REFLEXIVE COLLISION RESPONSE WITH VIRTUAL SKIN
Roadmap Planning Meets Reinforcement Learning

Mikhail Frank, Alexander Förster, and Jürgen Schmidhuber
Dalle Molle Institute for Artificial Intelligence (IDSIA), CH-6928 Manno-Lugano, Switzerland
Faculty of Informatics, Università della Svizzera italiana, CH-6904 Lugano
Dipartimento tecnologie innovative, Scuola universitaria professionale della Svizzera italiana, CH-6928 Manno-Lugano

Keywords: Roadmap Planning: Reinforcement Learning: Collision Avoidance: Robotics Framework

Abstract: Prevalent approaches to motion synthesis for complex robots offer *either* the ability to build up knowledge of feasible actions through *exploration*, *or* the *ability to react* to a changing environment, but *not both*. This work proposes a simple integration of roadmap planning with reflexive collision response, which allows the roadmap representation to be transformed into a Markov Decision Process. Consequently, roadmap planning is extended to changing environments, and the adaptation of the map can be phrased as a reinforcement learning problem. An implementation of the reflexive collision response is provided, such that the reinforcement learning problem can be studied in an applied setting. The feasibility of the software is analyzed in terms of runtime performance, and its functionality is demonstrated on the iCub humanoid robot.

1 INTRODUCTION

Currently available industrial robots are employed to do repetitive work in structured environments, and their highly specialized nature is therefore unproblematic, or even desirable. However the next generation of robot helpers is expected to tackle a much wider variety of applications, working alongside people in homes, schools, hospitals, and offices. The hardware exists already. State-of-the-art humanoid robots such as the NASA/GM Robonaut 2 (Diftler et al., 2011), the Willow Garage Personal Robot 2 (http://www.willowgarage.com), the iCub from the Italian Institute of Technology (IIT) (Metta et al., 2008), and Toyota's Partner Robots (Takagi, 2006; Kusuda, 2008) are impressive machines with many degrees of freedom (DOFs). Physically speaking, they should be capable of doing a much wider variety of jobs than their industrial ancestors. Behaviors however are still programmed manually by experts and the resulting programs are generally engineered to solve a particular instance (or at best a set of related instances) of a task. Consequently, these advanced robots are endowed with relatively few, highly specialized control programs, and their versatility remains quite limited.

In order to realize the potential of modern humanoid robots, especially with respect to service in unstructured, dynamic environments, we must find a way to improve their adaptiveness and exploit the versatility of modern hardware. This will undoubtedly require a broad spectrum of behaviors that are applicable under a variety of environmental constraints/configurations. At the highest level, the versatile agent/controller must solve a variety of different problems by identifying relevant constraints and developing or invoking appropriate behaviors. However we, as engineers and programmers, are not likely to be able to explicitly and accurately predict the wide range of constraints and operating conditions that will be encountered in homes, schools, hospitals, and offices, where the next generation of robots should serve. This is one motivation for the *developmental approach* to robotics, which focuses on systems that adaptively and incrementally build a repertoire of actions and/or behaviors from experience.

To effectively *learn* from experience, an agent/controller must *explore*. However given the fragility of robotic hardware, exploratory behavior can be quite hazardous. Self-collision and collision with fixed objects in the environment usually lead to calibration problems and/or broken components, both of which require time-consuming maintenance efforts that interrupt experimentation. Therefore, if exploration is to be implemented using physical hardware directly, an online monitoring mechanism is required to prevent harmful collisions. It is worthy of note that many biological organisms are equipped with exactly such a system. Skin facilitates collision detection, and reflexes help resolve

dangerous situations, allowing the agent to learn from mistakes without suffering catastrophic damage.

It is also noteworthy that in light of the fragility of robotic hardware, implementing exploratory behavior in simulation is an appealing alternative to the real world. In fact this is the approach favored in most of the path planning literature, however it requires the restrictive assumption that the configuration of the workspace/environment does not change between the simulated synthesis and validation of the behavior and the execution of the behavior by hardware in the real world. In practice, the static environment assumption may not hold, and therefore if motions/behaviors that are planned offline are to be used in any but the most controlled environment, an online monitoring mechanism is still required to prevent harmful collisions.

The remainder of this paper proceeds as follows: In section 2, the motion planning problem is defined. Prevalent approaches to motion planning and reactive collision avoidance are discussed in sections 3 and 4, respectively. Therein, we observe that currently available planning and collision avoidance methods offer *either* the ability to build up knowledge of feasible actions through *exploration*, *or* the *ability to react* to a changing environment, but *not both*. In section 5 we describe our assumptions concerning the motion planning problem for non-static environments, and we propose a very simple way to integrate roadmap planning with reactive collision response. In contrast to available methods, our reflexive collision response approach offers the considerable benefits that:

1. Roadmap planning is extended to non-static environments by transforming the roadmap graph into a Markov Decision Process (MDP).

