Research article

Real-time collision avoidance algorithm for robotic manipulators

Paul Bosscher and Daniel Hedman
Harris Corporation, Palm Bay, Florida, USA

Abstract
Purpose – The purpose of this paper is to present an algorithm for performing collision avoidance with robotic manipulators.
Design/methodology/approach – The method does not require any a priori knowledge of the motion of other objects in its environment. Moreover, it is computationally efficient enough to be implemented in real time. This is achieved by constructing limitations on the motion of a manipulator in terms of its allowable instantaneous velocity. Potential collisions and joint limits are formulated as linear inequality constraints. Selection of the optimal velocity is formulated as a convex optimization and is solved using an interior point method.
Findings – Experimental results with two industrial arms verify the effectiveness of the method and illustrate its ability to easily handle many simultaneous potential collisions.
Originality/value – The resulting algorithm allows arbitrary motions commanded to the robot to be modified on-line in order to guarantee optimal real-time collision avoidance behaviors.

Keywords Robots, Programming and algorithm theory, Collisions, Motion

Paper type Research paper

Introduction

Many applications exist where one or more robotic manipulator arms must work in a crowded or cluttered workspace and must avoid colliding with each other or with obstructions within the workspace. These include manufacturing applications, surgical robotics, and remote teleoperation (e.g. NASA's Robonaut (http://robonaut.jsc.nasa.gov/), explosive ordinance disposal), to name a few. Much work has been performed on off-line collision-free path planning (Ali *et al.*, 2002; Hosseini *et al.*, 2004; Yu *et al.*, 2007). However, pre-planning collision-free trajectories is only practical in highly structured environments with unchanging manipulator trajectories. If the environment is dynamically changing or if nearby manipulators are performing unplanned motions, an on-the-fly adjustment to the manipulator motion is necessary. This paper presents a method for calculating collision-free motion of manipulator arms in real time. This method is based on a reactive paradigm, where no a priori knowledge is assumed to be known of the motion of the objects to be avoided.

Motion planning for robotic manipulators near obstacles is a significant challenge. Over the last decade, successful solutions proposed for this problem utilize sampling-based algorithms. There are two primary types of sampling-based algorithms – multiple- and single-query methods. In multiple-query methods, significant pre-computation is performed so that multiple queries (paths planned) can be handled quickly. An example of this type of approach is probabilistic roadmaps (PRMs) (Kavraki *et al.*, 1996). Building on this approach, other roadmap methods based on importance sampling have been developed which concentrate samples in a non-uniform way, such as along the boundaries of the C-space (Amato *et al.*, 1998; Boor *et al.*, 1999) or medial axis (Holleman and Kavraki, 2000; Pisula *et al.*, 2000; Wilmarth *et al.*, 1999). PRMs have also been extended to solve motion planning problems for closed kinematic chains (Cortés *et al.*, 2002; Han and Amato, 2000; Yakey *et al.*, 2001), multiple robots (Svestka and Overmars, 1995), and non-holonomic robots (Sekhavat *et al.*, 1998).

Owing to their considerable pre-computation time, multiple-query methods are often not practical for dynamically changing environments, thus alternate methods were created for solving single-query problems (Bohlin and Kavraki, 2000; Hsu *et al.*, 1999; LaValle, 1998; Mazer *et al.*, 1998; Sánchez and Latombe, 2001). Rapidly exploring random trees (RRTs) were created primarily for solving single-query holonomic problems and problems with differential constraints (LaValle, 1998; Kuffner and LaValle, 2000; LaValle and Kuffner, 2001). The successful application of RRTs to many motion planning applications has prompted development of many extensions of this method. RRTs have been extended to manipulation problems and motion planning for closed articulated chains (Cortés and Siméon, 2004; Kagami *et al.*, 2003; Kallmann *et al.*, 2003; Siméon *et al.*, 2004).

Despite the many recent advances in multiple- and single-query motion planning for manipulation, these methods all plan in the robot's configuration space (C-space) (Brock *et al.*, 2008). Obstacles must be mapped to the C-space, which is

Industrial Robot: An International Journal
38/2 (2011) 186–197
© Emerald Group Publishing Limited [ISSN 0143-991X]
[DOI 10.1108/01439911111106390]

computationally very complex, even for low degree-of-freedom (DOF) manipulators in simple static environments. In applications with moving obstacles the geometric models of the obstacles must continuously be re-mapped to the C-space. Motion planning in C-space is unreasonable in this case. Our proposed approach circumvents these issues by performing all motion planning in the task space (operational space) of the manipulator and thus exploits the fact that the task space is three-dimensional regardless of the complexity of the robot and its environment.

Other researchers have proposed simpler methods where the path of the robot is not modified, but the motion of the robot along the desired path is slowed in order to avoid collision (Spencer *et al.*, 2008). However, such a method is only effective if the objects being avoided will move out of the path of the robot after a period of time. Significant work has also been done on collision avoidance for redundant manipulators utilizing only the redundant DOFs (i.e. self-motion) (Zhang *et al.*, 2008; Zlajpah and Nemec, 2002; Maciejewski and Klein, 1985; Galicki, 2001). However, these methods only apply to redundant manipulators and do not avoid collisions between the end-effector and other objects.

On the whole, these existing collision avoidance methods have not found their way into mainstream use. Because of the computational demands associated with the previously described collision avoidance methods, practical implementations have often favored more rudimentary approaches, such as keep-out zones (Gilliland and Gilliland, 1998; Kato and Nagayama, 2001) and occupancy grids (Pollack and Hoffman, 1992; Shaffer and Herb, 1992). However, these methods tend to overly constrain the allowed motion by preventing motion into large portions of the workspace. The objective of this paper is to create a method for performing real-time collision avoidance that retains the best characteristics of both the complex and simple approaches. That is, we aim to generate optimal collision-free motion in real time with an algorithm that is computationally efficient enough for practical implementation.

The organization of this paper is as follows. First an overview of the steps involved in the collision avoidance algorithm is given. The method of geometric modeling of objects and how distance checking are handled is presented. Following that is a description how joint angle limits, joint velocity limits and collision avoidance limits are combined to create a set of constraints on the manipulator velocity. We then describe the constrained optimization method used to solve for the optimal collision-free robot velocity. The algorithm is then implemented in an experimental setup that demonstrates robot-to-robot collision avoidance as well as collision avoidance based on machine vision. Lastly, we end with some conclusions and discussion of future work.

Algorithm overview

The overall flow of the proposed algorithm is shown in Figure 1. It is assumed that the manipulator is being controlled at discrete time intervals – that is, updated motion commands are being sent to the robot at small periodic intervals, where the duration of these intervals is a few tens of milliseconds (at most) for most applications. In the implementation discussed later in this paper we use 20-ms intervals (50 Hz). The rate at which the algorithm must be executed is largely dependent on the speeds and accelerations of the robot and surrounding objects.

The algorithm is executed at every time step and the output of the algorithm is a set of joint velocities that the robot will be commanded to execute during the following time interval. Note that if the manipulator is position controlled and cannot directly execute velocity commands, the desired joint velocities can be integrated over the interval to produce the desired joint positions at the end of the interval.

The first step in the process is the measurement of the robot and its surroundings. For the robot avoiding collisions (indicated by the "Robot" block), this will typically be known from its built-in joint angle sensors. Sensing of other objects may be done by a variety of methods. Stationary objects may be modeled statically in advance if their location and geometry is known. The "Implementation and experimental results" section will give specific examples of how this was done with two different experimental setups.

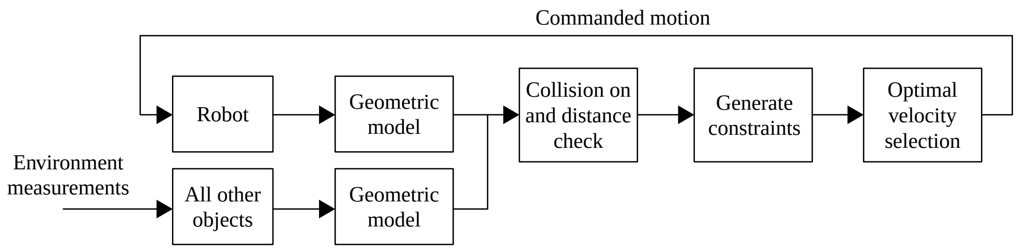
Based on these measurements, the physical poses of all bodies in the workspace are calculated with respect to the fixed global frame[1]. In the case of robotic manipulators, the forward (position) kinematics must be calculated. Once the physical location/pose of each body is determined, a geometric model of each body is created.

