Modeling, Identification and Control, Vol. 29, No. 3, 2008, pp. 77–92

Iterative Solutions to the Inverse Geometric Problem for Manipulators with no Closed Form Solution

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@itk.ntnu.no

Abstract

A set of new iterative solutions to the inverse geometric problem is presented. The approach is general and does not depend on intersecting axes or calculation of the Jacobian. The solution can be applied to any manipulator and is well suited for manipulators for which convergence is poor for conventional Jacobian-based iterative algorithms. For kinematically redundant manipulators, weights can be applied to each joint to introduce stiffness and for collision avoidance. The algorithm uses the unit quaternion to represent the position of each joint and calculates analytically the optimal position of the joint when only the respective joint is considered. This sub-problem is computationally very efficient due to the analytical solution. Several algorithms based on the solution of this sub-problem are presented. For difficult problems, for which the initial condition is far from a solution or the geometry of the manipulator makes the solution hard to reach, it is shown that the algorithm finds a solution fairly close to the solution in only a few iterations.

Keywords: Robotics, Kinematics, Inverse Kinematics

1 Introduction

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 solving the *inverse kinematic problem*, which finds the joint velocities from the desired end-effector velocities. Operational space control has many advantages and is fast to compute. A drawback is that it strongly depends on 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 solving 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 does not depend on finding a suitable inverse of the Jacobian, such as the Moore-Penrose generalised inverse, as for the inverse kinematic problem.

Another advantage of the inverse geometric approach is that each joint can be controlled more directly and

ISSN 1890–1328 doi:10.4173/mic.2008.3.1

given the desired characteristics such as joint stiffness, energy consumption, maximum velocity and obstacle avoidance. For the inverse Jacobian approach these characteristics must be added through the choice of the Jacobian. In some cases, such as the minimisation of energy through the Moore-Penrose, this is both efficient and elegant, but for other characteristics such an inverse Jacobian may be very hard or impossible to find.

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. The results presented in this paper are based on the preliminary results presented in From and Gravdahl (2007). Here the inverse geometric problem is treated as a pure optimisation problem. This allows the programmer to include a secondary objective which is used to give the manipulator motion the desired characteristics (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.

Six 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 three 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 both the search direction and the step size of the steepest descent is presented.

It is shown that the algorithms that approximate the steepest descent have very good convergence and reliability for difficult problems. However, for easy problems, when the initial guess is close to the solution, the convergence is better for conventional Jacobian-based algorithms than the algorithms presented in this paper. For problems for which the Jacobian-based algorithms have poor convergence or reliability, the algorithms presented are a better choice. A combination of the algorithms presented may give good and reliable performance for difficult problems but also reasonably good convergence close to the solution.

2 Representing Rotations

2.1 The Unit Quaternion

Any positive rotation ϕ about a fixed unit vector $\boldsymbol{n}$ can be represented by the quadruple (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 $Q_I = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$. A multiplication of two quaternions is given by a quaternion product and is written in vector algebra notations as

$$P * Q = \begin{bmatrix} p_0 q_0 - \boldsymbol{p} \cdot \boldsymbol{q} \\ p_0 \boldsymbol{q} + q_0 \boldsymbol{p} + \boldsymbol{p} \times \boldsymbol{q} \end{bmatrix}. \tag{3}$$

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 can then be written as the quaternion product as

$$P * Q = \begin{bmatrix} p_0 q_0 - p_1 q_1 - p_2 q_2 - p_3 q_3 \\ p_0 q_1 + p_1 q_0 + p_2 q_3 - p_3 q_2 \\ p_0 q_2 + p_2 q_0 + p_3 q_1 - p_1 q_3 \\ p_0 q_3 + p_3 q_0 + p_1 q_2 - p_2 q_1 \end{bmatrix}. \tag{4}$$

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 $\boldsymbol{v} = \begin{bmatrix} 0 & \bar{\boldsymbol{v}}^\mathsf{T} \end{bmatrix}^\mathsf{T}$. The conjugate of a quaternion is defined as

$$Q^* = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{bmatrix}^\mathsf{T}. \tag{5}$$

2.2 Quaternions and Rotations

Let a vector, $\bar{\boldsymbol{v}}_1$, be represented by the pure quaternion $\boldsymbol{v}_1$. This vector can be rotated the angle ϕ about the axis $\boldsymbol{n}$ by (Kuipers, 2002)

$$\boldsymbol{v}_2 = Q * \boldsymbol{v}_1 * Q^*. \tag{6}$$

Every vector $\bar{\boldsymbol{v}} \in \mathbb{R}^3$ can be represented by a pure quaternion, hence $\boldsymbol{v}$ is not necessarily a unit quaternion. The quaternion $Q(\phi, \boldsymbol{n})$ however is unitary. This represents the angle and the axis that the vector $\bar{\boldsymbol{v}}_1$ is to be rotated about. 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. The quaternion representation also leads to a useful formula for finding the shortest rotation from one orientation to another. Let P and Q be two orientations. Then, by taking

$$E = P^* * Q, \tag{7}$$

E will rotate P into Q by the shortest rotation. That is, E is the quaternion, out of all the quaternions that take P into Q, with the largest scalar part and thus the smallest angle.

Note that Equation (7) rotates one frame into another frame. By a *frame* is meant a coordinate system in $\mathbb{R}^3$ using Cartesian coordinates. One frame with respect to another frame represents three degrees of freedom and is referred to as an *orientation*. Equation (6) rotates one vector into another vector and has two degrees of freedom, in the same way as a point on a sphere can be defined by two coordinates. A unit vector with respect to a unit reference vector is referred to as a *direction*. Henceforth, when referred to direction, this is the direction of the z-axis of the body frame with respect to the z-axis of the reference frame, as the z-axis of the end effector is our main concern in this paper.

3 Quaternion Space Metric

The axis of a revolute joint, represented in the coordinate frame of the joint, is always constant. This is used in the following to simplify the computations. First the proximity of two frames is discussed, then this is applied to each joint to find the optimal position of the joint. By optimal position is meant the position of the joint that minimises the end effector orientation error, position error, or both.

There are many ways to represent the proximity or distance between two orientations (Yuan, 1988; Wen and Kreutz-Delgado, 1991; Hanson, 2006). One example which is proportional to the length of the geodesic path on the 4-dimensional unit sphere is

$$\Psi_r = \arccos(e_0) \tag{8}$$

where e_0 is taken from $E = P^* * Q$. The cost function in (8) can be identified with a physical property and is a metric function. The formal proof that (8) is a metric function is given in the Appendix. The geodesic path describes the shortest path from one orientation to another. Choosing that path on the 4-dimensional unit sphere gives a well-defined and computationally efficient metric.

A computationally more efficient cost function representing the rotational part is given simply by

$$\Psi_r = 1 - e_0. \tag{9}$$

This cost function lacks the property that it can be identified with a physical property directly and it is not a metric function. Also, its minimum is given by $e_0 = \pm 1$, for which the two orientations are identical, and the maximum is given by zero, for which the orientations point in the opposite directions. However, due to the light computational complexity, this cost function turns out to be very efficient.

A cost function on $SE(3)$ will depend on the weighing of the rotational and translational part. On its general form, it is given by

$$\Psi = w_t \Psi_t + w_r \Psi_r \tag{10}$$

where w_t and w_r are the weights, the translational part, Ψ_t is chosen as the standard Euclidean norm and the rotational part is the metric in (8)

$$\Psi = w_t \left\| p_0 - p_1 \right\| + w_r \arccos(e_0). \tag{11}$$

or alternatively the cost function in (9)

$$\Psi = w_t \left\| p_0 - p_1 \right\| + w_r (1 - e_0). \tag{12}$$

Definition 3.1 (Quaternion Space Proximity) *Given two orientations represented by the two quaternions P and Q. Let the* error quaternion *be denoted*

$$E = P^* * Q. \tag{13}$$

Then the scalar part of E, e_0, describes the proximity of the two frames.

