Modeling, Identification and Control, Vol. 29, No. 2, 2008, pp. 39–49

General Solutions to Functional and Kinematic Redundancy

Pål Johan From [1] Jan Tommy Gravdahl [1]

[1] *Department of Engineering Cybernetics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway. E-mail:* {from,tommy.gravdahl}@itk.ntnu.no

Abstract

A systematic and general approach to represent functional redundancy is presented. It is shown how this approach allows the freedom provided by functional redundancy to be integrated into the inverse geometric problem for real-time applications and how it can be utilised to improve performance. A set of new iterative solutions to the inverse geometric problem, well suited for kinematically redundant manipulators, is also presented. *

Keywords: Robotics, Kinematics, Redundancy

1 Introduction

This paper addresses functional and kinematic redundancy. Both types of redundancy provide a freedom that should be utilised in order to improve the performance of the manipulator. Two separate optimisation problems are formulated; (a) if functional redundancy is present, choose the desired (optimal) end-effector configuration; and (b) if kinematic redundancy is present, find the joint positions from the end-effector position/orientation that optimises a given cost function.

In general, motion control is performed in operational space or joint space (Khalil and Dombre, 2002). Operational space control has the advantage that the end-effector position and orientation are given in the Cartesian space. For operational space control, the transformation from operational to joint space is obtained by the *inverse kinematic problem*, which finds the joint velocities from the desired end-effector velocities. The operational space control has many advantages and is fast to compute. A drawback is that it is very dependent of the inverse Jacobian and that the transformation from operational to joint space is performed inside the feedback loop so that the time-step of the controller strongly depends of the complexity of this transformation (Perdereau et al., 2002).

For joint space control, the transformation from operational space to joint space is obtained by the *inverse geometric problem*, i.e. to find the joint positions from the desired end-effector position/orientation. Then some joint space control scheme, independent of the task, can be designed. The disadvantage of this approach is that the inverse geometrics is a time-consuming problem to solve. The advantage is that the transformation from operational to joint space is moved outside the control loop. When kinematic redundancy is present, the inverse geometric approach also allows for optimising a general secondary criteria, and is not dependent of finding a suitable inverse of the Jacobian, such as the Moore-Penrose generalised inverse, as for the inverse kinematic problem.

Functional redundancy comes from a freedom in the specifications of the end-effector configuration and is hence task dependent. One example of functional redundancy is the pointing task where only the direc-

ISSN 1890–1328 **doi:10.4173/mic.2008.2.1**

tion of the end-effector is specified. A systematic and general approach to represent functional redundancy is presented. The freedom due to the pointing task is represented by a continuous set of orientations, called a quaternion volume, and the optimal orientation is chosen among these. It is shown that if an orientation error is allowed, this can also be represented by a quaternion volume. It is also shown that performance is improved in both cases.

When path planning is to be integrated into the inverse geometric problem, this can be done by methods based on global or local path information. Most of the solutions presented in literature are local (Khalil and Dombre, 2002; Perdereau et al., 2002; Grudic and Lawrence, 1993). Optimal global solutions are far more complex and for the time being not suitable for real-time applications. One approach to solve the path-planning problem globally is to find a sub-optimal solution of a general minimisation problem. The problem is then to formulate some cost function, representing both global and local objectives. To illustrate the advantages of the general representation of the freedom provided by the functional redundancy, a sub-optimal approach based on complete path information which does not increase the computation time notably is presented. This approach can be integrated into *any* inverse geometric/kinematic algorithm and is suitable for real-time applications.

Closed-form solutions to the inverse geometric problem are only known for certain types of robotic manipulators, so numerical approaches are widely studied and in many cases, such as for most redundant manipulators, represent the only solution to the problem. Numerical solutions are in general more time-consuming than closed-form solutions and are hence more suitable for off-line path planning. In this paper, the inverse geometric problem is treated as a pure optimisation problem which allows the programmer to include a secondary objective (Grudic and Lawrence, 1993; Wang and Chen, 1991; Luenberger, 2003). When redundancy is present, the redundant degrees of freedom are used to optimise this objective.

The novelty of the method presented is that the minimum of the cost function with respect to each joint is found analytically and this is exploited to develop a set of computationally efficient algorithms. The solution is shown for a cost function representing the position and orientation error of the end effector but can be expanded to include a general class of cost functions representing both global and local objectives.

Five algorithms are presented. The first three use coordinate descent which looks at one joint at the time. It is well known that the convergence of coordinate descent is slower than steepest descent and Newton's method. The advantage is that the analytic solution presented is a lot faster to solve than search algorithms in general. The last two methods can be looked upon as approximations of steepest descent where the gradient is estimated. It is argued that the step size can be set as a constant. Hence, an analytic and computationally efficient alternative to the steepest descent is presented.

2 Representing Rotations

2.1 The Unit Quaternion

Any positive rotation ϕ about a fixed unit vector $\boldsymbol{n}$ can be represented by the four-tuple (Kuipers, 2002)

$$Q = \begin{bmatrix} q_0 \\ \boldsymbol{q} \end{bmatrix}, \tag{1}$$

where $q_0 \in \mathbb{R}$ is known as the scalar part and $\boldsymbol{q} \in \mathbb{R}^3$ as the vector part. The unit quaternion $Q(\phi, \boldsymbol{n})$ is written in terms of ϕ and $\boldsymbol{n}$ by

$$q_0 = \cos(\frac{\phi}{2}), \qquad \boldsymbol{q} = \sin(\frac{\phi}{2})\boldsymbol{n}, \tag{2}$$

where $\boldsymbol{n}$ is unitary. Note that Q and $-Q$ represent the same rotation. This is referred to as the dual covering. The quaternion identity is given by $Q_I = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$. Let $P = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \end{bmatrix}^\mathsf{T}$ and $Q = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^\mathsf{T}$. A multiplication of two quaternions is given by the quaternion product

$$P * Q = \begin{bmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2 \\ p_0q_2 + p_2q_0 + p_3q_1 - p_1q_3 \\ p_0q_3 + p_3q_0 + p_1q_2 - p_2q_1 \end{bmatrix}. \tag{3}$$

A pure quaternion is a quaternion with zero scalar part. Any vector, $\bar{\boldsymbol{v}} = \begin{bmatrix} x & y & z \end{bmatrix}^\mathsf{T}$ can be represented by a pure quaternion $\boldsymbol{v} = \begin{bmatrix} 0 & \bar{\boldsymbol{v}} \end{bmatrix}^\mathsf{T}$. The conjugate of a quaternion is defined as $Q^* = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{bmatrix}^\mathsf{T}$.