In most applications, the geometry of the manipulators and surrounding objects is reasonably complex. For example, a simple case of two manipulators on a tabletop is shown on the left in Figure 2. Were we to keep all of the details shown in this CAD drawing, the next step in our algorithm (collision and distance check) would become extremely cumbersome. To avoid this, we use a simplified geometric model that allows for fast calculation of the nearest points between bodies and the corresponding distances between them. In reality, the actual situations may have even greater geometric complexity than the example we are considering. Thus, the principle goal in the "Geometric model" step is to create a geometric representation of the manipulator and its surroundings that facilitates fast/easy geometric measurements. The details of the specific method used in our implementation are included in the "Geometry" section.

Given the geometric model for the instantaneous poses of all bodies in the workspace, a distance check is performed between all of the bodies (indicated by the "Collision and Distance Check" box in Figure 1). The details of how this is performed are described in the "Geometry" section. Based on these distance checks, a set of linear inequality constraints on the instantaneous joint velocities of the manipulator are created (shown as the "Generate Constraints" box in Figure 1). These constraints can be thought of as providing limits on the motion of any part of the robot towards a nearby object. The details of how these constraints are created are described in the "Constraint generation" section.

Once these constraints are created, the optimal velocity of the manipulator must be found (shown as the "Optimal Velocity Selection" box in Figure 1). This problem is formulated as a convex optimization problem whose solution is the manipulator velocity that most closely matches the desired velocity while not violating any of the velocity constraints. Once this optimal velocity has been found, the controller commands the manipulator to execute that set of joint velocities. After the discrete time interval has passed, the manipulator and surrounding objects have moved a small

Figure 1 Flow chart of collision avoidance process



amount and the entire process is then repeated in order to determine the joint velocity command at the next time step.

It is important to note that this method is not path based. That is, there is no recalculation of a collision-free path should an obstacle come near the robot. Rather, at each instant the optimal collision-free velocity is calculated in order to achieve as close as possible the desired velocity (e.g. velocity towards a goal pose). The robot travels a path as a result of this sequence of optimal velocities, but it is never necessary to calculate a complete robot path/trajectory.

Geometry

Geometric modeling

Our implementation of the "Geometric Model" step (in Figure 1) represents all objects as swept spherical bodies, each of which consists of the set of all points that are at a specified distance/radius from a geometric primitive. In our implementation, the geometric primitives are points, line segments, and rectangles. If the primitive is a point, the body is a sphere. If the primitive is a line segment, the body is a cylinder with spherical endcaps (also called a "cylisphere" (Holleman and Kavraki, 2000). If the primitive is a rectangle, the body is a box with rounded edges. For brevity, we will simply refer to these bodies as "spherical shells." As the right side of Figure 2 shows, a geometrically complex set of objects can be approximated by a collection of spherical shells. To more clearly illustrate the distinct spherical shells, the geometric primitive for each shell is superimposed over the shell. Each robot in our example is approximated by eight spherical shells (as can be seen from the eight-line segments for each). As a result, the complete geometric model of the robot and its surroundings at any instant can be represented by a list of bodies, the respective body types (i.e. point, line segment, or plane), the locations of the corner/end points on each body, and the radius associated with each body.

The primary advantage of using such a method for approximating the objects in our system is the simplicity in calculating the distance between any two objects. The distance $d(j, k)$ between objects j and k is simply:

$$d(j,k) = d_{\mathrm{p}}(j,k) - r_j - r_k \qquad (1)$$

where, $d_{\mathrm{p}}(j, k)$ is the distance between the primitives of objects j and k and r_j and r_k are the radii of objects j and k, respectively. Calculating the distance between the primitives (points, line segments, and rectangles) has a relatively straightforward closed-form solution in all cases.

Increasing the number of shells used per object can improve the accuracy of the geometry model, but at the cost of increased computation time for the algorithm. Based on our experiments, the level of detail shown in Figure 2 is sufficient to achieve very effective collision avoidance. It is also worth noting that this is not the only possible method of simplifying the geometric model of the system. Any representation of the system that allows geometric calculations to be performed quickly would be an acceptable substitute for the "Geometric Model" block.

As the "Constraint generation" section describes, the constraint generation portion of the algorithm creates limits on the motion of the robot based on how close the robot is to colliding with another object. In the implementation described in this paper, we accomplish this by creating a set of three spherical shells for each geometric primitive. This is shown in Figure 3, where we approximate the geometry of one link of the robot. The geometric primitive chosen is a line segment (not shown). Spherical shells with three different radii are then constructed from this single geometric primitive. The smallest (inner) shell is the "safety shell." This shell is the lower bound on how close an object can be to it. Specifically, the safety shell of an object should never be allowed to overlap any of the safety shells of the robot. The middle shell is the "equilibrium shell" and the largest (outer) shell is the "reaction shell." The radii of these shells are referred to

Figure 2 CAD model of robots and table (a) and reduced complexity geometric model (b)

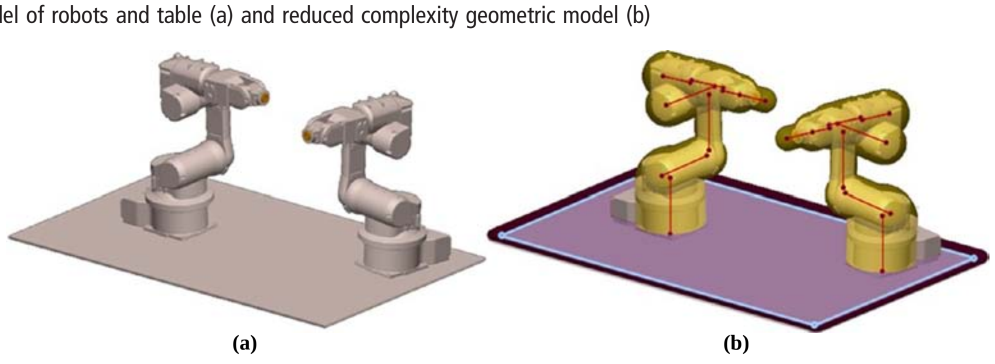
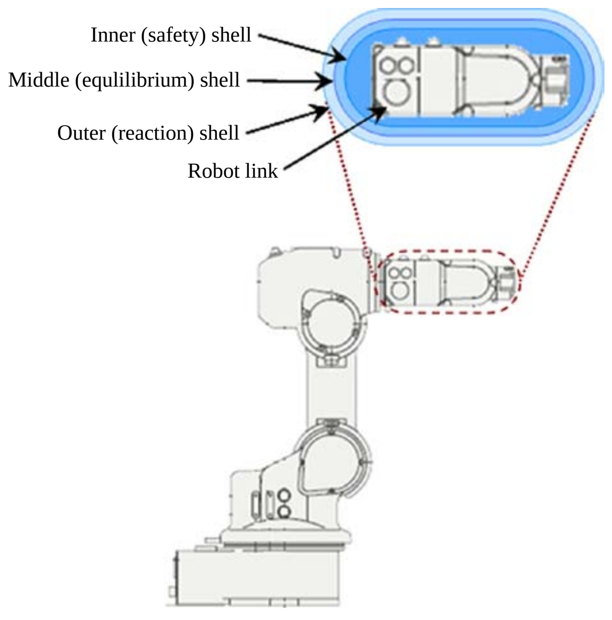


Figure 3 Spherical shell model of robot link



as the "safety radius," "equilibrium radius," and "reaction radius,", respectively, for a given body. These shells are used to gradually apply limits to the robot's motion as it approaches an object. The "Constraint generation" section discusses the details of how these shells are used.

Distance checking

Once the geometric modeling is complete, distances between them must be checked. For compactness, we will consider each geometric primitive and the three shells associated with it to collectively be a "body," denoted b. The set of bodies can be divided into two sets. The set of "robot" bodies, R, are all bodies that are a part of the controlled robot (i.e. the robot avoiding collisions)[2]. The set of "object" bodies, o, are all other bodies (including other robots in the workspace).

For each robot body $b_j \in R$ we must then determine the set of all bodies $b_k \in (R \cup O)$ that could possibly collide with it. This is done in three steps. The first step is performed off-line prior to running the collision avoidance algorithm and consists of manually removing potential collision pairs (b_j, b_k) from the list of possible collisions. This is primarily used to allow neighboring bodies to overlap each other without being treated as a collision by the algorithm. Collision between adjacent links is instead avoided through establishing appropriate joint limits (which are discussed in the "Constraint generation" section). For example, in Figure 2(b) it is clear that many of the robot bodies overlap each other even though the robot is not colliding with itself. In these cases, we would ignore the overlap between neighboring bodies by removing those pairs from the list of possible collisions. However, this is not the case for all pairs of robot bodies. For example, we want to prevent collision between the end-effector and the base of the robot, thus we would keep the collision pair for those bodies on our list of possible collisions. Body pairs that are not eliminated by step one must be checked in each iteration of the algorithm.

not needed 4 midterm → robot can collide w/itself/ no problem

The second and third steps are performed within the algorithm as the "Collision and Distance Check" in Figure 1.

