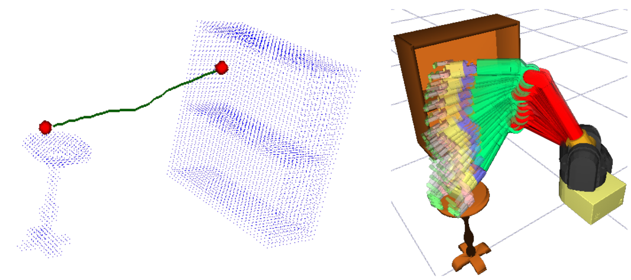
Deterministic Motion Planning for Redundant Robots along End-Effector Paths

Ana Huamán Quispe and Mike Stilman

Abstract—**In this paper we propose a deterministic approach to solve the *Motion Planning along End-Effector Paths problem* (MPEP) for redundant manipulators. Most of the existing approaches are based on local optimization, hence they do not offer global guarantees of finding a path if it exists. Our proposed method is resolution complete. This feature is achieved by discretizing the Jacobian nullspace at each waypoint and selecting the next configuration according to a given heuristic function. To escape from possible local minima, our algorithm implements a backtracking strategy that allows our planner to recover from erroneous previous configuration choices by performing a breadth-first backwards search procedure. We present the results of simulated experiments performed with diverse manipulators and a humanoid robot.**



(a) Workspace Path (b) Execution

Fig. 1. Experiment with a 7-DOF BarretWMA Arm

I. Introduction

[**?**] Task redundancy is a desirable feature in robotic manipulators. The additional degrees of freedom allow the robot to not only achieve a primary goal, such as following an end-effector trajectory, but it also endows the system with multiple ways to execute the same task successfully. Diverse uses of redundancy include obstacle avoidance in cluttered environments, joint velocity control, singularity avoidance, torque minimization among others.

Although redundancy is advantageous, it comes at a cost. The complexity of the manipulator increases, which means that the solution of the *inverse kinematics problem* does not have a closed-form solution except for particular configurations. Multiple approaches have been proposed, most of them tailored to solve the problem under specific assumptions and hence assessing different success criteria depending on the particular task [**?**].

The inverse kinematics problem was initially analyzed under an industrial perspective, where criteria such as minimization of kinetic energy and manipulatability were of vital interest. Since robots are slowly making the transition of factories to more challenging environment such as homes, the manipulator requirements are more complex, hence redundancy is a useful and needed feature.

The present paper focuses on solving a task typically encountered by a manipulator: Tracking an end-effector trajectory such that the manipulator avoids collisions in a static environment. This kind of task is important not only in the context of the robot accomplishing a task but it is also a helpful tool for robot learning, facilitating the demonstration of a task executed by a human (in which only the end-effector movement becomes the key feature, leaving to the robot the task of figuring out how to replicate the movement with its own particular kinematic configuration that may or may not be anthropomorphic).

We present background inverse kinematic theory and some of the existing approaches in Section II. On Section III we describe our proposed algorithm. Results of series of simulated experiments are presented in section IV. We end this paper with conclusions and future work in Section V.

The authors are with the Center for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA. ahuaman3@gatech.edu, mstilman@cc.gatech.edu

II. Background

The forward kinematics of a manipulator is given as follows:

$$x = f(q) \tag{1}$$

and its differential version is expressed as:

$$\dot{x} = J\dot{q} \tag{2}$$

where $x \in R^m$, $q \in R^n$, and m and n are the dimensions of the task and joint space respectively. If $m < n$, the manipulator is *task redundant*.

Manipulator tasks are varied; however, here we briefly describe two of the most common (as it is similary explained in [**?**]):

- *End-Point Goal Task:* The manipulator is given start and end task space points. The task is to find a sequence of joint space configurations that connect them. This is most known as a *path planning problem*
- *Point-to-Point Task:* The manipulator is given a task space trajectory to be mapped to a joint space trajectory. This is most known as a *tracking problem*, which will be the focus of this paper.