Definition 3.2 (Minimal Rotation) *The bigger (closer to 1)[1] the error quaternion scalar part e_0, the closer are the two orientations P and Q.*

That this is a perfectly good description of the proximity of two frames even though it does not represent a physical property directly. The geodesic path can, however, be found through Equation (8) .

Consider the set of orientations for which the identity frame is rotated about the z-axis. The problem to find the orientation P_z from this set that is closest to some arbitrary orientation Q is considered.

[1] Note that an equally good description of proximity is given when e_0 approaches -1. As $\cos(\frac{\psi}{2})$ is positive for ψ in the chosen interval $(-\pi, \pi)$, the positive value of e_0 is chosen.

Proposition 3.1 (Optimal Rotation) *Consider an orientation* $Q = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$. *The orientation described by the quaternion* $P_z = \begin{bmatrix} p_0 & 0 & 0 & p_3 \end{bmatrix}^T$ *that is closest to* Q *(by Definitions 3.1 and 3.2) is given by*

$$p_0 = \frac{\pm_s q_0}{\sqrt{q_0^2 + q_3^2}} \tag{14}$$

$$p_3 = \frac{\pm_s q_3}{\sqrt{q_0^2 + q_3^2}} \tag{15}$$

where the two $\pm_s$ *have the same sign.*

Proof $E = P^* * Q$ can be written

$$\begin{bmatrix} e_0 \\ e_3 \end{bmatrix} = \begin{bmatrix} p_0 & p_3 \\ -p_3 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_3 \end{bmatrix} \tag{16}$$

$$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} p_0 & p_3 \\ -p_3 & p_0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \tag{17}$$

By Definitions 3.1 and 3.2, the quaternion P_z that is closest to Q is found by the error quaternion with e_0 closest to 1.

$$e_0 = p_0 q_0 + p_3 q_3 \tag{18}$$

$$= q_0 \cos(\frac{\psi}{2}) + q_3 \sin(\frac{\psi}{2}). \tag{19}$$

$$\frac{\mathrm{d}e_0}{\mathrm{d}\psi_0} = -\frac{q_0}{2}\sin(\frac{\psi}{2}) + \frac{q_3}{2}\cos(\frac{\psi}{2}). \tag{20}$$

Let $\frac{\mathrm{d}e_0}{\mathrm{d}\psi_0} = 0$. Then

$$\tan(\frac{\psi}{2}) = \frac{q_3}{q_0}. \tag{21}$$

Then by using $\arctan(x) = \arcsin\left(\frac{x}{\sqrt{1+x^2}}\right)$ (Bronshtein et al., 2003), ψ is written as

$$\psi = 2\arctan(\frac{q_3}{q_0}) \tag{22}$$

$$= 2\arcsin\left(\frac{\frac{q_3}{q_0}}{\sqrt{1 + (\frac{q_3}{q_0})^2}}\right) \tag{23}$$

$$= 2\arcsin\left(\frac{q_3}{\sqrt{q_0^2 + q_3^2}}\right). \tag{24}$$

From the definition of the quaternion

$$\psi = 2\arcsin(p_3). \tag{25}$$

By comparing Equations (24) and (25), Equation (15) is given. Similarly, by $\arctan(x) = \arccos\left(\frac{1}{\sqrt{1+x^2}}\right)\mathrm{sgn}(x)$,

$$\psi = 2\arctan(\frac{q_3}{q_0}) \tag{26}$$

$$= 2\arccos\left(\frac{1}{\sqrt{1 + (\frac{q_3}{q_0})^2}}\right)\mathrm{sgn}(\frac{q_3}{q_0}) \tag{27}$$

$$= 2\arccos\left(\frac{q_0}{\sqrt{q_0^2 + q_3^2}}\right)\mathrm{sgn}(\frac{q_3}{q_0}). \tag{28}$$

Note that the sign of $\psi = 2\arccos(p_0)\mathrm{sgn}(\psi)$ is given by Equation (25). Hence, Equation (14) is found. For ψ to be in the interval $(-\pi, \pi)$, the sign $\pm_s$ is chosen so that e_0 is positive.

Similar results are found when P rotates about the x- and y-axis. The largest rotation is given when e_0 is close to zero.

$$e_0 = p_0 q_0 + p_3 q_3 \tag{29}$$

$$= q_0 \cos(\frac{\psi}{2}) + q_3 \sin(\frac{\psi}{2}) = 0. \tag{30}$$

$$\tan(\frac{\psi}{2}) = -\frac{q_0}{q_3}. \tag{31}$$

Similar to the proof of Proposition 3.1, the orientation P_z furthest away from Q is given by

$$p_0 = \frac{\pm_s q_3}{\sqrt{q_0^2 + q_3^2}} \tag{32}$$

$$p_3 = \frac{\pm_t q_0}{\sqrt{q_0^2 + q_3^2}} \tag{33}$$

where the $\pm_s$ and $\pm_t$ have opposite signs.

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 ($\nabla f = 0$) for descent algorithms. A similar proof cannot be given for algorithms that are not descent.

4.2 Steepest Descent

The most common method for the minimisation of a function of several variables is the steepest decent, or the gradient method. The steepest descent is given by the iterative algorithm

$$x^{k+1} = x^k - \alpha^k \nabla f(x^k)^\mathsf{T} \tag{34}$$

where α^k is a non-negative scalar minimising $f(x^k - \alpha^k \nabla f(x^k))$. 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 $\nabla f(x) = 0$ can be proven (Luenberger, 2003).

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 $i = 1 \dots n$. The convergence of coordinate descent is in general poorer than the steepest descent. Coordinate descent is, however, easy to implement and, as the gradient is not required, a fast solution to the sub-problem makes these algorithms relatively fast.

4.4 Position and Orientation Error

This section presents a set of algorithms that solve the inverse geometric problem as seen from one joint. The solution of this sub-problem is the basis for all the algorithms presented in the next sections. Assume that only one joint can be moved, and consider the problem of finding the joint position which minimises the given cost function. All the algorithms presented are based on the analytical solution of a minimisation problem on $SE(3)$. This analytical solution guarantees that every sub-problem is computationally very efficient.

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.

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. In this case the solution depends on the choice of units. As angles and lengths cannot simply be added together, care must be taken.