2.2 Quaternions and Rotations

Let a vector, $\bar{\boldsymbol{v}}_1$, be represented by the pure quaternion $\boldsymbol{v}_1$. This vector is rotated ϕ radians around the axis $\boldsymbol{n}$ by

$$\boldsymbol{v}_2 = Q * \boldsymbol{v}_1 * Q^*. \tag{4}$$

Every vector $\bar{\boldsymbol{v}} \in \mathbb{R}^3$ can be represented by a pure quaternion. The resulting vector, $\bar{\boldsymbol{v}}_2$, is then of the same length as $\bar{\boldsymbol{v}}_1$ if and only if Q is a unit quaternion.

3 Quaternion Volumes

3.1 General Definition

A systematic approach on how to represent sets of orientations, as presented in From and Gravdahl (2007), is given. A set of frames that correspond to a reference frame by a rotation about a fixed axis, $\boldsymbol{n}$, can be represented by a quaternion and some restriction[1]

$$Q(\phi, \boldsymbol{n}), \quad \text{for } \phi_{min} < \phi < \phi_{max}. \tag{5}$$

When restrictions are not limited to one axis only, a more general description of all allowed orientations can be defined by a combination of rotations given by the quaternion product of two or more quaternions and their restrictions. In this paper, only sets of frames that can be described by a sequence of rotations about fixed axes are treated.

Definition 3.1 (Quaternion Volume) *A quaternion volume, $Q^{\otimes}$, is defined as*

$$\begin{aligned} Q^{\otimes} \triangleq \{ Q(\phi_1, \ldots, \phi_n, \boldsymbol{n}_1, \ldots, \boldsymbol{n}_n) | \phi_{1,min} \leq \phi_1 \leq \phi_{1,max} \\ \vdots \\ \phi_{n,min} \leq \phi_n \leq \phi_{n,max} \} \end{aligned} \tag{6}$$

for $n \geq 1$ and where

$$Q(\phi_1, \ldots, \phi_n, \boldsymbol{n}_1, \ldots, \boldsymbol{n}_n) = Q(\phi_1, \boldsymbol{n}_1) * \cdots * Q(\phi_n, \boldsymbol{n}_n). \tag{7}$$

3.2 Quaternion Volumes by Rotations Sequences

A rotation sequence describes a rotation about one coordinate axis followed by a rotation about another of the coordinate axes in the rotated coordinate system. A general framework on how to construct easily visualisable quaternion volumes by rotation sequences is presented. The rotation sequence starts with two subsequent rotations about two coordinate axes, represented by the quaternion Q_s. This defines the direction of the z-axis. The last degree of freedom is added by a rotation about the direction vector, here the z-axis, by Q_z. In Equation (4), let Q_z represent the vector to be rotated and Q_s the quaternion describing the direction of this vector. Then the rotation sequence

$$\mathcal{V} = Q_s * Q_z * Q_s^* \tag{8}$$

represents the direction of the z-axis for a given rotation Q_s given by the direction of the vector part of $\mathcal{V}$ and the rotation about the z-axis given by the scalar part or length of the vector part of $\mathcal{V}$ by $\psi = 2\arcsin(\|\bar{\boldsymbol{v}}\|) = 2\arccos(v_0)\text{sgn}(\psi)$. Henceforth, $\mathcal{V}$ is called a *visualising quaternion*. Note that $\mathcal{V}$ does not represent a rotation. It is used as a tool to visualise rotations and as a help to define an appropriate set of frames for different applications. The visualising quaternion and the corresponding quaternion should be viewed upon as a pair, $(Q, \mathcal{V})$, where the visualising quaternion, $\mathcal{V}$, gives an intuitive description of a rotation of a frame by Q.

Let the vector part of the visualising quaternion be plotted as a point in the xyz-sphere. Then the direction of the z-axis, rotated by the corresponding quaternion is given by the vector from the origin to this point, and the rotation about the z-axis itself is given by the length of this vector. Hence, a continuous set of quaternions (a quaternion volume) is represented by a "cloud" in the xyz-sphere describing the corresponding set of orientations.

The quaternion that rotates the reference frame into the orientation described by Equation (8) is then given by

$$Q = Q_s * Q_z. \tag{9}$$

Finally, the quaternion volume is given by restricting the allowed rotations of each quaternion.

Given a visualising quaternion volume by the sequence

$$\mathcal{V}^{\otimes} = Q_s^{\otimes} * Q_z^{\otimes} * (Q_s^{\otimes})^* \tag{10}$$

and the restrictions on $Q_s^{\otimes}$ and $Q_z^{\otimes}$. Then the corresponding quaternion volume that results in the set of orientations described by $\mathcal{V}^{\otimes}$ is given by

$$Q^{\otimes} = Q_s^{\otimes} * Q_z^{\otimes} \tag{11}$$

with the same restrictions applied to $Q^{\otimes}$ as to $\mathcal{V}^{\otimes}$.

Figure 1 shows the difference between the quaternion volume and the visualising quaternion volume plotted in the xyz-sphere. Note that the dual covering also applies to the visualising quaternion volume.

3.3 Reorientation of Quaternion Volumes

Let $Q^{\otimes}$ be a quaternion volume and the quaternion P represent some transformation on $Q^{\otimes}$. Then the transformation $Q_P^{\otimes} = P * Q^{\otimes}$ rotates this set of frames by a rotation P.