The inverse kinematics problem consists on finding a mapping such that $q = f^{-1}(x)$. For redundant manipulators there are potentially infinite values q for a given x, which can be obtained by solving (2). A particular solution is given by:

$$\dot{q} = J^{\dagger}\dot{x} \tag{3}$$

where $J^{\dagger}$ is a generalized inverse that can be chosen to minimize a specific criterion. In [**?**], Whitney used the Moore-Penrose pseudo-inverse to minimize the norm of $\dot{q}$. Whitney further proposed to use a pseudo-inverse Jacobian weighted with the inertia matrix in order to minimize the kinematic energy of the system. A generalization of this is the weighted least-norm method (WLM) presented in [**?**], which proved to be particularly effective to avoid joint limits. Recent extensions to this approach include the work of Xiang with GWLM [**?**].

(3) is a particular solution of 2. In general, the set of possible solutions can be expressed as:

$$\dot{q} = J^{\dagger}\dot{x} + (I - J^{\dagger}J)\dot{q}_0 \tag{4}$$

The second right-hand term represents the homogeneous solutions or *self-motions*, that is, motions in the configuration space that do not produce motions in the task space.

Homogeneous solutions are configurations in the Jacobian nullspace. These can be found by projecting an arbitrary vector $\dot{q}_0$ on the nullspace, such as in (4) where the columns of $(I - J^{\dagger}J)$ are the basis of the nullspace of J. Most of the proposed methods differ in their choosing of $\dot{q}_0$. The most widely used method is the *Gradient projection method* (GPM) proposed by Liegeois in [**?**]. The GPM method consists on defining an optimization function $H(q)$ to be minimized. By defining:

$$\dot{q}_0 = -\alpha\frac{\partial H}{\partial q} \tag{5}$$

it produces solutions that move the manipulator away from undesirable configurations. Diverse H functions have been used in the literature, such as the measure manipulatability in [**?**], used to avoid singularities. Other applications include obstacle avoidance [**?**], minimization of torques [**?**], avoidance of joint limits [**?**], among others. An excellent overview of these early methods can be found in [**?**]. For problems with more than one subtask, nested approaches have been proposed [**?**] [**?**], based on the relative priority between each subtask. Other methods instead define a weigthed sum of optimization functions, such as in [**?**].

There are other approaches that tackle the redundancy problem by defining additional constraints such that the problem is no longer redundant. Examples of these are the Extended Jacobian [**?**] and the Augmented Jacobian [**?**] [**?**].

The methods mentioned above solve different kinds of problems; however all of them share the same weakness of being local, which precludes them to fall in local minima. There are a few global approaches in the literature, but these are rarely used in practice due to the intensive computation required (Add citations). In the next section we propose a method that attempts to escape the local minima curse by using backtracking in the discretized nullspace of the manipulator.

III. Proposed Algorithm

A. Basics

We saw in section II that the solution to (2) has the general form:

$$\dot{q} = J^{\dagger}\dot{x} + (I - J^{\dagger}J)\dot{q}_0 \tag{6}$$

where the second right-hand term represents the self-motions in the Jacobian nullspace, which is a subspace of dimensions $(n - m)$. Hence, we can write (6) equivalently as:

$$\dot{q} = J^{\dagger}\dot{x} + \sum_{i=1}^{n-m} w_i\hat{e}_i \tag{7}$$

where:

- $\hat{e}_i$: Normalized basis of the Jacobian nullspace
- w_i: Coefficients of each $\hat{e}_i$

Both representations are equivalent, however (7) is of special interest to us since instead of having to define a vector $q_0 \in R^n$, we define the homogeneous solution in terms of the $(n - m)$ coefficient w_i. The problem, however still has potentially infinite solutions.

Our approach proposes, instead of finding a q_0 vector (as GPM does), to effectuate a search in the discretized Jacobian nullspace. The discretization is carried out by selecting a range of values for each w_i. Notice that, since we are considering the tracking of a task space *trajectory*, we do only consider homogeneous solutions that are realizable in one time step. To make this clear, please refer to Fig.2. Figures 2.a and 2.c show the one-step self-motions that can be achieved in one time step. Figures 2.b and 2.d show the subset of additional self-motions corresponding to these that would require more than one time-step to be executed, hence they are not considered in the nullspace search explained in this paper[1].