4.4.1 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 $P_c = \begin{bmatrix} 0 & x_c & y_c & z_c \end{bmatrix}^\mathsf{T}$ of the end effector be given in the frame of joint i. Assume that the current position can be rotated about the z-axis, and hence represents one degree of freedom, given by all quaternions on the form $Q_z = \begin{bmatrix} \cos(\frac{\psi}{2}) & 0 & 0 & \sin(\frac{\psi}{2}) \end{bmatrix}^\mathsf{T}$ for $-\pi < \psi < \pi$. Then, the solution to the problem of finding the quaternion that takes P_c as close to P_d as possible is given by minimising

$$g_p(\psi) = (x_d - \hat{x}_c)^2 + (y_d - \hat{y}_c)^2 + (z_d - \hat{z}_c)^2, \tag{35}$$

where

$$\hat{P}_c = Q_z * P_c * Q_z^*. \tag{36}$$

By noting that

$$\hat{P}_c = \begin{bmatrix} 0 \\ \hat{x}_c \\ \hat{y}_c \\ \hat{z}_c \end{bmatrix} = \begin{bmatrix} 0 \\ x_c \cos\psi - y_c \sin\psi \\ y_c \cos\psi + x_c \sin\psi \\ z_c \end{bmatrix} \quad \text{for} - \pi < \psi < \pi, \tag{37}$$

$g_p(\psi)$ can be written as

$$g_p(\psi) = K_\psi + a_\psi \cos(\psi) + b_\psi \sin(\psi), \tag{38}$$

where

$$K_\psi = x_d^2 + y_d^2 + z_d^2 + x_c^2 + y_c^2 + z_c^2 - 2z_d z_c, \tag{39}$$
$$a_\psi = -2(x_d x_c + y_d y_c), \tag{40}$$
$$b_\psi = 2(x_d y_c - y_c x_d). \tag{41}$$

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

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

where

$$K_\theta = x_d^2 + y_d^2 + z_d^2 + x_c^2 + y_c^2 + z_c^2 - 2y_d y_c, \tag{43}$$
$$a_\theta = -2(x_d x_c + z_d z_c), \tag{44}$$
$$b_\theta = 2(z_d x_c - x_d z_d). \tag{45}$$

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 $\frac{\mathrm{d}g_p(\theta)}{\mathrm{d}\theta} = 0$:

$$\psi_{min} = \arctan 2\left(\frac{b_\psi}{a_\psi}\right) \pm \pi, \tag{46}$$

$$\theta_{min} = \arctan 2\left(\frac{b_\theta}{a_\theta}\right) \pm \pi \tag{47}$$

for a rotation about the z- and y-axes, respectively. In order to choose the solution that corresponds to the minimum and not the maximum value of g, choose the solution for which

$$\frac{\mathrm{d}^2 g_p(\theta)}{\mathrm{d}\theta^2} > 0. \tag{48}$$

Which solution to choose can also be determined by the following lemma.

Lemma 4.1 *Given a function $g(\theta)$ on the form*

$$g(\theta) = K + a\cos(\theta) + b\sin(\theta), \tag{49}$$

evaluated on $-\pi < \theta < \pi$. Let θ_{min} minimise $g(\theta)$. Then the following is always true

$$b > 0 \Rightarrow \theta_{min} < 0, \tag{50}$$
$$b < 0 \Rightarrow \theta_{min} > 0. \tag{51}$$

Proof The lemma is proved by contradiction. Let θ_{min} minimise $g(\theta)$. Assume that $b > 0$ and $\theta_{min} > 0$. Then on the interval $-\pi < \theta < \pi$, we have that $a\cos(\theta_{min}) = a\cos(-\theta_{min})$ and $b\sin(\theta_{min}) > b\sin(-\theta_{min})$. Thus we have that $g(\theta_{min}) > g(-\theta_{min})$ which is a contradiction as θ_{min} was assumed to minimise $g(\theta)$. Similarly for $b < 0$.

4.4.2 Orientation Cost Function

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

$$\hat{C} = Q_z * C = \begin{bmatrix} c_0\cos(\frac{\psi}{2}) - c_3\sin(\frac{\psi}{2}) \\ c_1\cos(\frac{\psi}{2}) - c_2\sin(\frac{\psi}{2}) \\ c_2\cos(\frac{\psi}{2}) + c_1\sin(\frac{\psi}{2}) \\ c_3\cos(\frac{\psi}{2}) + c_0\sin(\frac{\psi}{2}) \end{bmatrix}, \quad \text{for } -\pi < \psi < \pi \tag{52}$$

The orientation error is then given by

$$\begin{aligned} g_o(\psi) =& (d_0 - \hat{c}_0(\psi))^2 + (d_1 - \hat{c}_1(\psi))^2 \\ &+ (d_2 - \hat{c}_2(\psi))^2 + (d_3 - \hat{c}_3(\psi))^2 \\ =& 2 - 2(c_0d_0 + c_1d_1 + c_2d_2 + c_3d_3)\cos(\frac{\psi}{2}) \\ &+ 2(c_3d_0 + c_2d_1 - c_1d_2 - c_0d_3)\sin(\frac{\psi}{2}). \end{aligned} \tag{53}$$

$g_o(\psi)$ can be written as

$$g_o(\psi) = K_\psi + c_\psi\cos(\frac{\psi}{2}) + d_\psi\sin(\frac{\psi}{2}), \tag{54}$$

where

$$K_\psi = 2, \tag{55}$$
$$c_\psi = -2(c_0d_0 + c_1d_1 + c_2d_2 + c_3d_3), \tag{56}$$
$$d_\psi = 2(c_3d_0 + c_2d_1 - c_1d_2 - c_0d_3). \tag{57}$$

Similarly when the y-axis is the revolute axis.

$$\begin{aligned} g_o(\theta) =& 2 - 2(c_0d_0 + c_1d_1 + c_2d_2 + c_3d_3)\cos(\frac{\theta}{2}) \\ &+ 2(c_2d_0 - c_3d_1 - c_0d_2 + c_1d_3)\sin(\frac{\theta}{2}). \end{aligned} \tag{58}$$

$g_o(\theta)$ can then be written as

$$g_o(\theta) = K_\theta + c_\theta\cos(\frac{\theta}{2}) + d_\theta\sin(\frac{\theta}{2}), \tag{59}$$

where

$$K_\theta = 2, \tag{60}$$
$$c_\theta = -2(c_0d_0 + c_1d_1 + c_2d_2 + c_3d_3), \tag{61}$$
$$d_\theta = 2(c_3d_0 + c_2d_1 - c_1d_2 - c_0d_3). \tag{62}$$

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).

4.5 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 (38) and (54), respectively, so that the minimum of $g(\psi)$ is given by

$$\frac{\mathrm{d}g(\psi)}{\mathrm{d}\psi} = 0 \tag{63}$$

where

$$\frac{\mathrm{d}g(\psi)}{\mathrm{d}\psi} = b_\psi\cos(\psi) + d_\psi\cos(\frac{\psi}{2}) - a_\psi\sin(\psi) - c_\psi\sin(\frac{\psi}{2}). \tag{64}$$

This can be turned into an equation of degree four which can be solved analytically, for example by Ferrari's method. This will give four solutions. The solution that results in the smallest value of $g(\psi)$ is then chosen.

However, by avoiding the half angles in Equation (64), the solution is found simply by the inverse tangent and the computational complexity is reduced. In Wang and Chen (1991) a ψ is found by maximising $g(\psi)$. 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, which was not the case in Wang and Chen (1991). 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 ${}^{x}Q_d = Q_d * \boldsymbol{e}_i * Q_e^*$. ${}^{y}Q_d$ and ${}^{z}Q_d$ 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

$$ {}^{x}Q_d = Q_d * \boldsymbol{e}_i * Q_d^* = \begin{bmatrix} 0 \\ q_0^2 + q_1^2 - q_2^2 - q_3^2 \\ 2(q_1 q_2 + q_0 q_3) \\ 2(q_1 q_3 - q_0 q_2) \end{bmatrix}, \quad (65) $$

$$ {}^{y}Q_d = Q_d * \boldsymbol{e}_j * Q_d^* = \begin{bmatrix} 0 \\ 2(q_1 q_2 - q_0 q_3) \\ q_0^2 - q_1^2 + q_2^2 - q_3^2 \\ 2(q_0 q_1 + q_2 q_3) \end{bmatrix}, \quad (66) $$

$$ {}^{z}Q_d = Q_d * \boldsymbol{e}_k * Q_d^* = \begin{bmatrix} 0 \\ 2(q_0 q_2 + q_1 q_3) \\ 2(q_2 q_3 - q_0 q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}. \quad (67) $$

