SOFTWARE AND ALGORITHMS FOR MOTION PLANNING IN VIRTUAL ENVIRONMENTS

Omar A.A. Orqueda

Grupo de Robótica y Simulación - Departamento de Electrotecnia
Universidad Tecnológica Nacional, Facultad Regional Bahía Blanca
11 de abril 461, B8000LMI Bahía Blanca, Argentina
e-mail: orqueda@ieee.org, web page: http://www.frbb.utn.edu.ar

Key Words: Motion Planning, Virtual Environments, Nonholonomic Planning.

Abstract. Motion planning in virtual environments is an exciting research field that has been studied during last three decades. It has evolved from the simplest problem of planning a 2D path of a mobile robot using geometric methods, to the very difficult problem of planning 3D motions of multiple robots with many degrees of freedom, or, more surprisingly, graphic animation, surgical planning, computational biology, or automatic sensing.

This work presents a generic software package, MPSLab, that is aimed for motion planning and simulation of robots with multiple degrees of freedom (dof) in virtual environments. This package can represent and simulate multiple robots with several dofs moving among fixed polyhedral obstacles. The motion planning algorithms implemented in this packages are probabilistic RRT-based algorithms, some of them developed by the author of this work. With the motion planning algorithms, problems involving holonomic, nonholonomic, and generic constraints can be solved. To implement these algorithms, the package possesses differential-equation solvers, optimization algorithms, collision detection/distance computation libraries, and fast computation of artificial potential fields.
1 INTRODUCTION

The Motion Planning Problem (MPP)\textsuperscript{1} have attracted the attention of many researchers during the last three decades. With the increase of computational power capability of today’s computer, there exists the possibility of giving more autonomy to any system, if not full autonomy.

In its simpler form, the MPP consists in finding a collision-free path for a robot among rigid static obstacles connecting an initial state with a final desired one. The basic MPP has been traditionally solved using geometric approaches, but this problem is computationally hard for robots with several degrees of freedom, i.e., more than five.

In 1969, Nilsson described a mobile robot system with motion planning capabilities using a visibility graph method combined with the $A^*$ search algorithm to find the shortest collision-free path.\textsuperscript{1} The robot was represented by a point amidst polygonal obstacles. Udupa solved collision avoidance by introducing the idea of shrinking a robot to a point.\textsuperscript{2} In 1979, Lozano-Pérez and Wesley introduced the concept of configuration space and solved the MPP for polygonal or polyhedral robots translating among polygonal or polyhedral obstacles.\textsuperscript{3,4} This year, Reif showed that path planning for a 3D linkage made of polyhedral links is PSPACE-hard and that there is strong evidence that any complete planner\textsuperscript{2} requires time that grows exponentially with the number of dof of the robot to find a solution.\textsuperscript{5-7}

The complexity of complete path planners and their lack of robustness have motivated the development of heuristic planners. Two of the most popular approaches are approximate cell decomposition and artificial potential field.\textsuperscript{7-11} In the approximate cell decomposition, the free space is represented by a collection of simple cells, whereas artificial potential fields are used to move the robot under the local effects of repulsive fields associated to obstacles and the attractive field pulling toward the goal.

Both approaches are resolution-complete\textsuperscript{3} and can solve complex path planning problems in 2D and 3D configuration spaces, but none of these approaches extends well to robots with more than 3 dofs, because the number of cells becomes too large, or the potential field can stuck the robot into local minima before reaching the goal position.

In 1991, a randomized planner which alternated down motions to track the negated gradient of a potential field and random motions to escape local minima was introduced.\textsuperscript{12} This planner was able to solve complex path planning problems for many-dof robots. Because the problems of local minima caused by a deterministic potential field persist, another type of randomized planner was developed. This planner consists of sampling the configuration space at random and connecting the samples in free space by local paths,

\textsuperscript{1}Motion planning is a general term that refers to either path planning or trajectory planning.

\textsuperscript{2}A complete, or exact, path planner is one which returns a collision-free path whenever one exists, and indicates that no such path exists otherwise.

\textsuperscript{3}A resolution-complete planner is one which finds a path if it exists and if the resolution parameter, the size of the smallest cells or the resolution of the grid, is set fine enough.
typically straight paths, thus creating a probabilistic roadmap (PRM)\(^4\)\(^{13-16}\) Experiments with PRM planners have been quite successful, showing that they are both fast and reliable even with robots with many dofs. Formal analysis supports this experimental observation by showing that PRM planning is complete in a probabilistic sense\(^5\)\(^{17}\) PRM planners are also robust to floating-point approximations and easy to implement. A number of variants applying different sampling strategies have been recently developed.\(^{17-21}\)

Most of PRM-based algorithms are checked for collision using fast collision checkers,\(^{22-34}\) because, despite of some attempts to compute an explicit representation of the free space,\(^32\) it has been shown that this computation is computationally prohibitive.