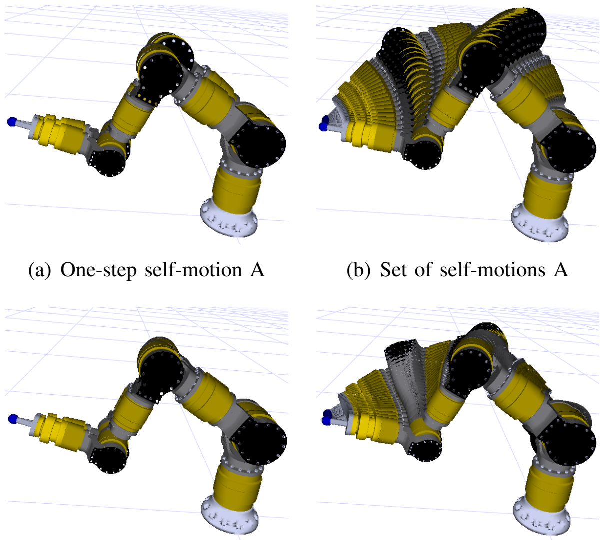
Once the set of self-motions is generated we proceed to eliminate the configurations that are in collision, keeping only the set of *valid* configurations, from which any can be a possible solution for the task space point evaluated. Rather than picking a random configuration from the nullspace set, we choose a configuration such that it maximizes a performance optimization function. Previous work such as [**?**] suggests a serie of diverse functions to assess the desirability of each configuration. In particular, we have evaluated the so-called Joint Range Availability measurement (JRA):

$$JRA(q) = \sum_{i=1}^{n} \frac{q_i - \bar{q}_i}{q_{i_{Max}} - q_{i_{Min}}} \tag{8}$$

in our experiments. We save the nullspace sets as priority queues ordered w.r.t. the JRA measure of each configuration. The approach explained so far is shown in Algorithm 1

Algorithm 1 is prone to fall in local minima. This can be easily seen in Line 3 in Algorithm 1, which reports failure if

[1]Notice that in the general case of tracking a *path*, *all* the self-motion configurations should also be considered. We are currently working on this topic



(a) One-step self-motion A (b) Set of self-motions A

(c) One-step self-motion B (d) Set of self-motions B

Fig. 2. Illustration of self-motion considered

Algorithm 1: Simple_Track($\mathcal{P}$, q_0, $\mathcal{Q}$)

Input: Workspace trajectory $\mathcal{P}$ and initial joint configuration q_0

Output: Jointspace trajectory $\mathcal{Q}$

```
q ← q0
for i ← 1 to P.size() do
    if Generate_Sol(q, P[i], NS[i]) is false then
        return false
    else
        q ← NS[i][0] ; // Top priority element
        Q.push_back(q)
return true
```

Algorithm 2: Generate_Sol(q, p, ns)

Input: Current joint configuration q, next workspace point p

Output: Nullspace queue ns corresponding to q

```
ds ← (p − ToTaskSpace(q))
q_p ← J†(q) · ds ;                // Least-norm sol
ê = J(q).kernel() ;               // J nullspace basis
forall combinations of w_j do
    q_ns ← q + q_p + Σ_{j=1}^{m−n} w_j ê_j
    if || p - ToTaskSpace(q_ns) || < thresh then
        if InCollision(q_ns) is false and
        InLimits(q_ns) is true then
            ns.Push_Queue(q_ns, JRA(q_ns))
if ns.size() > 0 then
    return true
else
    return false
```

a solution is not found for a workspace point. We know that a solution might exist if previous configurations were chosen differently. Hence, our method improves this by implementing a *backtracking schema* that operates whenever local solutions fail.