Then the cost function can be written as

$$ g(\psi) = w_p g_p(\psi) + w_o g_o(\psi) \quad (68) $$

where w_p and w_o are constant weights, $g_p(\psi)$ is given by Equation (38) 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 ${}^{x}Q_d = \begin{bmatrix} 0 & {}^{x}x_d & {}^{x}y_d & {}^{x}z_d \end{bmatrix}$. Assume that the z-axis is the revolute axis. Then the position of the unitary x-axis is given by ${}^{x}Q_c = \begin{bmatrix} 0 & \cos(\psi) & \sin(\psi) & 0 \end{bmatrix}$ and the difference is written as

$$ \begin{aligned} {}^{x}g_o(\psi) &= ({}^{x}x_d - \cos(\psi))^2 + ({}^{x}y_d - \sin(\psi))^2 + ({}^{x}z_d - 0)^2 \\ &= 2 - 2{}^{x}x_d \cos(\psi) - 2{}^{x}y_d \sin(\psi), \end{aligned} \quad (69) $$

and similarly for the y- and z-axes. By adding these three to Equation (38), $g(\psi)$ can be written as

$$ \begin{aligned} g(\psi) &= w_p g_p(\psi) + w_o({}^{x}g_o(\psi) + {}^{y}g_o(\psi) + {}^{z}g_o(\psi)) \\ &= K_\psi + a_\psi \cos(\psi) + b_\psi \sin(\psi) \end{aligned} \quad (70) $$

where

$$ \begin{aligned} K_\psi &= w_p \left(x_d^2 + y_d^2 + z_d^2 + x_c^2 + y_c^2 + z_c^2 - 2z_d z_c\right) \\ &\quad + w_o \left(6 - 2{}^{z}z_d\right), \\ a_\psi &= -2w_p(x_d x_c + y_d y_c) - 2w_o({}^{x}x_d + {}^{y}y_d), \\ b_\psi &= 2w_p(x_d y_c - y_d x_c) + 2w_o({}^{y}x_d - {}^{x}y_d). \end{aligned} $$

Similarly when the y-axis is the revolute axis

$$ g(\theta) = K_\theta + a_\theta \cos(\theta) + b_\theta \sin(\theta) \quad (71) $$

where

$$ \begin{aligned} K_\theta &= w_p \left(x_d^2 + y_d^2 + z_d^2 + x_c^2 + y_c^2 + z_c^2 - 2y_d y_c\right) \\ &\quad + w_o \left(6 - 2{}^{y}y_d\right), \\ a_\theta &= -2w_p(x_d x_c + z_d z_c) - 2w_o({}^{x}x_d + {}^{z}z_d), \\ b_\theta &= 2w_p(z_d x_c - x_d z_c) + 2w_o({}^{x}z_d - {}^{z}x_d). \end{aligned} $$

The minimum of the cost function, with respect to each joint, is given by Equation (46) and the error is given by $E = K + a$ (set $\psi = \theta = 0$ in (70) and (71)).

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

$$ g(\psi) = w_p g_p(\psi) + w_o g_o(\psi) + w_r g_r(\psi). \quad (72) $$

Whenever g_r can be written on the form of (70) 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.

Note also that for the pointing task, Equation (70) reduces to

$$ g(\psi) = w_p g_p(\psi) + w_o {}^{z}g_o(\psi) \quad (73) $$

which is widely used in applications such as spray painting, welding and high pressure water jets. In this case only the direction of the end-effector tool is considered and thus the computational complexity is reduced.

5 Solutions to the Inverse Geometric Problem

5.1 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 Section 5.2).

The cost function must be objective, i.e. independent of the coordinate frame in which it is measured (Lin and Burdick, 2000), and preferably describing some physical property, as the sum of the position and orientation error. Objectivity is important because all the calculations are done in local coordinates, and thus the coordinate frame changes for each iteration. Objectivity is in this case sufficient to guarantee that the algorithm is descent and convergent (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 4.5 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. The cost functions representing rotation or orientation error only are also well-defined on $SO(3)$ and $\mathbb{R}^3$, respectively. They can also be combined as described in Section 4.5 to a metric function on $SE(3)$. Caution must be taken when dealing with metrics on $SE(3)$, as it will depend on the choice of units and an unfortunate implementation of the algorithm may cause the algorithm to fail to converge. This is, as will be clear in the following, for example the case when iteratively optimising with respect to orientation and position error.

Three different approaches are presented:

Algorithm 1a: Loop until the error is under a threshold limit or a maximum number of iterations is performed.

- for each joint, in a pre-defined order, find the joint position that locally minimises the position error of the end-effector, as in Section 4.4.1.
- for each joint, in a pre-defined order, find the joint position that locally minimises the orientation error of the end-effector, as in Section 4.4.2.

Algorithm 1b: Loop until the error is under a threshold limit or a maximum number of iterations is performed. For each joint, in a pre-defined order

- find the joint position that locally minimises the position error of the end-effector, as in Section 4.4.1.
- find the joint position that locally minimises the orientation error of the end-effector, as in Section 4.4.2.

Algorithm 1c: Loop until the error is under a threshold limit or a maximum number of iterations is performed. For each joint, in a pre-defined order

- Minimise a cost function representing the sum of the position and orientation error, as in Section 4.5.

The change of reference frame on the cost function must be studied. The cost function needs to be objective, as defined in Lin and Burdick (2000), if not, the algorithm may fail to converge. A well defined metric function will guarantee that the the value of the cost function does not change with the change of reference frame which again means that it does not change with the joint. The cost function must also be so that the total error decreases when iterating between $SO(3)$ and $\mathbb{R}^3$ such as in Algorithms 1a and 1b. This is not guaranteed by just successively iterating between position and orientation as a decrease in the orientation error might cause an increase in the position error and vice versa. There is no guarantee that the total error decreases for every iteration.

5.2 Algorithm 2 - Modified Gauss-Southwell

The Gauss-Southwell Method determines the largest component of the gradient $\nabla g(x)$ 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 (46). 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.

5.3 Algorithm 3 - Gauss-Southwell

The method presented above can be modified somewhat so that each joint is chosen by the steepest descent instead of maximum possible descent. Assume

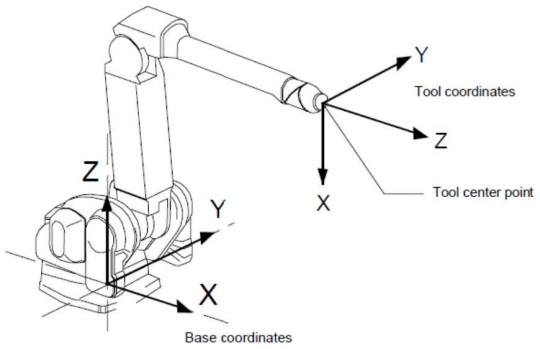

Figure 1: General structure of a robotic manipulator.

that the position of each joint that results in the minimum of the cost function $g(x)$ is found. Denote this by $\hat{x}_i^k$ for joint i and iteration k. The *rate* of decrease with respect to this joint is estimated by

$$\frac{\partial g(x_i^k)}{\partial x_i^k} \approx \frac{g(\hat{x}_i^k) - g(x_i^k)}{\left|\hat{x}_i^k - x_i^k\right|}, \qquad \text{for } i = 1 \dots n. \tag{74}$$

This is a good estimate only when $\left|\hat{x}_i^k - x_i^k\right|$ is small. Then the joint with the largest corresponding absolute value of the "gradient" is chosen. This approach is different from the solution given in Section 5.2 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

$$x_i^{k+1} = x_i^k + w_i(\hat{x}_i^k - x_i^k), \qquad \text{with } 0 < w_i \leq 1. \tag{75}$$

5.4 Algorithm 4 - Steepest Descent

Equation (74) gives information about all the joints. This information can be exploited by applying (75) to all the joints for every iteration. As the optimal position of each joint is found, assuming that all the other joints are fixed, the weights $0 < w_i \leq 1$ need to be chosen conservatively. As this approach requires approximately the same computational burden as the approach in the previous section but all joints are moved, the convergence can be improved substantially. The algorithm is not descent, and convergence cannot be proven. This is due to the fact that Equation (74) is an estimate of the gradient and not the actual gradient. In some cases the w_i's must be chosen very small which makes the convergence very slow.