Proposition 3.1 (Transformation of QV) *Any quaternion volume, $Q^{\otimes}$, represented with respect to*

[1] The dual covering allows every rotation to be described twice. In this paper, however, it is only described once, so that all angles are assumed to be in the interval $(-\pi, \pi)$. It is also assumed that all angles of inverse trigonometric functions are in this interval with the correct sign. For arctan, this is denoted $\arctan 2$.

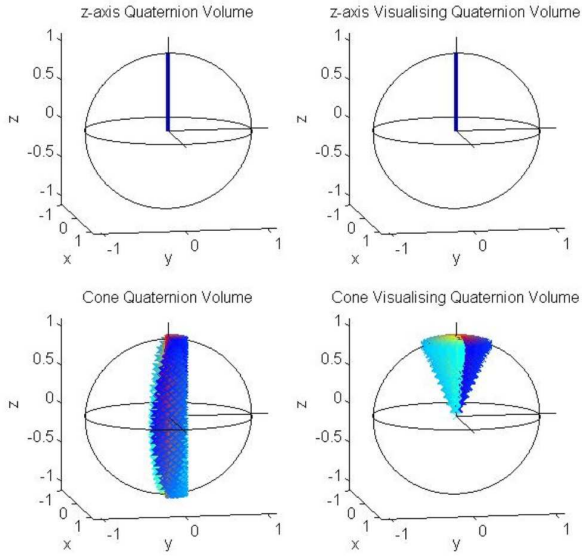

Figure 1: The quaternion volume and the visualising quaternion volume in the xyz-sphere. The upper plots show a freedom about the z-axis and the lower plots show all directions that span out a cone and the orientations about these direction vectors. The visualising quaternion volume gives a more intuitive picture of the orientations described by the quaternion volume.

the identity frame can be transformed into another quaternion volume by

$$Q_P^{\otimes} = P * Q^{\otimes}, \tag{12}$$

where the orientations represented by $Q_P^{\otimes}$ relate to P in the same way as $Q^{\otimes}$ relates to the identity frame.

proof The proof is given in From and Gravdahl (2007).

Similarly, the transformation $Q_P^{\otimes} = P^* * Q^{\otimes}$ allows the set of frames represented by the quaternion volume to be represented with respect to a new reference frame P. The transformation induced by changing from one reference orientation to another is called *reorientation* (Alpern et al., 1993).

4 Optimisation Algorithms

4.1 Descent Methods

This section presents some important approaches to solve a general optimisation problem by iterative algorithms (Luenberger, 2003).

Definition 4.1 (Descent Algorithm) *An algorithm that for every new point generated, decreases the corresponding value of some function, is called a descent algorithm.*

If an algorithm is not descent, it is not guaranteed that the cost function decreases at every iteration. This property is desirable, but not required. Luenberger (2003) shows that the first order necessary condition is satisfied ($\boldsymbol{\nabla} f = \mathbf{0}$) for descent algorithms. A similar proof cannot be given for algorithms that are not descent.

4.2 Steepest Descent

The most common method to minimise a function of several variables is the steepest decent, or the gradient method. The steepest descent is given by the iterative algorithm (Luenberger, 2003)

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha^k \boldsymbol{\nabla} f(\boldsymbol{x}^k)^{\mathsf{T}} \tag{13}$$

where α^k is a non-negative scalar minimising $f(\boldsymbol{x}^k - \alpha^k \boldsymbol{\nabla} f(\boldsymbol{x}^k))$. α_k is found by a search in the direction of the negative gradient for a minimum of this line. Convergence to a point where $\boldsymbol{\nabla} f(\boldsymbol{x}) = \mathbf{0}$ can be proven.

4.3 Coordinate Descent Methods

The coordinate descent algorithm optimises a given cost function $f(x)$, $x \in \mathbb{R}^n$, by sequentially minimising with respect to each of the components, x_i, for $i = 1 \ldots n$. The convergence of coordinate descent is in general poorer than the steepest descent. However, they are easy to implement and, as the gradient is not required, a fast solution to the sub-problem makes these algorithms relatively fast.

One important issue that needs to be addressed is objectivity (observer indifference). In this paper, the cost function is an error metric, i.e. a distance metric on $SE(3)$. Note that an algorithm might be descent for one metric, but not for another. An unfortunate choice of distance metric on $SE(3)$ may cause the algorithm to fail to converge (Gwak et al., 2003).

5 Solutions to the Inverse Geometric Problem

If kinematic redundancy is present, this should be included in the inverse geometric algorithm to improve performance. This section presents a set of algorithms, all based on the same analytical solution to a minimisation problem on $SE(3)$.

In the following, the principal cost function, representing the position and orientation error is presented.

All cost functions presented are well-defined. If the cost function is extended to also include some secondary objective, this will depend on the task, and must be worked out in each case. The problem is solved for revolute joints only.

5.1 Position and Orientation Error

The algorithms in this section are based on two different optimisation problems. One with the position and orientation treated separately, and one where the cost function represents the sum of the position and orientation error.

