Numerical robot kinematics based on stochastic and molecular simulation methods

Thomas Kastenmeier* and Franz J. Vesely†
Institute of Experimental Physics, University of Vienna,
Boltzmanngasse 5, A-1090 Vienna, Austria, Europe.

(Submitted to Robotica, August 1995,
accepted September 1995)

Abstract

Multilink robot arms are geometrically similar to chain molecules. We investigate the performance of molecular simulation methods, combined with stochastic methods for optimization, when applied to problems of robotics. An efficient and flexible algorithm for solving the inverse kinematic problem for redundant robots in the presence of obstacles (and other constraints) is suggested. This "Constrained Kinematics / Stochastic Optimization" (CKSO) method is tested on various standard problems.

*e-mail: kast@pap.univie.ac.at
†e-mail: ves@pap.univie.ac.at

1 Motivation

There is an impressive arsenal of analytical and semianalytical methods for treating the mechanics of multilink manipulators [1, 2]. Generally, one attempts to set up a small (minimum?) set of internal coordinates $\mathbf{q}$ that determine the world coordinates $\mathbf{x}$ of the ultimate link (the “end effector”). A simple example would be a two-dimensional chain consisting of three links of fixed lengths, coupled by three joints (the first joint being fastened at the coordinate origin.) Here the obvious choice for the set of internal coordinates are the joint angles, α_{1-3}. In a more general context the Denavit-Hartenberg parameters [3], quaternions [5] or other generalized coordinates may be preferred.

Assuming the position of the first joint to be fixed, our simple manipulator has 3 degrees of freedom. The set $\mathbf{q} = \{\alpha_1, \alpha_2, \alpha_3\}^T$ uniquely determines the 2 world coordinates of the end point:

$$\mathbf{x} = \mathbf{x}(\mathbf{q}), \quad \text{with} \quad \mathbf{x} \equiv \{x_1, x_2\}^T \tag{1}$$

For some given temporal evolution $\mathbf{q}(t)$ of the internal coordinates the velocity components of the end effector may be computed from

$$\dot{\mathbf{x}}(t) = \mathbf{J}(\mathbf{q}) \cdot \dot{\mathbf{q}}(t) \tag{2}$$

where

$$\mathbf{J}(\mathbf{q}) \equiv \left\{ \frac{\partial x_\alpha}{\partial q_\beta} \right\} \tag{3}$$

is the Jacobian matrix of the transformation $\mathbf{q} \rightarrow \mathbf{x}$. As a rule $\mathbf{J}$ is *not* a square matrix, since the rank of $\mathbf{x}$ is lower than that of $\mathbf{q}$. As a consequence, there is no unique inverse to $\mathbf{J}$, meaning that different $\mathbf{q}(t)$ may result in the same effector trajectory: we are dealing with a *redundant* robot.

The formal solution of the underdetermined system 2 involves the *pseudoinverse* $\tilde{\mathbf{J}}$ defined by the equation

$$\dot{\mathbf{q}} \equiv \tilde{\mathbf{J}} \cdot \dot{\mathbf{x}} \tag{4}$$

and by the additional requirement that the quantity

$$A \equiv \dot{\mathbf{q}}^T \cdot \dot{\mathbf{q}} \tag{5}$$

be a minimum. This leads to the Moore-Penrose generalized inverse matrix

$$\tilde{\mathbf{J}} = \mathbf{J}^T \cdot \left[\mathbf{J} \cdot \mathbf{J}^T \right]^{-1} \tag{6}$$

Since the solution $\mathbf{q}(t)$ of the inverse kinematic problem is not unique, additional constraints or requirements may be introduced at will. These may involve circumventing obstacles, minimizing angular velocities or accelerations at the joints, or avoiding extreme – stretched or folded – configurations of the multilink structure.

Depending on the specific structure of the matrix $\mathbf{J}$, as defined by the geometry of the manipulator, various strategies for the inversion of equ. 2 with consideration of additional requirements may be devised. While these methods are mathematically elegant, they usually lack generality and flexibility. As a rule, any change of robot geometry, desired effector trajectory, or shape and position of external obstacles requires a completely new analysis and a change of computational strategy [4].

In the context of statistical-mechanical simulation the dynamics of chain molecules has long become routine. The use of internal angular coordinates was abandoned after very few attempts [6]. It became clear that the use of an overcomplete set $\mathbf{r}$ of cartesian coordinates describing the positions of the joints, together with geometrical constraints (usually, but not exclusively, referring to fixed bond lengths) is superior to the more elegant but less computer-suited formulation in terms of generalized coordinates $\mathbf{q}$.

In the case of our simple two-dimensional robot the preferred set of coordinates would be $\mathbf{r} \equiv \{\mathbf{r}_1, \ldots \mathbf{r}_4\}$, where $\mathbf{r}_1 = \{0, 0\}^T$ is the (trivial) position vector of the robot base, $\mathbf{r}_2$ and $\mathbf{r}_3$ are the positions of the two intermediate joints, and $\mathbf{r}_4 (\equiv \mathbf{x})$ is the position vector of the end effector. The 3 length constraints $|\mathbf{r}_{12}| = l_{12}$, $|\mathbf{r}_{23}| = l_{23}$, $|\mathbf{r}_{34}| = l_{34}$ and the 2 constraints $\mathbf{r}_1 = \mathbf{0}$ carve a set of 3 degrees of freedom out of an abundance of 8 coordinates. If we relax the condition $\mathbf{r}_1 = \mathbf{0}$ – by placing the robot on a vehicle, or suspending it in a satellite orbit – 5 degrees of freedom remain.

The most successful method for handling holonomic constraints of the type $|\mathbf{r}_j - \mathbf{r}_i| = const$ is due to Ryckaert et al. [7]. Their SHAKE algorithm consists of two steps. First the individual "interaction centers" (atoms or atomic groups) are allowed to move independently for one time step, subject only to the influence of inertia and of the interaction forces present in the multiparticle system. Second, the bond constraints along the chain are enforced by an iterative scheme.

Such a procedure seems inaccurate and clumsy, but one finds that a) errors stemming from the second step are of the same order as those of any numerical integrator for mechanical equations of motion; and b) the computational operations involved are so simple, and the iteration converges so fast, that the procedure is truly competitive. Its main advantage is that the introduction of orientational coordinates, with their inconvenient metric, is completely avoided. At all times the positions of the interaction sites are given in cartesian "world" coordinates. The internal angles at each time step may be evaluated from the scalar products of successive link vectors; however, they do not enter in the integration of the equations of motion.