5.5 Algorithm 5 - Manipulator Dependent Steepest Descent

The manipulator structure can 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 1, 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 1:

$$W = w_s \begin{bmatrix} 1 & 0.5 & 0.5 & 0.3 & 0.3 & 0.3 \end{bmatrix}. \tag{76}$$

As the previous algorithm, this algorithm is not descent. However, w_s can be set so that the behaviour of the algorithm resembles that of a descent algorithm. This is done at the cost of fast convergence. A simple approach to make the algorithm behave like a descent algorithm is to perform a test for every iteration to check whether the cost function has decreased or not. Then, if it has not, w_s should be reduced until a decrease in the cost function is obtained. For the steepest descent, a decrease of the cost function can be guaranteed as $w \to 0$. As $\nabla g(x)$ is only an approximation of the gradient, this cannot be guaranteed in this case.

5.6 Algorithm 6 - Steepest Descent with Gradient Estimate

Equation (74) can also be used to make an estimate of the gradient of the cost function. If the absolute sign is removed, the gradient of $g(x^k)$ can be estimated as

$$\widehat{\nabla} g(x^k) \approx \begin{bmatrix} \frac{g(\hat{x}_1^k) - g(x_1^k)}{\hat{x}_1^k - x_1^k} \\ \vdots \\ \frac{g(\hat{x}_n^k) - g(x_n^k)}{\hat{x}_n^k - x_n^k} \end{bmatrix} \tag{77}$$

As $g(x)$ is on the form of (70), $\left|\widehat{\nabla} g(x_i^k)\right| \leq \left|\nabla g(x_i^k)\right|$ for all i so that $\widehat{\nabla} g(x)$ is a conservative estimate of $\nabla g(x)$.

Now, Equation (77) can be applied to Equation (34) directly. The "step size" can be set similar to Equation (76) with (somewhat conservatively) $w_s = \min_{i=1\dots n} \left|\hat{x}_i^k - x_i^k\right|$. 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 (75) is applied to all joints, or the estimate of the gradient is applied in Equation (34), the algorithm is not descent. Again, however, the behaviour of the algorithm can be made descent by choosing the weights conservatively.

The steepest descent with gradient estimate differs from Algorithm 4 steepest descent in that for the steepest descent the optimal solution for each joint looked at separately is found, and then the update is done for all joints. For the steepest descent with the gradient estimate is the well known steepest descent method, but with an estimate of the gradient.

6 Numerical Examples

All the inverse geometric algorithms have been tested for a great variety of problems with the cost functions given in Sections 4.4 and 4.5. For comparison, the same test has also been done for a Jacobian-based inverse geometric algorithm. The Jacobian-based algorithm used in the simulations is an iterative algorithm based on the pseudo-inverse of the manipulator Jacobian, as the one presented in Robotics Toolbox (Corke, 1996). The convergence of the algorithms are tested for very difficult problems and very easy problems. Difficult problems are problems for which the solution is very far from the initial guess or the geometric considerations makes it difficult to "move" the manipulator from the initial condition to the solutions. For the easy problems the initial guess is chosen close to one of the solutions. 20 difficult and 20 easy problems are chosen and convergence is investigated for the two cases for all algorithms presented. The convergence for easy and difficult problems for all algorithms presented are plotted with respect to iterations and time in Figures 4-7.

6.1 Algorithm 1 - Coordinate Descent

The conventional CCD presented in Section 5.1 is computationally fast. The convergence of the CCD algorithms can be found in Figures 2-3. The following algorithms are tested:

- Alg1a $(6 \rightarrow 1)$
- Alg1b $(6 \rightarrow 1)$
- Alg1c $(6 \rightarrow 1)$
- Alg1c $(1 \rightarrow 6)$
- Alg1c (double sweep)

where Alg1x refers to the algorithms in Section 5.1 and $(6 \rightarrow 1)$ means that the algorithm works its way through the chain from the end effector to the base.

It is clear that the first two algorithms that optimise iteratively between orientation and rotation error are not descent and convergence is poor. It is found that optimising with respect to one criteria, while disregarding the other, will not necessarily decrease the sum of the two cost functions.

The three algorithms presented that are based on a cost function representing the sum of the orientation and position error are all descent algorithms and convergence is reasonably good due to the analytical solution of each sub-problem. The algorithm that starts at the base and works its way towards the end of the chain has fastest convergence in the beginning an also finds the most accurate solution. The fast analytical solution to the sub-problem, presented in Section 4.5 makes this algorithm reasonably good. Alg1c $(6 \rightarrow 1)$ is chosen to compare convergence in Figures 4-7.

6.2 Algorithm 2 - Modified Gauss-Southwell

Gauss-Southwell is computationally slower as it finds the minimum for all the joints but only one joint is chosen for decrease. As the Modified Gauss-Southwell finds the minimum possible value of the cost function by moving one joint only, it has the best convergence in the beginning among the algorithms that move only one joint at the time. This makes this algorithm a very good choice when an approximate solution to the problem is needed. Convergence is very good for 5-10 iterations. After this the convergence flattens out and one should switch to another algorithm to find an exact solution.

6.3 Algorithm 3 - Gauss-Southwell

Also the Gauss-Southwell has good convergence in the beginning, but only for about 5 iterations. Then it flattens out and the closest solution found is farther from the desired solution than for the Modified Gauss-Southwell. The algorithm easily gets stuck, and for the majority of the problems, it does not converge toward a correct solution. The algorithm can only be said to perform satisfactory for the first few iterations.

6.4 Algorithm 4 - Steepest Descent

The Steepest Descent moves all joints for every iteration which results in very good convergence. For a weight $w \approx 0.5$, the behaviour of the algorithm is very stable and a very accurate solution is found reasonably fast. This is the algorithm presented that best competes with the Jacobian approach when the initial condition is close to a solution. Also for more difficult problems, this is the algorithm that finds the most accurate solution if many iterations are allowed.