With the introduction of differential constraints, a challenging problem emerges that involves both nonlinear control and traditional path planning issues. This problem is often referred to as nonholonomic planning,\(^{36-38}\) or kinodynamic planning.\(^{40-42}\) The design of a roadmap-based algorithm is more challenging because of the increased difficulty of connecting pairs of states in the presence of the constraints, also referred to as the steering problem.\(^36\) Several randomized approaches to kinodynamic planning appeared based on Rapidly-exploring Random Trees for static and time-varying environments, nonholonomic and dynamic constraints, optimization criteria, moving obstacles, and/or flexible robots.\(^{43-46}\)

In this work, it is presented a novel software package for motion planning and simulation of robots with multiple degrees of freedom (dof) in virtual environments, MPSLab. This package can represent and simulate multiple robots with several dofs moving between fixed polyhedral obstacles. The motion planning algorithms implemented in this packages are probabilistic RRT-based algorithms, some of them developed by the author of this work.\(^{11,43,44,46,47}\) This motion planning algorithms can solve MPPs involving holonomic, nonholonomic, and generic constraints. It possess several differential-equation solvers and optimization algorithms, collision detection/distance computation libraries, and fast computation of artificial potential fields.

The work is organized a follows: In Section 2, some mathematical expressions and notions on constraints are resumed. In Section 3, the Motion Planning Problem is formalized. In Section 4, the main characteristics of the software package MPSLab is introduced. In Section 5, new fields of application are briefly discussed. Conclusions, future work, and references close the work.

### 2 MATHEMATICAL PRELIMINARIES

Let \(\mathcal{W}\) denote the world space, or the physical space in which robots and obstacles exist. Using the configuration space concept, a robot, \(\mathcal{A}\), is represented as a point, \(q\), called a configuration, in a parameter space encoding the robot’s dofs, the configuration space,

\(^4\)A **roadmap** is a network of collision-free paths that captures the configuration-space topology, and is generated by preprocessing the configuration space independently of any initial-goal query.

\(^5\)Under reasonable geometric assumptions on the free space, the probability that a PRM planner fails to find a path while one exists decreases exponentially toward 0 with the number of samples.
\( \mathcal{C} \). The obstacles in the workspace, \( \mathcal{B}_i, i = 1, 2, \ldots \), map as forbidden regions into the configuration space, \( \mathcal{CB}_i = \{ q \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{B}_i \neq \emptyset \} \), or \( \mathcal{C} \)-obstacle. The union of all the \( \mathcal{C} \)-obstacles in \( \mathcal{C} \) is called the \( \mathcal{C} \)-obstacle region. The complement of the \( \mathcal{C} \)-obstacle region is the free space, \( \mathcal{C}_{\text{free}} \). Path planning for a dimensioned robot is thus reduced to the problem of planning a path for a point in a space that has as many dimensions as the robot has does.\(^7\)

Let \( \mathcal{C}_{\text{valid}} \subseteq \mathcal{C} \) denote the subset of valid postures of the robot \( \mathcal{A} \). Let \( \tau : \mathcal{I} \longrightarrow \mathcal{C} \) denote a motion trajectory or path for \( \mathcal{A} \) expressed as a function of time, where \( \mathcal{I} \) is an interval \([t_0, t_1]\). \( \tau(t) \) represents the configuration \( q \) of \( \mathcal{A} \) at time \( t \), with \( t \in \mathcal{I} \). A trajectory \( \tau \) is said to be collision-free if \( \tau(t) \in \mathcal{C}_{\text{free}} \) for all \( t \in \mathcal{I} \).

The differential equations describing a nonlinear system affine in control that represents the motion of \( \mathcal{A} \) can be described in state space form by\(^6\):

\[
\dot{x} = f_0(x) + \sum_{i=1}^{m} f_i(x) u_i = f_0(x) + f(x) u
\]

(1)

In the general case, the state variable \( x \) evolves over a real-analytic, connected manifold \( \mathcal{M} \in \mathbb{R}^n \), with \( q \in \mathcal{C} \subseteq \mathcal{M} \), the controls \( u_i \) are assumed to lie in the Sobolev space \( H = H^k[0, T] \), while the \( f_i \) are smooth \( C^\infty(\mathcal{M}) \) vector fields.

