s\theta$ and $c\theta$ are short for $\sin(\theta)$ and $\cos(\theta)$. From Part 1, we have the relationship

$$R = [\omega]_x R$$

$$\dot{R} R^T = [\omega]_x$$

$$\forall \gamma \in [\omega]_x$$

which gives us the result

$$\omega = \left( \begin{array}{c} \dot{\alpha} \\ \dot{\beta} \\ 0 \end{array} \right) = \left( \begin{array}{c} s\gamma c\beta + c\gamma b \theta \\ -c\gamma s\beta + s\gamma b \theta \\ 0 \end{array} \right)$$

$$= \left( \begin{array}{c} s\gamma \\ c\gamma \\ 0 \end{array} \right)$$

$$= A(\Gamma) \dot{\Gamma}$$

(34)

where $A(\Gamma)$ 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

$$J_A(q)(q) = J_r(\Gamma) J(q)$$

$$= \left( \begin{array}{c} 1_{3x3} \\ 0_{3x3} \end{array} \right) \left( \begin{array}{c} A^{-1}(\Gamma) \end{array} \right) J(q).$$

(35)

In the case where $\beta = \pm 90^\circ$, $A$ will be singular, and its inverse does not exist. Therefore, for applications involving 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 $J_A(q)$ is typically used in applications that have a task-space acceleration term. We can obtain $\dot{J}_A(q)$ from (35) using the product rule:

$$\dot{J}_A(q) = \frac{dJ_r(\Gamma)}{dt} J(q) + J_r(\Gamma) \frac{dJ(q)}{dt}$$

$$= \frac{dJ_r(\Gamma)}{dt} J(q) + J_r(\Gamma) (H(q) \dot{q})$$

(36)

where the derivative of the augmented Jacobian $J_r(\Gamma)$ is

$$\frac{dJ_r(\Gamma)}{dt} = \left( \begin{array}{c} 0_{3x3} \\ 0_{3x3} \end{array} \right) \left( \begin{array}{c} A^{-1}(\Gamma) \end{array} \right).$$

(37)

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

**ADVANCED VELOCITY CONTROL**

Many modern manipulators are redundant—they have more than 6 degrees of freedom (DoF). We can exploit this redundancy by having the robot optimize some performance measure while still achieving the original goal. In this section, we start with the RRMC algorithm explained in Part 1 of this tutorial:

$$\dot{q} = J(q)^T v.$$

(38)

**NULL-SPACE PROJECTION**

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

$$\dot{q} = J(q)^T v + (1 - J(q)^T J(q)) \dot{q}_{null}$$

(39)

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

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

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

$$m(q) = \sqrt{\det(J(q) J(q)^T)}$$

(40)

where $J(q) \in \mathbb{R}^{3 \times n}$ is either the translational or rotational rows of $J(q)$ causing $m(q)$ to describe the corresponding component of manipulability. “Excursus 1: Mixed-Joint Manipulators” highlights considerations for manipulators with mixed joint types.

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

$$\frac{d}{dt} \frac{m(t)}{dt} = \frac{1}{2} \frac{m(t)}{dt} \det(J(q) J(q)^T)$$

(41)

we can write this as

$$\dot{m} = J_r(q) \dot{q}$$

(42)

**Excursus 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 that the performance is acceptable, or a scaling approach must be used [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.
where

\[
J_m(q) = m \begin{bmatrix}
\text{vec}(\dot{J}(q)\dot{H}_1(q)^T) \\
\text{vec}(\dot{J}(q)\dot{H}_2(q)^T) \\
\vdots \\
\text{vec}(\dot{J}(q)\dot{H}_6(q)^T)
\end{bmatrix} \text{vec}(\langle J(q)\dot{J}(q)^T \rangle^{-1})
\]

is the manipulability Jacobian \( J_m \in \mathbb{R}^n \) and where the vector operation \( \text{vec}(\cdot) : \mathbb{R}^{n \times b} \rightarrow \mathbb{R}^b \) converts a matrix columnwise into a column vector, and \( \dot{H}_i \in \mathbb{R}^{3 \times a} \) is the translational or rotational component [matching the choice of \( J(q) \)] of \( \dot{H}_i \in \mathbb{R}^{6 \times a} \), which is the \( i \)th component of the manipulator Hessian tensor \( H_i \in \mathbb{R}^{n \times 6 \times a} \).