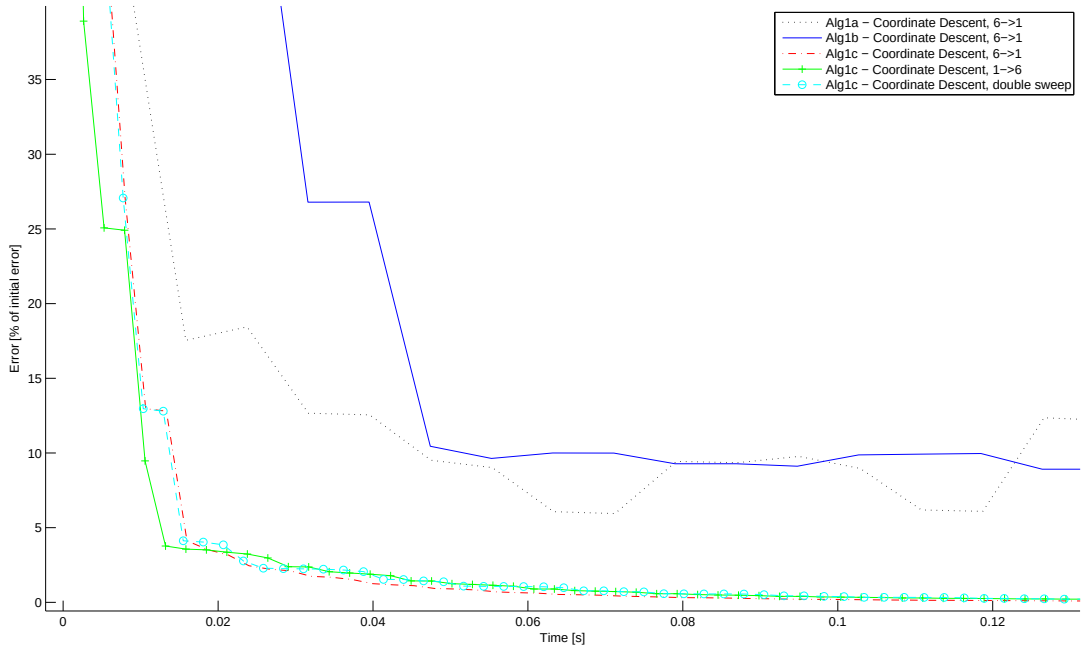

Figure 2: Convergence of Coordinate Cyclic Descent Algorithms that move one joint at the time. Initial conditions is set far from a solution.

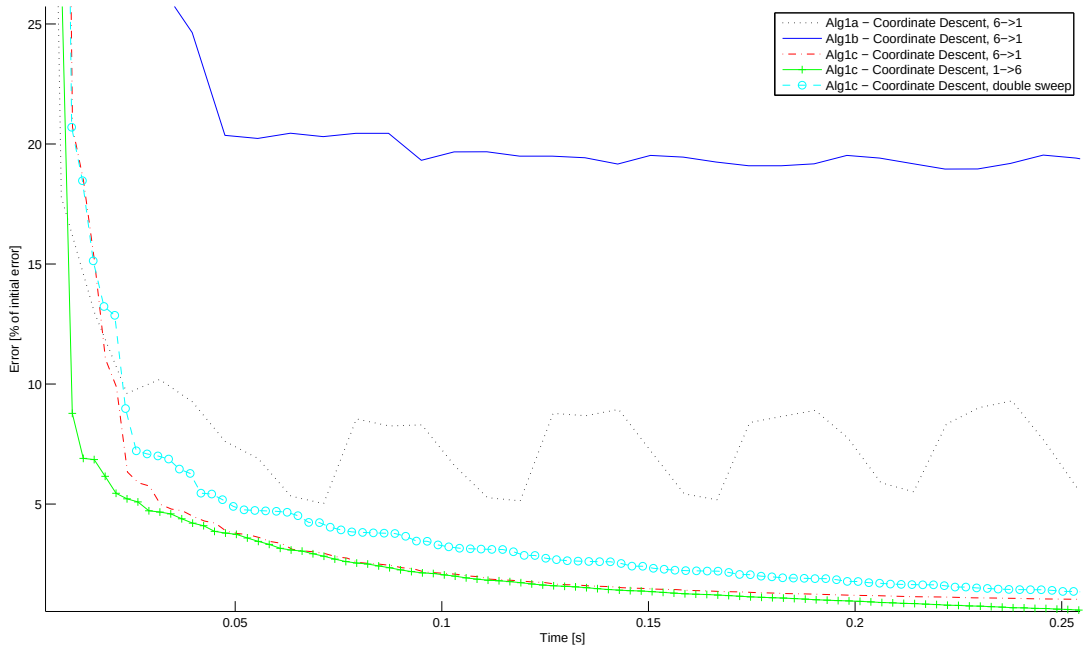

Figure 3: Convergence of Coordinate Cyclic Descent Algorithms that move one joint at the time. Initial conditions is set close to a solution.

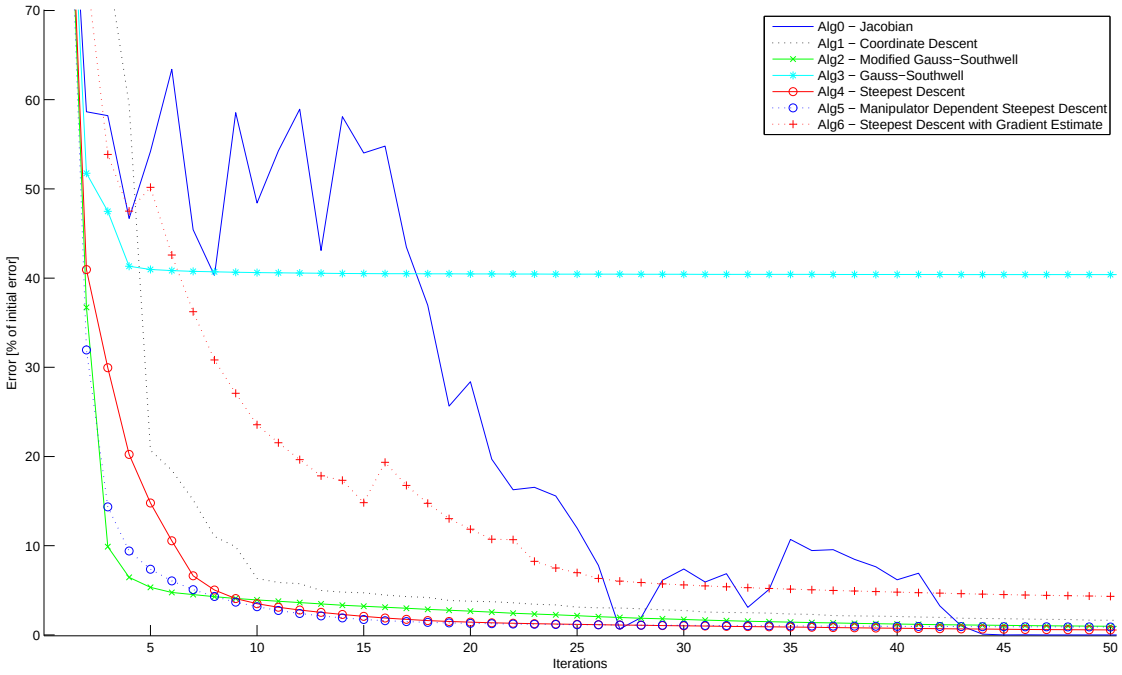

Figure 4: Convergence of algorithms with initial conditions far from a solution.