System motion can be subjected to constraints that may arise from the structure of the mechanism, or from the way in which it is actuated and controlled. Constraints may be expressed as equalities or inequalities, bilateral or unilateral constraints, respectively, and they may explicitly depend on time or not, rheonomic or scleronomic constraints.

Motion restrictions that may be put in the form

\[
h_i(q) = 0, \quad i = 1, \ldots, k < n, \quad h_i : \mathcal{Q} \rightarrow \mathbb{R}, \text{ smooth}
\]

(2)

are called holonomic constraints. Its effect is to confine the attainable system configurations to an \((n-k)\)-dimensional smooth submanifold of \( \mathcal{Q} \). The problem is solved by defining \( n-k \) new coordinates on the restricted submanifold that characterize the actual degrees of freedom of the system. For simulation purposes, a Differential-Algebraic Equation (DAE) system solver is used.

System constraints whose expression involves generalized coordinates and velocities in the form

\[
a_i(q, \dot{q}) = 0, \quad i = 1, \ldots, k < n, \quad a_i : \mathcal{Q} \rightarrow \mathbb{R}^n, \text{ smooth}
\]

are referred to as kinematic constraints. These will limit the admissible motions of the system by restricting the set of generalized velocities that can be attained at a given configuration. In mechanics, such constraints are usually encountered in the Pfaffian form

\[
a^T_i(q) \dot{q} = 0, \quad i = 1, \ldots, k < n, \quad \text{or} \quad A^T(q) \dot{q} = 0,
\]

(3)

\(^6\)Not every robot has an associated differential model. This cases are addressed by equating the differentials to zero, and computing the next state by interpolation. 

2358
It may happen that the kinematic constraints (3) are not integrable, i.e., cannot be put in the form (2). In this case, the constraints and the mechanical system itself are called nonholonomic.

3 MOTION PLANNING PROBLEM

Given $q_{\text{init}} \in C_{\text{valid}}$ and $q_{\text{goal}} \in C_{\text{valid}}$, the main goal of the planning algorithm is to compute a continuous motion planning trajectory $\tau$, or equivalently, a control law $u(t)$, such that $\forall t \in [t_0, t_1], \tau(t) \in C_{\text{valid}}$, and $\tau(t_0) = q_{\text{init}}$ and $\tau(t_1) = q_{\text{goal}}$.

The idea behind the basic Probabilistic Roadmap Method (PRM) is to represent and to capture the connectivity of $C_{\text{free}}$ by a random network, a roadmap, whose nodes and edges correspond to randomly selected configurations and path segments, respectively. In a preprocessing step, or a learning phase, a large number of points are distributed uniformly at random in $C$, and those found to be in $C_{\text{free}}$ are retained as nodes in the roadmap. A local planner is then used to find paths between each pair of nodes that are sufficiently close together. If the planner succeeds in finding a path between two nodes, they are connected by an edge in the roadmap. In the query phase, the user specified start and goal configurations are connected to the roadmap by the local planner. Then the roadmap is searched for a shortest path between the given points, Fig. 1.

A strategy for building a Basic-PRM can be summarized as follows, Alg. 1: Let $G$ be a graph composed by a set $E$ of edges that connect two configurations by a free path. The path represents an edge, the set $V$ contains all the configuration nodes of the graph. A configuration $q_{\text{rand}}$ is randomly chosen each iteration with uniform distribution. If this configuration belongs to $C_{\text{free}}$ then it is added to $V$. Then it is tested the connection with its neighbors. The neighbors of this configuration are defined as the nodes whose distance to the configuration $q_{\text{rand}}$ is less than $d_{\text{max}}$.

The Basic-PRM in its original form, is a purely geometric planner. There exist several modification to take into account system constraints and/or dynamics, but none of them are generic solution for rapid development: a careful study of the resulting trajectories of a particular system must be obtained.
Algorithm 1 PRM-Basic

PRM_BASIC ($q_{init}$)
1: $V \rightarrow \emptyset$; $E \rightarrow \emptyset$; $n_{nodes} \rightarrow 0$;
2: while $n_{nodes} < N_{max}$ do
3:     $q_{rand} \leftarrow \text{RANDOM\_CONFIG}()$;
4:     if $q_{rand} \in \mathcal{C}_{free}$ then
5:         $V \rightarrow V \cup \{q_{rand}\}$;
6:         $n_{nodes} \leftarrow n_{nodes} + 1$;
7:     for all $v \in V_{c}$ in increasing order of distance do
8:         if $\neg\text{connected} (q_{rand}, v) \& \mathcal{L}(q_{rand}, v) \in \mathcal{C}_{free}$ then
9:             $E \leftarrow E \cup \{(q_{rand}, v)\}$;
10:         end if
11:     end for
12:     end if
13: end while

The Rapidly-Exploring Random Tree (RRT)\textsuperscript{51} is an exploration algorithm for quickly searching high-dimensional spaces that have both global constraints, arising from workspace obstacles and velocity bounds, and differential constraints, arising from system kinematics and dynamics. The key idea is to bias the exploration toward unexplored portions of the space by randomly sampling points in the state space and incrementally pulling the search tree toward them.