2. Adaptation of the map to changes in the environment is phrased as a reinforcement learning (RL) problem.

3. Topological changes to the state-action map can be handled within the MDP framework.

4. Provided that unexpected contact with obstacles can be detected physically, the response behavior need not be generated by a computational model of the robot.

In section 6, we discuss implementation issues and in section 7, we conduct a brief feasibility study on the implemented software, which was developed in collaboration with the EU project IM-CLeVeR. Finally, in section 8 we discuss outstanding issues and future work .

2 THE MOTION PLANNING PROBLEM

For an arm (or a humanoid upper body) working in 3D space, the motion planning problem is formalized as follows: The workspace,

$$W = \mathbb{R}^3 \tag{1}$$

contains a robot composed of n links:

$$A(q) = \bigcup_{i=1}^{n} A_i(q) \subset W \tag{2}$$

Each link, A_i, is represented by a semi-algebraic model. The vector of joint positions

$$q \in C = \mathbb{R}^n \tag{3}$$

denotes the *configuration* of the robot A, and kinematic constraints yield a proper functional mapping $q \rightarrow A(q)$. Furthermore, there exists an obstacle region composed of m obstacles:

$$O = \bigcup_{i=1}^{m} O_i \subset W \tag{4}$$

Each obstacle, O_i, is also expressed as a semi-algebraic model.

To find feasible motions, we must disambiguate the feasible region of the configuration space, $C_{free} \subset C$, from the infeasible region:

$$C_{infeasible} = C_O \cup C_{self} \subset C \tag{5}$$

C_O denotes the set of configurations q for which $A(q)$ intersects O:

$$C_O = \{q \in C \mid A(q) \cap O \neq \varnothing\} \subset C \tag{6}$$

In equation 5 C_{self} denotes the set of configurations q for which $A(q)$ is self-intersecting. To handle this, let P denote a set of pairs of indices $(j,k) \in P$ such that $j \neq k$ and $j,k \in \{1,2,...n\}$, which correspond to the poses of two links A_j and A_k that are not allowed to collide. As a matter of convenience, indices of links that share a common joint are typically excluded from P. The set of self colliding poses can then be expressed:

$$C_{self} = \bigcup_{[j,k] \in P} \{q \in C \mid A_j(q) \cap A_k(q) \neq \varnothing\} \subset C \tag{7}$$

The motion planning problem is essentially to find a trajectory $T(q_{initial}, q_{goal}) \subset C_{free}$ that interpolates initial and goal configurations while not intersecting $C_{infeasible}$.

3 PLANNING ALGORITHMS

One approach to solving the above problem is to plan ahead of time. Algorithms that take this approach are generally called "Path Planning" or "Motion Planning" algorithms, as they plan and validate feasible motions, which can later be passed to the robot as reference trajectories. There exists a vast literature on this topic, and we refer the interested reader to the excellent, recent text book by Stephen LaValle (LaValle, 2006). Here we focus on the only class of motion planning algorithms that scale to the high dimensional configuration spaces of humanoid robots.

Sampling based motion planning algorithms probe the configuration space with some sampling scheme. The samples q are mapped to poses $A(q)$, which are in turn used to do collision detection computations, revealing whether $q \in C_{free}$ or $q \in C_{infeasible}$. In this way, feasible motions are constructed piece-by-piece. This can either be done on an as-needed basis (single query planning) (LaValle, 1998; Perez et al., 2011), or the results of queries can be aggregated and stored such that future queries can be fulfilled faster (multiple query planning) (Latombe et al., 1996; Li and Shie, 2007). Following, we consider in broad terms the benefits and drawbacks of these two approaches.

The strength of single query planning algorithms is that they directly and effectively implement exploration by searching for feasible motions through trial and error. Some algorithms, such as RRT (LaValle, 1998), and its many descendants, even offer asymptotic completeness, guaranteeing a solution in the limit of a dense sampling sequence, if one exists. Moreover, since these algorithms answer each query by starting a search from scratch, they can readily adapt to different C_{free} and $C_{infeasible}$ from one call to the next. It is however noteworthy that C_{free} and $C_{infeasible}$ must remain constant during the course of planning. The primary drawback of single query algorithms is their high complexity, which is $O(m^n)$, where m is linear sampling density and n is the dimensionality of the configuration space. A recent state-of-the-art algorithm, BT-RRT (Perez et al., 2011), requires 10 seconds to find a feasible solution to a relatively easy planning problem, wherein two arms (12 DOF) must circumvent the edge of a table. Moreover, the initial solution, the result of stochastic search, is quite circuitous, and BT-RRT requires an additional 125 seconds to smooth the motion by minimizing a cost function in the style of optimal control.