The resulting table $\{\mathbf{r}^n \equiv \mathbf{r}(t_n);\ n = 1, \ldots\}$ of internal coordinates is one particular, more or less arbitrary, solution to the inversion problem. It represents the motion that would result if we were to force the effector to go through the predetermined trajectory with the desired velocity, having the other elements of the chain follow in accordance with their respective masses.

Next we want to introduce additional constraints or requirements. If these are of time-local character, i.e. pertaining to momentary positions or velocities, they may be cast in the form of a "cost function" that is to be minimized. Examples of such cost functions are the modulus of the largest angular velocity (or any power of this modulus); the largest angular acceleration; the largest velocity of any link or joint in the chain. Alternatively, the sum of squared angular velocities etc. could be used, but as a rule the largest element is preferable for identifying a locally optimal trajectory.

Out of the many available methods for minimizing a cost function depending on a large number of variables we will choose the *simulated annealing technique* (SA) due to Kirkpatrick et al. [9]. This stochastic minimization method is extremely flexible and easy to implement. It is capable of overcoming barriers in configuration space, such that the danger of mistaking a local minimum for the global one is small. In many cases this danger is negligible to begin with. It is possible in such instances to convert the SA method to a simple random search strategy just by choosing one parameter (the "temperature") to be small.

By combining the SHAKE algorithm and the SA (or the random search) technique we obtain a powerful numerical method for solving the inverse kinematic problem in the presence of time-local constraints. Since the use of generalized coordinates is avoided, a wide class of multilink bodies and a large variety of constraints may be treated by a single, easily adaptable computer program. In the following the proposed numerical scheme will be denoted by the abbreviation CKSO, for "constrained kinematics/stochastic optimization".

In Sections 2 and 3 the SHAKE and SA techniques, respectively, are reviewed under the aspect of their application in robotics. Section 4 discusses some useful terms to be incorporated in the cost function. (Depending on the specific task, or the given constraints, additional cost function terms may be invented.) Section 5 relates the results of test calculations on various two-dimensional multilink arms deftly moving amidst work space obstacles.

Future investigations will cover the generalization of the CKSO method to three dimensions, telescope arms, minimization of mechanical power, and inclusion of deformations.

2 Constrained kinematics using SHAKE

Chain molecules are often described in terms of the *Kramers model* consisting of atoms or atomic groups connected by bonds of fixed lengths. The motion of a particular atom i is restricted by conditions of the form

$$\sigma(t) \equiv |\mathbf{r}_{ij}(t)|^2 - l_{ij}^2 = 0 \tag{7}$$

where in a linear chain j may equal $i \pm 1$; for more complex molecular topologies, containing loops or branches, other particle indices may be involved. Introducing a Lagrange-type constraint force $\mathbf{c}_i$ acting to enforce $\sigma(t) = 0$ at all times, we have

$$\mathbf{c}_i(t) = -\lambda \nabla_i \, \sigma(t) = 2\lambda \mathbf{r}_{ij}(t) \tag{8}$$

$$\mathbf{c}_j(t) = -\lambda \nabla_j \, \sigma(t) = -2\lambda \mathbf{r}_{ij}(t) \tag{9}$$

A straightforward numerical strategy would then consist in evaluating this constraint force at time t_n and integrating the equations of motion for atoms i and j up to time $t_{n+1} \equiv t_n + \Delta t$. Since the numerical errors accumulating over many time steps would gradually disrupt the molecular structure, alternative methods had to be found.

According to Ryckaert et al. [7] a bond constraint is most effectively accounted for by the following formalism. Let $\mathbf{r}'_i$, $\mathbf{r}'_j$ be the *unconstrained* positions of the objects i and j at time t_{n+1}; by this we mean the positions the objects would attain if they were free to follow solely the influence of inertia and any potential forces. The constraint force is now assumed to act along the interparticle vector $\mathbf{r}_{ij}^n$ given *at the previous time step*, but with an adjustable prefactor such that $\sigma = 0$ is maintained at time t_{n+1}. Thus one writes

$$\mathbf{r}_i^{n+1} = \mathbf{r}'_i + \gamma \frac{(\Delta t)^2}{m_i} \mathbf{r}_{ij}^n \tag{10}$$

and similar for $\mathbf{r}_j^{n+1}$ (m_i denoting the respective masses of the atoms.) The free parameter γ, which may be thought of as an effective Lagrange factor, is calculated from the requirement that $\sigma(t_{n+1}) = 0$.

Instead of solving at once the quadratic equation for γ following from the bond constraint at time t_{n+1}, Ryckaert et al. proposed to use a linear iterative scheme. This makes sense in view of the fact that by repairing one constraint in a chain one necessarily destroys others. It is therefore necessary to go through the entire chain repeatedly until all constraints are met up to some given tolerance. Since each individual chain link has thus to be visited several times, an approximate treatment at each such visit will suffice to obtain the desired accuracy.

The detailed procedure is as follows:

- Given $\mathbf{r}'_i$ and $\mathbf{r}'_j$, compute

$$\sigma' \equiv \sigma(\mathbf{r}'_{ij}) \equiv \left|\mathbf{r}'_{ij}\right|^2 - l_{ij}^2 \tag{11}$$

- Try to improve the positions according to

$$\mathbf{r}_i'' = \mathbf{r}_i' + \frac{a}{m_i}\mathbf{r}_{ij}^n \tag{12}$$

$$\mathbf{r}_j'' = \mathbf{r}_j' - \frac{a}{m_j}\mathbf{r}_{ij}^n \tag{13}$$

  where a is determined from the requirement that $\sigma'' = 0$ (neglecting the quadratic term in a):

$$a = \frac{\sigma' \mu}{2\mathbf{r}_{ij}^n \cdot \mathbf{r}_{ij}'} \tag{14}$$

  (with $1/\mu \equiv 1/m_i + 1/m_j$.)

- Repeat this procedure for all elements of the chain and iterate several times through the entire chain until the absolute values of all σ' are small enough to be negligible.

The application of SHAKE to the inverse kinematic problem of robotics consists of the following steps: 1) define the desired trajectory of the end effector by a list of positions as a function of time, $\{\hat{\mathbf{x}}^n \equiv \hat{\mathbf{x}}(t_n)\}$; 2) for each time step $n \to n+1$, move the end effector to the required position and apply SHAKE to adjust the intermediate joints and limbs accordingly. This is most effectively done by fixing the masses (appearing in equs. 12-14) of the robot base and the end effector at infinity, while the intermediate joints are given a mass of 1. In this way, find $\mathbf{r}^{n+1} \equiv \mathbf{r}(t_{n+1})$ describing a new configuration of the entire robot arm.

