PATH PLANNING FOR REDUNDANT MANIPULATORS USING ITERATIVE OPTIMIZATION TECHNIQUE

D. K. Fenger and W.-S. Lu

Department of Electrical and Computer Engineering
University of Victoria, Victoria, BC, V8W 3P6

ABSTRACT

This paper presents a fairly general solution to the problem of how best to use the extra degree(s) of freedom that redundant manipulators allow. One of the particular strengths of the algorithm presented here is that it can easily use a broad range of possible 'cost functions' in the optimization. The main goal of the algorithm, however, is to make the optimization less expensive, computationally. It is hoped that this approach may lead to practical (suboptimal) solutions that can run in real time or nearly so, on the next generation of computers.

1. INTRODUCTION

A redundant manipulator is one that has more degrees of freedom than the number of constraints imposed by specifying the location and orientation of the manipulator's end effector. (e.g. a 4 DOF planar manipulator would be considered redundant, as only 3 constraints are imposed by position and orientation. Likewise, a 7 DOF 3-dimensional manipulator would be redundant.) The trajectory planning problem for redundant manipulators involves taking an end effector trajectory (specifying position and orientation over time) and converting it into the corresponding joint space trajectory. The difficulty that arises is that for every degree of redundancy, there is a corresponding degree of freedom in the trajectory at each point in time.

1.1 The Inverse Kinematics Problem

While the greatest strength of redundant robots is their improved flexibility, it is also the biggest problem. With that improved flexibility, they can satisfy position and orientation constraints in an infinite number of ways, making the inverse kinematics problem significantly more difficult. Many methods have been proposed to solve this problem, ranging from the simple - such as adding constraints to the problem until the number of constraints m equals the number of degrees of freedom, n[4] - to the complex, such as techniques that use Lagrangian methods to analytically optimize some function of the robot configuration over the trajectory. [2]

Here, we shall consider an iterative optimization technique to minimize a cost function of the robot trajectory over the space of all possible trajectories. This requires parameterizing the set of trajectories in some reasonable fashion.

In general, the relation between the task space variables of the robot, $x(t)$, and the joint configuration of the robot, $\theta(t)$, is represented by:

$$x = F(\theta). \tag{1}$$

One method for solving the inverse of this relation works with the first derivative of (1),

$$\dot{x} = J\dot{\theta}, \tag{2}$$

where J is the Jacobian of F.

The simplest solution to this in the redundant case, where J is a m-by-n (non-square) matrix, is to use the Moore-Penrose pseudo-inverse of J:

$$J^{\#} = J^T\left(JJ^T\right)^{-1}. \tag{3}$$

The general solution to (2) becomes:

$$\dot{\theta} = J^{\#}\dot{x} + \left(I - J^{\#}J\right)\Phi, \tag{4}$$

where $J^{\#}\dot{x}$ is the particular solution, and $(I - J^{\#}J)\Phi$ is the homogeneous solution. The operator $(I - J^{\#}J)$ represents the null space of $J^{\#}$, and Φ is an arbitrary vector.

The problem then becomes one of choosing Φ in some optimal sense. The approach taken here, is to define

$$\Phi = \varepsilon(t), \tag{5}$$

and parameterize $\varepsilon(t)$ as noted below, then optimize on these parameters.

The simplest definition of $\varepsilon(t)$ is as a set of polynomials in time:

$$\varepsilon(t) = e_0 + e_1 t + e_2 t^2 + \ldots + e_p t^P, \tag{6}$$

where the e_i are vectors of length n (the degree of freedom of the robot). This can be expressed more compactly as:

$$\varepsilon(t) = E_{n\times p}\left[1\ t\ t^2 \ldots t^P\right]^T. \tag{7}$$

The variable of the cost function, x, can now be defined as:

$$x = \left[e_{00}\ e_{01} \ldots e_{0p}\ e_{10} \ldots e_{np}\right]^T. \tag{8}$$

In this case, p was taken to be 3. Thus $\varepsilon(t)$ becomes a cubic polynomial, about the simplest that is likely to be useful in modeling.

2. IMPLEMENTATION

It is necessary to define a cost function for the optimization to be performed. In the work done to date, the cost function has been based on keeping the robot as far from singular configurations as is possible.