Practically speaking, a latency of tens to hundreds of seconds with respect to a robot's response to commands is often unacceptable. This is the primary motivation for multiple query planning algorithms, which typically utilize a "roadmap" data structure in the form of a graph $G(V,E)$, where $V = \{v_1, v_2, ...v_n\} \in C_{free}$ and E is a set of pairs of indices $(j,k) \in V$ such that $j \neq k$ and $j,k \in \{1,2,...n\}$. With each member of E is associated a verified collision free trajectory $T(E_i) \subset C_{free}$. The roadmap approach reduces each query from a search in $\mathbb{R}^n$ to graph search, which can be carried out by Dijkstra's shortest path algorithm in $O(|V| \cdot log(|V|))$ time. Importantly, the roadmap graph represents a natural crossroads between motion planning as an engineering discipline and the field of artificial intelligence, as it is a special case of an MDP, where "states" are configurations $q \in C_{free}$, "actions" are trajectories $q(t) \subset C_{free}$, and all state transition probabilities are equal to one.

Early versions of the roadmap approach, such as PRM (Latombe et al., 1996), first constructed the map offline, then queried it to move the robot (Latombe et al., 1996). Whereas more recent versions can build the map incrementally on an as-needed basis by extending the current map toward unreachable goal configurations using single query algorithms (Li and Shie, 2007). The ability of roadmap planners to quickly satisfy queries, even for complex robots with many DOFs, makes them an appealing choice for practical application in experimental robotics. Moreover, when roadmaps are constructed incrementally by single query algorithms, the resulting system is one that builds knowledge from experience gained through exploratory behavior. As the roadmap grows over time, it becomes more competent, particularly with respect to navigating the regions of the configuration space in which the planner has operated in the past.

Although the incrementally learned roadmap (Li and Shie, 2007) makes significant steps toward the autonomous development/acquisition of reusable behaviors, it is plagued by one very unrealistic assumption, namely that $T(E_i) \subset C_{free}$ for all time. In other words, neither the obstacle region O nor the robot itself A can change in a way that might have unpredictable consequences with respect to the roadmap $G(V,E)$. Therefore, the roadmap approach implicitly prohibits the robot from grasping objects, which would change A, and even if objects could be moved without grasping, that is anyway prohibited also, as it would change O.

With respect to the goal of adaptively building knowledge of feasible actions from exploratory behavior, we summarize the benefits and drawbacks of single query vs. multiple query motion planning algorithms as follows. Single query planning algorithms such as RRT effectively implement exploration

and can readily adapt to changes in the environment from one query to the next, however they produce circuitous motions that require smoothing, and they are not fast enough to be applied online in practice. Multiple query roadmap based algorithms are quite fast, as they reduce the planning query to graph search, and when the map is constructed incrementally by single query algorithms, the resulting system clearly aggregates knowledge from experience through exploration. The drawback of the incrementally constructed roadmap is that it requires a static environment, which is not practical in the unstructured environments of interest discussed in section 1.

4 REACTIVE COLLISION AVOIDANCE

An alternative to planning feasible actions preemptively is to react to impending collisions as they are detected. This approach was pioneered by Oussama Khatib (Khatib, 1986), under the moniker "real time obstacle avoidance". However, it has become widely known as the "potential field" approach, and is formulated as follows for non-redundant manipulators in terms of the notation from section 2 above: Consider a point $x \in A_i(q) \subset W$, and let

$$U_O(x) = \sum_{i=1}^{m} U_{Oi}(x) \tag{8}$$

be a potential field function, which represents the influence of m obstacles in the workspace on x. Let each U_i be a continuous, differentiable function, defined with respect to an obstacle region $O_i \subset W$, such that U_i is at a maximum in the neighborhood of the boundary of O_i and goes to zero far from O_i. Khatib suggests:

$$U_{Oi}(x) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho(x)} - \frac{1}{\rho_0})^2 & : \rho(x) \leq \rho_0 \\ 0 & : \rho(x) > \rho_0 \end{cases} \tag{9}$$

where ρ is the shortest distance from x to O_i, and ρ_o controls the range of the potential field's influence. The defining characteristic of the 'potential field' approach is that the robot is controlled such that x descends the gradient of the potential field, U_O. The control input is computed as follows: The influence of the potential field U_O on the robot is first computed as a force f that acts on the robot at x:

$$f = -\frac{\partial U_O}{\partial x} = -\sum_{i=1}^{m} \frac{\partial U_{Oi}}{\partial x} \tag{10}$$

The force f is then transformed via the Jacobian matrix to yield joint torques τ.

$$\tau(x) = J^T(q)\Lambda(x)f(x) \tag{11}$$

Here, $\Lambda(x)$ is a quadratic form, a "kinetic energy matrix" that captures the inertial properties of the end effector.

In a similar fashion, another potential field U_G can be defined to attract the end effector to the goal position, and a third can be defined directly in the configuration space to push q away from joint limits.

Khatib's approach is very effective with respect to quickly generating evasive motions to keep the manipulator away from obstacles, however it offers a very brittle solution to the motion planning problem presented in section 2. It is often the case that the superposition of potential functions creates local minima, attractors in which the controller gets stuck.