The purpose of the second step is eliminate potential collision pairs (b_j, b_k) that are far enough apart that they can be ignored at this instant. We have implemented this step by constructing bounding boxes around every body in the workspace. The bounding boxes of each potential collision pair (b_j, b_k) are compared and if the bounding boxes do not overlap then that (b_j, b_k) pair can be excluded from the list of possible colliding bodies. This step is very fast and can quickly eliminate the vast majority of potential collisions from consideration.

for me is each point in the robot arm & the obstacle

In the third step, we take all (b_j, b_k) pairs that were not eliminated in the first two steps and calculate the actual distance between the bodies. As the following section shows, we need only find the distance between the geometric primitives associated with the bodies. Because of the simple geometric primitives chosen, this can be calculated very quickly. The output of this step is the shortest distance $d_{\mathrm{p}}(j, k)$ between the primitives of (b_j, b_k), and the points on each of the primitives corresponding to this shortest distance. For the ith pair of (b_j, b_k) considered we refer to this distance as d_{p_i} and we refer to these points as cp_i (the "collision point"), and ip_i (the "interfering point"), where cp_i is on body b_j and ip_i is on body b_k.

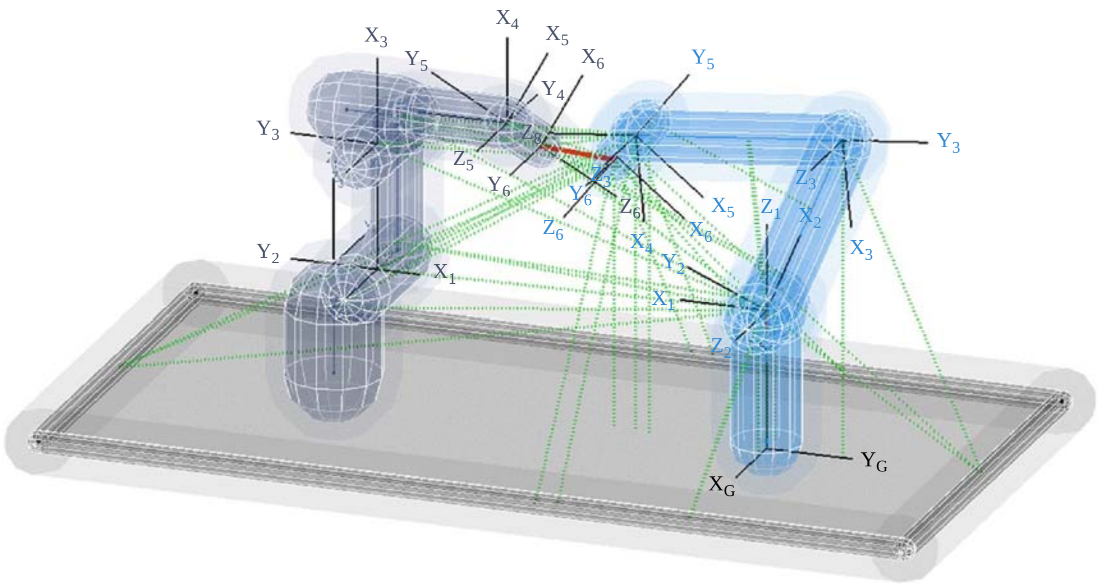
define "safety threshold" $\Leftarrow$

This process is illustrated in Figure 4, which shows a Matlab simulation of two robots on a table top, where the robot on the right is avoiding collisions with the robot on the left. The left robot is approximated by eight bodies and the robot on the right is approximated by six bodies. When the first step of distance checking is performed, several pairs of collisions are removed from consideration. For example, we do not check for collision between the adjacent links that join at the elbow of the robot, as they nominally overlap each other (collision is avoided via joint limits). The green lines represent the shortest distance between all (b_j, b_k) pairs after the first step is complete (for illustration only – not all distances need to be calculated). When the second step (bounding box check) is performed, nearly all (b_j, b_k) pairs are removed from consideration. The only pair that remains corresponds to the bodies that are associated with the end-effectors of the robots. The third step produces d_{p_i}, cp_i, and ip_i for the potential collision between the two end-effectors. This potential collision is shown in Figure 4 by the red-line segment between the two end-effectors. It is worth noting that the collision and distance check method described here would need to be modified if a different geometric modeling approach were used.

Constraint generation

Given the current state of the system (as represented by the geometric model), the desired velocity of the end-effector (based on the task the manipulator must perform), and the set of all possible collisions, the next step is to generate constraints that limit the motion of the robot in order to avoid these collisions. In addition to avoiding collisions (both with other objects and with itself), the manipulator's commanded motion must not violate its joint angle limits or joint velocity limits. The method that we propose is to create a set of linear inequality constraints on the commanded velocity of the robot. For convenience, we formulate these limits in the end-effector (task) space in the implementation described here. Note that because the Jacobian mapping of joint velocities to end-effector velocities is linear, these constraints can be expressed in either the joint space or the task space and they will still be linear[3]. As a result, the set of allowable

Figure 4 Simulation of distance checking



velocities of the robot at this instant forms a convex set. We can take advantage of this structure by formulating the velocity selection problem as a convex optimization. ? This optimization can be calculated very efficiently which allows collision-free motions to be computed in real time.

The form of the convex optimization problem we are working with is:

$$\begin{aligned} &\text{minimize} \quad f_0(\mathbf{x}) \\ &\text{subject to} \quad f_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, \kappa \end{aligned} \tag{2}$$

where, there are κ simultaneous inequality constraints. The objective function $f_0(\mathbf{x})$ can be any convex function that produces "optimal" behavior when minimized. We simply wish to minimize the error between the actual end-effector linear and angular velocity $(v^T \omega^T)^T_{ee}$ and the desired end-effector linear and angular velocity $(v^T \omega^T)^T_{ee,d}$. Thus, we have chosen to use:

$$f_0(\mathbf{x}) = ((\mathbf{x} - \mathbf{x}_d)^T (\mathbf{x} - \mathbf{x}_d))^{1/2} \tag{3}$$

where:

$$\mathbf{x} = \mathbf{W}\begin{pmatrix} v \\ w \end{pmatrix}_{ee} \quad \mathbf{x}_d = \mathbf{W}\begin{pmatrix} v \\ w \end{pmatrix}_{ee,d} \tag{4}$$

and where, $\mathbf{W}$ is a weighting matrix with appropriate unit terms such that all of the elements of $\mathbf{x}$ have the same units:

$$\mathbf{W} = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \alpha\mathbf{I}_{3\times3} \end{bmatrix}. \tag{5}$$

For example, a valid choice would be $\alpha = 1\, m/rad$. The choice of equation (3) as our objective function will result in selection of an optimal velocity that minimizes the weighted Euclidian norm of the difference between the desired and actual end-effector velocities.

In other cases, we may wish to have additional criteria included in the objective function. For example, with redundant manipulators additional terms can be added to the objective function to represent optimal use of the self-motion of the robot, provided the resulting $f_0(\mathbf{x})$ is a convex, twice-differentiable function.

Given the objective function, it is necessary to define the constraint functions $f_i(\mathbf{x})$. These constraints are due to joint angle limits, joint velocity limits and potential collisions.