One of the better known measures used to determine the distance of the robot from a 'singular configuration' is Yoshikawa's *measure of manipulability*: [3,4]

$$\sqrt{\det\left(J(\theta)J^T(\theta)\right)} \quad (9)$$

The objective is to maximize this value, or minimize its negative. Thus, the most obvious cost function, $F(x)$, associated with this measure would be:

$$F(x)=\int_0^{t_f}\left[-\sqrt{\det\left(J(\theta)J^T(\theta)\right)}\right]dt \quad (10)$$

However, the addition of a joint velocity term to the minimization is desirable, as it prevents the singularity avoidance term's tendency to increase the joint velocity to impractical levels. The addition of a weighted $\dot{\theta}^T\dot{\theta}$ term is inexpensive, as $\dot{\theta}$ over time has already been calculated. With this new weighting factor, the cost function becomes:

$$F(x)=\int_0^{t_f}\left[\frac{1}{2}\dot{\theta}^T W\dot{\theta}-\mu\sqrt{\det\left(J(\theta)J^T(\theta)\right)}\right]dt \quad (11)$$

It turns out that for the optimization in question, it is desirable to have a fixed lower bound to the cost function. Since the upper bound to equation (9) is difficult to determine precisely, and tends to be quite large (for the limited test case shown below, one estimate of the upper bound is 2500), the inverse was used, rather than the negative. The resulting cost function was:

$$F(x)=\int_0^{t_f}\left[\frac{1}{2}\dot{\theta}^T W\dot{\theta}+\mu\frac{1}{\det\left(J(\theta)J^T(\theta)\right)}\right]dt \quad (12)$$

The square root was removed inadvertently in the algorithm, and left as coded as the results were good.

2.1 Basic Algorithm

To evaluate $F(x)$ for a given x:

1) Solve the ODE in (4) numerically, using 'ode23' in Matlab.
2) Use the resulting (θ, t) pairs to find the corresponding $\dot{\theta}$, again using (4).
3) Evaluate the integrand of $F(x)$ (11) at the known $\dot{\theta}$, θ, t) points.
4) Do an approximate numerical integration, interpolating the integrand between the known points, to generate $F(x)$.

The integration in step 4 was done fairly crudely, to improve computational speed. A quick interpolation through the points generated by a numeric solution of the ODE was used. This introduces some significant possibility of error, particularly if it leads the algorithm to select a trajectory that makes the ODE algorithm generate 'better' points, rather than selecting on reducing $F(x)$. As long as the errors are generally consistent, however, the result of the optimization will not be greatly affected. This issue needs further examination.

To evaluate $G(x)$, the gradient of $F(x)$, for a given x:

1) Evaluate $F = F(x)$
2) For each component of $G(x)$:

$$G_i(x)\cong\frac{F\left(x+e_i\delta_G\right)-F}{\delta_G}, \quad (13)$$

where e_i is the ith unit vector of length n.

Admittedly, this is a rather crude approach, and expensive, requiring $n(p+1)+1$ evaluations of $F(x)$. However, with the various values of x acting through the ODE in (4), then the resulting θ being used in the calculation of the Jacobian, calculating the effects of infinitesimal changes in x is expected be quite difficult, if tractable at all, and higly variant on the selection of cost functions. It has not yet been attempted by this researcher.

In cases where the gradient in a given direction was needed, the following calculation for Gs^T was used:

$$G(x,s)\cong\frac{F\left(x+\frac{s\delta_G}{\|s\|}\right)-F(x)}{\delta_G}\|s\| \quad (14)$$

The savings per evaluation of Gs^T is obvious - only two calculations of $F(x)$ as opposed to $n(p+1)+1$.

The value for δ_G used above was determined more or less empirically, balancing the errors created by taking too large a distance between the evaluated points against those created by rounding when subtracting two large, similar numbers.

2.2 Optimization Algorithm

The algorithm used to minimize $F(x)$ was a fairly generic Broyden family Quasi-Newton algorithm, using the Fletcher switch to determine whether the BFGS or DFP algorithm was used on any given iteration. This is the most effective algorithm presented in Elec 503, that does not require the calculation of the Hessian of $F(x)$. Seeing how expensive the gradient is, and that the Hessian would require about $(np)^2$ evaluations of $F(x)$, it is obvious that avoiding such a calculation is good.

3. OBSERVATIONS

A number of test runs have been done with the above algorithm, mostly with a simulated 3 DOF 2-dimensional robot, for simplicity.

The algorithm performed quite consistently in terms of the final joint-space results, although it did tend to converge to different values of x if given slightly different starting points. A large part of this is due to the overspecification in $\varepsilon(t)$. Since the null-space operator in (4), $(I - J^{\#}J)$, has a rank equal to the redundant DOF of the robot, it is obvious that number of independent entries in $\varepsilon(t)$ only needs to be the same for complete parameterization. The problem of exploiting this feature efficiently without throwing away information from the null-space operator has not yet been approached.