Subsequent work has reformulated the above to improve the robustness of the implied global plan. For example, (Kim and Khosla, 1992) uses harmonic potential functions, which guarantee that no local minimum exists other than the global minimum, or alternatively, that point x above, if treated as a point robot, will always be pushed to the goal from any initial condition. Still though there may exist structural local minima, configurations where non-point robots will not be able to proceed although they are being forced by the potential gradient.

Another reformulation is known as "attractor dynamics" (Schoner and Dose, 1992), wherein the robot does not descend the gradient of the potential function, but rather moves with constant velocity, adjusting its heading according to a dynamical system that steers toward the goal, but away from obstacles that lie in the robot's path. The method was developed for mobile robotics, however it has also been applied to manipulators in a manner similar to the above (Iossifidis and Schoner, 2004; Iossifidis and Schoner, 2006). And again, although "attractor dynamics" improves robustness over the original "potential field" approach, this time by keeping the state of the robot in the neighborhood of a stable attractor, it is still a heuristic planner that bases decisions on local information only, and it can therefore get stuck.

A third reformulation called "Elastic Strips" (Brock and Khatib, 2000) combines the local reactivity of the potential field approach with the more global framework of a roadmap planner. The edges of the roadmap graph are trajectories that are parameterized in such a way as to be deformable under the influence of a potential field. Again, this approach does improve robustness with respect to global planning, however it still suffers from structural local minima, and the elastic graph edges may not be able to circumvent certain obstacles. Failure to circumvent

an obstacle while traversing an elastic edge causees the roadmap planner to fail exactly as its non-elastic counterpart would. Since re-planning is limited to local deformation of the current trajectory, the approach cannot cope with topological changes in the roadmap. Worse yet, after failure, the configuration of the robot does not lie on any of the nodes of the roadmap graph, nor on its undeformed edges. Therefore, a single query planner must be invoked to find a feasible path back to a node of the roadmap graph, and this could be problematic in a dynamic environment.

All approaches based on the potential field idea use local information from the workspace, and transform it into motor commands according to some heuristics. It is therefore not surprising that these approaches excel at fast, reactive obstacle avoidance while they have trouble with global planning tasks. Accordingly, potential field approaches have become popular in the context of safety and human-robot interaction (De Santis et al., 2007; Dietrich et al., 2011; Stasse et al., 2008; Sugiura et al., 2007). In these applications a potential filed U_G, to attract the robot to the goal, is not defined. Instead, in the absence of influence from obstacles and joint limits, some other planner/controller system is allowed to operate freely.

Once again, to relate this discussion back to the topic of adaptively building knowledge of feasible actions from exploratory behavior, we make the following observation: Reactive collision avoidance is by definition quite contrary to exploration, however it is nonetheless a highly desirable, even necessary component of any learning system that hopes to differentiate feasible sequences of actions from infeasible ones in a changing environment.

5 APPROACH

The motion planning literature provides many algorithms that perform well in static environments. On the other hand, the control literature provides methods for quickly reacting to avoid collisions in dynamic environments. However, to the best of our knowledge, surprisingly little attention has been paid to the pertinent question, "How can we plan and execute motions robustly in environments that change sporadically but drastically?"

Consider for a moment a roadmap, $G(V,E)$ that covers the configuration space of a humanoid upper-body reasonably well. Such a graph will undoubtedly allow us to move the robot around while avoiding self-collision, and that will remain true regardless of how the environment changes. Therefore, if we were to say, put a table in front of the robot, it seems likely that we would prefer to adapt G to the new constraints, rather than throw it away and start from scratch. The addition of the table however, has certainly invalidated many edges in G, as $T(E_i) \not\subset C_{free} \forall i \in \{a,b,c,...\}$, drastically changing its topology.

Before the addition of the table, G had been a valid MDP, with states V and actions E, and all state transition probabilities equal to one. After the addition of the table, the MDP became invalid. The probability of transitioning from V_a to V_b along $T(E_i)$ had been equal to one, now it is equal to zero. Therefore, the state transition probabilities no longer sum to one for the actions E_i in states V_a. In order to repair the MDP, we require an alternative state transition, in fact one that can actually be implemented for a real physical robot.

We propose the simplest possible solution, should the transition from V_a to V_b along $T(E_i)$ fail, retreat to V_a. Now, every state/action combination in G has two possible state transitions associated with it: $V_a \rightarrow V_b$ and $V_a \rightarrow V_a$. And the state transition probabilities can be maintained such that they always sum to one. With the validity of the MDP restored, we can treat discovery and avoidance of the table as a RL problem.

This very simple behavior, which is inspired by the reflexive response to surprise that is easy to observe in humans, has the following important consequences:

1. We are able to generalize roadmap planning to non-static environments, which is accomplished by adopting probabilistic state transitions and casting the roadmap graph into an MDP.
2. Within the MDP, the discovery and avoidance of novel objects/obstacles can be phrased as a reinforcement learning problem.
3. In contrast (Brock and Khatib, 2000), which also generalizes roadmap planning to non-static environments, our method is able to cope with changes to the topology of the roadmap. This is also a consequence of adopting the probabilistic framework.
4. Provided that unexpected contact with obstacles can be detected physically, the response behavior need not be generated by a computational model of the robot. This is to say that the only information required to facilitate the collision response is the history of the trajectory $T(E_i)$.