It should be noted that a very similar procedure can be applied to enforce orientational or angular constraints, which may be necessary for the generalization to three dimensions and for implementing extendable shafts for telescope arms.

3 Path optimization by simulated annealing

Finding the minimum of a function $U(y_1, \ldots y_d)$ of a large number of variables is a notoriously hard task. The regular scan of a dense coordinate grid in, say, 8 dimensions takes very long indeed, and for 16 dimensions it is out of question. An alternative method is the *random search*. It consists in starting from some arbitrary point $\mathbf{y}$, making a trial step $\Delta\mathbf{y}$ with random direction and size, and computing the new value of U. If the difference $\Delta U \equiv U(\mathbf{y}+\Delta\mathbf{y}) - U(\mathbf{y})$ is negative, accept the new position and reiterate. If $\Delta U > 0$, the trial step is rejected.

A possible problem with this procedure is that the configuration space point $\mathbf{y}$ can move "downhill" only. A minimum of U is accessible only if it may be reached by such a downhill path. If a global minimum is separated from the starting region by a ridge, it can never be found, and the system point will come to rest in a local minimum.

To remove this flaw, Kirkpatrick et al. [9] introduced a slight but decisive change in the above prescription. Instead of rejecting a random step that has $\Delta U > 0$, accept it with probability $p \equiv \exp(-\beta\Delta U)$, where $\beta > 0$ is an adjustable control parameter. Small values of β make the random walk insensitive to the variations of U. If β is large, uphill steps become increasingly difficult, and with $\beta \to \infty$ the algorithm is equivalent to the simple random search. By starting out with a small β and increasing it gradually in the course of the random walk through configuration space we have a fair chance to find the *global* minimum of U.

The Kirkpatrick technique is really just a particular application of the Monte Carlo method devised by N. Metropolis [10]. Interpreting β as a reciprocal temperature according to $\beta = 1/kT$ we see that the transition probability p is just the quotient of two Boltzmann factors. Increasing β in the course of the calculation is tantamount to lowering the temperature, which explains the name "simulated annealing method."

How does all this relate to robotics? Assuming that by applying SHAKE we have found a new configuration of the multilink arm, we want to make sure that some cost function be minimized. A time sequence of such optimized configurations defines an "optimal path".

Since the inverse kinematic problem for a redundant manipulator is underdetermined, we may try out slight variations of the new internal coordinates $\mathbf{r}^{n+1}$ (keeping $\mathbf{x}^{n+1}$ as well as the robot base $\mathbf{r}_1$ fixed) and look for lower values of $U(\mathbf{r}^{n+1})$. To this end we perform a random walk in the space of valid robot configurations for each timestep. Let the robot arm (having N joints) be represented by an open polygon, with the vertices numbered from 1 (=base) to $N+1$ (=end point). Both the base and the end point remain fixed during the following procedure. First an intermediate joint $(i = 2, \ldots N)$ is picked at random; its position is changed by a random vector $\Delta\mathbf{r}_i$. Then SHAKE is applied once more to re-enforce the constraints, leading to a new valid trial robot configuration for the current time step. Now the corresponding value U of the cost function is calculated, which can be a function of any time-local variable, such as the coordinates of a joint, its velocity or acceleration components, the angle between successive limbs, the angular velocity or angular acceleration, or even the time itself (see Section 4 below).

To decide whether to accept or reject this trial step depending on U we apply the simulated annealing method explained before. By repeating the described procedure several times, while gradually increasing β (i.e. lowering the temperature T), we get close to the global minimum of the cost function. Now we have found a near-optimal configuration for the current time.

This random walk is performed at each time step, leading to a robot trajectory which is optimized with respect to the given time-local conditions. Thus the inverse kinematic problem with additional constraints is solved in a step-by-step manner using the simulated annealing method.

The method described here may be seen as applying random forces to the intermediate joints to try out various valid robot configurations in the course of the random walk. The masses of these joints (which enter SHAKE; see equs. 12-14) barely affect the result if they do not enter the cost function, because then they have no influence on its global minimum. This is the reason why in such cases they may all be set to 1.

In order to minimize the time needed for finding the global minimum of the cost function and thus speeding up the whole algorithm we would like to start the random walk from a configuration that is already quite close to the global minimum. Then we can start at a lower temperature, and the annealing procedure will take a smaller number of steps. The starting configuration involves the initial $\mathbf{r}'_i(t_{n+1})$, i.e. the unconstrained positions of the intermediate joints i at time t_{n+1} (before applying SHAKE). The best first guess for these proved to be a rectilinear uniform translation, meaning that $\mathbf{r}'_i(t_{n+1}) = 2\mathbf{r}_i(t_n) - \mathbf{r}_i(t_{n-1})$. Having moved the effector to the required position $\hat{\mathbf{x}}(t_{n+1})$ we can apply SHAKE and get a good valid starting configuration.

Test simulations of several different (2-dimensional) problems have shown that the result is still a well-optimized robot trajectory if we choose a very low temperature T right from the start, instead of gradually lowering it. This means that for the investigated cost functions a simple (and fast) *random search strategy* is good enough, provided that we use the above-explained starting configuration.

It is very easy to calculate the robot trajectory not only for given effector positions as a function of time, but also for additionaly given effector orientations. One simply sets the "mass" m_N of the last joint at a high value, calculates and updates its given position for the starting configuration, and in the subsequent random walk varies only the positions of the remaining intermediate joints, $i = 2, \ldots, N-1$. Due to this very high mass as compared to the masses of the other intermediate joints (which are taken to be = 1), SHAKE will not alter the position of the last joint. By the same token it does not change the positions of the origin and the end of the robot arm, their "masses" being infinite. Since the positions of last joint and end point define the orientation of the last limb, such a procedure yields valid robot configurations, suiting the required effector position *and orientation* at each timestep.