3.1 Sample Trajectory

The simulated 2-dimensional robot used had 3 links of length [2 2 1], and a starting configuration of:

$$\theta_0 = \begin{bmatrix} 0 & \frac{\pi}{2} & -\frac{\pi}{2} \end{bmatrix}. \tag{15}$$

The trajectory used was:

$$\dot{x} = \begin{bmatrix} -3\pi\sin(t\pi) \\ -2\pi\sin(t\pi) \end{bmatrix}. \tag{16}$$

This trajectory was chosen in hopes of forcing the manipulator near a singular configuration. From the standard starting configuration, this specifies a trajectory that passes straight through the origin, fairly quickly. While this configuration is not singular (only the edges of the workspace are for this robot), it is closer to singularity than much of the workspace.

Typical values for W and μ were I_3 and 1, respectively.

Despite this attempt to force singularity, even the particular solution (achieved when $\varepsilon(t)=0$) was well away from singular configurations. In fact, though the algorithm did smooth out the joint velocities some, it had very little effect on the measure of manipulability component, even when μ was increased to 50. Increasing μ to 1000 only succeeded in increasing the joint velocities, and had little more effect than the μ=50 case.

The improvement in joint velocities can be seen by comparing Figures 1 and 2, the graphs of $\dot{\theta}$ for the x_0=0 and x^* (optimal) cases, respectively.

In all the runs tried, examining the plots of x and $F(x)$ as a function of the number of iterations, showed a 'surging' effect where x would nearly converge, then suddenly jump and start converging to a new value, with a corresponding drop in $F(x)$. (Figure 3 shows the surging quite clearly, at 8, 20 and 30 iterations, with a large surge on iteration 31.) While the cause of these cycles is unknown, it seems suggestive of the optimization algorithm getting caught in a succession of local minima that it subsequently escapes, due to the iterative estimation of the inverse of the Hessian in the Quasi-Newton optimization algorithm. As the state converges towards the local minima, the estimate of the local value of the inverse Hessian improves, allowing the algorithm to find its way out of the trap.

This suggests that the algorithm is not actually finding a global minima, just another local one, and tightening the termination conditions would cause the algorithm to progress further, finding yet another, lower local minima. It is hoped that a good local minima arrived at quickly will be sufficient for purposes of robot control.

The value of the joint-velocity term was evaluated directly, by eliminating the term and attempting the optimization. The system failed to converge, with the values of x increasing without bound. Since the x=0 case minimizes the integral of $\dot{\theta}^T\dot{\theta}$, the reason for the need for the term becomes obvious.

Computational Cost

In general, this algorithm is very expensive to run. A short run that converges in 8 iterations can easily use 4 million flops or more. A longer run, such as the 31 iteration one illustrated in Figure 3, can take over 23 million flops (Matlab flops). (Note that the cost of the last few iterations tends to increase as the Fletcher's line search has more difficulty finding a result.) Since the greatest improvements in $F(x)$ show up in the first 2 to 4 iterations, it is possible that one might, in time-sensitive cases, terminate the optimization at whatever number of iterations is convenient, based on available computing resources.

It is expected that the algorithm will scale as at least n^2, more likely n^3. This will be offset, however, by the fact that the number of independent elements in $\varepsilon(t)$ needs only be the number of redundant DOF of the manipulator, as noted above.

Comparison with Other Works

Direct comparison of results from other papers has not been done, as most global optimization algorithms in this class constrain start and finish joint positions to be the same for cyclic paths, which has not yet been done here. The computational costs could not be compared as none of the relevant papers listed them.

In general, this approach seems to have not been tried before. The only use of the homogeneous solution to (4) noted in Nenchev's review paper[4], is the Gradient Projection method. This is a local optimization technique that uses the gradient of a function of robot configuration for Φ in (4). Papers such as Dubey's[1] or Kang and Freeman's[6] that look at this are concerned with finding useful functions of configuration to use, and not considering any form of iterative optimization. (Which is impractical in a real-time situation.) This technique is really not comparable, in that the Gradient Projection method is a local optimization for real-time use, while the algorithm at hand attempts to find a global optimum and is an off-line technique in its current form.

The global approaches to optimization of the trajectory tend to be quite a bit more analytical, using Lagrangian techniques to optimize the integral of $G(\dot{\theta}, \theta, t)$, some function of robot configuration. In the paper by Martin, et al.[2], a globally optimal solution for the problem is given, based on these

techniques. Solving it involves determination of sufficient boundary conditions, and then solution of a rather complex second order differential equation. Numerically, one would expect this to be cheaper than the iterative optimization technique outlined in this report.