B. Backtracking

Algorithm 4 implements a recursive backtracking procedure. It accepts as input the current configuration, the stored nullspace sets of the trajectory mapped so far and the maximum number of backtracking steps allowed. When a failure is detected at time i, our algorithm goes back one step $(i-1)$, pop up the failing configuration and choose a different one from the $\mathcal{NS}[i-1]$. Notice that since $\mathcal{NS}$ stores the nullspace sets as priority queues, the next configuration to be evaluated is the next best, according to our optimization function. In case the backtrack fails again in producing a valid solution, it backtracks to $i-2$ and it will repeat the procedure until it finds a solution or until it reaches the maximum allowed number of backtracking steps.

Of course, backtracking can be computationally intensive if the depth is too big. In order to avoid backtracking as much as possible, we must use a reasonable heuristic function such that backtrack is only used in extreme cases. In our experiments we only used the JRA function, which seemed to produce adequate results.

Our proposed method is shown in Algorithm 3. The recursive backtracking routine (which actually implements a breadth-first search) is presented in Algorithm 4. Notice that the latter algorithm depends on the `ForwardSearch` routine(Algorithm 5) which searches for a solution from the backtracked configuration $(i-step)$. Also note that each time a backtrack is executed, the nullspace sets generated previously from $(i-step-1)$ to i are cleared and updated according to the new configuration found in $(i-step)$.

IV. Experiments and Results

In this section we present a series of simulated experiments with diverse manipulators. The simulations were made using the integrated GRIP + DART[2] software platform. For collision detection we used the VCOLLIDE package [?]. All the experiments were executed on a Intel 2 Core Duo (2.80GHz).

A. Preliminaries

Before presenting the experiments, some details should be mentioned:

- All the experiments presented are *end-effector position tracking* problems. We chose to focus this paper on position (rather than pose) tracking since the manipulator

[2] GRIP (https://github.com/golems/grip) is a robotics simulator based on the physical engine DART (https://github.com/golems/dart). Both packages are OpenSource projects currently developed at Georgia Tech

Algorithm 3: Complete_Track($\mathcal{P}$, q_0, $\mathcal{Q}$, *maxStep*)

Input: Workspace trajectory $\mathcal{P}$ and initial joint configuration q_0, maximum backtrack allowed *maxStep*
Output: Jointspace trajectory $\mathcal{Q}$

$q \leftarrow q_0$
for $i \leftarrow 1$ **to** $\mathcal{P}$.size() **do**
  if Generate_Sol(q, $\mathcal{P}[i]$, $\mathcal{NS}[i]$) **is** false **then**
    step $\leftarrow 1$
    b = Backtrack(*i*, *step*, *maxStep*, $\mathcal{NS}$, $\mathcal{P}$)
    if *b is* false **then**
      return false
    else
      Update($\mathcal{Q}$, *i*, *step*, $\mathcal{NS}$)
  else
    $q \leftarrow \mathcal{NS}[i][0]$
    $\mathcal{Q}$.push_back(q)
return true

Algorithm 4: Backtrack(*i*, *step*, *maxStep*, $\mathcal{NS}$, $\mathcal{P}$)

Input: current index i, current backtrack *step*, maximum backtrack *maxStep*, nullspace sets $\mathcal{NS}$, workspace trajectory $\mathcal{P}$
Output: Updated $\mathcal{NS}$

if *step* $<$ *maxStep* **then**
  if ForwardSearch(*i*, *step*, *maxStep*, $\mathcal{NS}$, $\mathcal{P}$) **is** false **then**
    return Backtrack(*i*, *step*++, *maxStep*, $\mathcal{NS}$, $\mathcal{P}$)
  else
    return true
else
  return ForwardSearch(*i*, *step*, *maxStep*, $\mathcal{NS}$, $\mathcal{P}$)
return true

has more redundant degrees of freedom with respect to this kind of tasks. Having less constraints poses a more challenging scenario to test our algorithm.

- The workspace trajectories used in the experiments were generated with a low-level workspace planner described in [**?**]
- All the kinematics evaluations were handled by the DART dynamics package. Additional linear algebra calculations (such as the nullspace basis or the Jacobian pseudo-inverse) were performed using the Eigen[3] C++ library.
- All the experiments used a discretization of 10 values per each w_i. The range of each w_i was $[-10, 10]$. The nullspace basis $\hat{e}_i$ were normalized to have the same relative norm w.r.t each other.