5.1.1 Position Cost Function

Let the desired position $P_d = \begin{bmatrix}0 & x_d & y_d & z_d\end{bmatrix}^\mathsf{T}$ and current position $P_c = \begin{bmatrix}0 & x_c & y_c & z_c\end{bmatrix}^\mathsf{T}$ be given in the frame of joint i. Assume that the current position can be rotated about the z-axis, and hence represents a 1 degree of freedom, given by the quaternion volume $Q_z^\otimes = \begin{bmatrix}\cos(\frac{\psi}{2}) & 0 & 0 & \sin(\frac{\psi}{2})\end{bmatrix}^\mathsf{T}$ for $-\pi < \psi < \pi$. Then, the solution to the problem of finding the quaternion within the quaternion volume that takes P_c as close to P_d as possible is given by minimising

$$g_p(\psi) = (x_d - \hat{x}_c)^2 + (y_d - \hat{y}_c)^2 + (z_d - \hat{z}_c)^2, \quad (14)$$

where

$$\hat{P}_c^\otimes = Q_z^\otimes * P_c * (Q_z^\otimes)^*. \quad (15)$$

By noting that

$$\hat{P}_c^\otimes = \begin{bmatrix}0 \\ x_c\cos\psi - y_c\sin\psi \\ y_c\cos\psi + x_c\sin\psi \\ z_c\end{bmatrix} \quad \text{for} -\pi < \psi < \pi, \quad (16)$$

$g_p(\psi)$ can be written as

$$g_p(\psi) = K_\psi + a_\psi\cos(\psi) + b_\psi\sin(\psi), \quad (17)$$

where

$$K_\psi = x_d^2 + y_d^2 + z_d^2 + x_c^2 + y_c^2 + z_c^2 - 2z_dz_c, \quad (18)$$
$$a_\psi = -2(x_dx_c + y_dy_c), \quad (19)$$
$$b_\psi = 2(x_dy_c - y_cx_d). \quad (20)$$

Similarly when the freedom is given about the y-axis, $g_p(\theta)$ is given by

$$g_p(\theta) = K_\theta + a_\theta\cos(\theta) + b_\theta\sin(\theta), \quad (21)$$

where

$$K_\theta = x_d^2 + y_d^2 + z_d^2 + x_c^2 + y_c^2 + z_c^2 - 2y_dy_c, \quad (22)$$
$$a_\theta = -2(x_dx_c + z_dz_c), \quad (23)$$
$$b_\theta = 2(z_dx_c - x_dz_d). \quad (24)$$

The rotation that minimises the position error of the end effector is given by setting $\frac{\mathrm{d}g_p(\psi)}{\mathrm{d}\psi} = 0$ and $\frac{\mathrm{d}g_p(\theta)}{\mathrm{d}\theta} = 0$:

$$\psi_{min} = \arctan 2\left(\frac{b_\psi}{a_\psi}\right), \qquad \theta_{min} = \arctan 2\left(\frac{b_\theta}{a_\theta}\right) \quad (25)$$

for a rotation about the z- and y-axes, respectively.

5.1.2 Orientation Cost Function

Similarly, the orientation error can be given by the difference between the desired orientation, D, and $\hat{C}^\otimes$. Let D and $\hat{C}^\otimes$ be given in the frame of joint i and let $\hat{C}^\otimes = Q_z^\otimes * C$ represent all the reachable orientations by rotating about z-axis. Then,

$$g_o(\psi) = (\hat{c}_0(\psi) - d_0)^2 + (\hat{c}_1(\psi) - d_1)^2 + (\hat{c}_2(\psi) - d_2)^2 + (\hat{c}_3(\psi) - d_3)^2. \quad (26)$$

$g_o(\psi)$ is given by

$$g_o(\psi) = K_\psi + c_\psi\cos(\frac{\psi}{2}) + d_\psi\sin(\frac{\psi}{2}), \quad (27)$$

where

$$K_\psi = 2, \quad (28)$$
$$c_\psi = -2(c_0d_0 + c_1d_1 + c_2d_2 + c_3d_3), \quad (29)$$
$$d_\psi = 2(c_3d_0 + c_2d_1 - c_1d_2 - c_0d_3). \quad (30)$$

Similarly when the freedom is given about the y-axis, $g_o(\theta)$ is given by

$$g_p(\theta) = K_\theta + c_\theta\cos(\frac{\theta}{2}) + d_\theta\sin(\frac{\theta}{2}), \quad (31)$$

where

$$K_\theta = 2, \quad (32)$$
$$c_\theta = -2(c_0d_0 + c_1d_1 + c_2d_2 + c_3d_3), \quad (33)$$
$$d_\theta = 2(c_3d_0 + c_2d_1 - c_1d_2 - c_0d_3). \quad (34)$$

The advantage of this approach is that the cost function can be used as an error measure directly. The quaternion representation also allows the optimal rotation to be computed somewhat faster, but then the error function needs to be calculated separately as in Johnson (1995) and From (2006).

5.2 Orientation and Position Cost Function

The total position and orientation error can be given by $g(\psi) = g_p(\psi) + g_o(\psi)$. $g_p(\psi)$ and $g_o(\psi)$ are taken

from Equations (17) and (27), respectively, so that the minimum of $g(\psi)$ is given by

$$\frac{\mathrm{d}g}{\mathrm{d}\psi} = b_\psi \cos(\psi) + d_\psi \cos(\frac{\psi}{2}) - a_\psi \sin(\psi) - c_\psi \sin(\frac{\psi}{2}) = 0. \tag{35}$$

This can be turned into an equation of degree four which can be solved analytically, for example by Ferrari's method.

However, by avoiding the half angles, the solution is found simply by the inverse tangent and the computational complexity is reduced.

In Wang and Chen (1991) a cost function of ψ is found, and this is maximised for every iteration. In the following, a cost function, representing the sum of the position and orientation error is presented. This cost function can then be used as a threshold limit directly. The approach resembles the one in Ahuactzin and Gupka (1999), but allows the programmer to weigh the importance of the position and orientation error.