6 IMPLEMENTATION

We have implemented the reflexive collision response behavior described in section 5 within a soft-

ware framework that we call "Virtual Skin". The source code is available through the software repository of the EU project IM-CLeVeR. In this section we will describe the software at a high level, then discuss design decisions and implementation issues.

Virtual Skin is a module for YARP. A popular open source "robotics platform", YARP is essentially middleware that allows roboticists to create distributed systems of loosely coupled modules, without having to worry about the details of network protocols and operating systems. As a YARP module, Virtual Skin can easily be used with any robot for which YARP drivers have been implemented. The software is primarily intended to facilitate research on the RL problem formulated in section 5, in an applied setting. We have tried in our implementation to maximize its generality by focusing on transparency and modularity. The software consists of three primary components:

1. A kinematic model of the robot/workspace system that is dynamically updated in real-time by sensory/state information from the hardware.
2. A port filter that allows Virtual Skin to act as a proxy between an arbitrary planning/control module and the robot.
3. A collision response behavior.

The trio of components works as follows: The planner/controller, which can be any program capable of controlling the robot, connects to Virtual Skin's proxy versions of the robot's motor control ports. The proxy is completely transparent, in fact the planner/controller does not even know that it is not connected to the real robot. The robot responds to control commands passed through the proxy, and asynchronous messages are streamed back across the network, communicating the state of the hardware. These messages are received by Virtual Skin, and they are used to update the kinematic model. Collision detection computations are done on the model in real-time, and when an unwanted collision is detected, the port filter cuts off the controlling program from the robot. The robot is then stopped, and the collision response behavior is executed. When the behavior finishes executing, control is restored to the planner/controller. The architecture of the implementation is shown graphically in figure 1.

It is important to mention that Virtual Skin is not intended to be used as a foolproof safety mechanism in the sense that it does not provide guarantees that collisions will be prevented regardless of the robot's inertial state. Instead it is intended to simulate full body tactile feedback (hence the name Virtual Skin) to facilitate research on embodied AI. We therefore

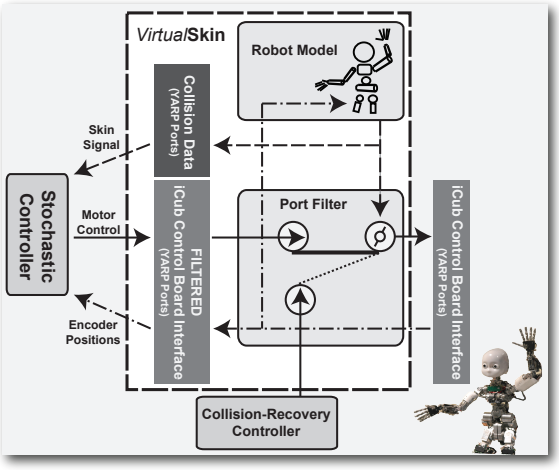

Figure 1: **Simplified Architecture** - Virtual Skin wraps the iCub robot's motor control interface (right). A stochastic, exploratory control program is connected to a proxy motor control interface (left). A port filter connects the motor control interface to its twin (middle). The kinematic robot model (top), driven by streams of motor encoder positions from the robot, does collision detection and regulates the state of the port filter. When the model detects collision, the port filter cuts off the stochastic controller and invokes an alternative user-defined controller (bottom) to recover from the dangerous configuration.

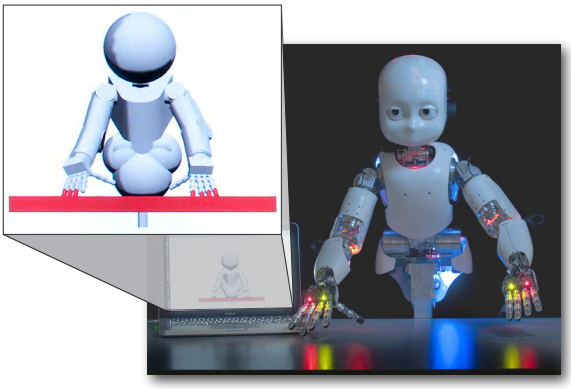

Figure 2: Virtual Skin detects impending collision between the iCub humanoid robot and a table. Darkened (red) geometries in the model (left) are colliding.

encourage *careful* exploratory behavior. That is to say that obstacles should be modeled with generous bounding volumes, and motor velocities should selected in accordance with the safety margin afforded by nearby bounding volumes.

6.1 Kinematic Model

Although the reflexive behavior discussed above, which motivates the implementation of Virtual Skin, does not require a model of the robot to compute the

collision response trajectory, we do need a way to detect impending collisions. Since neither the iCub, nor the Katana (nor any other available robot we know of) is completely wrapped in skin-like tactile sensors, a computational model is required to represent the robot/environment system.