The complete equation proposed by Park et al. [5] is

\[
\dot{q} = J(q)^T \nu + \frac{1}{\lambda}((1_n - J(q)^TJ(q))J_m(q))
\]

where \( \lambda \) is a gain that scales the magnitude of the null-space velocities. This equation will choose joint velocities \( \dot{q} \) which will achieve the end-effector spatial velocity \( \nu \) 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 toward some desired pose. The PBS scheme is

\[
\nu = k\dot{e}
\]

where \( \nu \) is the desired end-effector spatial velocity to be used in (44), and the choice and calculation of \( k \) and \( \dot{e} \) are detailed in (37)–(43) of Part 1. Figure 2 shows the difference in the final joint configuration between RRMC and the controller of Park et al. [5] (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 maximization. Any subtask that can be expressed as a differentiable function of \( q \) can be used, and multiple subtasks can be individually weighted and added together to be used as \( \dot{q}_{\text{null}} \). For example, Baur et al. [8] used the manipulability Jacobian and an additional weighting on joint positions, which discouraged joints from getting too close to their physical limits.

**QUADRATIC PROGRAMMING**

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

\[
\begin{align*}
\min_x & \quad f_o(x) = \frac{1}{2}x^TQx + c^Tx, \\
\text{subject to} & \quad Ax = b, \\
& \quad A_s x \leq b, \\
& \quad g \leq x \leq h
\end{align*}
\]

where \( f_o(x) \) is the objective function, which is subject to the equality and inequality constraints, and \( A \) and \( g \) represent the upper and lower bounds of \( x \). A quadratic program is strictly convex when the matrix \( Q \) is positive definite [9]. This framework allows us to solve the same problems as those described earlier, 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 QP:

\[
\begin{align*}
\min_q & \quad f_o(q) = \frac{1}{2}q^T Q q, \\
\text{subject to} & \quad J(q)\dot{q} = \nu, \\
& \quad \dot{q}^- \leq \dot{q} \leq \dot{q}^+
\end{align*}
\]
where we have imposed $\dot{q}^-$ and $\dot{q}^+$ as the upper and lower joint-velocity limits. If the manipulator has more DoF 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

$$
\min_{\dot{q}} \quad f_o(\dot{q}) = \frac{1}{2} \dot{q}^T \lambda \dot{q} - J_m(q)^T \dot{q},
$$

subject to

$$
J(q) \dot{q} = v
$$

(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:

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

(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, $\rho_i$ is the influence distance within which to activate the damper, and $\rho_s$ is the stopping distance in which the distance $\rho$ will never be able to reach or enter.

In a robot with mixed joint types, see “Excurs 1: Mixed-Joint Manipulators.” We can stack velocity dampers to perform joint-position-limit avoidance for each joint within a robot and incorporate this into our QP (48) as an inequality constraint:

$$
\begin{bmatrix}
\rho_0 - \rho_s \\
\rho_i - \rho_s \\
\vdots \\
\rho_n - \rho_s \\
\rho_i - \rho_s
\end{bmatrix}
\leq
\eta
\begin{bmatrix}
\dot{q}_1 \\
\dot{q}_2 \\
\vdots \\
\dot{q}_n
\end{bmatrix}
$$

(50)

where the identity $1_n$ is included to show how the equation fits into the general form $A_x \leq b$ 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 $e$ 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 six. 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 $v$. 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

$$
\min f_o(x) = \frac{1}{2} x^T Q x + C^T x,
$$

subject to

$$
Jx = v,
$$

$$
Ax \leq B,
$$

$$
x^- \leq x \leq x^+
$$

(51)

with

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

(52)

$$
Q = \begin{bmatrix} \lambda_0 1_n & 0_{6 \times n} \\ 0_{n \times 6} & \lambda_\delta 1_6 \end{bmatrix} \in \mathbb{R}^{(n+6) \times (n+6)}
$$