[3]http://eigen.tuxfamily.org

Algorithm 5: ForwardSearch(*i*, *step*, *maxStep*, $\mathcal{NS}$, $\mathcal{P}$)

Input: current index i, current backtrack *step*, maximum backtrack *maxStep*, nullspace sets $\mathcal{NS}$
Output: Updated $\mathcal{NS}$

ForwardClear_NS(*i*, *step*, $\mathcal{NS}$)
j $\leftarrow$ *i* - *step*
found $\leftarrow$ false
while $\mathcal{NS}[j]$.size() > 0 **and** *found* **is** false **do**
  b $\leftarrow$ Generate_Sol($\mathcal{NS}[j][0]$, $\mathcal{P}[j+1]$, $\mathcal{NS}[j+1]$)
  if *b is* true **then**
    if *step* > 1 **then**
      found $\leftarrow$ ForwardSearch(*i*, *step-1*, *maxStep*, $\mathcal{NS}$, $\mathcal{P}$)
    else
      found $\leftarrow$ true
  else
    $\mathcal{NS}[j]$.Pop_Queue()
if *found* **is** false **then**
  if *step* **is** *maxStep* **then**
    return false
  else
    $\mathcal{NS}[j+1]$.Pop_Queue()
else
  return true

Procedure Update($\mathcal{Q}$, *i*, *step*, $\mathcal{NS}$)

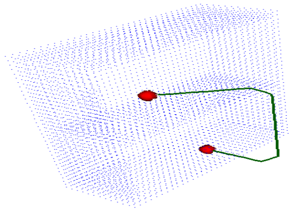
for $j \leftarrow 1$ **to** *step* **do**
  // Update the changed joint trajectory points
  $\mathcal{Q}[i-j] \leftarrow \mathcal{NS}[i-j][0]$
end

B. Experiment 1: LWA3

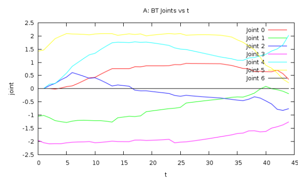
Our first series of experiments were done with a 7 DOF LWA3 arm, as shown in Fig.3. The initial configuration of the robot has its end effector below the cabinet. The workspace trajectory to follow is shown in Fig.3(b). Notice that the end-effector trajectory is very close to the cabinet, hence the tracking is harder since there are more chances of the manipulator links colliding with the object. The resulting execution is shown in Fig.3(c).

Procedure ForwardClearNS(*i*, *step*, $\mathcal{NS}$)

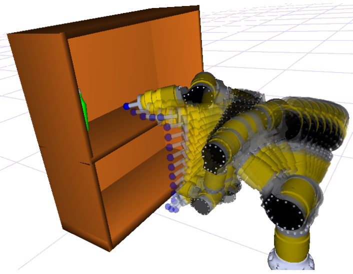
if *step* > 1 **then**
  for $j \leftarrow 1$ **to** *step* **do**
    $\mathcal{NS}[i-j]$.clear()
  end
end



(a) Workspace trajectory

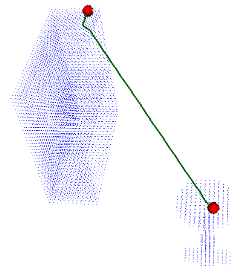


(b) Joint trajectory

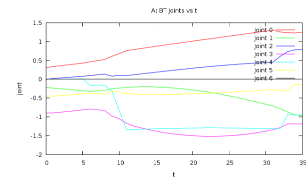


(c) Execution

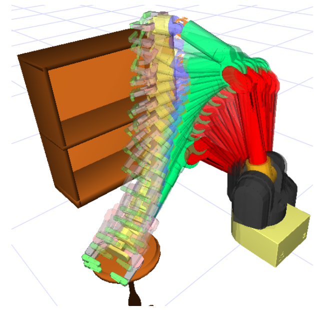
Fig. 3. Experiment with a 7-DOF LWA3 Schunk arm