Constraints due to joint limits

There are two different cases of joint limits: joint angle limits and joint velocity limits. We wish to accommodate both cases with a single set of upper and lower joint velocity limits. To do this, we form a vector that contains all of the upper limits on allowable joint velocities at this instant:

$$\dot{\mathbf{q}}_{ul} = \begin{bmatrix} \dot{q}_{ul_1} \\ \vdots \\ \dot{q}_{ul_n} \end{bmatrix} \tag{6}$$

where, the value of each $\dot{q}_{ul_i}$ depends on the current angle of joint i:

$$\dot{q}_{ul_i} = \begin{cases} \dot{q}_{upper_i} & q_i < q_{upper_i} \\ 0 & q_i \geq q_{upper_i} \end{cases} \tag{7}$$

That is, the joint i is nominally upper bounded by a user-specified velocity limit $(\dot{q}_{upper_i})$ (typically, the rated velocity limit of the joint[4]), but if joint i has begun to exceed its joint limit (q_{upper_i}) its upper velocity limit is 0 and the angle is not allowed to increase further. A similar vector of lower limits on allowable joint velocities is created:

$$\dot{\mathbf{q}}_{ll} = \begin{bmatrix} \dot{q}_{ll_1} \\ \vdots \\ \dot{q}_{ll_n} \end{bmatrix} \tag{8}$$

where:

$$\dot{q}_{ll_i} = \begin{cases} \dot{q}_{lower_i} & q_i > q_{lower_i} \\ 0 & q_i \leq q_{lower_i} \end{cases} \tag{9}$$

Given the values for the upper and lower joint velocity limits in equations (7) and (9), we can state the instantaneous limit on $\dot{q}_i$ is:

$$\dot{q}_{ll_i} \leq \dot{q}_i \leq \dot{q}_{ul_i}. \tag{10}$$

We can restate equation (10) using equations (6) and (8) as:

$$\hat{\mathbf{s}}_i^T \dot{\mathbf{q}}_{ll} \leq \hat{\mathbf{s}}_i^T \dot{\mathbf{q}} \leq \hat{\mathbf{s}}_i^T \dot{\mathbf{q}}_{ul} \tag{11}$$

where, $\hat{\mathbf{s}}_i$ is a vector that selects the terms corresponding to the ith joint:

$$\hat{\mathbf{s}}_1 = [1 \quad 0 \quad \cdots \quad 0]^T. \tag{12}$$

Here, the $(\hat{\ })$ notation indicates that a vector is a unit vector.

Let us begin by creating the constraint functions $f_i(\mathbf{x})$ corresponding to the right portion of equation (11) (i.e. the upper limits):

$$\hat{\mathbf{s}}_i^T \dot{\mathbf{q}} \leq \hat{\mathbf{s}}_i^T \dot{\mathbf{q}}_{ul}. \tag{13}$$

Using the Jacobian relationship for serial manipulators:

$$\begin{pmatrix} v \\ w \end{pmatrix}_{ee} = \mathbf{J}\dot{\mathbf{q}} \tag{14}$$

and assuming that the manipulator is not in a singular configuration we can invert $\mathbf{J}$ and substitute equation (14) into equation (13):

$$\hat{\mathbf{s}}_i^T \mathbf{J}^{-1} \begin{pmatrix} v \\ w \end{pmatrix}_{ee} \leq \hat{\mathbf{s}}_i^T \dot{\mathbf{q}}_{ul}. \tag{15}$$

Using the fact that $\mathbf{W}^{-1}\mathbf{W} = \mathbf{I}$, we can insert it into equation (15) and group terms:

$$\underbrace{\hat{\mathbf{s}}_i^T \mathbf{J}^{-1} \mathbf{W}^{-1}}_{\mathbf{k}_{l_i}^T} \mathbf{W} \begin{pmatrix} v \\ w \end{pmatrix}_{ee} \leq \hat{\mathbf{s}}_i^T \dot{\mathbf{q}}_{ul} \tag{16}$$

where:

$$\mathbf{k}_{l\,i}^T = \hat{\mathbf{s}}_i^T \mathbf{J}^{-1} \mathbf{W}^{-1}. \tag{17}$$

Dividing through by the magnitude of equation (17) produces:

$$\underbrace{\frac{\mathbf{k}_{l\,i}^T}{\|\mathbf{k}_{l\,i}^T\|}}_{\hat{\mathbf{a}}_{ul_i}^T} \underbrace{\mathbf{W} \begin{pmatrix} v \\ w \end{pmatrix}_{ee}}_{\mathbf{x}} \leq \underbrace{\frac{\hat{\mathbf{s}}_i^T}{\|\mathbf{k}_{l\,i}^T\|} \dot{\mathbf{q}}_{ul}}_{b_{ul_i}}. \tag{18}$$

Thus, the upper joint limit on joint i reduces to:

$$\hat{\mathbf{a}}_{ul_i}^T \mathbf{x} \leq b_{ul_i}. \tag{19}$$

We then create the constraint functions $f_i(x)$ corresponding to the left portion of equation (11) (i.e. the lower limits) similarly:

$$-\hat{\mathbf{s}}_i^T \dot{\mathbf{q}} \leq -\hat{\mathbf{s}}_i^T \dot{\mathbf{q}}_{ll} \tag{20}$$

$$-\mathbf{k}_{l\,i}^T \mathbf{W} \begin{pmatrix} v \\ w \end{pmatrix}_{ee} \leq -\hat{\mathbf{s}}_i^T \dot{\mathbf{q}}_{ll} \tag{21}$$

$$\underbrace{\frac{\mathbf{k}_{l\,i}^T}{\|\mathbf{k}_{l\,i}^T\|}}_{\hat{\mathbf{a}}_{ll_i}^T} \underbrace{\mathbf{W} \begin{pmatrix} v \\ w \end{pmatrix}_{ee}}_{\mathbf{x}} \leq \underbrace{\frac{\hat{\mathbf{s}}_i^T}{\|\mathbf{k}_{l\,i}^T\|} \dot{\mathbf{q}}_{ll}}_{b_{ll_i}} \tag{22}$$

$$\hat{\mathbf{a}}_{U_i}^T \mathbf{X} \leq b_{U_i}. \tag{23}$$

Combining coefficients of equations (19) and (23) for all n joints produces:

$$\mathbf{A}_{ul} = \begin{bmatrix} \hat{\mathbf{a}}_{ul_1}^T \\ \vdots \\ \hat{\mathbf{a}}_{ul_n}^T \end{bmatrix} \quad \mathbf{b}_{ul} = \begin{bmatrix} b_{ul_1} \\ \vdots \\ b_{ul_n} \end{bmatrix} \tag{24}$$

$$\mathbf{A}_{ll} = \begin{bmatrix} \hat{\mathbf{a}}_{ll_1}^T \\ \vdots \\ \hat{\mathbf{a}}_{ll_n}^T \end{bmatrix} \quad \mathbf{b}_{ll} = \begin{bmatrix} b_{ll_1} \\ \vdots \\ b_{ll_n} \end{bmatrix} \tag{25}$$

Then our set of $2n$ constraints due to joint limits simplifies to:

$$\mathbf{A}_l \mathbf{x} \leq \mathbf{b}_l \tag{26}$$

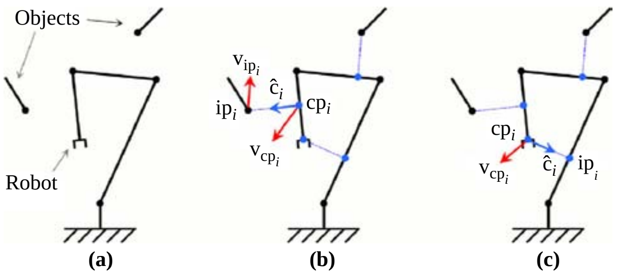
where:

$$\mathbf{A}_l = \begin{bmatrix} \mathbf{A}_{ul} \\ \mathbf{A}_{ll} \end{bmatrix} \quad \mathbf{b}_l = \begin{bmatrix} \mathbf{b}_{ul} \\ \mathbf{b}_{ll} \end{bmatrix}. \tag{27}$$

Constraints due to potential collisions

Recall from the "Geometry" section that the collision and distance check generated pairs of points that could be colliding: the collision point cp$_i$ and the interfering point ip$_i$. For each potential collision, we wish to constrain the velocity of cp$_i$ (on the robot) towards ip$_i$. Graphically, we can see this shown in Figure 5. In Figure 5(a) a robot and two objects near it are represented by their respective geometric primitives (line segments in this case)[5]. In Figure 5(b) and (c), there are dashed lines between the three pairs of possible collision points. Figure 5(b) shows the potential collision between an external object and the robot and Figure 5(c) shows the

Figure 5 Diagram of collision nomenclature



potential collision between the robot and itself. In each case, we construct a unit vector $\hat{\mathbf{c}}_i$ in the collision direction (directed from cp$_i$ towards ip$_i$). In addition, we denote the velocity of cp$_i$ as $\mathbf{V}_{\text{cp}_i}$ and the velocity of ip$_i$ as $\mathbf{V}_{\text{ip}_i}$.