(53)

$$
J = \begin{bmatrix} J_m \\ I_{6 \times 6} \end{bmatrix} \in \mathbb{R}^{(n+6)}
$$

(54)

$$
C = \begin{bmatrix} \rho \end{bmatrix} \in \mathbb{R}^{n+6}
$$

(55)

$$
A = \begin{bmatrix} \rho_0 - \rho_s \\ \rho_i - \rho_s \\ \vdots \\ \rho_n - \rho_s \\ \rho_i - \rho_s \end{bmatrix} \in \mathbb{R}^n
$$

(56)

$$
B = \begin{bmatrix} \rho_0 - \rho_s \\ \rho_i - \rho_s \\ \vdots \\ \rho_n - \rho_s \\ \rho_i - \rho_s \end{bmatrix} \in \mathbb{R}^n
$$

(57)

$$
x^- = \begin{bmatrix} \dot{q}^- \\ \delta^- \end{bmatrix} \in \mathbb{R}^{(n+6)}
$$

(58)

where $\delta \in \mathbb{R}^6$ 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 optimizer, $\dot{q}^- \dot{q}^+$ are the minimum and maximum joint velocities, and $\delta^- \delta^+$ are the minimum and maximum slack velocities. Each of the gains can be adjusted dynamically. For example, $\lambda_\delta$ is typically large when far from the goal, but it reduces toward zero as the goal approaches.

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

$$
\nu(t) - \delta(t) = J(q) \dot{q}(t)
$$

(59)

which clearly demonstrates that the slack is essentially intentional error, where the optimizer can choose to move components of the desired end-effector motion into the slack vector. For both redundant and nonredundant 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 IK

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

To begin, we first consider the IK solver based on the Newton–Raphson (NR) method, which we described in Part 1 as the iteration

$$q_{k+1} = q_k + (\mathbf{J}(q_k))^\top \mathbf{e}_k$$

until the desired end-effector pose is reached, where $\mathbf{e}$ 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 $\mathbf{J} = \mathbf{J}(q)$ 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 program will find the minimum-norm solution for $\dot{q}$ 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 Kinematics and Dynamics Library IK will not allow joint limits to be exceeded during solutions that exceed iteration limits and joint limits. We can improve this approach by augmenting the vector $\mathbf{e}_k$ with the same null-space motion defined in (62):

$$\mathbf{e}_k = \begin{cases} \mathbf{e}_k & q_i \geq \bar{q}_M, \\ \mathbf{e}_k & q_i \leq \bar{q}_m, \\ 0 & \text{otherwise} \end{cases}$$

where the maximum and minimum joint angles are specified by $q_M$ and $q_m$, respectively, while the maximum and minimum joint angle thresholds are specified by $\bar{q}_M$ and $\bar{q}_m$, respectively. Once this threshold is passed, further progress toward the joint limit is penalized by $\Sigma$. 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):

$$q_{null} = \left( (\mathbf{I}_n - \mathbf{J}(q_k))^\top \mathbf{J}(q) \left( \frac{1}{\lambda_m} \Sigma + \frac{1}{\lambda_m} \mathbf{J}_m(q) \right) \right)^{\frac{1}{2}}$$

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

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

$$\mathbf{q}_{k+1} = \mathbf{q}_k + (\mathbf{J}_k)^\top \mathbf{g}_k$$

$$\mathbf{A}_k = \mathbf{J}(q_k)^\top \mathbf{W}_n \mathbf{J}(q_k) + \mathbf{W}_s$$

$$\mathbf{g}_k = \mathbf{J}(q_k)^\top \mathbf{W}_n \mathbf{e}_k$$

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

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

$$\mathbf{g}_k = \mathbf{J}(q_k)^\top \mathbf{W}_n \mathbf{e}_k + \mathbf{q}_{null}.$$  

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

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \dot{q}.$$  

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 program with a custom error metric.