There is a promising alternative method to solve the inverse kinematic problem of redundant robots, which has not been fully investigated yet (but will be part of further studies). Instead of using SHAKE combined with simulated annealing it is also possible to use the Moore-Penrose pseudoinversion combined with simulated annealing (or possibly, with random search). The principle of such a "Rate Control / Stochastic Optimization" (RCSO) method is the following:

If any solutions $\dot{\mathbf{q}}$ to equ. 2 exist, the complete set of solutions is formally given by

$$\dot{\mathbf{q}} = \tilde{\mathbf{J}} \cdot \dot{\mathbf{x}} + \left(\mathbf{I} - \tilde{\mathbf{J}} \cdot \mathbf{J}\right) \cdot \mathbf{z} \tag{15}$$

where $\mathbf{I}$ is the unit matrix, $\tilde{\mathbf{J}}$ is the Moore-Penrose generalized inverse of $\mathbf{J}$ (see equ. 6), and $\mathbf{z}$ is an arbitrary vector in $d\mathbf{q}/dt$ space. The step-by-step solution $\mathbf{q}(t_n)$ for a given effector trajectory $\hat{\mathbf{x}}(t_n)$ may be found by numerical integration of eq. 15:

$$\mathbf{q}^{n+1} = \mathbf{q}^n + \tilde{\mathbf{J}}(\mathbf{q}^n) \cdot \left(\hat{\mathbf{x}}^{n+1} - \mathbf{x}^n\right) + \left(\mathbf{I} - \tilde{\mathbf{J}}(\mathbf{q}^n) \cdot \mathbf{J}(\mathbf{q}^n)\right) \cdot \mathbf{z}^n \Delta t \tag{16}$$

where n is the discrete time, $\mathbf{x}^n$ is the effector position (and orientation) at time t_n, which due to small discretization errors is not quite the same as $\hat{\mathbf{x}}^n$, and Δt is the size of the time interval.

The search for the global minimum of a given cost function can now be performed by again applying simulated annealing. This time we create a random walk in the space of the arbitrary vector $\mathbf{z}$ for each timestep by randomly picking one component of $\mathbf{z}$ and adding a random Δz_i. Thus a new trial robot configuration for the current time step is obtained by simply adding the i-th column of the matrix $\mathbf{I} - \tilde{\mathbf{J}} \cdot \mathbf{J}$, multiplied by Δz_i times Δt, to $\mathbf{q}(t_{n+1})$. Now its cost function value is calculated and the decision whether to accept or reject is made as in the CKSO method. A continuous $\mathbf{z}(t)$ leads to a continuous $\dot{\mathbf{q}}(t)$, as evident from equ. 15. It is therefore reasonable to use, as a starting vector $\mathbf{z}$ at the beginning of each random walk, the "optimized" $\mathbf{z}$ obtained in the previous time step.

The advantage of using Moore-Penrose pseudoinversion instead of SHAKE is a faster random walk loop, since by varying $\mathbf{z}$ to produce other trial configurations the constraints are not violated, which removes the need of repeatedly invoking SHAKE at each random walk step. The disadvantage is that the algorithm loses some flexibility, since the Jacobian matrix $\mathbf{J}$ has to be derived anew for every specific kind of robot, while SHAKE may be implemented in a very general way, thus being applicable to a large class of different robot geometries. Also, Δt may be chosen more freely with SHAKE, because there the accuracy of the effector position and orientation at the discrete time steps t_n does not depend on Δt. (It depends on the given tolerance for the constraints in SHAKE.)

An efficient combination of both methods, together with simulated annealing, seems to be possible:
a) Use Moore-Penrose pseudoinversion for a faster random walk loop; b) at the end of the random walk loop move the effector to its exact required position (and orientation) and then apply SHAKE just once for higher accuracy with respect to the given effector trajectory (possibly putting up with a slight loss of optimality).

4 The cost function

The crucial step in the CKSO and RCSO methods for use with redundant robots is to set up a cost function whose minimum defines a "good" robot trajectory. In the 2-dimensional task simulations we have performed the cost function relates to minimal angular velocities and accelerations, joint limits avoidance, singularity avoidance, avoidance of collisions with obstacles, as well as finding *periodic* robot trajectories for given periodic effector motions. All these requirements may be fulfilled by using one single cost function. The general form of the investigated cost function is

$$U = \sum_k c_k f_k \tag{17}$$

where f_k denotes the individual contributing terms, and c_k are their respective weights. Depending on these relative weights, different terms will be more strictly minimized. Thus the weights decide which are the requirements that to fulfill the redundant degrees of freedom are used predominantly.

It is advisable to adapt the various weighting factors to each different kind of task to be performed by the robot; a trial and error strategy is recommended. One will start the simulation of a certain kind of task using some reasonable standard cost function, with an intelligent guess for the weighting coefficients. In the course of the investigations these factors are modified, and additional terms (e.g. concerning angle positions) may have to be added to the cost function, until the resulting robot trajectory is satisfactory. Thus as a rule simulations are necessary before a certain kind of task can be performed by the robot, as a suitable cost function has to be found first. However, it turned out that with a "certain kind of task" a large class of similar tasks (e.g. with different obstacle positions or another effector trajectory) may be performed as well.

Angular velocities: In order to minimize the absolute value of the angular velocities $d\alpha_i/dt$ (of angles between successive limbs) the term

$$f_{av} = \max_i \dot{\alpha}_i^2 \tag{18}$$

or, alternatively,

$$f_{av} = \frac{1}{N} \sum_{i=1}^{N} \dot{\alpha}_i^2 \tag{19}$$

is used, where N is the number of robot joints. In the calculations $d\alpha/dt$ is usually approximated by the central difference quotient:

$$\frac{d\alpha}{dt} \approx \frac{\alpha^{n+1} - \alpha^{n-1}}{2\,\Delta t} \tag{20}$$

where α^{n+1} denotes an angle at time t_{n+1}; it is varied in the course of the random walk, while α^{n-1} remains fixed.

Angular accelerations: To minimize the absolute value of the angular accelerations $d^2\alpha_i/dt^2$, either of the terms

$$f_{aa} = \max_i \ddot{\alpha}_i^2 \quad \text{or} \quad f_{aa} = \frac{1}{N}\sum_{i=1}^{N} \ddot{\alpha}_i^2 \tag{21}$$

may be included in the cost function. $d^2\alpha/dt^2$ is conveniently approximated by the second central difference quotient:

$$\frac{d^2\alpha}{dt^2} \approx \frac{\alpha^{n+1} - 2\alpha^n + \alpha^{n-1}}{(\Delta t)^2} \tag{22}$$

Preferred angles: To prefer certain regions of joint angles and avoid others (such as the joint limits), the use of a term like

$$f_{pa} = \frac{1}{N}\sum_{i=1}^{N} \alpha_i^4 \tag{23}$$

is suggested, with α_i denoting the varied angle at time t_{n+1}. In 23 it is assumed that for all joints the preferred angle is zero (i.e. a stretched configuration) and that their angular working ranges are about the same; thus limit angles which are beyond this range are automatically avoided. (If different angular regions are preferred, a trivial modification takes care of that.) A power of 4 (rather than, say, 2) is used to have a flat $f(\alpha)$ near the preferred angular region, thus providing more freedom for minimizing other terms in the cost function. As the effector has to follow its given trajectory, this term results in a completely unfolded, though arched, robot arm with some more or less equal offsets from the preferred angles.