We can restate our limit on the motion of cp$_i$ towards ip$_i$ as:

$$\hat{\mathbf{c}}_i^T \mathbf{V}_{\text{cp}_i} \leq v_{\text{a}_i} \tag{28}$$

where, v_{a_i} is the maximum allowed "approach velocity" of cp$_i$ towards ip$_i$. The manner in which v_{a_i} is specified can be varied depending on what the desired collision avoidance behavior is. In our implementation, we have constructed multiple shells (reaction, equilibrium, and safety) for each body in order to create a smoothed approach velocity v_{a_i}, which is expressed as:

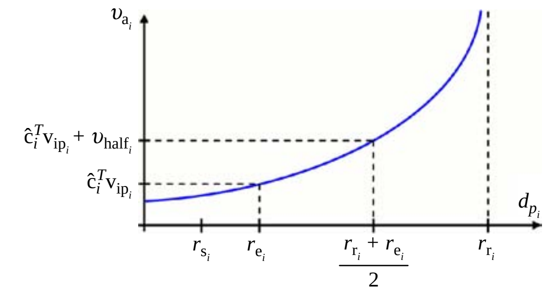
$$v_{ai} = \hat{\mathbf{c}}_i^T \mathbf{V}_{\text{ip}_i} + \frac{v_{\text{half}_i}}{\ln(0.5)} \ln\left(\frac{r_{\text{r}i} - d_{\text{p}_i}}{r_{\text{r}i} - r_{\text{e}i}}\right) \tag{29}$$

where, d_{p_i} is the distance between cp$_i$ towards ip$_i$, $r_{\text{r}i}$ is the sum of the reaction radii of the two bodies, $r_{\text{e}i}$ is the sum of the equilibrium radii of the two objects, and v_{half_i} is the maximum approach velocity allowed when d_{p_i} is halfway between $r_{\text{r}i}$ and $r_{\text{e}i}$. The maximum approach velocity v_{half_i} is a parameter that can tuned in order to adjust how rapidly the relative velocities of the bodies are reduced. The resulting maximum allowed approach velocity is plotted in Figure 6 as a function of the separation distance. Note that the equation is only valid (and thus only used) when $r_{\text{r}i} > d_{\text{p}_i} > 0$. When the objects are separated by exactly the sum of their equilibrium radii, the avoidance velocity must equal the velocity of the interfering point along the collision direction to ensure that they do not get any closer. As the separation distance increases to the sum of the reaction radii, the avoidance velocity increases to infinity, effectively ignoring the constraint. When the objects are closer than the sum of their equilibrium radii, the avoidance velocity is lowered to force them apart. If d_{p_i} ever drops below $r_{\text{s}i}$ (the sum of the safety radii of the two bodies) an emergency stop of the system is triggered. Note that the rule we choose here in equation (29) can be replaced by any equivalent function.

Given v_{ai}, we must now expand the left side of equation (28) to express the velocity of cp$_i$ in terms of the robot joint velocities. This is accomplished by treating cp$_i$ as if it were the end-effector of the robot and creating a "partial Jacobian matrix" ($\mathbf{J}_{\text{cp}_i}$) for this point. In other words, we virtually truncate the robot at cp$_i$ and only consider the effects of joints between cp$_i$ and the robot base. Utilizing the Jacobian relationship for serial manipulators we express $\mathbf{v}_{\text{cp}_i}$ as:

$$\mathbf{v}_{\text{cp}_i} = \mathbf{J}_{\text{cp}_i} \dot{\mathbf{q}}. \tag{30}$$

Figure 6 Approach velocity vs separation distance



In the case shown in Figure 5(b) where the potential collision is between the robot and another object $\mathbf{J}_{\text{cp}_i}$ is:

$$\mathbf{J}_{\text{cp}_i} = \begin{bmatrix} \frac{(\mathbf{v}_{\text{cp}_i})_x}{\dot{q}_1} & \cdots & \frac{(\mathbf{v}_{\text{cp}_i})_x}{\dot{q}_\mu} & 0 & \cdots & 0 \\ \frac{(\mathbf{v}_{\text{cp}_i})_y}{\dot{q}_1} & \cdots & \frac{(\mathbf{v}_{\text{cp}_i})_y}{\dot{q}_\mu} & 0 & \cdots & 0 \\ \frac{(\mathbf{v}_{\text{cp}_i})_z}{\dot{q}_1} & \cdots & \frac{(\mathbf{v}_{\text{cp}_i})_z}{\dot{q}_\mu} & 0 & \cdots & 0 \end{bmatrix} \tag{31}$$

where, there are μ joints between cp$_i$ and the ground. In the case of potential collision between the robot and itself as is shown in Figure 5(c), the motion of the joints below ip$_i$ do not affect the distance between cp$_i$ and ip$_i$. Thus, we can ignore the effects of these joints and formulate $\mathbf{J}_{\text{cp}_i}$ as:

$$\mathbf{J}_{\text{cp}_i} = \begin{bmatrix} 0 & \cdots & 0 & \frac{(\mathbf{v}_{\text{cp}_i})_x}{\dot{q}_\eta} & \cdots & \frac{(\mathbf{v}_{\text{cp}_i})_x}{\dot{q}_\mu} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \frac{(\mathbf{v}_{\text{cp}_i})_y}{\dot{q}_\eta} & \cdots & \frac{(\mathbf{v}_{\text{cp}_i})_y}{\dot{q}_\mu} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \frac{(\mathbf{v}_{\text{cp}_i})_z}{\dot{q}_\eta} & \cdots & \frac{(\mathbf{v}_{\text{cp}_i})_z}{\dot{q}_\mu} & 0 & \cdots & 0 \end{bmatrix} \tag{32}$$

where there are $\eta - 1$ joints between ip$_i$ and the ground. Substituting equation (30) into equation (28) produces:

$$\hat{\mathbf{c}}_i^T \mathbf{J}_{\text{cp}_i} \dot{\mathbf{q}} \leq v_{\text{a}i}. \tag{33}$$

Using equation (14), we substitute for $\dot{q}$:

$$\hat{\mathbf{c}}_i^T \mathbf{J}_{\text{cp}_i} \mathbf{J}^{-1} \begin{pmatrix} v \\ w \end{pmatrix}_{ee} \leq v_{\text{a}i}. \tag{34}$$

and using the same approach as equation (16), we insert $\mathbf{W}^{-1}\mathbf{W}$ and group terms:

$$\underbrace{\hat{\mathbf{c}}_i^T \mathbf{J}_{\text{cp}_i} \mathbf{J}^{-1} \mathbf{W}^{-1}}_{\mathbf{k}_{ci}^T} \mathbf{W} \begin{pmatrix} v \\ w \end{pmatrix}_{ee} \leq v_{ai} \tag{35}$$

where:

$$\mathbf{k}_{ci}^T = \hat{\mathbf{c}}_i^T \mathbf{J}_{\text{cp}_i} \mathbf{J}^{-1} \mathbf{W}^{-1}. \tag{36}$$

Equation (35) is thus reduced to:

$$\mathbf{k}_{ci}^T \mathbf{W} \begin{pmatrix} v \\ w \end{pmatrix}_{ee} \leq v_{\text{a}i}. \tag{37}$$

Dividing through by the magnitude of equation (36) produces:

$$\underbrace{\frac{\mathbf{k}_{ci}^T}{\|\mathbf{k}_{ci}^T\|}}_{\hat{\mathbf{a}}_{ci}^T} \underbrace{\mathbf{W} \begin{pmatrix} v \\ w \end{pmatrix}_{ee}}_{\mathbf{x}} \leq \underbrace{\frac{v_{ai}}{\|\mathbf{k}_{ci}^T\|}}_{b_{ci}}. \tag{38}$$

Thus, the limits due to the *i*th potential collision (equation (28)) reduce to:

$$\hat{\mathbf{a}}_{ci}^T \mathbf{X} \leq b_{ci}. \tag{39}$$

Combining the coefficients of equation (39) for all m potential collisions produces:

$$\mathbf{A}_c = \begin{bmatrix} \hat{\mathbf{a}}_{c1}^T \\ \vdots \\ \hat{\mathbf{a}}_{cm}^T \end{bmatrix} \quad \mathbf{b}_c = \begin{bmatrix} b_{c1} \\ \vdots \\ b_{cm} \end{bmatrix} \tag{40}$$

which can be combined with equation (26):

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_l \\ \mathbf{A}_c \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_l \\ \mathbf{b}_c \end{bmatrix} \tag{41}$$

such that all constraints on the manipulator's velocity due to joint angle limits, joint velocity limits, potential collisions with other objects, and potential collisions with itself are expressed within the single set of equations:

$$\mathbf{Ax} \le \mathbf{b}. \tag{42}$$

To achieve the desired form of the constraints, we simply rearrange equation (42) to:

$$\mathbf{Ax} - \mathbf{b} \le 0 \tag{43}$$

where each $f_i(\mathbf{X})$ is represented by the ith row of the left side of equation (43). Thus, the total number of constraints is $k = 2n + m$.