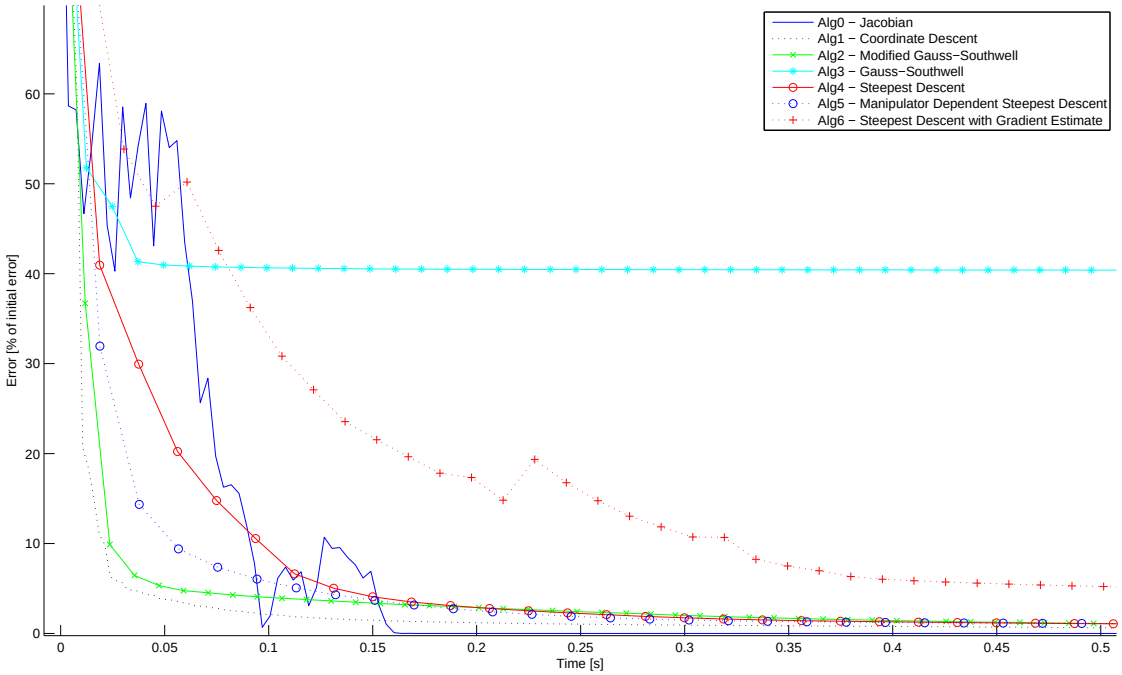

Figure 5: Convergence of algorithms with initial conditions far from a solution with respect to time.

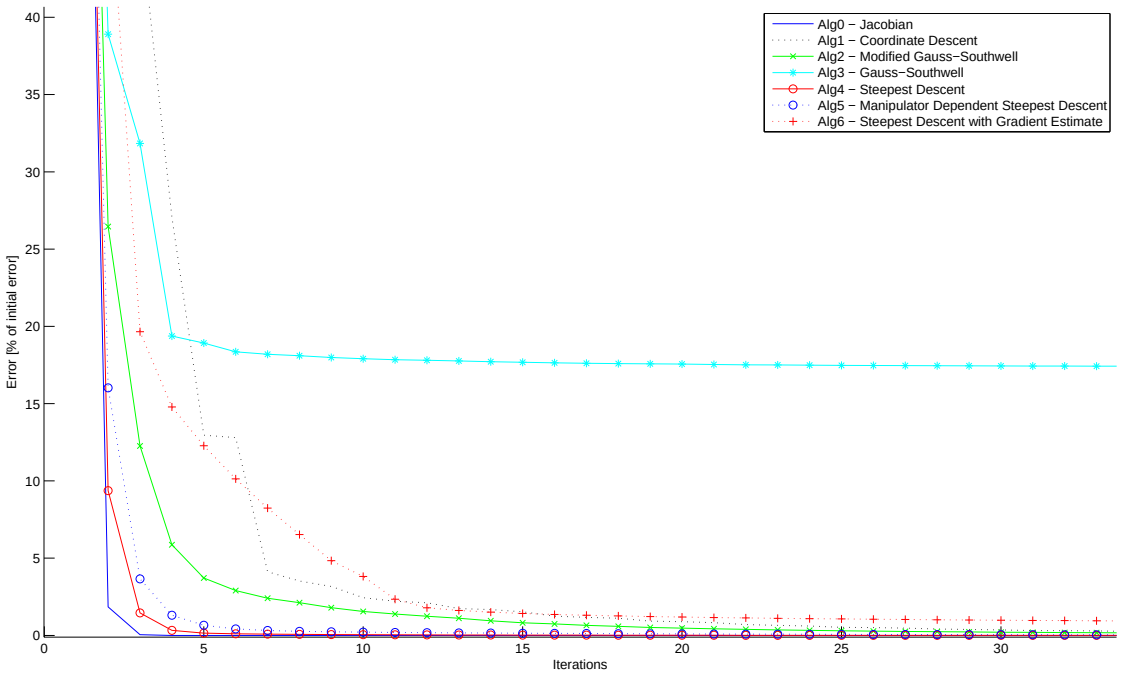

Figure 6: Convergence of algorithms with initial conditions close to solution.

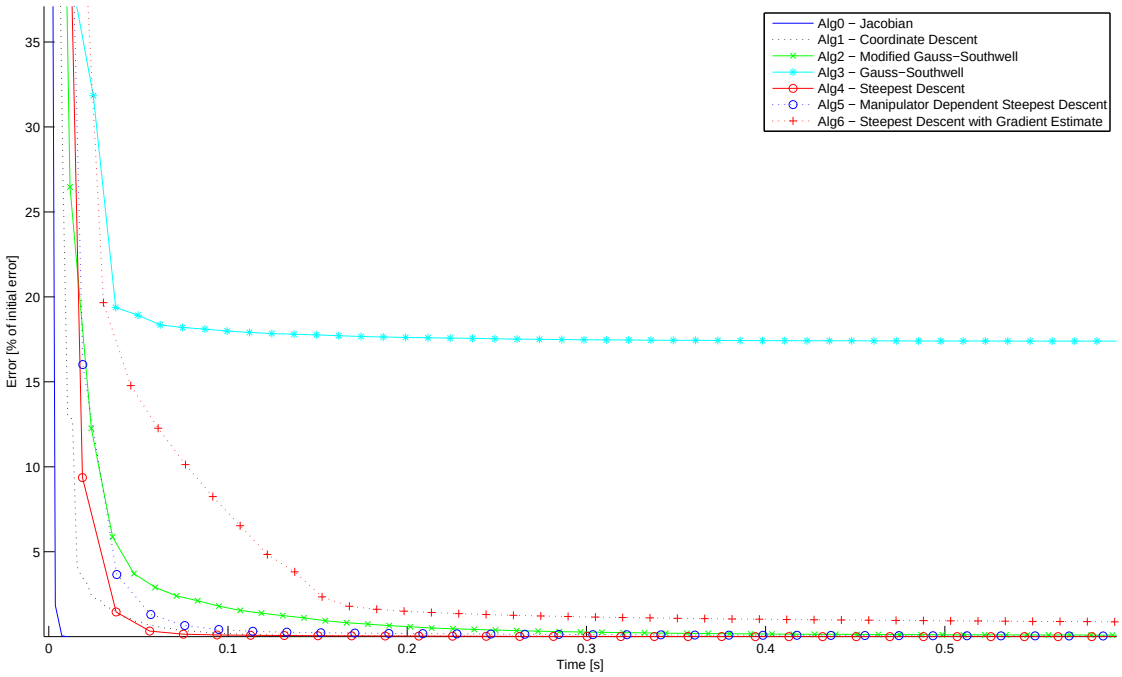

Figure 7: Convergence of algorithms with initial conditions close to solution with respect to time.

6.5 Algorithm 5 - Manipulator Dependent Steepest Descent

The convergence of the Manipulator Dependent Steepest Descent is about the same as the Steepest Descent, but convergence is better in the beginning for difficult problems. An algorithm that applies a few (5-10) iterations of the Manipulator Dependent Steepest Descent and then changes to Steepest Descent will give a fast and reliable algorithm which is easy to implement as the two algorithms are almost equal when it comes to implementation.