IK COMPARISON

We show a comprehensive comparison of various numerical IK 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 the results are summarized 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” columns report 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 (+) indicate that solutions with a joint-limit violation are treated as a failure, and another attempt is performed. Methods marked with $\mathbf{q}_{null}(\Sigma)$ have joint-limit avoidance projected into the null space using (62). Methods marked with $\mathbf{q}_{null}(\Sigma, \mathbf{J}_m)$ have joint-limit avoidance and manipulability maximization projected into the null space using (64). For the QP method, the symbols $\Sigma$ and $\mathbf{J}_m$ indicate that joint-limit avoidance and manipulability maximization have been incorporated, respectively. Note that null-space methods cannot be used on the Universal Robotics (UR5) manipulator, as it is not redundant.
| METHOD                  | MEAN ITERATIONS | MEDIAN ITERATIONS | INFEASIBLE COUNT | MEAN SEARCHES | MAXIMUM SEARCHES | JOINT-LIMIT VIOLATIONS | TIME PER ITERATION | MEDIAN TIME |
|------------------------|-----------------|-------------------|------------------|---------------|------------------|------------------------|-------------------|-------------|
| NR                     | 27.96           | 16                | 0                | 1.44          | 25               | 0                      | 1.68              | 26.9        |
| LM (Chan)              | 15.52           | 8                 | 0                | 1.21          | 14               | 0                      | 1.04              | 8.35        |
| LM+ (Wampler)          | 23.75           | 13                | 0                | 1.35          | 20               | 0                      | 1                 | 13          |
| LM+ (Chan)             | 15.52           | 8                 | 0                | 1.21          | 14               | 0                      | 1.04              | 8.29        |
| LM+ (Sugihara)         | 21.89           | 13                | 0                | 1.27          | 19               | 0                      | 1.07              | 13.97       |
| QP (Jₘ)                | 15.93           | 8                 | 0                | 1.22          | 13               | 0                      | 3.12              | 24.99       |

**TABLE 2. Numerical IK methods compared on 10,000 problems with a 7-DoF Panda manipulator.**

| METHOD                  | MEAN ITERATIONS | MEDIAN ITERATIONS | INFEASIBLE COUNT | MEAN SEARCHES | MAXIMUM SEARCHES | JOINT-LIMIT VIOLATIONS | TIME PER ITERATION | MEDIAN TIME |
|------------------------|-----------------|-------------------|------------------|---------------|------------------|------------------------|-------------------|-------------|
| NR                     | 27.88           | 16                | 0                | 1.43          | 12               | 6,705                  | 1.71              | 27.35       |
| LM (Chan)              | 11.91           | 8                 | 0                | 1.12          | 7                | 5,394                  | 1.06              | 8.46        |
| NR+ (Wampler)          | 139.56          | 80                | 104              | 7.5           | 100              | 0                      | 1.54              | 123.03      |
| LM+ (Wampler)          | 127.61          | 76                | 102              | 7.11          | 98               | 0                      | 1                 | 76          |
| LM+ (Chan)             | 37.55           | 18                | 91               | 3.81          | 86               | 0                      | 1.03              | 18.54       |
| LM+ (Sugihara)         | 50.13           | 26                | 89               | 3.64          | 76               | 0                      | 1.07              | 27.81       |
| NR+ qnull(Σ)           | 347.7           | 219               | 254              | 15.88         | 99               | 0                      | 3.02              | 661.59      |
| LM+ (Wampler) qnull(Σ) | 353.84          | 196               | 190              | 14.02         | 99               | 0                      | 2.67              | 523.24      |
| LM+ (Chan) qnull(Σ)    | 37.4            | 18                | 91               | 3.79          | 86               | 0                      | 2.73              | 49.16       |
| LM+ (Sugihara) qnull(Σ)| 44.63           | 24                | 99               | 2.85          | 97               | 0                      | 2.77              | 66.43       |
| NR+ qnull(Σ, Jₘ)       | 232.16          | 132               | 135              | 10.19         | 99               | 0                      | 4.68              | 618.41      |
| LM+ (Wampler) qnull(Σ, Jₘ)| 178.22     | 103               | 105              | 8.58          | 100              | 0                      | 4.23              | 435.74      |
| LM+ (Chan) qnull(Σ, Jₘ)| 37.33           | 18                | 90               | 3.77          | 86               | 0                      | 4.25              | 76.53       |
| LM+ (Sugihara) qnull(Σ, Jₘ)| 49.55     | 26                | 89               | 3.6            | 97               | 0                      | 4.29              | 111.5       |
| QP (Σ, Jₘ)             | 42.42           | 14                | 76               | 2.12          | 86               | 0                      | 3.95              | 55.36       |