The paper by Martin, et al. is also interesting in that it does prove the existence of classes of local minima solutions to periodic trajectories that are not global minima. Thus, the presence of local minima to catch the iterative optimization is assured.

The global approaches are generally much more concerned with boundary conditions, as the second order (or higher) differential equations they propose require two or more boundary conditions. It is perhaps notable that the algorithm in this report worked quite satisfactorily with only one boundary condition. However, more attention does need to be paid to the use of alternative boundary conditions with this iterative algorithm. In general, one would like to convert boundary conditions to constraints on the variable, x, of the optimization, preferably removing elements of it outright so that the optimization remains an unconstrained one.

4. CONCLUSIONS

The algorithm introduced here produces useful results, although they are not likely to be globally optimal. Notably, it generalizes easily to any integral criteria of the form $G(\dot{\theta}, \theta, t)$. It can improve on the basic inverse Jacobian methods, and hopefully be eventually shown as an improvement over locally optimal methods. Further work is needed to determine the quality of the local optima that this algorithm finds.

Some potential exists for a real-time system exists, although it would require knowledge of the manipulator's trajectory some fraction of a second in advance. This would be a system that takes the first local minima found, and passes the associated $\varepsilon(t)$ parameters to the next layer of the control scheme. While the computational cost of this algorithm is fairly high, on modern or upcoming processors it should be workable.

4.1 Future Directions

The accuracy of the algorithm is being examined in more detail. In particular, the degree of sensitivity of the optimization to the approximations used in its calculation needs to be determined. Work to include a variation on Deo and Walker's[5] Singularity Robust Inverse to improve the handling of near-singular regions is also ongoing.

References

[1] R. Dubey and J. Y. S. Luh, "Redundant robot control using task based performance measures," *Journal of Robotic Systems*, 5, 409-432 (1988).

[2] D. P. Martin, J. Baillieul, J. M. Hollerbach, "Resolution of Kinematic Redundancy Using Optimization Techniques," *IEEE Transactions on Robotics and Automation*, 5, 529-533, (1989)

[3] Y. Nakamura, Advanced Robotics - Redundancy and Optimization, Addison-Wesley, New York, 1991

[4] D. N. Nenchev, "Redundancy Resolution - Global vs. Local Optimization: A Review," *Journal of Robotic Systems*, 6, 769-798 (1989)

[5] A. S. Deo, I. D. Walker, "Robot Subtask Performance with Singularity Robustness using Optimal Damped Least-Squares", *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, 434-441 (1992)

[6] H.-J. Kang, R. A. Freeman, "Null Space Damping Method for Local Joint Torque Optimization of Redundant Manipulators", *Journal of Robotic Systems*, 10, 249-270 (1993)

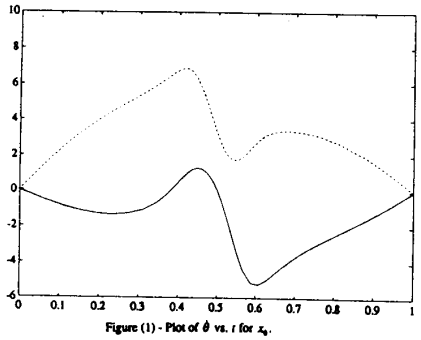

Figure (1) - Plot of $\dot{\theta}$ vs. t for x_0.

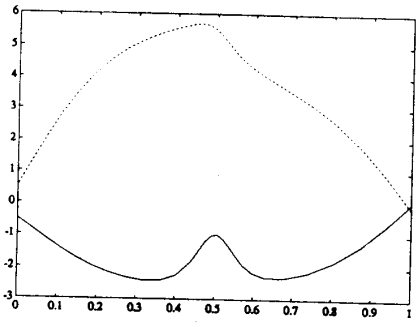

Figure (2) - Plot of $\dot{\theta}$ vs. t for x^*.

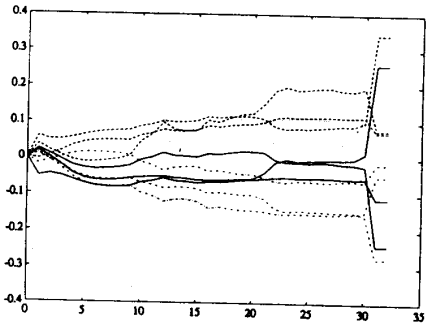

Figure (3) - Plot of x vs. number of iterations.