(a) Workspace path (x, y, z)



(b) Tracking



(c) Execution

Fig. 4. Experiment with a 7-DOF BarretWMA

C. Experiment 2: BarretWMA

Our second experiment was done with a 7 DOF BarretWMA manipulator. The scenary kept the same as in the experiment with the LWA3 but the initial configuration of the arm changed as it is shown in Fig.4. The arm has its end effector located on top of the cabinet and its target location is resting on top of the table. The resulting trajectory is shown in Fig.4.(c). In this execution, two backtrack calls were needed (at time step 3 and 31 out of the total trajectory of 35 points) both of them requiring a depth of 2. Notice that a simple search such as GPM is not able to solve this problem for this particular workspace to follow.

D. Experiment 3: Mitsubishi

1) *Case 1: Straight Line:*

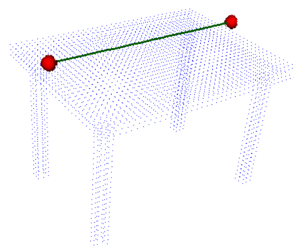
2) *Case 2: Straight Line:*

E. Experiment 4: Dual-Armed robot

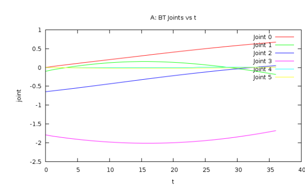
V. Conclusions and Future Work

We have presented an approach to solve the trajectory tracking problem for redundant manipulators based on the discretization of the Jacobian nullspace and a backtracking strategy to prevent local minima traps. In contrast with previous approaches, we exploit the nullspace by considering more than one possible configuration and allow our method to be more flexible. In contrast with previous work in which a set of joints were arbitrarily selected as non-redundant, our approach uses the nullspace basis of the Jacobian to perform the search with the minimum required values.

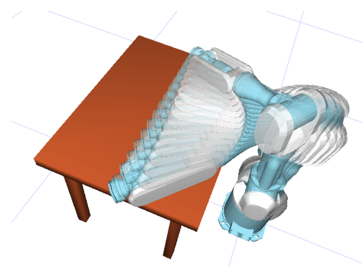
As future work, we intend to devise more simulated experiments with more challenging scenarios and further test the presented method in a physical manipulator. Additionally, we plan to test our approach with *pose tracking* problems to verify its effectiveness.



(a) Workspace trajectory



(b) Tracking



(c) Execution

Fig. 5. Experiment with a 6-DOF manipulator

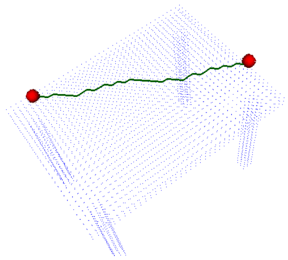
References

[1] G. Oriolo, M. Ottavi, and M. Vendittelli, "Probabilistic Motion Planning for Redundant Robots along Given End-Effector Paths," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

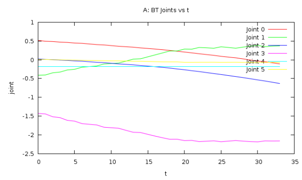
[2] R. Hooper and D. Tesar, "Motion Coordination based on multiple performance criteria with a hyper-redundant serial robot example," in *Proceedings of the IEEE International Conference on Intelligent Control*, 1995, pp. 133–138.

[3] S. Seereeram and J. Wen, "A Global Approach to Path Planning for Redundant Manipulators," in *IEEE Transactions on Robotics and Automation*, 1995, pp. 152–160.