**TABLE 3. Numerical IK methods compared on 10,000 problems with a 13-DoF (waist, arm, and index finger) Valkyrie humanoid.**

| METHOD                  | MEAN ITERATIONS | MEDIAN ITERATIONS | INFEASIBLE COUNT | MEAN SEARCHES | MAXIMUM SEARCHES | JOINT-LIMIT VIOLATIONS | TIME PER ITERATION | MEDIAN TIME |
|------------------------|-----------------|-------------------|------------------|---------------|------------------|------------------------|-------------------|-------------|
| NR                     | 27.96           | 16                | 0                | 1.44          | 25               | 0                      | 1.68              | 26.9        |
| LM (Chan)              | 15.52           | 8                 | 0                | 1.21          | 14               | 0                      | 1.04              | 8.35        |
| LM+ (Wampler)          | 23.75           | 13                | 0                | 1.35          | 20               | 0                      | 1                 | 13          |
| LM+ (Chan)             | 15.52           | 8                 | 0                | 1.21          | 14               | 0                      | 1.04              | 8.29        |
| LM+ (Sugihara)         | 21.89           | 13                | 0                | 1.27          | 19               | 0                      | 1.07              | 13.97       |
| QP (Jₘ)                | 15.93           | 8                 | 0                | 1.22          | 13               | 0                      | 3.12              | 24.99       |
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 based on the time per iteration and the median number of iterations.

Table 1 displays the results on a 6-DoF UR5 manipulator, where each joint can articulate ±180°—this means that, for each joint, every angle is achievable, which is a larger range than that of most other manipulators. Table 2 displays the results on a 7-DoF Panda manipulator, where each joint has unique articulation limits, more typical of other manipulators. Table 3 displays the results on a 13-DoF 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 nonredundant, redundant, and hyperredundant 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 maximization. The QP IK solver is clearly the most robust, uses the fewest iterations in the median, and will use fewer attempts to find a solution, but it is one of the slowest algorithms presented. As more DoF 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 maximization. Although the latter methods will typically arrive at a solution in fewer iterations, the extra computation time makes the median time much worse than that of 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 DoF if a joint is at its limit, if the arm is fully extended, or when two or more joints 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], [22], [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 \( x = f(q) \in \mathbb{R}^6 \) given by the forward kinematics. Introducing a small change to the joint coordinates \( \Delta q \), we can write

\[
x + \Delta x = f(q + \Delta q)
\]

and the Taylor series expansion is

\[
x + \Delta x = f(q) + \frac{\partial f}{\partial q} \Delta q + \frac{1}{2} \left( \frac{\partial^2 f}{\partial q^2} \right) \Delta q + \ldots
\]

\[
\Delta x = J(q) \Delta q + \frac{1}{2} (H(q) \Delta q) \Delta q + \ldots
\]