As Virtual Skin is to operate online in realtime, we designed it around a kinematic robot model, the configuration of which is supplied by streams of motor encoder positions from hardware. To minimize the computational burden, we model geometries as unions of geometric primitives, currently supported are spheres, cylinders, and boxes. This has the added benefit that we can easily model non-convex objects. The model also records its history in a large circular buffer that can be annotated in realtime by sending messages to a particular RPC port, and queried as necessary.

The robot model is specified via an XML configuration file. The kinematic constraints are expressed using "Zero Position Displacement Notation" (Gupta, 1986), which is significantly less complex and more intuitive than the popular Denavit-Hartenburg convention. And geometries can be attached to the model by specifying them directly in the XML hierarchy.

The configuration of objects in the world can be specified similarly by another XML configuration file. This is the method used to specify the table in section 7.2. Alternatively, objects in the robot's environment can be added, transformed, and destroyed dynamically at any time via a remote procedure call (RPC) interface. This allows the model to be maintained in real time by a computer vision or other sensory module if one is available.

Collision detection in Virtual Skin is handled by the Software Library for Interference Detection (SOLID) (van den Bergen, 2004), which is highly optimized and supports primitives, Minkowski sums, and polyhedra. To keep the collision detection computations as efficient as possible, we employ hierarchical pruning to reduce the number of pairs of geometries to be tested within the kinematic tree. Moreover, objects in the robot's environment are not tested against each other, so the approximate complexity of collision detection computations is $O(n^2 \cdot m)$ where n is the number of objects in the robot model and m is the number of objects in the environment.

The model is implemented as a static library that can be compiled separately, independent of YARP. Thus in addition to being an integral part of Virtual Skin, the kinematic model is a stand-alone planning environment that can be reused easily in other projects/modules.

6.2 Port Filter

In order to enforce modularity and allow Virtual Skin to function with the widest possible range of planners/controllers, we have embedded it in a port filter that acts as a proxy. The port filter is written entirely in the YARP API, and it filters a "ControlBoardInterface" object which consists of several YARP ports (see YARP documentation). The most important consequence of this design is that the workings of Virtual Skin are completely transparent to the client program, meaning that *any* program that can connect to a robot and control it can also do so through Virtual Skin.

6.3 Reflex Behavior

When the reflex behavior is triggered, the configuration history of the robot is first queried from the robot model. Then, the poses from the history $q_i \in \{q_t, q_{t-1}, q_{t-2}, ...q_{t-n}\}$ are sent to the robot as sequential position move commands, causing the robot to retrace its steps.

The model continues doing collision detection and adding to the history. When the reflex behavior starts, the poses being logged are colliding. In order to avoid that colliding poses remain in the buffer, as soon as the model is no longer colliding, the pose buffer is reinitialized with the non-colliding pose.

Recall that the history can be annotated via an RPC port. If we desire that the reflexive retreat should stop at a particular configuration, as described in section 5, then the planner/controller must annotate "waypoints" in the history, which correspond, for example, to vertices in a roadmap graph. The reflexive behavior provided in the Virtual Skin distribution, retraces the history until either an annotated waypoint or the end of the buffer is encountered.

Lastly, the reader should be aware that the modular nature of the Virtual Skin implementation makes this collision response behavior easy to replace with arbitrary code.

7 Feasibility Study

In this section, we discuss the results of two experiments running Virtual Skin. The first experiment analyzes the scalability of the software with respect to the complexity of robot models and environments. The second, verifies that Virtual Skin adequately protects the iCub humanoid from harm while under the control of a purely random exploratory control policy, which amounts to motor babbling using all DOFs of the iCub's upper-body.

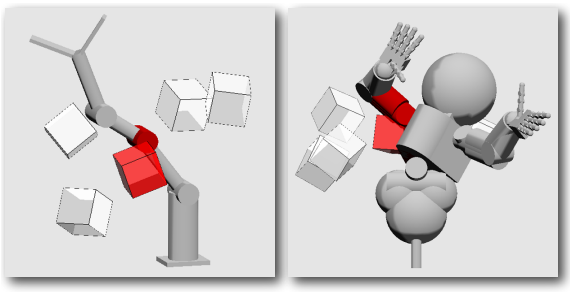

Figure 3: **Kinematic Robot Models** - The Katana arm (left) and the iCub humanoid (right) collide with random obstacles. Darkened (red) geometries are colliding.