It turns out that the resulting robot configurations keep well away from any singular arrangement. A suitably defined distance of a given robot configuration from a kinematic singularity is the manipulability d_m; it is defined by [11]

$$d_m \equiv \sqrt{\det\left(\mathbf{J}\cdot\mathbf{J}^T\right)} \tag{24}$$

We have found that another, more conveniently accessible quantity may be used to monitor the distance from singularities. Let $\overline{s}_{iter}$ denote the number of necessary iterations per call to SHAKE, averaged over all such calls at the current time. Then the "convergence speed" of the SHAKE algorithm, defined as

$$v_c \equiv \frac{1}{\sqrt{\overline{s}_{iter}}} = d'_m \tag{25}$$

measures the effectiveness of the routine re-enforcing the constraints up to the given tolerance. It is calculated for each random walk loop. The lower the convergence speed of SHAKE, the closer we are to a singular position; if it turns out that SHAKE does not converge at all, meaning that not all constraints can be reinstalled at the same time, we obviously require something impossible from the robot arm. This, however, is equivalent to saying that we have a singular configuration. The expected qualitative similarity in the behaviour of v_c and d_m was checked, and proved, in the simulations [8]. Of course, this alternative measure for singularity distance is very specific to the use of SHAKE.

In case an explicit singularity avoidance is needed, it is possible to provide for this by using an additional cost function term. One approach would be to maximize the manipulability as defined in equ. 24. Alternatively – and more efficiently – the singularities (regions of zero manipulability) for the given robot structure could first be calculated analytically or numerically, then a suitably defined distance (such as a norm) between the momentary configuration and each such singularity (or singular region) may be included in the cost function.

The above contributions to the cost function, equs. 18 (or 19), 21 and 23, should always be included. If particular joints are "more important" than others, further weighting-factors (involving specific angular regions, velocities, etc.) may be inserted in these terms. The following terms are more problem-dependent ones:

Limit angles: Equ. 23 by itself leads to a preference for certain angular regions, but does not strongly enforce limit angles. If there are such limit values of joint angles that should not be far exceeded, we additionally include, with a very large weighting factor, terms such as

$$f_{la} = \begin{cases} (\alpha_i - \alpha_i^{max})^2 & (\alpha_i > \alpha_i^{max}) \\ 0 & (\alpha_i^{max} \geq \alpha_i > \alpha_i^{min}) \\ (\alpha_i - \alpha_i^{min})^2 & (\alpha_i^{min} \geq \alpha_i) \end{cases} \tag{26}$$

in the cost function.

Obstacle avoidance: One of the main objects of using redundant robots is their ability to avoid collisions with obstacles (see Figure 1. It is here that the flexibility of the method described above is most evident. The pertinent contributions to the cost function may be defined at will, its derivatives need never be computed, etc. In this treatment the obstacles are represented by polygons and circles (in 2 dimensions) or polyhedrons and spheres (in 3 dimensions). The mathematical description of the robot as a sequence of line segments requires that the half-diameter r of the limbs be added to the dimensions of the (thus "blown up") obstacles. Alternatively, it is possible to avoid any collisions by keeping a minimum distance r to the unmodified ("real") obstacles.

In order to make the distances between the robot arm and the obstacles as large as possible, all relevant combinations of distances between "robot elements" and "obstacle elements" are calculated. Let the number of pertinent distances d_j be z; then the cost function is augmented by

$$f_{oa} = \frac{1}{z} \sum_{j=1}^{z} \frac{1}{d_j^2 - r_j^2} \tag{27}$$

or a similar reciprocal distance term. Minimizing a cost function with this term leads to large distances d_j. The "strength of repulsion" may be varied via its weighting-factor; the parameter r_j allows us to select an individual minimal distance. It is evident that at a large distance d_j the respective contribution has hardly any influence on the minimum of the cost function, and therefore on the robot trajectory. This is desirable, since in these regions the robot's redundancy is used to minimize other terms.

For collision avoidance in two dimensions every d_j is calculated as the minimal distance between a point and a line segment (see Fig. 2). Thus level lines of equal distance between a point and a line AB are as shown in Fig. 3. The d_j are now the minimal distances of all robot joints (whose positions are varied in the course of the random walk) to all straight lines representing the obstacle walls, as well as the distances of all obstacle corners (polygon points) or circle centers to all robot limbs represented by straight line segments. In three dimensions the straight line representing a obstacle wall becomes a plane, the obstacle corner will be a obstacle edge, and a circle center turns into a sphere center. Of course, all segments and joints of the robot arm must avoid not only external obstacles, but also each other, and in particular the end effector. Normally, the term 23 will take care of this in a "soft" manner; if need be, additional d_j's describing these distances (effector position - robot limbs) may be used. Quite generally, moving obstacles (of which the effector is only one particular example) may be implemented easily, as the cost function can be time-dependent.

Effector alignment: Near a obstacle corner (2 dim.) or edge (3 dim.) it may be necessary to request a specific orientational alignment of the last limb, such that the effector can move around the edge without causing excessive angular velocities. This may be achieved by the use of the term

$$f_{ea} = \frac{(\gamma - \gamma_0)^2}{s^2 + b} \tag{28}$$

where s denotes the distance between the effector (i.e. endpoint of the robot polygon) and the edge, γ is the alignment of the ultimate robot limb (in 2 dimensions, $\gamma = \sum_{i=1}^{N} \alpha_i$), and γ_0 its desired value. With the parameter b the strength of alignment near the edge can be adjusted. The term s guarantees that the limb is aligned *only* if it is *near* the edge (see Fig. 4).

Periodic paths: When we solve the inverse kinematic problem for a redundant robot, requiring a periodic effector trajectory does not necessarily lead to a periodic sequence of internal coordinates. However, such a periodic path in configuration space is desirable for many reasons. In particular, the internal configuration data for periodic effector motions may then be stored in a table of finite length; also, after one period no further surprises (collisions with obstacles etc.) are to be expected.