The cost function can be written as a function of ψ by representing the desired orientation of each joint by a rotation of the three unit vectors by ${}^xQ_d = Q_d * \boldsymbol{e}_i * Q_e^*$. ${}^yQ_d$ and ${}^zQ_d$ are constructed similarly from $\boldsymbol{e}_j$ and $\boldsymbol{e}_k$ where $\boldsymbol{e}_i, \boldsymbol{e}_j, \boldsymbol{e}_k$ are the unitary axes. Then the unitary axes are transformed by the quaternion Q_d into

$${}^xQ_d = Q_d * \boldsymbol{e}_i * Q_d^* = \begin{bmatrix} 0 \\ q_0^2 + q_1^2 - q_2^2 - q_3^2 \\ 2(q_1q_2 + q_0q_3) \\ 2(q_1q_3 - q_0q_2) \end{bmatrix}, \tag{36}$$

$${}^yQ_d = Q_d * \boldsymbol{e}_j * Q_d^* = \begin{bmatrix} 0 \\ 2(q_1q_2 - q_0q_3) \\ q_0^2 - q_1^2 + q_2^2 - q_3^2 \\ 2(q_0q_1 + q_2q_3) \end{bmatrix}, \tag{37}$$

$${}^zQ_d = Q_d * \boldsymbol{e}_k * Q_d^* = \begin{bmatrix} 0 \\ 2(q_0q_2 + q_1q_3) \\ 2(q_2q_3 - q_0q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}. \tag{38}$$

Then the cost function can be written as

$$g(\psi) = w_p g_p(\psi) + w_o g_o(\psi) \tag{39}$$

where w_p and w_o are constant weights, $g_p(\psi)$ is given by Equation (17) and $g_p(\psi)$ is found similarly by representing the difference between the desired position of the unitary axes and the current position of the same axes. The desired position for the x-axis is given by ${}^xQ_d = \begin{bmatrix} 0 & {}^xx_d & {}^xy_d & {}^xz_d \end{bmatrix}$. Assume that the z-axis is the revolute axis. Then the position of the unitary x-axis is given by ${}^xQ_c = \begin{bmatrix} 0 & \cos(\psi) & \sin(\psi) & 0 \end{bmatrix}$ and the difference is written as

$$\begin{aligned} {}^xg_o(\psi) &= ({}^xx_d - \cos(\psi))^2 + ({}^xy_d - \sin(\psi))^2 + ({}^xz_d - 0)^2 \\ &= 2 - 2{}^xx_d \cos(\psi) - 2{}^xy_d \sin(\psi), \end{aligned} \tag{40}$$

and similarly for the y- and z-axes. By adding these three equations to Equation (17), $g(\psi)$ can be written as

$$\begin{aligned} g(\psi) &= w_p g_p(\psi) + w_o({}^xg_o(\psi) + {}^yg_o(\psi) + {}^zg_o(\psi)) \\ &= K_\psi + a_\psi \cos(\psi) + b_\psi \sin(\psi) \end{aligned} \tag{41}$$

where

$$\begin{aligned} K_\psi &= w_p \left(x_d^2 + y_d^2 + z_d^2 + x_c^2 + y_c^2 + z_c^2 - 2z_dz_c\right) \\ &\quad + w_o \left(6 - 2{}^zz_d\right), \\ a_\psi &= -2w_p(x_dx_c + y_dy_c) - 2w_o({}^xx_d + {}^yy_d), \\ b_\psi &= 2w_p(x_dy_c - y_dx_c) + 2w_o({}^yx_d - {}^xy_d). \end{aligned}$$

Similarly when the y-axis is the revolute axis

$$g(\theta) = K_\theta + a_\theta \cos(\theta) + b_\theta \sin(\theta) \tag{42}$$

where

$$\begin{aligned} K_\theta &= w_p \left(x_d^2 + y_d^2 + z_d^2 + x_c^2 + y_c^2 + z_c^2 - 2y_dy_c\right) \\ &\quad + w_o \left(6 - 2{}^yy_d\right), \\ a_\theta &= -2w_p(x_dx_c + z_dz_c) - 2w_o({}^xx_d + {}^zz_d), \\ b_\theta &= 2w_p(z_dx_c - x_dz_c) + 2w_o({}^xz_d - {}^zx_d). \end{aligned}$$

The minimum of the cost function, with respect to each joint, is given by Equation (25) and the error is given by $E = K + a$ (set $\psi = \theta = 0$ in (41) and (42)).

For redundant manipulators, the cost function can be expanded to include an addition term

$$g(\psi) = w_p g_p(\psi) + w_o g_o(\psi) + w_r g_r(\psi). \tag{43}$$

Whenever g_r can be written on the form of (41) the same analytical solution to the sub-problem can be found. This is a large class of cost functions that allows a great variety of secondary objectives to be included in the cost function, such as distance to obstacles and elbow position.

5.3 Algorithm 1 - Coordinate Descent

The coordinate descent algorithm optimises a cost function with respect to each of the variables of the cost function (Wang and Chen, 1991). That is, for each joint in the chain, the minimum of the cost function, when only the respective joint is moved, is found.

There are several different ways the algorithm can work its way through the chain:

- Start from the end and work its way towards the base.
- Start from the base and work its way towards the end.
- Start from one end and sweep its way towards the other and then back (Aitken double sweep method).
- If the gradient is known, select the coordinate (in this case the joint) that corresponds to the largest (in absolute value) component of the gradient vector (Gauss-Southwell Method, presented in the sub-section 5.4).

The cost function must be objective, preferably describing some physical property, as the sum of the position and orientation error. Objectivity is important and in this case sufficient to guarantee that the algorithm is descent (to a point satisfying the first order necessary condition). The cost function should also be computationally efficient, i.e. the minimum of the cost function should be found analytically.

The cost function presented in Section 5.2 have these properties. This cost function, together with an algorithm that starts from the end and moves its way towards the base, is a fast and stable algorithm.

5.4 Algorithm 2 - "Gauss-Southwell" with Cost Function

The Gauss-Southwell Method determines the largest component of the gradient $\nabla g(\boldsymbol{x})$ and chooses this for descent. This sub-section presents an alternative approach, where the minimum of the cost function is found for each joint. The joint that corresponds to the smallest possible value of the cost function is then chosen. This is found simply by Equation (25). This approach is computationally more efficient than to compute the gradient. It will also converge faster (at least in the beginning) because the joint that corresponds to the maximum possible decrease of the cost function is always chosen. This algorithm is descent.