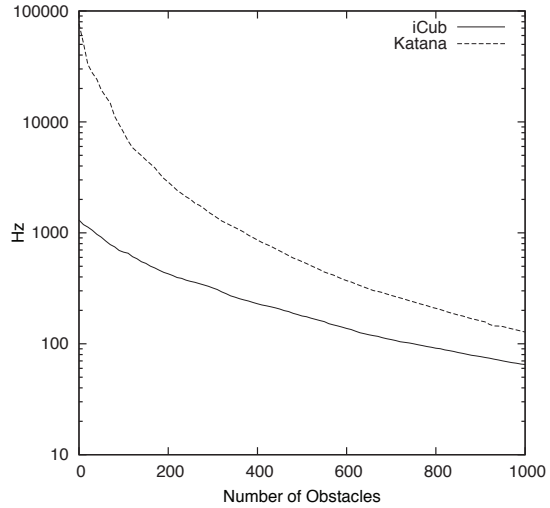

Figure 4: **Scalability of Virtual Skin** - A simple robot model (Katana - 9 primitives) is compared with a complex one (iCub - 129 primitives). The curves show the number of times collision detection can be computed per second (y-axis) given a particular number of obstacles modeled in the robot's environment (x-axis).

7.1 Runtime Performance - Scalability

This experiment analyzes the performance of the kinematic model within Virtual Skin, as the number of modeled objects in the robot's environment grows. The chosen experimental setup is a path planning scenario, wherein the kinematic/geometric model (figure 1 - top) is run offline without a port filter (figure 1 - middle) or a physical robot (or simulator) to supply streams of configurations. Instead, each joint is driven through its entire range of motion in a series of discrete steps by a simple oscillator. The motion continues uninterrupted (collision events to not stop the model or alter the behavior of the oscillators) for several minutes as obstacles are randomly added to the robot's workspace. The obstacles chosen for this experiment are boxes, as they are the most computationally expensive of the currently supported primitives.

The experiment was run on a dual-core 2.4GHz laptop with 4GB of memory, and two different robot models were used; a 6 DOF Katana arm (9 primitives), and the 41 DOF upper-body of an iCub humanoid robot (129 primitives). Figure 3 shows snapshots from the early stages of the running experiments, and the results are plotted in figure 4. Based on this experiment, we make the following two claims:

1. For simple arms in simple environments, Virtual Skin can keep pace with even the fastest control frequencies encountered in industrial practice.
2. Virtual Skin can compute hundreds of poses per second for a humanoid with hundreds of obstacles in the environment, which is adequate for most applications in developmental robotics.

7.2 Motor Babbling on The iCub Humanoid Robot

The software architecture employed in this experiment is shown in figure 1. The stochastic control policy is simply randomly generated configurations sent to the robot through VIrtual Skin as position move commands. All joints on the iCub upper-body except those in the hands are controlled. When invoked, the collision recovery controller, tracks a trajectory comprised of the previous n states in the robot's history, returning control as soon as the robot model is in a non-colliding configuration. The modeled environment consists of a table, as pictured in figures 2 and 5. The experiment has been run for approximately two hours over several trials of 5 to 20 minutes each with joint velocity limited to a moderate 20% of maximum. Video excerpts of some of these trials are available as supplemental material on the authors' web sites.

As expected, the stochastic controller quickly produced many motions, which uninterrupted would have resulted in destructive collisions. However Virtual Skin effectively prevented all of them. Included were several commonly occurring self collisions, such as elbow vs. hip, and upper-arm vs. chest, as well as many collisions between the hands/forearms and the table.

The experiment demonstrates that the Virtual Skin framework is a feasible tool to facilitate study the RL problem formulated in section 5 in an applied setting on a real humanoid robot.

8 Discussion

We motivated the work presented here around the need for modern, advanced robots such as humanoids

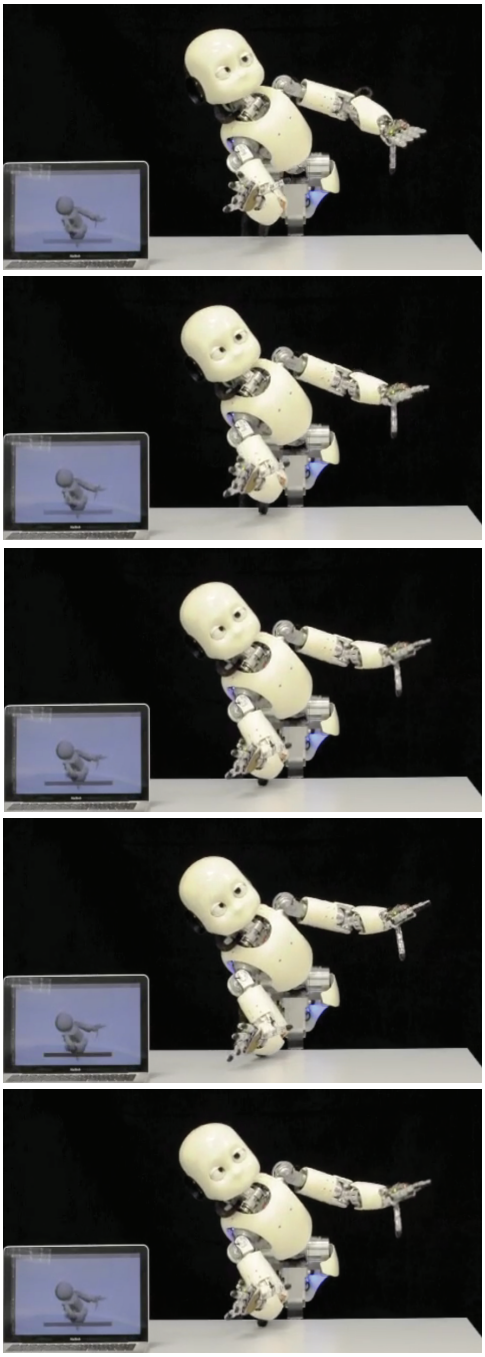

Figure 5: Time lapse images of Virtual Skin detecting impending collision with the table and invoking reflexive response. Collision is detected in frame 4, and response begins in frame 5.