As a simple but efficient way to obtain a periodic robot trajectory for a given periodic effector trajectory, we suggest to include an additional term in the cost function: A data vector is set up which can store the discrete configurations of the robot for exactly one period of effector motion. As the calculation of the time development proceeds, the "oldest" (i.e. by exactly one period preceding) configuration is overwritten by the current one, so that this data vector always contains exactly one period of the most recent configurations up to the current time t. First a few effector cycles are performed considering only the cost terms described so far. (At least one such period is necessary to fill the data vector, but things may settle down a bit if several cycles are calculated.) Now the requirement for a periodic robot trajectory is "switched on" (at time t_p) by including the following term in the cost function:

$$f_{pp} = (t - t_p) \cdot \Theta\,(t - t_p) \cdot \max_i |\mathbf{r}_i - \mathbf{r}_i^p| \qquad (29)$$

where $\mathbf{r}_i$ are the coordinates of the i-th robot joint whose position is varied in the course of the random walk, $\mathbf{r}_i^p$ are the coordinates of the same robot joint exactly one period of time earlier (as taken from the data vector), and $\Theta(t - t_p)$ is the Heaviside step function responsible for the switching on. This term is *gradually* enforced due to the multiplicative factor $(t - t_p)$ in equ. 29 (how fast depends on the weighting factor), while further time steps are calculated as so far. This causes the current configuration to join in a smooth way to the end of the data vector, since the positions of the intermediate joints are gradually forced to approach those of one period earlier, as $\max_i |\mathbf{r}_i - \mathbf{r}_i^p|$ is minimized. The calculation of further timesteps is terminated as soon as $\max_i |\mathbf{r}_i - \mathbf{r}_i^p|$ is smaller than some tolerance limit, which must not be chosen too small since the accuracy attainable by the *stochastic* method of simulated annealing is limited. Now the data vector will contain a robot trajectory which is periodic within the scope of the attainable accuracy. It should be noted that any representation of the robot configurations may be used. Instead of the cartesian coordinates of the individual joints used in eq. 29, some kind of generalized robot coordinates, such as angles and sliding shaft lengths, may be inserted in 29 to define the distance between the current configuration and the one which is a full period older.

5 Task simulations

The CKSO method described in the preceding sections was tested by application to various tasks that are frequently treated in the literature. A simulation program was written in C language, to be executed on a general-purpose PC. The high flexibility of the method is expressed by the fact that widely varying tasks can be performed using the same computer code. The parameters describing both the robot's structure and the task, such as the number of joints and limbs, the form of the desired effector trajectory and the positions and shapes of external obstacles, are simply input parameters to be set at run time. The same holds for the weight factors of the various contributions to the cost function; if need be, these coefficients may be adjusted by trial and error.

The set of cost function contributions that have been included in the current version of the code consists of the seven elements described in Section 4. Additional terms may be defined and added easily, requiring only a few additional lines of code.

The computation time depends mainly on the number of elements in the polygons describing the robot and the obstacles, and on details of the simulated annealing procedure, such as starting temperature, annealing speed, etc. On a 486-type PC the computer time required per time step is typically 0.2 seconds (for a relative SHAKE tolerance of 10^{-5} and a total of 400 random walk steps in the annealing loop). The following tasks, with periods of 150 time steps, required computing times between 30 and 50 seconds for one period. If a periodic trajectory is required, 2-3 such periods have to be performed.

At first sight these timings seem poor in comparison with current techniques. However, a fair comparison must include two factors which more than outweigh the longer simulation times: a) traditional methods require a lengthy analysis *preceding* the actual calculation; in contrast, the CKSO algorithm may be cast into a general-purpose computer code applicable to a wide class of robot architectures and problems; b) there are several conceivable ways to parallelize the CKSO calculation; in particular, the individual terms in the cost function require the same input and may be performed in parallel; also, multiple trial steps in the random walk may be introduced to speed up minimisation.

Circular motion; comparison with Moore-Penrose pseudoinversion: Let us return to the 2-dimensional 3-link robot. Its initial configuration is $\mathbf{r}_{1..3} = \{(0/0), (-1/4), (2/2)\}$, $\mathbf{x}(\equiv \mathbf{r}_4) = (4/2)$. As a "warm-up exercise" we will require the end point $\mathbf{x}$ to move along a circle with $\hat{x}(t) = 4 - 2\sin\omega t$ and $\hat{y}(t) = 2\cos\omega t$. Choosing $\omega\,\Delta t = 2\pi/150$ (and $\Delta t = 0.04$), we prepare a table $\hat{\mathbf{x}}(t_n)$ for two full circles (i.e. $n = 1, \ldots, 300$). For each time step, $\mathbf{r}_4$ is changed to the new $\hat{\mathbf{x}}(t_n)$; SHAKE is invoked to move the intermediate links and joints to a suitable position. As our cost function (to be minimized by simulated annealing) we choose the sum of squares of the angular velocities according to equ. 19.

As mentioned above, the use of this particular cost function should yield the same results as the semianalytical Moore-Penrose pseudoinversion; by this we mean the use of equs. 4-6, together with some numerical integrator to treat equ. 4. Indeed, test calculations have shown that the sum squared deviation between the joint positions calculated in both manners never exceeded a value of 0.067, which includes the error due to the use of a straightforward Euler-Cauchy integrator in the pseudoinversion calculation. Thus practically the same sequence of robot configurations is obtained by both methods.

Fetch through a hole: A redundant robot (4 joints) is commanded to move its end effector through a hole in a wall, then to the ground. The cost function consists of terms involving angular velocities (equ. 19) and accelerations (equ. 21), angles (23) and – most important in this application – the distances between the elements of the robot and the contours of the wall cross section depicted in Figure 5.

As an additional exercise, the manipulator is required to return through the hole and move the effector to the ground in front of the wall ("retrieving" task; see Fig. 6.) To facilitate the retrieving task, an additional term was included favouring positive base joint angles (meaning that the first limb leans towards the left of the vertical).

We also require the manipulator to perform a *periodic* motion by incorporating the term 29 in the cost function. The optimal trajectory as sketched in Figure 6 refers to such a periodic motion.

Processing a circular object: Manipulators with one and with two redundant degrees of freedom were ordered to move their end effector with constant velocity along a circular contour, keeping the last limb aligned perpendicular to the circle. In Figure 7 the resulting trajectory is depicted for the 5-joint (2-redundant) manipulator. Results for the 4-joint robot are similar, though less smooth with respect to angular and translational velocities of the joints.

Processing an object with corners: Moving the ultimate limb of a manipulator around a sharp corner may give rise to undesirably high angular accelerations and velocities. To avoid such sudden "jerks", we need some specific alignment of the end effector when being close to any one of the four front corners (see Fig. 8; manipulator approaching the bottom segment of the object). This is achieved by including four cost terms of the form 28. Figure 9 shows the optimized trajectory of a 3-joint (i.e. 1-redundant) manipulator operating on an object with internal and external corners. The same problem was investigated previously by Allgeuer using a variant of the

"Resolved Motion Rate Control" (RMRC) method [2].

Vanishing self-motion: It is desirable that the self-motion of the manipulator stops (and stops gradually) when the motion of the effector comes to an end (i.e. $\dot{\mathbf{x}} = 0$). For redundant manipulators, such a vanishing motion of the joints is not necessarily implied by a vanishing effector motion. In the CKSO and RCSO methods, however, this desired property is automatically achieved if the cost function contains the terms referring to minimal angular velocities (f_{av}) and accelerations (f_{aa}). This is demonstrated by Fig. 10 which shows the right-left motion and stopping of an effector under the influence of f_{av}, f_{aa} and f_{pa} only.