6.6 Algorithm 6 - Steepest Descent with Gradient Estimate

The Steepest Descent with Gradient Estimate is hard to tune and the weights need to be chosen relatively small for the algorithm to behave stable. This results poor convergence. The convergence is about the same as the Coordinate Descent methods, but the computational complexity makes this algorithm slower. The weight used in the simulations was $w_s = 0.05$.

6.7 Iteration Speed

The simulations were performed on an 2GHz processor. Table 1 shows the iteration speed of each algorithm. For algorithms 1-6, this is the time needed to analytically solve the optimisation problem and to update the joint position and the value of the objective function.

Algoritm	Iteration Speed [ms]
Alg0	3.85
Alg1	2.62
Alg2	12.36
Alg3	12.46
Alg4	18.87
Alg5	18.88
Alg6	15.27

Table 1: Iteration speed of each algorithm

7 Conclusions

A new class of solutions to the inverse geometric problem is presented. Convergence is found to be very good for problems which cannot be solved efficiently or cannot be solved at all with Jacobian-based algorithms. For all tests, an approximate solution was found in only a few iterations. The analytical solution to the sub-problem guarantees computational efficiency. A combination of the algorithms presented will give a stable and fast solution to any inverse geometrics problem. For problems with initial condition close to a solution, conventional Jacobian-based algorithms converge faster. The algorithms presented are thus well suited to find an initial condition for the Jacobian-based algorithms in order to improve convergence and guarantee that a solution is found.

Acknowledgments

The authors wish to acknowledge the support of the Norwegian Research Council and the TAIL IO project for their continued funding and support for this research. The TAIL IO project is an international cooperative research project led by StatoilHydro and an R&D consortium consisting of ABB, IBM, Aker Kvaerner and SKF. During the work with this paper the first author was with the University of California at Berkeley.

Appendix I - Formal Metric Proof

A metric on a set X is a function

$$\Psi : X \times X \to \mathbb{R} \tag{78}$$

which for all $x, y, z \in X$ satisfy the following conditions

1. $\Psi(x, y) \geq 0$
2. $\Psi(x, y) = 0$ if and only if $x = y$
3. $\Psi(x, y) = \Psi(y, x)$
4. $\Psi(x, z) \leq \Psi(x, y) + \Psi(y, z)$

Let $\mathbb{U}$ define the set of all quaternions of unit length

$$\mathbb{U}=\{(q_0, q_1, q_2, q_3)|q_0, q_1, q_2, q_3 \in \mathbb{R},\ q_0^2+q_1^2+q_2^2+q_3^2 = 1\} \tag{79}$$

Further let e_0 be the scalar part of $E = P * Q^*$ given by

$$e_0 = p_0q_0 + p_1q_1 + p_2q_2 + p_3q_3. \tag{80}$$

We will, without loss of generality, assume that all angles are in the interval $-\pi \leq \phi \leq \pi$.

Proposition 7.1 *The function*

$$\Psi_r = \mathbb{U} \times \mathbb{U} \to \mathbb{R} \tag{81}$$

given by $\Psi_r = \arccos(e_0)$*, is a metric function.*

Proof For all $P, Q, R \in \mathbb{U}$ we have

1. $\Psi(P, Q) \geq 0$
   We have

$$-1 \leq e_0 \leq 1 \Rightarrow \arccos(e_0) \geq 0. \tag{82}$$

2. $\Psi(P,Q) = 0$ if and only if $P = Q$
   We have
   $$\arccos(e_0) = 0 \tag{83}$$
   if and only if
   $$e_0 = 1 \tag{84}$$
   for which $P = Q$.

3. $\Psi(P,Q) = \Psi(Q,P)$
   $$\Psi(P,Q) = \arccos(p_0q_0 + p_1q_1 + p_2q_2 + p_3q_3) = \arccos(q_0p_0 + q_1p_1 + q_2p_2 + q_3p_3) = \Psi(Q,P). \tag{85}$$

4. $\Psi(P,R) \leq \Psi(P,Q) + \Psi(Q,R)$
   By definition the rotation $E = P * R^*$ takes P into R by the shortest rotation. This is obtained by the rotation
   $$\phi_{PR} = 2\arccos(e_0^{PR}) \tag{86}$$
   where e_0^{PR} is the scalar part of $P * Q^*$. Thus we have that
   $$\phi_{PR} \leq \phi_{PQ} + \phi_{QR}. \tag{87}$$
   Because the rotation from P to Q followed by the rotation from Q to R also take P into R, and from (86) and (87) we have
   $$\begin{aligned} \phi_{PR} &\leq \phi_{PQ} + \phi_{QR} \\ \frac{1}{2}\phi_{PR} &\leq \frac{1}{2}\phi_{PQ} + \frac{1}{2}\phi_{QR} \\ \arccos(e_0^{PR}) &\leq \arccos(e_0^{PQ}) + \arccos(e_0^{QR}) \\ \Psi(P,R) &\leq \Psi(P,Q) + \Psi(Q,R) \end{aligned} \tag{88}$$
   which concludes the proof.

Finally we show, by contradiction that the function

$$\Psi_r = \mathbb{U} \times \mathbb{U} \to \mathbb{R} \tag{89}$$

given by $\Psi_r = 1 - e_0$, is *not* a metric function.
Given the triangular inequality

$$\begin{aligned} \Psi(P,R) &\leq \Psi(P,Q) + \Psi(Q,R)) \\ (1 - e_0^{PR}) &\leq (1 - e_0^{PQ}) + (1 - e_0^{QR}) \\ -e_0^{PR} &\leq -(e_0^{PQ} + e_0^{QR} - 1) \\ e_0^{PR} &\geq e_0^{PQ} + e_0^{QR} - 1 \end{aligned} \tag{90}$$

Consider the following rotations

$$\begin{aligned} P &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T} \\ Q &= \begin{bmatrix} \cos(\frac{\phi}{2}) & 0 & \sin(\frac{\phi}{2}) & 0 \end{bmatrix}^\mathsf{T} \\ R &= \begin{bmatrix} \cos(\frac{2\phi}{2}) & 0 & \sin(\frac{2\phi}{2}) & 0 \end{bmatrix}^\mathsf{T} \end{aligned} \tag{91}$$

Then we have that both $P * R^*$ and $P * Q^*$ followed by $Q * R^*$ will take P into R. If we set $\phi = 0.1$ we have

$$\begin{aligned} e_0^{PQ} &= 0.9988 \\ e_0^{QR} &= 0.9988 \\ e_0^{PR} &= 0.9950 \end{aligned} \tag{92}$$

and we have

$$\begin{aligned} 0.9950 &\geq 0.9988 + 0.9988 - 1 \\ 0.9950 &\geq 0.9975 \end{aligned} \tag{93}$$

and thus a contradiction.

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.

Bronshtein, I. N., Semendyayev, K. A., Musiol, G., and Muehlig, H. *Handbook of Mathematics*. Springer, 2003.

Corke, P. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 1996. 3(1):24–32.

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. General solutions to kinematic and functional redundancy. *Proc. 46th IEEE Conf. on Decision and Control*, 2007.

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.

Hanson, A. J. *Visualizing Quaternions*. Morgan Kaufmann, 2006.

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.

Lin, Q. and Burdick, J. W. Objective and frame-invariant kinematic metric functions for rigid bodies. *International Journal of Robotics Research*, 2000. 19.

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.

Wen, J. T.-Y. and Kreutz-Delgado, K. The attitude control problem. *IEEE Transactions on Automatic Control*, 1991. 30 no. 10.

Yuan, J. S. C. Closed-loop manipulator control using quaternion feedback. *IEEE Journal of Robotics Automation*, 1988. 4, no. 4.