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 NR approach to solve for \( \Delta q \) in (71). First, we rearrange (71) as

\[
 g(q_k) = J(q_k) \Delta q_k + \frac{1}{2} (H(q_k) \Delta q_k) \Delta q_k - \Delta x = 0
\]

where \( q_k \) and \( \Delta q_k \) are the manipulator’s current joint coordinates and change in joint coordinates, respectively, and \( \Delta x \) is the desired change in end-effector position. Taking the derivative of \( g \) with respect to \( \Delta q \),

\[
\frac{\partial g(\Delta q_k)}{\partial \Delta q} = J(q_k) + H(q_k) \Delta q_k
\]

\[
= J(q_k, \Delta q_k)
\]

we obtain a new Jacobian \( J(q_k, \Delta q_k) \). Using this Jacobian, we can create a new linear system

\[
\dot{J}(q_k, \Delta q_k) \delta_{\Delta q} = g(\Delta q_k)
\]

\[
\delta_{\Delta q} = \dot{J}(q_k, \Delta q_k)^{-1} g(\Delta q_k)
\]

where \( \delta_{\Delta q} \) is the update to \( \Delta q_k \). The changes in joint coordinates at the next step are

\[
\Delta q_{k+1} = \Delta q_k - \delta_{\Delta q}.
\]

In the case that \( \Delta q = 0 \), 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 zero may be used to seed the initial value with \( \Delta q = (0.1, \ldots, 0.1) \), or the pseudo-inverse approach could be used:

\[
\Delta q = \mathbf{J}(q)^\top \Delta \mathbf{x}.
\]  

(76)

This controller can be used in the same manner as the RRMC described in Part 1. The manipulator Jacobian and Hessian are calculated in the robot’s base frame, and the end-effector velocity \( \Delta \mathbf{x} \) 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 IK algorithm using any technique described in the “Inverse Kinematics” section in Part 1 or the “Advanced IK” section of this article.

In the literature [21, 22, 23, 24], the manipulator Hessian is reported as taking too long to compute when the robot is in or near a singularity. Therefore, a value near zero may be used to seed the initial value. This method is no longer valid.

CONCLUSION

In this tutorial’s final installment, 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 IK solvers can be extended to avoid joint limits and maximize manipulability.

Finally, we described how the manipulator Hessian is used to create quadratic-rate motion control: a controller that 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.

ACKNOWLEDGMENT

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 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 the associated Jupyter Notebooks.

AUTHORS

Jesse Haviland. Centre for Robotics, Queensland University of Technology, Brisbane, Queensland 4001, Australia. E-mail: j.haviland@qut.edu.au.

Peter Corke. Centre for Robotics, Queensland University of Technology, Brisbane, Queensland 4000, Australia. E-mail: peter.corke@qut.edu.au.

REFERENCES

[1] J. Haviland and P. Corke. “Manipulator differential kinematics – Part I. Kinematics, velocity, and applications,” IEEE Robot. Autom., vol. 31, no. 4, pp. 149–158, 2023, doi: 10.1109/MRA.2023.3270228.

[2] P. Corke and J. Haviland, “Not your grandmother’s toolbox – The robotics toolbox reinvented for Python,” in Proc. IEEE Int. Conf. Robot. Automat. (ICRA), 2021, pp. 11,357–11,363, doi: 10.1109/ICRA48506.2021.9561366.

[3] P. Corke, Robotics: Vision and Control. Fundamental Algorithms in Python, 3rd ed. Cham, Switzerland: Springer Nature, 2023.

[4] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation,” IEEE J. Robot. Autom., vol. 3, no. 1, pp. 43–53, Feb. 1987, doi: 10.1109/JRA.1987.1087066.

[5] J. Park, C. Wangkyun, and Y. Younigl, “Computation of gradient of manipulability for kinematically redundant manipulators including dual manipulators systems,” Trans. Contr., Autom. Syst. Eng., vol. 1, no. 1, pp. 8–15, Jun. 1999.

[6] T. Yoshikawa, “Manipulability of robotic mechanisms,” Int. J. Robot. Res., vol. 2, no. 2, pp. 3–9, Jun. 1985, doi: 10.1177/027836498500200201.