Acknowledgements

Entering robotics from another field (molecular simulation), we needed some friendly advice to get started. Such help was graciously provided by K. Desoyer, P. Kopacek and I. Troch, as well as by S. Stifter.

References

[1] Desoyer, K., Kopacek, P., Troch, I.: Industrieroboter und Handhabungssysteme. R. Oldenburg Verlag, München-Wien, 1985

[2] Allgeuer, H.: Kinematische Steuerung von Robotern mit redundanten Freiheitsgraden. Dissertation, Technische Universität Wien, 1992

[3] Pfeiffer, F., and Reithmeier, E.: Roboterdynamik. Teubner, Stuttgart 1987

[4] Abdel-Rahman, T.M., The Intern. J. of Rob. Res. 10/4, 382-395 (1991)

[5] Kneller, G.R. and Hinsen, K., Phys. Rev. E 50/2, 1559-1564 (1994)

[6] Barojas, J., and Levesque, D., Phys. Rev. A 7/3, 1092-1105 (1973)

[7] Ryckaert, J.-P., Ciccotti, G., and Berendsen, H.J.C., J. Computat. Phys. 23, 327-341 (1977)

[8] Kastenmeier, Th.: "Numerische Roboterkinematik mit stochastischen und molekulardynamischen Simulationsmethoden, insbesondere für redundante Roboter", diploma thesis, University of Vienna, 1995

[9] Kirkpatrick, S., Gelatt, C.D.,Jr., Vecchi, M.P., Science 220, Nr. 4598, 671-680 (1983)

[10] Metropolis, N., and Ulam, S., J. Amer. Statist. Assoc. 44, 335-341 (1949)

[11] Yoshikawa, T., in: Brady, M., and Paul, R., Eds., The First International Symposium of Robotics Research, New Hampshire, USA, Aug. 25-Sept.2, 1983; The MIT Press, Cambridge, Massachusetts, 1984; pp.735-747