5.5 Algorithm 3 - Gauss-Southwell with Gradient

The method presented above can be modified somewhat so that each joint is chosen by the steepest descent instead of maximum possible descent. Assume that the position of each joint that results in the minimum of the cost function $g(\boldsymbol{x})$ is found. Denote this by $\hat{x}_i^k$ for joint i and iteration k. Then, the *rate* of decrease with respect to this joint is estimated by

$$\frac{\partial g(x_i^k)}{\partial x_i^k} \approx \frac{g(\hat{x}_i^k) - g(x_i^k)}{\left|\hat{x}_i^k - x_i^k\right|}, \qquad \text{for } i = 1 \dots n. \tag{44}$$

Then the joint with the largest corresponding absolute value of the "gradient" is chosen. This approach is different from the solution given in Section 5.4 in that not only the absolute minimum is taken into account, but also how much the manipulator has to move reflects the choice of search direction, which leads to a more energy preserving solution. The joint update is then given by

$$x_i^{k+1} = x_i^k + w_i(\hat{x}_i^k - x_i^k), \qquad \text{with } 0 < w_i \leq 1. \tag{45}$$

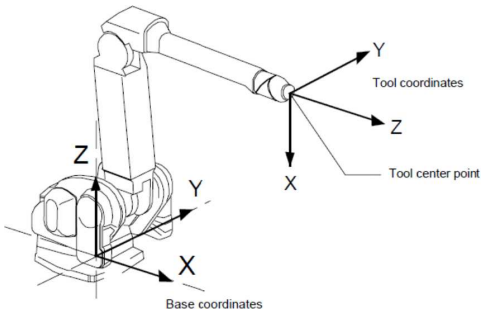

Figure 2: General structure of a robotic manipulator.

5.6 Algorithm 4 - Manipulator Dependent Steepest Descent

Equation (47) gives information about all the joints. Another approach is thus to apply (45) to all the joints for every iteration. The next step is then to find the weights, w_i.

The manipulator structure should be taken into account to improve convergence. For instance if two joints work in the same "direction" in the operational space, they should be scaled down so that the sum of the two joints will result in the desired movement, and not each one looked at separately. By studying the structure of the manipulator in Figure 2, joint 1 is seen to be very much decoupled from the others when it comes to the effect on the end-effector position and orientation, and thus x_1^{k+1} is set close to $\hat{x}_1^k$. Joint 2 and 3, however, are strongly coupled, so $w_{2,3}$ should be set to about 0,5. The three wrist joints should also be scaled due to coupling. In addition, this scaling vector should be scaled down somewhat by a factor $0 < w_s \leq 1$, to ensure convergence. The following scaling vector is suggested for a manipulator with a structure similar to the one in Figure 2:

$$W = w_s \begin{bmatrix} 1 & 0.5 & 0.5 & 0.3 & 0.3 & 0.3 \end{bmatrix}. \tag{46}$$

5.7 Algorithm 5 - Steepest Descent with Gradient Estimate

Equation (44) can also be used to make an estimate of the gradient of the cost function. If the absolute sign is removed, the gradient of $g(\boldsymbol{x}^k)$ can be estimated as

$$\widehat{\boldsymbol{\nabla}} g(\boldsymbol{x}^k) \approx \begin{bmatrix} \frac{g(\hat{x}_1^k) - g(x_1^k)}{\hat{x}_1^k - x_1^k} \\ \vdots \\ \frac{g(\hat{x}_n^k) - g(x_n^k)}{\hat{x}_n^k - x_n^k} \end{bmatrix} \tag{47}$$

As $g(\boldsymbol{x})$ is on the form of (41), $\left|\widehat{\boldsymbol{\nabla}} g(x_i^k)\right| \leq \left|\boldsymbol{\nabla} g(x_i^k)\right|$ for all i so that $\widehat{\boldsymbol{\nabla}} g(\boldsymbol{x})$ is a conservative estimate of $\boldsymbol{\nabla} g(\boldsymbol{x})$.

Now, Equation (47) can be applied to Equation (13) directly. The "step size" can be set similar to Equation (46) with (somewhat conservatively) $w_s = \min_{i=1\ldots n} \left|\hat{x}_i^k - x_i^k\right|$. When the solution approaches zero, the it can be simplified to $w_s = \left|\hat{x}_1^k - x_1^k\right|$.

It should be noted that when Equation (45) is applied to all joints, or the estimate of the gradient is applied in Equation (13), the algorithm is not descent.

6 Pointing Task Improved Trajectory Planning

Functional redundancy, as opposed to kinematic redundancy, does not increase production cost of the manipulator, nor does it make the manipulator structure more complex. It is important to recognise functional redundancy and to analyse how overall performance changes when introducing this freedom. For the pointing task, only the direction of the central axis (in this case the z-axis, see Figure 2) of the end effector is specified, so that the orientation about the central axis can be chosen freely, and thus represents one degree of freedom (1 DOF). This functional redundancy can be exploited to improve the overall performance of the manipulator in many ways. In this paper, this redundancy is exploited to improve the trajectory velocity of the end effector.

Assume that the end effector is to follow a path in the xy-plane, as in Figure 3, with constant velocity[2]. In general, the joints close to the base require the most energy due to the total mass to be moved. A simple approach to minimise the velocity of joint 1 is presented. This is obtained by, for every time step, setting the

[2]In this paper, when it is stated that the end-effector velocity is constant, it is meant that the velocity of some critical point of the tool is constant. The location of this point depends on the task, and in most cases is a point on the surface, which is the case for welding, coating, etc.