[7] L. J. Stocco, S. E. Salcudean, and F. Sassani, “On the use of scaling matrices for task-specific robot design,” IEEE Trans. Robot. Autom., vol. 15, no. 5, pp. 958–965, Nov. 1999, doi: 10.1109/70.795800.

[8] J. Baur, J. Pfaff, H. Ulbrich, and T. Villgrattner, “Design and development of a redundant modular multipurpose agricultural manipulator,” in Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics (AIM), 2012, pp. 823–830, doi: 10.1109/ AIM.2012.6265928.

[9] J. Nocedal and S. Wright, Numerical Optimization. New York, NY, USA: Springer Science & Business Media, 2006.

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

[11] B. Faverjon and P. Tournassoud, “A local based approach for path planning of manipulators with a high number of degrees of freedom,” in Proc. IEEE Int. Conf. Robot. Automat., 1987, vol. 1, no. 1, pp. 1159–1163, doi: 10.1109/ROBOT.1987.1087982.

[12] J. Haviland and P. Corke, “NEO: A novel expedient optimisation algorithm for reactive control of manipulators,” IEEE Robot. Autom. Lett., vol. 6, no. 2, pp. 1043–1050, Feb. 2021, doi: 10.1109/LRA.2021.3056060.

[13] J. Haviland, N. Sünderhauf, and P. Corke, “A holistic approach to reactive mobile manipulation,” IEEE Robot. Autom. Lett., vol. 7, no. 2, pp. 3212–3219, 2022, doi: 10.1109/LRA.2022.3146554.

[14] S. Kumar, N. Sukavanam, and R. Balasubramanian, “An optimization approach to solve the inverse kinematics of redundant manipulator,” Int. J. Inf. Syst. Sci., vol. 6, no. 4, pp. 414–423, 2010.

[15] R. Smits. KDL: Kinematics and Dynamics Library. (2022). Orocos Kinematics and Dynamics. [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 Trans. Syst., Man, Cybern., vol. 16, no. 1, pp. 93–101, Jan. 1986, doi: 10.1109/TSMC.1986.289285.

[17] S. K. Chan and P. D. Lawrence, “General inverse kinematics with the error damped pseudoinverse,” in Proc. IEEE Int. Conf. Robot. Automat., 1988, pp. 834–839, doi: 10.1109/ROBOT.1988.12614.

[18] T. Sugihara, “Solvability-unconcerned inverse kinematics by the Levenberg–Marquardt method,” IEEE Trans. Robot., vol. 27, no. 5, pp. 984–991, Oct. 2011, doi: 10.1109/TRO.2011.2148230.

[19] P. Beeson and B. Ames, “TRAC-IK: An open-source library for improved solving of generic inverse kinematics,” in Proc. IEEE-RAS 15th Int. Conf. Humanoid Robots (Humanoids), 2015, pp. 928–935, doi: 10.1109/HUMANOIDS.2015.7363472.

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

[21] N. S. Bedrossian, “Classification of singular configurations for redundant manipulators,” in Proc. IEEE Int. Conf. Robot. Automat., 1990, pp. 818–823, doi: 10.1109/ROBOT.1990.126089.

[22] N. Bedrossian and K. F. Haeckegger, “Characterizing spatial redundant manipulator singularities,” in Proc. IEEE Int. Conf. Robot. Automat., 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 Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Hum. Robot Interact. Cooperative Robots, 1995, vol. 3, pp. 78–83.

[24] E. D. Pohl and H. Lipkin, “A new method of robotic rate control near singularities,” in Proc. IEEE Int. Conf. Robot. Automat., Sacramento, CA, USA: IEEE Computer Society, 1991, pp. 1708–1709, doi: 10.1109/ROBOT.1991.131866.

Authorized licensed use limited to: RMIT University. Downloaded on October 28,2022 at 10:37:35 UTC from IEEE Xplore. Restrictions apply.