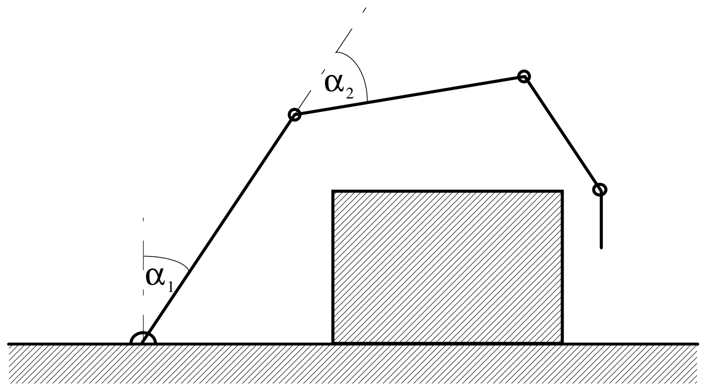

Figure 1: Redundant manipulator circumventing obstacle

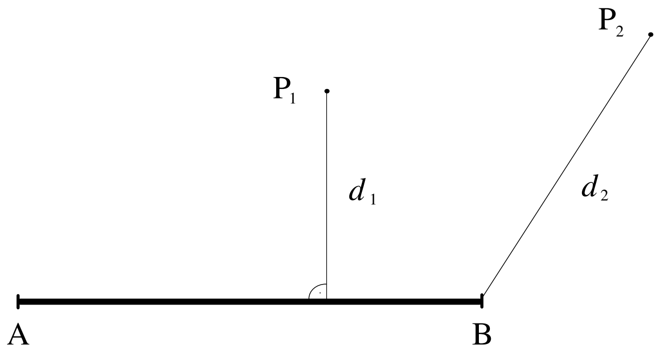

Figure 2: Minimal distance between points and a line segment

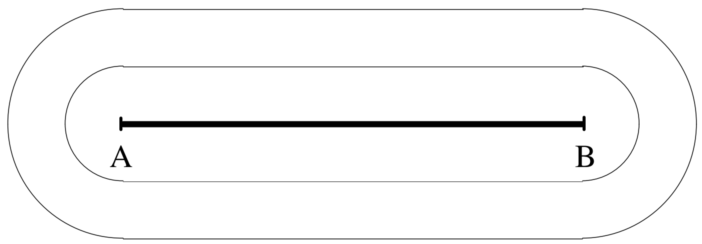

Figure 3: Level lines of equal distance from a line segment

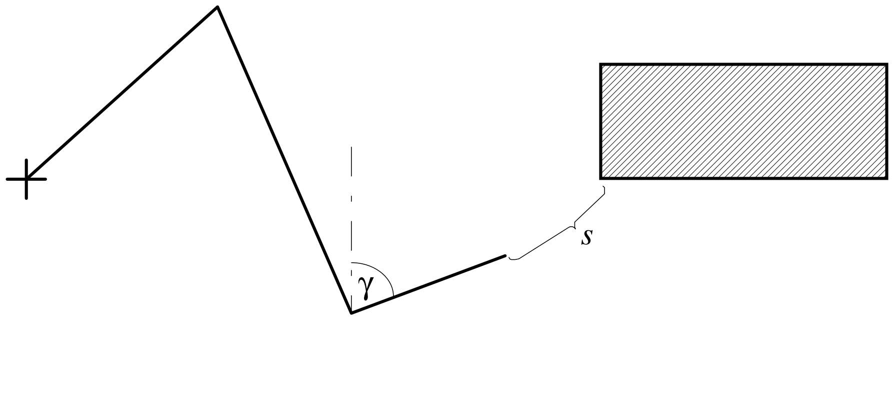

Figure 4: Alignment near an external corner

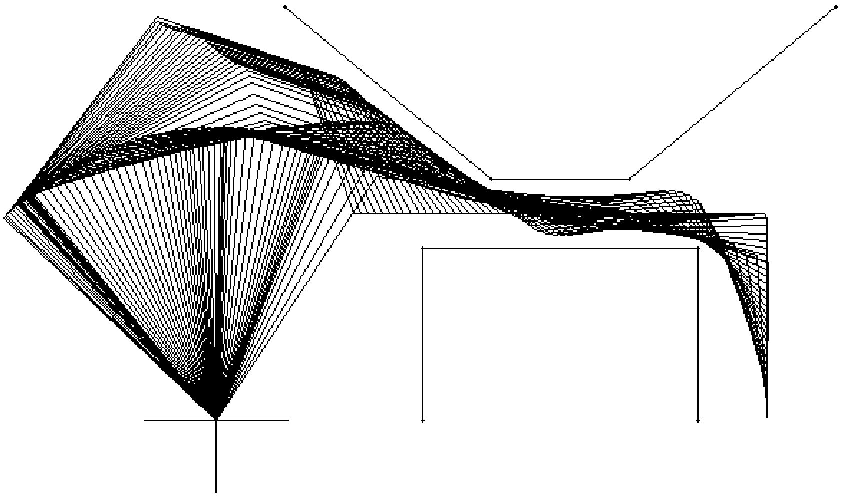

Figure 5: Grasp through a hole

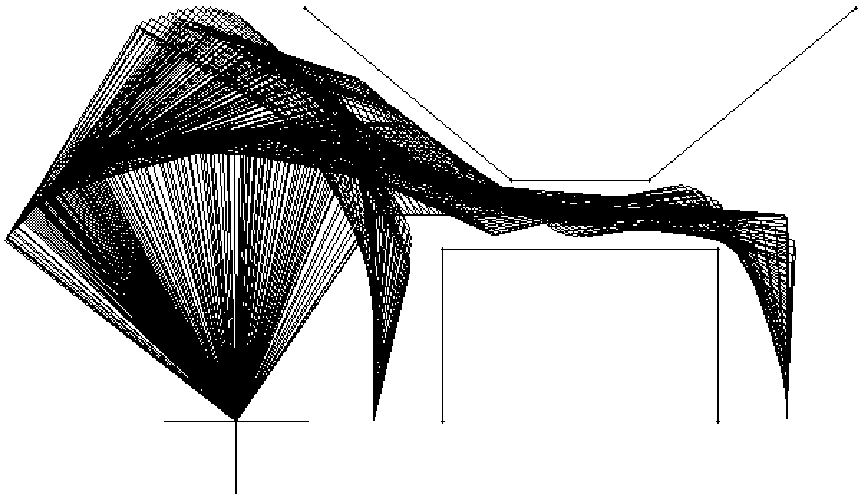

Figure 6: Retrieve through hole

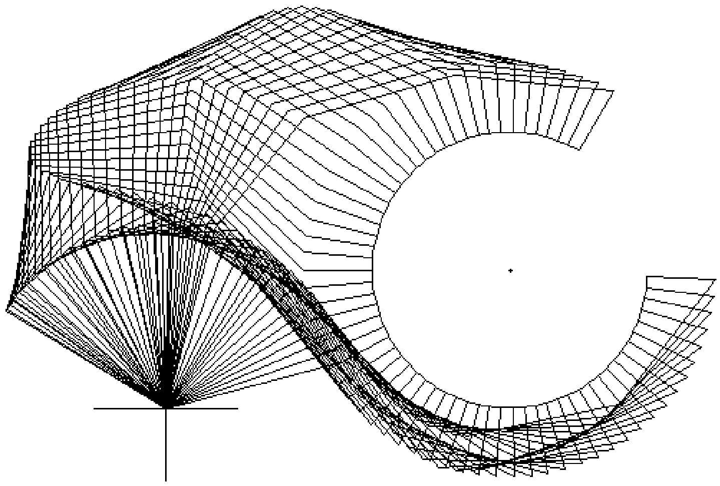

Figure 7: Processing a circular object with effector aligned normal to surface

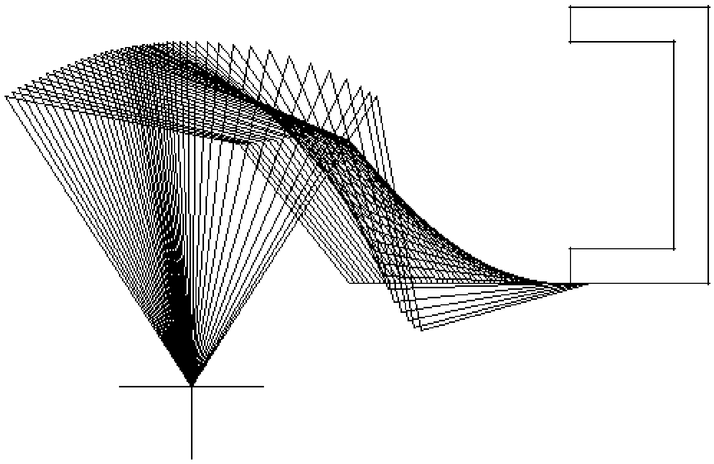

Figure 8: Approaching the object at a corner

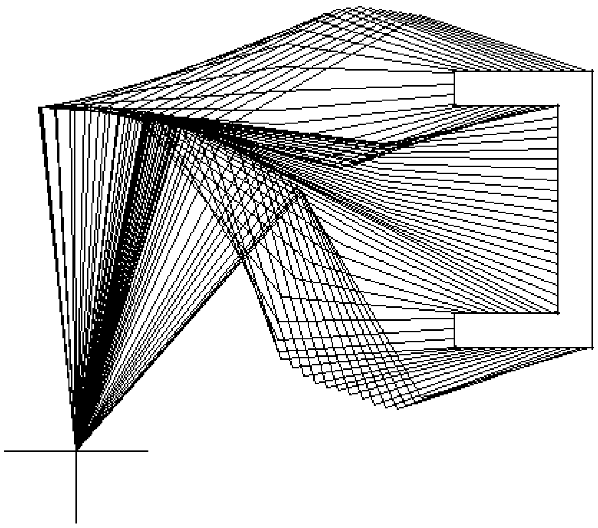

Figure 9: Processing an object with corners; periodic trajectory

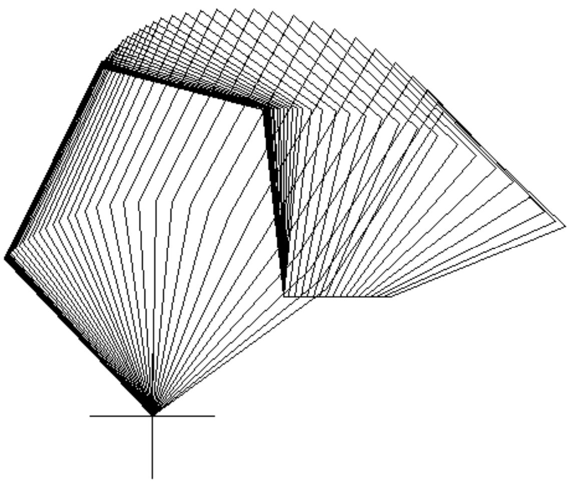

Figure 10: Vanishing self-motion on completion of task