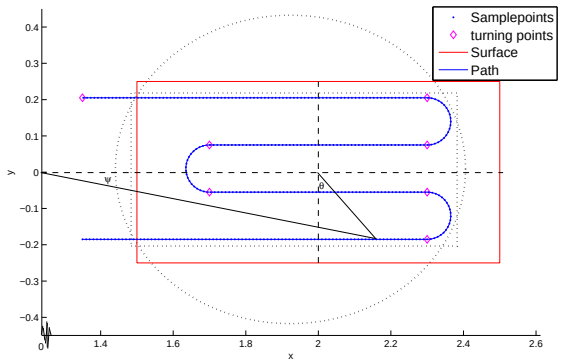

Figure 3: A simple path in the xy-plane. The dotted lines show two examples on how the cross section of the visualising quaternion volume in Section 8 can be chosen.

desired orientation by

$$Q_d^{\otimes} = Q_d * Q_z^{\otimes} \tag{48}$$

where Q_d can be set as a rotation of π about the y-axis (makes the end effector orthogonal to the surface) given by $Q_d = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^\mathsf{T}$ and $Q_z^{\otimes} = \begin{bmatrix} \cos\left(\frac{\psi}{2}\right) & 0 & 0 & \sin\left(\frac{\psi}{2}\right) \end{bmatrix}^\mathsf{T}$ is a rotation by ψ about the z-axis and does not affect the orthogonality. $Q_d^{\otimes}$ is then given by $Q_d^{\otimes} = \begin{bmatrix} 0 & \sin\left(\frac{\psi}{2}\right) & \cos\left(\frac{\psi}{2}\right) & 0 \end{bmatrix}^\mathsf{T}$ and some restriction on ψ. A simple and efficient way to choose ψ is by

$$\psi = \arctan 2\left(\frac{y}{x}\right) \tag{49}$$

where x and y are the desired positions in the xy-plane for the current time step. The physical interpretation of ψ is given in Figure 3.

7 Orientation Error

The freedom represented by the pointing task can be extended somewhat, so that, instead of a set of allowed orientations about the central axis, a cone or pyramid of allowed orientations pointing in approximately the same direction as the central axis can be allowed. See Section 3 and From and Gravdahl (2007) for details on how to construct such quaternion volumes.

Let ψ be found in the same way as in the previous section, but instead of a rotation about the z-axis (in the end-effector coordinate frame), a rotation about the x-axis is added. This will, unlike the approach from the previous section, add an orientation error. This orientation error will push the wrist of the manipulator towards the y-axis so that it allows the first joint to move less.

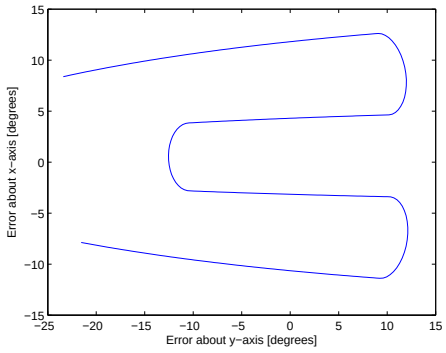

Figure 4: The orientation error by the method described in Section 7.

The same approach is done with the y-axis. From Figure 3, θ will add an orientation error that pushes the wrist towards the vertical line dividing the surface into two equal parts.

In many applications like coating, welding and steaming, a small orientation error will not reduce the quality of the job performed. Figure 4 shows how the orientation error around the y- and z-axes changes for the given path. Note that this closely resembles the path in the xy-plane.

8 Quaternion Volume

The approach presented above is suitable for simple cases as the one presented. Consider the more general case when the end effector must follow a given path on some surface. Much work has been done one finding an optimal path on a surface, and it is assumed that this path is given. The path given in Figure 3 is a good example of such a path for heating or coating a surface. Assume a plane or approximately plane surface. Further, the functional redundancy should be investigated (pointing task, orientation error, etc) and represented by an appropriate quaternion volume.

Assume the restriction on the allowed orientations are given by the quaternion volume in Equation (11). Then $Q_s^{\otimes}$ represents the freedom from the orientation error while $Q_z^{\otimes}$ is the freedom about the end-effector z-axis. The procedure can be summarised in two steps:

- Functional redundancy analysis: Given the surface and the specifications of the end-effector position/orientation, determine the quaternion volume.

- Integrated inverse geometric problem: Define some rule to chose the optimal or sub-optimal orientation that is inside the quaternion volume and solve the inverse geometric problem for this orientation, the desired position and subject to some cost function.

The functional redundancy analysis can again be divided into two parts, to determine $Q_s^{\otimes}$ and to determine $Q_z^{\otimes}$. A test to verify that a quaternion satisfies the restrictions given by the quaternion volume is given in From and Gravdahl (2007).

Assume, without loss of generality, that the surface is in the xy-plane. As $Q_s^{\otimes}$ describes the orientation errors, it can be visualised in this plane. This is done by looking at the effect the quaternion has on the central axis, see Section 3 and From and Gravdahl (2007). The quaternion volume is a continuous set of orientations, so the effect that this set has on a vector is a continuous set of vectors, represented by the visualising quaternion volume. The cross section of this set with the xy-plane gives much information about the quaternion volume. Let ϵ be the projection of the orientation error η into the cross section of the visualising quaternion volume. Some shapes of the cross section of special interest are listed in the following, where $\epsilon = d\sin(\eta)$ and d is the distance from the end effector to the surface and $\theta_{min} < \eta < \theta_{max}$.

Orientation Error	xy-plane cross section
η_x about the x-axis	The line from [0,-ϵ_x] to [0,ϵ_x]
η_y about the y-axis	The line from [-ϵ_y,0] to [ϵ_y,0]
The two previous	Square [-ϵ_y,-ϵ_x], [ϵ_y,-ϵ_x], [ϵ_y,ϵ_x], [-ϵ_y,ϵ_x]
Relative error η	The circle with radius ϵ

$Q_z^{\otimes}$ is the rotation about the central axis. For the pointing task, this can be set freely.

The quaternion volume can be chosen in many different ways according to the surface. The first step is to choose the shape of the cross section of the visualising quaternion volume. The shape of the cross section should resemble the shape of the surface. Two examples of how this cross section can be chosen is given in Figure 3.

When the shape of the cross section of the quaternion volume is chosen, the path is simply placed inside the cross section. This "path" then represents the orientation error. The idea behind this approach is the same as in the previous section. As the orientation error follows the same "path" as the end-effector position, it forces the wrist closer to the centre of the surface, which again requires less torque, especially from the main axes. The last step is to add the orientation about the central axis. This can be set by Equation (49).