to serve in challenging, unstructured environments such as homes, schools, hospitals, and offices. Since engineers and programmers are not likely to be able to adequately describe the wide range of constraints and operating conditions that will be encountered in such surroundings, we argued in favor of systems that adaptively and incrementally build a repertoire of actions and/or behaviors from experience.

Incrementally learned roadmap planners (Li and Shie, 2007) are an appealing approach to the problem, as they build up knowledge of feasible actions from exploratory behavior, and they also scale to the large configuration spaces of humanoid robots. However, they require us to assume a static environment, which is unrealistic in the environments listed above.

We proposed a simple integration of roadmap planning with reflexive collision response, allowing the roadmap representation to be transformed into a Markov Decision Process. Roadmap planning is thereby extended to changing environments, and the adaptation of the map can be phrased as a reinforcement learning problem.

We implemented the proposed reflexive collision response, and discussed its design. The feasibility of the software was also verified in experiments with the Katana Manipulator and the iCub humanoid upper-body.

Still to do is to analyze the robustness of our reflexive response behavior in dynamic environments, and to study the proposed RL problem as it applies to the iCub humanoid using the Virtual Skin framework.

REFERENCES

Brock, O. and Khatib, O. (2000). Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 550–555. IEEE.

De Santis, A., Albu-Schaffer, A., Ott, C., Siciliano, B., and Hirzinger, G. (2007). The skeleton algorithm for self-collision avoidance of a humanoid manipulator. In *Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on*, pages 1–6. IEEE.

Dietrich, A., Wimbock, T., Taubig, H., Albu-Schaffer, A., and Hirzinger, G. (2011). Extensions to reactive self-collision avoidance for torque and position controlled humanoids. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3455–3462. IEEE.

Diftler, M., Mehling, J., Abdallah, M., Radford, N., Bridgwater, L., Sanders, A., Askew, S., Linn, D., Yamokoski, J., Permenter, F., Hargrave, B., Platt, R., Savely, R., and Ambrose, R. (2011). Robonaut 2: The first humanoid robot in space. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

Gupta, K. (1986). Kinematic analysis of manipulators using the zero reference position description. *The International Journal of Robotics Research*, 5(2):5.

Iossifidis, I. and Schoner, G. (2004). Autonomous reaching and obstacle avoidance with the anthropomorphic arm of a robotic assistant using the attractor dynamics approach. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4295–4300. IEEE.

Iossifidis, I. and Schoner, G. (2006). Reaching with a redundant anthropomorphic robot arm using attractor dynamics. *VDI BERICHTE*, 1956:45.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90.

Kim, J. and Khosla, P. (1992). Real-time obstacle avoidance using harmonic potential functions. *Robotics and Automation, IEEE Transactions on*, 8(3):338–349.

Kusuda, Y. (2008). Toyota's violin-playing robot. *Industrial Robot: An International Journal*, 35(6):504–506.

Latombe, J., Kavraki, L., Svestka, P., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

LaValle, S. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa State University.

LaValle, S. (2006). *Planning algorithms*. Cambridge Univ Pr.

Li, T. and Shie, Y. (2007). An incremental learning approach to motion planning with roadmap management. *Journal of Information Science and Engineering*, 23(2):525–538.

Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: Yet Another Robot Platform. *International Journal of Advanced Robotic Systems*, 3(1).

Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, pages 50–56. ACM.

Perez, A., Karaman, S., Shkolnik, A., Frazzoli, E., Teller, S., and Walter, M. (2011). Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4307–4313. IEEE.

Schoner, G. and Dose, M. (1992). A dynamical systems approach to task-level system integration used to plan and control autonomous vehicle motion. *Robotics and Autonomous Systems*, 10(4):253–267.

Stasse, O., Escande, A., Mansard, N., Miossec, S., Evrard, P., and Kheddar, A. (2008). Real-time (self)-collision avoidance task on a hrp-2 humanoid robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3200–3205. IEEE.

Sugiura, H., Gienger, M., Janssen, H., and Goerick, C. (2007). Real-time collision avoidance with whole body motion control for humanoid robots. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2053–2058. IEEE.

Takagi, S. (2006). Toyota partner robots. *Journal of the Robotics Society of Japan*, 24(2):62.

van den Bergen, G. (2004). *Collision detection in interactive 3D environments*. Morgan Kaufmann.