Before we continue to performing the constrained optimization, it is important to discuss the special cases that cause this method to fail. First, it is necessary for the manipulator to be in a non-singular configuration. This is typically achieved through appropriate selection of joint limits to avoid singular configurations or using additional software to transition the robot through the singularity and then resuming the collision avoidance algorithm. Also, in the case where the range space of $\mathbf{J}_{\mathrm{cp}_i}$ (i.e. the set of all possible velocities of cp$_i$) is orthogonal to $\hat{\mathbf{c}}_i$, then the left side of equation (33) equals zero regardless of the choice of $\dot{q}$. In this case, the robot cannot move cp$_i$ away from ip$_i$ at this instant. This scenario is rare, and can be overcome by adding additional software logic that commands the robot to move cp$_i$ such that the orthogonality condition changes and the collision avoidance algorithm can resume.

Constrained optimization

Restating equation (2), our collision avoidance problem has been reduced to a constrained optimization of the form:

$$\begin{aligned} &\text{minimize} \quad f_0(\mathbf{x}) \\ &\text{subject to} \quad f_i(\mathbf{x}) \le 0, \quad i = 1, \ldots, \kappa \end{aligned}. \tag{44}$$

Our objective function $f_0(\mathbf{x})$ and constraint functions $f_1(\mathbf{x}), \ldots, f_\kappa(\mathbf{x})$ are detailed in equations (3) and (43), respectively. In addition, $f_0, \ldots, f_\kappa : \mathbf{R}^n \rightarrow \mathbf{R}$ are convex and twice continuously differentiable. Thus, convex optimization techniques are applicable.

Convex optimization is a subject of extensive study within mathematics. We have chosen to use an interior point method of solving the convex optimization. In particular, we are using the logarithmic barrier method. This method is very well-suited for solving this problem because it is very fast, can handle an arbitrarily large number of inequality constraints, and we can mathematically prove the accuracy of the result (bound the error).

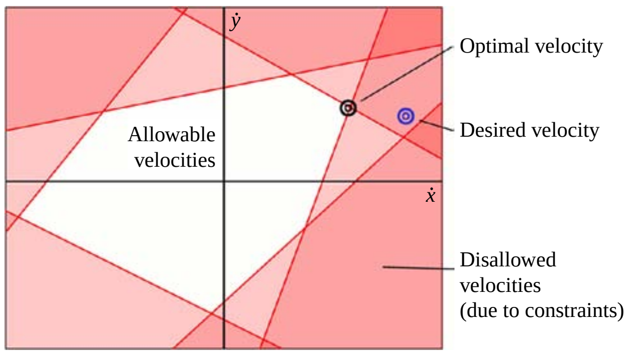
For brevity, we will not detail the method completely, but refer the reader to Hsu *et al.* (1999) for an excellent overview of the method. Briefly, the problem is shown in Figure 7, where each constraint $f_i(\mathbf{x})$ creates a set of velocities that are disallowed. When all κ constraints are combined, the remaining set of allowable velocities is convex. The basic approach of the logarithmic barrier method is to model each inequality constraint as a logarithmic penalty function that grows to infinity as $\mathbf{x}$ approaches the $f_i(\mathbf{x}) = 0$ barrier. These penalty functions are added to the objective function $f_0(\mathbf{x})$, and an unconstrained optimization of $\mathbf{x}$ is performed via an iterative descent method (typically Newton's method). This value of $\mathbf{x}$ is then refined by scaling the magnitude of the penalty functions and again optimizing $\mathbf{x}$ via the descent method. This process is iterated to find $\mathbf{x}_{opt}$, the optimal value of $\mathbf{x}$. Then using equations (4) and (14), the optimal joint velocities $\dot{\mathbf{q}}_{opt}$ are:

$$\dot{\mathbf{q}}_{opt} = \mathbf{J}^{-1}\mathbf{W}^{-1}\mathbf{x}_{opt}. \tag{45}$$

These joint velocities are commanded to the manipulator. After they are executed for the duration of the specified time interval, the algorithm is repeated.

It is worth noting at this point that while the algorithm presented here describes how to modify the motion of a single manipulator, it is easily extensible to any number of manipulators. In the case where several manipulators must simultaneously avoid collisions with each other (and any other nearby obstacles) they would each have their motion control governed by an independent collision avoidance algorithm running on their respective controller. While a large number of manipulators increases the number of collisions that must be checked for, the collision check step is very simple, and the resulting collision-checking computational burden for each controller would remain relatively unchanged. An increase in the number of active constraints for a robot (i.e. overlapping reaction shells) would increase computation time, but experimentation with the test-bed described in the following section has shown that hundreds of simultaneous active constraints can be handled with little impact to computation time, which should be sufficient even when a large number of manipulators are used.

Figure 7 Velocity constraints in the end-effector space



Implementation and experimental results

An experimental setup was created (shown in Figure 8(a)) to test the effectiveness of the proposed algorithm. This setup consists of two manipulators attached to a tabletop and controlled by a common master controller. The robot on the left (referred to as robot 1) is an Epson Pro-Six PS5 manipulator, which is a six DOF anthropomorphic arm. Robot 1 is mounted on a one DOF linear stage which can translate the robot along the length of the table. The robot on the right (robot 2) is a Mitsubishi PA 10-6 C manipulator, which is also a six DOF anthropomorphic arm. Note that while we use traditional six DOF anthropomorphic arms in this example, the collision-avoidance algorithm is applicable to manipulators of any number of DOFs.

The master controller of the system is implemented on a PC with an Intel Core2 Duo CPU with two 2.67 GHz cores and 2 GB RAM running Windows XP SP2. The control software on the master controller is written in the Matlab Simulink environment. In addition, Quanser Consulting, Inc.'s QuaRC software is used to allow the Simulink models to be executed in real time. This is accomplished with the timing capabilities of a Quanser Q8 Hardware-in-the-Loop Board, which is mounted in one of the PC's PCI slots. The master controller communicates with robot 1 through a serial connection with the Epson RC520 controller. Communication with robot 2 is provided by an ARCnet connection to the Mitsubishi controller. Note that joint velocity commands can be sent directly to the Mitsubishi controller, which then regulates motor currents to achieve the desired joint velocities. Both manipulators have joint angle sensors (encoders on robot 1 and resolvers on robot 2) and the current joint angles are reported back to the master controller by their respective controllers. The corresponding poses of the two robots are determined using the forward kinematic equations for each.

Robot-to-robot collision avoidance

In our first experiment, the collision avoidance algorithm is implemented on robot 2. Robot 2 is programmed to move towards a desired pose (the pose shown in Figure 8(a)) but to avoid collisions with robot 1. Robot 1 is then given a somewhat arbitrary pre-programmed trajectory to follow that takes it through the workspace of robot 2 from a variety of angles. In this scenario, we essentially give precedence to robot 1 – it can move freely through its motions without any considerations for the other robot – while robot 2 behaves as a "subordinate" robot and must sacrifice its goal of achieving the desired pose in order to avoid collisions with robot 1. Thus, robot 2 starts in its desired pose, deviates from it as necessary to avoid collisions, and then returns to its desired pose when robot 1 moves out of the way.

The geometric model of each robot consists of eight bodies, all with line segment primitives (similar to the model shown in Figure 4). The radius of each safety shell was chosen to be just large enough to fully enclose the associated portion of the robot. The tabletop and linear stage are modeled as bodies with rectangular primitives with safety shells of 0.02-m radii. The radius of the equilibrium shell for every body was then chosen to be 0.02 m larger than the safety shell radius of the body. Similarly, the radius of the reaction shell for every body was then chosen to be 0.04 m larger than the safety shell radius of the body.

The duration of the experiment was approximately 30 s, with robot 1 making five passes through the workspace of robot 2 in a variety of directions. The maximum speed of the end-effector of robot 1 during these passes was approximately 2 m/s. Robot 2 was able to avoid collisions with robot 1, as shown in Figure 8(b) and (c).