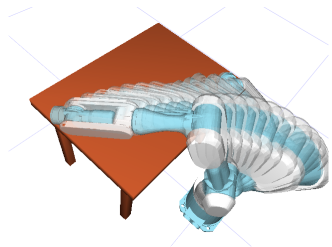
[4] D. E. Whitney, "Resolved Motion Rate Control of Manipulators and



(a) Workspace trajectory

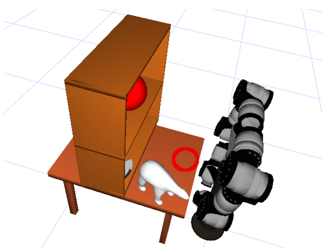


(b) Tracking

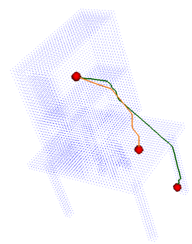


(c) Execution

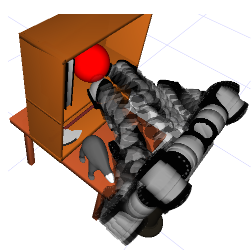
Fig. 6. Experiment with a 6-DOF manipulator



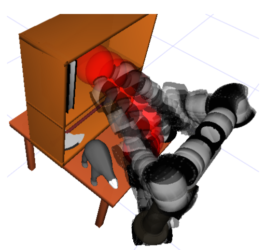
(a) Problem setup



(b) Workspace traj



(c) Reach execution



(d) Move execution

Fig. 7. Experiment with a Dual-Armed robot

Human Prostheses," in *IEEE Transactions on Man-Machine Systems*, 1969, vol. 2, pp. 47–53.

[5] T. F. Chan and R. Dubey, "A Weighted Least-Norm Solution Based Scheme for Avoiding Joint Limits for Redundant Joint Manipulators," in *IEEE Transactions on Robotics and Automation*, vol. 11, no. 2, April 1995, pp. 152–160.

[6] J. Xiang and C. Zhong, "General-Weighted Least-Norm Control for Redundant Manipulators," *IEEE Transactions on Robotics*, pp. 660–669, 2010.

[7] A. Liegeois, "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms," in *IEEE Transactions on Systems, Man and Cybernetics*, 1977, pp. 868–871.

[8] T. Yoshikawa, "Analysis and Control of Robot Manipulators with Redundancy," in *Robotics Research, The First International Symposium*. MIT Press, 1985, pp. 735–747.

[9] A. Maciejewski and C. Klein, "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," in *The International Journal of Robotics Research*, vol. 4, no. 3, 1985, pp. 109–117.

[10] J. Hollerbach and K. Suh, "Redundancy Resolution of Manipulators through Torque Optimization," in *IEEE Journal of Robotics and Automation*, 1985, pp. 308–316.

[11] B. Siciliano, "Kinematic Control of Redundant Robot Manipulators: A Tutorial," *Journal of Intelligent and Robotics Systems*, pp. 201–212, 1990.

[12] S. Chiaverini, "Singularity-Robust Task-Priority Redundancy Resolution for Real-Time Kinematic Control of Robot Manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 398–410, June 1997.

[13] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-Priority Based Redundancy Control of Robot Manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, June 1987.

[14] K.-K. Lee and M. Buss, "Redundandy Resolution with Multiple Criteria," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[15] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in *IEEE International Conference on Robotics and Automation*, 1985, pp. 722–728.

[16] L. Sciavicco and B. Siciliano, "A Solution Algorithm to the Inverse Kinematic Problem for Redundant Manipulators," *IEEE International Journal of Robotics and Automation*, vol. 4, no. 4, pp. 403–410, August 1988.

[17] O. Egeland, "Task-Space Tracking with Redundant Manipulators," *IEEE International Journal of Robotics and Automation*, vol. 3, no. 5, pp. 471–475, October 1987.

[18] C. Kapoor, M. Cetin, and D. Tesar, "Performance based redundancy resolution with multiple criteria," in *Proceedings of the ASME Design Engineering Technical Conference*, September 1998.

[19] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk, and D. Manocha, "V-COLLIDE: Accelerated Collision Detection for VRML," in *In Proceedings of VRML*, 1997, pp. 119–125.

[20] A. Huamán Quispe and M. Stilman, "Diverse Workspace Path Planning for Robot Manipulators," College of Computing, Georgia Institute of Technology, Tech. Rep. GT-GOLEM-2012-003, July 2012.