The advantage of the algorithm presented above is that it requires very little information about the path

and can thus be integrated in the inverse geometric algorithm. It exploits the functional redundancy, determined by the programmer, and finds the orientation of the end effector that allows the manipulator to traverse the trajectory with a higher velocity. The inputs required to the inverse geometric algorithm are:

- The coordinates of the centre of the surface and the maximum and minimum values of the surface (the corners if the surface is a square, or the radius, if the surface is a circle.)
- The shape and restrictions on the quaternion volume.

The desired orientation is found from the limited information about the surface and the current end-effector position only.

Note that if the cross section of the visualising quaternion volume is chosen as the square in Figure 3, this will result in approximately the same orientation error as in the previous section.

9 Numerical Examples

9.1 Inverse Geometric Algorithms

All the inverse geometric algorithms have been tested for a great variety of problems with the cost function given in Section 5.2. For comparison, the same test has also been done for a inverse Jacobian-based inverse geometric algorithm.

The conventional CCD is computationally fast and convergence is very good in the beginning. Gauss-Southwell is computationally slower as it finds the minimum for all the joints but only one joint is chosen for decrease. The maximum rate and maximum decrease of cost function as criteria for choosing the joint give approximately the same convergence, but maximum rate is preferred as this gives a better configuration of the manipulator, i.e. each joint moves less when the previous configuration is used as the initial guess. The convergence are about the same as for the Jacobian-based algorithm.

The three approaches that resemble steepest descent, and move all joints for every iteration, have better convergence, as expected. However, they are computationally more demanding and a good choice of weights ("step size") is required for stability and good convergence. When this is chosen appropriately, the convergence is better than for coordinate descent and the Jacobian-based algorithm.

All the algorithms have also been tested with an expanded cost function representing some additional requirement, in this case the elbow position. All the algorithms have the same, or approximately the same, performance for the two cases.

9.2 Functional Redundancy

A robot, similar to the one in Figure 2 is set to follow the path in Figure 3. Three simulations are done:

1. The orientation is set constant.
2. The orientation is set as in Section 6.
3. The orientation is set as in Section 7/8.

The torques needed for the first two joints for the three cases are shown in Figure 5. The maximum speeds that the manipulator can follow the path for the three cases were:

End-effector Orientation	Or. Error	Max speed
Constant	0°	0.91 m/s
As in Section 6	0°	1.13 m/s
As in Section 7/8	10°	1.35 m/s

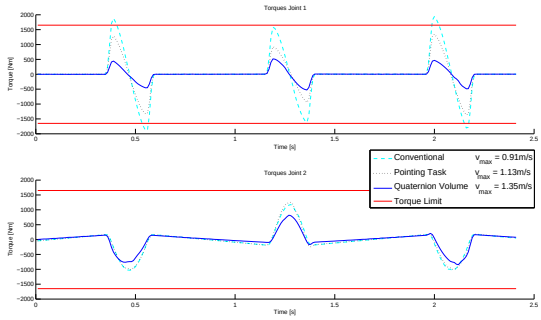

Figure 5: The torques of joints 1 and 2 for the given path.

10 Conclusions

A new class of solutions to the inverse geometric problem is presented. For a large class of cost functions, an analytical solution to the sub-problem can be found, which guarantees computational efficiency. Convergence is found to be about the same or a little better than for Jacobian-based algorithms. The main advantage is that a secondary objective can be included into the cost function without increasing the computation time.

A very simple framework on how to integrate the path planning into the inverse geometric algorithms is presented. By adding very little information to the inverse geometric algorithm, some sub-optimal path is

found, without increasing the computation time notably. It is also shown that an optimisation scheme based on the complete path information can be implemented in the inverse geometric algorithms in real-time. It is shown that the speed of the end effector can be increased by more than 20% for the pointing task, and by almost 50% if a small orientation error is accepted.

References

Ahuactzin, J. M. and Gupka, K. K. The kinematic roadmap: A motion planning based global approach for inverse kinematics of redundant robots. *IEEE Trans. on Robotics and Automation*, 1999. 15.

Alpern, B., Carter, L., Grayson, M., and Pelkie, C. Orientation maps: Techniques for visualizing rotations (a consumers guide). *IEEE Conference on Visualization*, 1993. pages 183–188.

From, P. J. *Modelling and Optimal Trajectory Planner for Industrial Spray Paint Robots*. Master's thesis, NTNU, 2006.

From, P. J. and Gravdahl, J. T. Representing attitudes as sets of frames. *Proc. American Control Conference*, 2007. pages 2465–2472.

Grudic, G. Z. and Lawrence, P. D. Iterative inverse kinematics with manipulator configuration. *IEEE Transactions on Robotics and Automation*, 1993. 9, no. 4:476–483.

Gwak, S., Kim, J., and Park, F. C. Numerical optimization on the euclidean group with applications to camera calibration. *IEEE Transactions on Robotics and Automation*, 2003. 19:65–74.

Johnson, M. P. *Exploiting Quaternions to Support Expressive Interactive Character Motion*. Ph.D. thesis, MIT, 1995.

Khalil, W. and Dombre, E. *Modeling, Identification and Control of Robots*. Hermes Penton, 2002.

Kuipers, J. B. *Quaternions and Rotation Sequences*. Princeton University Press, 2002.

Luenberger, D. G. *Linear and Nonlinear Programming*. Kluwer Academic Publishers, 2003.

Perdereau, V., Passi, C., and Drouin, M. Real-time control of redundant robotic manipulators for mobile obstacle avoidance. *Robotics and Autonomous Systems*, 2002. 41.

Wang, L.-C. T. and Chen, C. C. A combined optimization method for solving the inverse kinematics problem of mechanical manipulators. *IEEE Trans. on Robotics and Automation*, 1991. 7, no. 4.