Recall from the "Geometry" section that each potential collision pair (bj, b_k) is examined to determine whether collision could occur. All potential collision pairs are compared via a bounding-box method to determine whether the exact distance between the bodies needs to be calculated. Of the five passes that robot 1 made, passes 2 and 3 resulted in the greatest number of potential collisions. During this 11-s time period in the experiment, 28 different potential collision pairs required calculation of the distances between the bodies. To verify the effectiveness of the algorithm, these distances are plotted in Figure 9. Specifically, the distances between the safety shells of the two bodies are plotted vs time for all 28 cases, where every curve (each one plotted in a different color) represents a different collision case that is being tracked. In addition, the equilibrium radii of all bodies are identically 0.02 m and the reaction radii of all bodies are identically 0.04 m. Thus, we have superimposed threshold lines at 0.04 m and 0.08 m, which represent the distance at which the equilibrium shells of the two bodies overlap (short-dashed line) and the distance at which the reaction shells of the two bodies overlap (long-dashed line), respectively.

As Figure 9 shows, the distance between the safety shells of any two bodies never drops to zero, thus collision never occurs. Moreover, we can see that pairs of bodies rarely get closer than having their equilibrium shells overlap. This intuitively makes sense, because this is the distance at which the maximum

Figure 8 Collision avoidance between two robots with robot 2 (on right) actively avoiding robot 1 (on left)

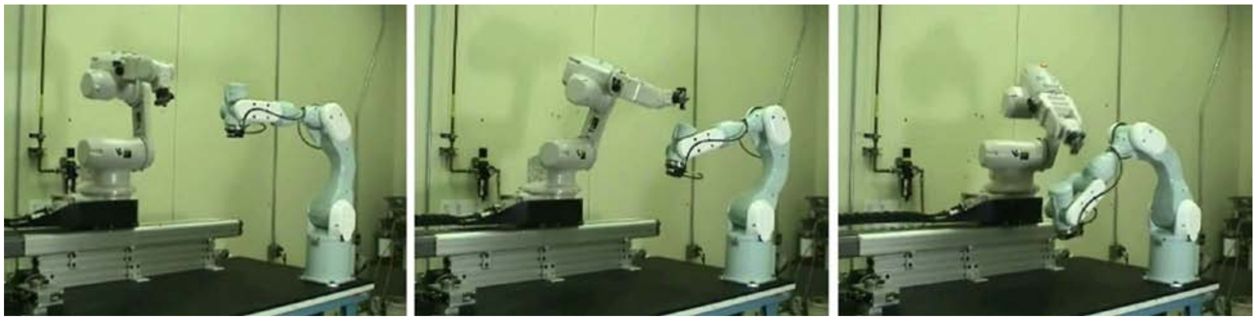
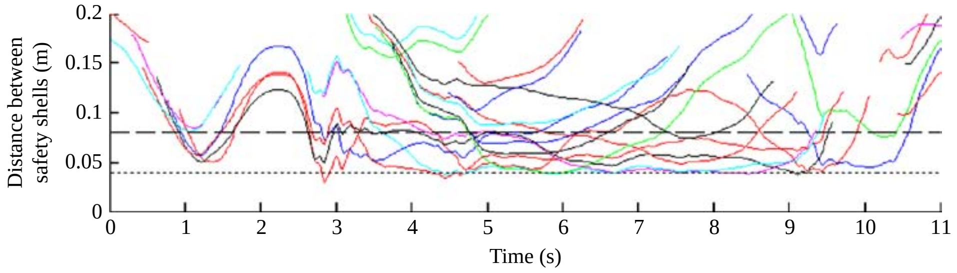

(a) (b) (c)

Figure 9 Distance between the safety shells of various bodies on robots 1 and 2 during a portion of the first experiment



approach velocity of robot 2 towards robot 1 equals zero. The few times that the distance drops below this threshold are due to acceleration of robot 1 towards robot 2. Note that the max allowable acceleration towards the robot is a function of sensing latency, the rate at which the algorithm is executed, and the value of the v_{ai} function (29) when $d_{p_i} = 0$.

Figure 9 also shows that up to nine collision pairs are close enough that their reaction shells overlap (at approximately 4.9 s). Recall that a constraint on the commanded robot 2 velocity is created for each instance where this occurs. The increase in calculations required did not adversely affect the performance of the algorithm. However, through a separate experiment not discussed in this paper we were able to verify that our experimental setup can handle up to 240 simultaneous constraints on the manipulator velocity. Moreover, this number could be significantly increased through the use of more efficient software. Simulink is an excellent tool for quickly prototyping algorithms but is not very efficient in terms of the speed of execution of the compiled Simulink/QuaRC code.

Vision-based collision avoidance

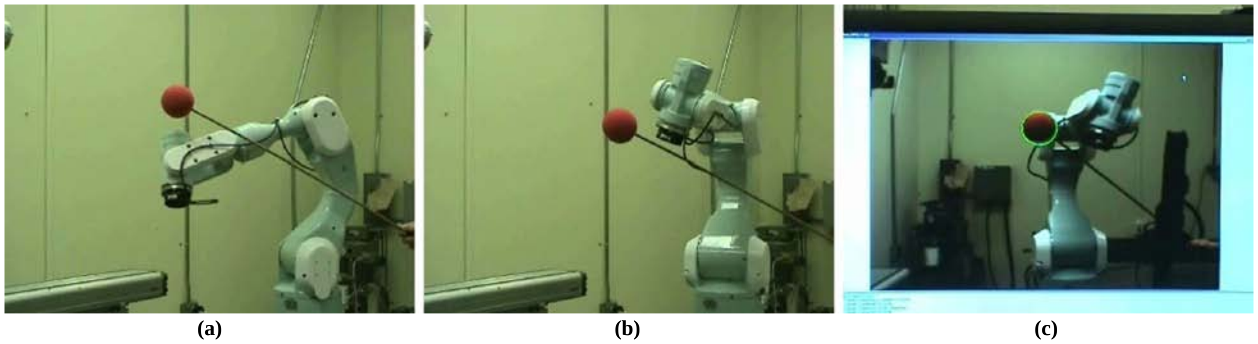
In our second experiment, the collision avoidance algorithm is again implemented on robot 2. The geometric model of robot 2 is unchanged from the first experiment. As in the first experiment, robot 2 is programmed to have a desired pose (the same pose as in the first experiment), but in this case robot 2 is programmed to avoid collision with a foam ball that is attached to the end of a stick. A person then manually moves the ball around the workspace of robot 2 and it avoids collision with the ball. In this experiment the position of the ball is found using a machine vision system. A single color camera is mounted to the left side of the table approximately 0.7 m above the surface of the table (not shown in Figure 8 or Figure 10). The camera is a Prosilica GC1020C 1024 × 768 GigE camera that runs at 30 fps. It is connected by gigabit Ethernet to a second PC that performs machine vision processing to determine the position of the ball within the workspace. This processing is partly performed on the PC CPU (an Intel Core2 Quad CPU with four 2.83 GHz cores). It is sped up by parallelizing some of the processing and executing it on the graphics card (an NVidea GeForce 9600 GT GPU). We take advantage of a priori knowledge about the ball (i.e. its size, shape, and color) to enable this process[6]. Based on hue the ball is segmented from the background and the location of its center is calculated. Based on the known diameter of the ball and the apparent size of the ball in the image we can calculate the distance of the ball from the camera. Combining this with the ball centroid location we extract the 3D coordinates of the ball and send the location information to the master controller. This processing can be executed at approximately 30 Hz.

The results of the experiment were very successful. The robot is able to avoid collisions with the ball for a wide variety of ball motions at a maximum speed of approximately 0.5 m/s. Collision with the ball could not be avoided for very fast motion of the ball (> 1 m/s) due to latency in the vision system sensing of the ball position. If faster objects need to be avoided with a system that has this level of latency the reaction shell radii should be increased for all objects. Figure 10 shows the robot successfully avoiding collisions with the ball. Figure 10(c) shows a screenshot of the PC running the machine vision algorithm. The camera image is shown with the edges of the ball highlighted to indicate the size and location of the ball that was found with the vision system.

Conclusions and future work

This paper illustrates a novel algorithm (patent pending) for generating collision-free motion of robotic manipulator arms in real time. The method is based on formulation of

Figure 10 Collision avoidance between robot and red ball sensed with machine vision system



instantaneous inequality constraints on the velocity of the manipulator corresponding to limits imposed by nearby objects or joint constraints. As a result, the problem was reduced to a convex optimization with inequality constraints, which was solved with an interior point method. Experimental implementation of the algorithm verifies the practicality of the algorithm. Performance of the algorithm in these experiments was very good, with adequate stand-off distance between the robot and other objects maintained at all times.

Future work on this algorithm includes integration with a global planner, which could provide additional trajectory modification based on predicted motion of objects in the robot's workspace.

Notes

1 In this paper, all calculations, task space quantities (e.g. velocities) and transformations (e.g. Jacobian matrices) are expressed in a single, fixed, global coordinate frame.

2 Note that only bodies associated with moving robot links are included in R. Bodies associated with fixed portions of the robot (such as the robot base) are included in O.

3 In some cases, it is not possible to formulate the velocity limits and perform the velocity optimization purely in the end-effector space (e.g. redundant manipulators) as it does not have sufficient DOFs. In those cases, it will be necessary to either augment the end-effector space with additional DOFs (e.g. corresponding to redundant motion) or form the limits and perform the velocity optimization in the joint space.

4 Note that the velocity limit of each joint need not be constant. If the instantaneous joint velocity that joint i can attain is time varying (e.g. due to robot dynamics or actuator acceleration limits) then $\dot{q}_{upper_i}$ and $\dot{q}_{lower_i}$ can be implemented as functions of the appropriate variables.

5 The end-effector tool at the end of the robot in Figure 5 is shown for illustration purposes only and is not a geometric primitive.

6 In general cases where object geometry is not known a priori, a system for detecting and localizing unknown 3D objects (e.g. LIDAR, stereo vision) would need to be used.

References

Ali, M., Babu, N. and Varghese, K. (2002), "Offline path planning of cooperative manipulators using co-evolutionary genetic algorithm", *International Symposium on Automation and Robotics in Construction, Washington, DC, USA, September*, pp. 415-24.

Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C. and Vallejo, D. (1998), "OBPRM: an obstacle-based PRM for 3D workspaces", *Proceedings of the Workshop on Algorithmic Foundations of Robotics, Houston, TX*, pp. 155-68.

Bohlin, R. and Kavraki, L. (2000), "Path planning using lazy PRM", paper presented at the IEEE International Conference Robotics and Automation, San Francisco, CA.

Boor, V., Overmars, N.H. and van der Stappen, A.F. (1999), "The gaussian sampling strategy for probabilistic roadmap planners", *IEEE International Conference on Robotics and Automation*, pp. 1018-23.

Brock, O., Kuffner, J. and Xiao, J. (2008), "Motion for manipulation tasks", in Siciliano, B. and Khatib, O. (Eds), *Handbook of Robotics*, Springer, Berlin.

Cortés, J. and Siméon, T. (2004), "Sampling-based motion planning under kinematic loop-closure constraints", *6th International Workshop on Algorithmic Foundations of Robotics, Utrecht, The Netherlands*, pp. 59-74.

Cortés, J., Siméon, T. and Laumond, J.P. (2002), "A random loop generator for planning the motions of closed kinematic chains using PRM methods", paper presented at the IEEE International Conference on Robotics and Automation.

Galicki, M. (2001), "Robot motions in a dynamic environment", *Second Workshop on Robort Motion and Control, October*, pp. 175-80.

Gilliland, M.T. and Gilliland, K.A. (1998), "Method for simultaneous operation of robot welders", US Patent No. 5798627, August.

Han, L. and Amato, N.M. (2000), "A kinematics-based probabilistic roadmap method for closed kinematic chains", *Proceedings of the Workshop on Algorithmic Foundations of Robotics*.

Holleman, C. and Kavraki, L.E. (2000), "A framework for using the workspace medial axis in PRM planners", *IEEE International Conference on Robotics and Automation, San Francisco, CA, USA*, pp. 1408-13.

Hosseini, A., Keskmiri, M. and Marzban, H.R. (2004), "Application of direct methods in optimal path planning of redundant cooperative robots", *IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan*, pp. 3619-24.

Hsu, D., Latombe, J.-C. and Motwani, R. (1999), "Path planning in expansive configuration spaces", *Int. J. Comput. Geom. & Appl.*, Vol. 4, pp. 495-512.

Kagami, S., Kuffner, J., Nishiwaki, K., Okada, K. and Inaba, M. (2003), "Humanoid arm motion planning using stereo vision and RRT search", paper presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan.

Kallmann, M., Aubel, A., Abaci, T. and Thalmann, D. (2003), "Planning collision-free reaching motions for interative object manipulation and grasping", *Eurographics*, Vol. 22 No. 3.

Kato, T. and Nagayama, A. (2001), "Method of avoiding interference of industrial robot", US Patent No. 6212444, April.

Kavraki, L.E., Svestka, P., Latombe, J.-C. and Overmars, M.H. (1996), "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Trans. Robot. & Autom.*, Vol. 12 No. 4, pp. 566-80.

Kuffner, J.J. and LaValle, S.M. (2000), "RRT-connect: an efficient approach to single-query path planning", *Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, USA*, pp. 995-1001.

LaValle, S.M. (1998), "Rapidly-exploring random trees: a new tool for path planning", Technical Report No. 98-11, Computer Science Department, Iowa State University, Ames, IA, October.

LaValle, S.M. and Kuffner, J.J. (2001), "Rapidly-exploring random trees: progress and prospects", in Donald, B.R., Lynch, K.M. and Rus, D. (Eds), *Algorithmic and Computational Robotics: New Directions*, A K Peters, Wellesley, MA, pp. 293-308.

Maciejewski, A. and Klein, C. (1985), "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments", *Int. J. Robot. Res.*, Vol. 4 No. 3, pp. 109-17.
Mazer, E., Ahuactzin, J.M. and Bessiere, P. (1998), "The Ariadne's clew algorithm", *J. Artificial Intell. Res.*, Vol. 9, pp. 295-316.
Pisula, C., Hoff, K., Lin, M. and Manoch, D. (2000), "Randomized path planning for a rigid body based on hardware accelerated Voronoi sampling", *Proceedings of the Workshop on Algorithmic Foundation of Robotics*.
Pollack, S.H. and Hoffman, B.D. (1992), "Method and apparatus for anti-collision and collision protection for multiple robot systems", US Patent No. 5150452, September.
Sánchez, G. and Latombe, J.-C. (2001), "A single-query bi-directional probabilistic roadmap planner with lazy collision checking", paper presented at the International Symposium on Robotics Research.
Sekhavat, S., Svestka, P., Laumond, J.-P. and Overmars, M.H. (1998), "Multilevel path planning for nonholonomic robots using semiholonomic subsystems", *Int. J. Robot. Res.*, Vol. 17, pp. 840-57.
Shaffer, C.A. and Herb, G.M. (1992), "A real-time robot arm collision avoidance system", *IEEE Trans. on Robotics and Automation*, Vol. 8 No. 2, pp. 149-60.
Siméon, T., Laumond, J.P., Cortés, J. and Sahbani, A. (2004), "Manipulation planning with probabilistic roadmaps", *Int. J. Robot. Res.*, Vol. 23 Nos 7/8, pp. 739-46.
Spencer, A., Pryor, M., Kapoor, C. and Tesar, D. (2008), "Collision avoidance techniques for tele-operated and autonomous manipulators in overlapping workspaces", *IEEE International Conference on Robotics and Automation, Pasadena, CA, USA*, pp. 2910-5.
Svestka, P. and Overmars, M.H. (1995), "Coordinated motion planning for multiple car-like robots using probabilistic roadmaps", *IEEE International Conference Robotics and Automation*, pp. 1631-6.
Wilmarth, S.A., Amato, N.M. and Stiller, P.F. (1999), "MAPRM: a probabilistic roadmap planner with sampling on the medial axis of the free space", *IEEE International Conference on Robotics and Automation*, pp. 1024-31.
Yakey, J., LaValle, S.M. and Kavraki, L.E. (2001), "Randomized path planning for linkages with closed kinematic chains", *IEEE Transactions on Robotics and Automation*, Vol. 17 No. 6, pp. 951-8.
Yu, S.N., Lim, S.J., Kang, M.K., Han, C.S. and Kim, S.R. (2007), "Off-line robot palletizing simulator using optimized pattern and trajectory generation algorithm", *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Zurich, September*, pp. 1-6.
Zhang, W., Yuan, J., Li, Q. and Tang, A. (2008), "An automatic collision avoidance approach for remote control of redundant manipulator", *IEEE International Conference on Robotics and Automation, Changsha, June*, pp. 809-14.
Zlajpah, L. and Nemec, B. (2002), "Kinematic control algorithms for on-line obstacle avoidance for redundant manipulators", *IEEE International Conference on Intelligent Robots and Systems, September*, pp. 1898-903.

Further reading

Boyd, S. and Vandenberghe, L. (2004), *Convex Optimization*, Cambridge University Press, Cambridge.

Corresponding author

Paul Bosscher can be contacted at: paul.bosscher@harris.com