In order to build a Basic-RRT, Alg. 2, a simple iteration is performed in which each step attempts to extend the RRT by adding a new vertex that is biased by a randomly-selected configuration. The EXTEND function selects the nearest vertex already in the RRT to the given sample configuration, $q$. The nearest vertex is chosen according to a metric $\rho$. The function NEW\_STATE makes a motion toward $q$ by applying an input $u$ for some time increment $\Delta t$, with some fixed incremental distance $\epsilon$, and test for collision. This input can be chosen at random, or selected by trying all possible inputs and choosing the one that yields a new state as close as possible to the sample, $q$. NEW\_STATE implicitly uses the collision detection function to determine whether the new state and all intermediate states satisfy the global constraints. Three situations can occur: REACHED, in which $q$ is directly added to the RRT because it already contains a vertex within $\epsilon$ of $q$; ADVANCED, in which a new vertex $q_{new} \neq q$ is added to the RRT; TRAPPED, in which the proposed new vertex is rejected because it does not lie in $\mathcal{C}_{free}$.

Notwithstanding the basic-RRT can be used in isolation as a path planner, the problem is that, without any bias toward the goal, convergence might be very slow. Several improved planning algorithm have been proposed based on the basic-RRT, p.e., RRT-GoalBias, RRT-Connect, and MPC-RRT.

In RRT-GoalBias, RANDOM\_STATE is replaced by a biased coin that if it returns
Algorithm 2 \textit{RRT-Basic}

\textbf{BUILD\_RRT} ($q_{init}$)
1: $T$.	extit{init} ($q_{init}$)
2: \textbf{for} $k = 1$ to $K$ do
3: $q_{rand} \leftarrow$ \textsc{RANDOM\_CONFIG} ($T, q_{rand}$)
4: \textbf{end for}
5: \textbf{Return} $T$

\textbf{EXTEND} ($T, q$)
1: $q_{near} \leftarrow$ \textsc{NEAREST\_NEIGHBOR} ($T, q$)
2: \textbf{if} NEW\_CONFIG ($q, q_{near}, q_{new}$) \textbf{then}
3: $T$.\textit{add\_vertex} ($q_{new}$)
4: $T$.\textit{add\_edge} ($q_{near}, q_{new}, u_{new}$)
5: \textbf{if} $q_{new} = q$ \textbf{then}
6: \textbf{Return} \textsc{REACHED}
7: \textbf{else}
8: \textbf{Return} \textsc{ADVANCED}
9: \textbf{end if}
10: \textbf{Return} \textsc{TRAPPED}
11: \textbf{end if}

\textit{heads}, then $q_{goal}$ is returned; otherwise, a random state is returned. Even with a small probability of returning \textit{heads}, such as 0.05, \textit{RRT-GoalBias} converges to the goal much faster than the basic \textit{RRT}.

\textit{RRT-Connect} is a probabilistically complete planner designed specifically for path planning problems that involve no differential constraints, or that can be integrated in time reversed. The method is based on two ideas: CONNECT heuristic that attempts to move over a longer distance, and the growth of \textit{RRT}s from both $q_{init}$ and $q_{goal}$. The CONNECT heuristic is a greedy function that can be considered as an alternative to the \textsc{EXTEND} function of the basic-	extit{RRT}, Alg. 3. Instead of attempting to extend an \textit{RRT} by a single $\epsilon$ step, the CONNECT heuristic iterates the \textsc{EXTEND} step until $q$, an obstacle, or an infeasible configuration is reached. This operation has a similar function than the artificial potential function in a randomized potential field approach.\textsuperscript{43} With the CONNECT heuristic, the basin of attraction continues to move around as the \textit{RRT} grows, as opposed to an artificial potential field method, in which the basin of attraction remains fixed at the goal. Two trees $T_a$ and $T_b$ are maintained at all times until they become connected and a solution is found. In each iteration, one tree is extended, and an attempt is made to connect the nearest vertex of the other tree to the new vertex. Then the roles are reversed by swapping the two trees. This causes both trees to explore $C_{free}$, while trying to establish a connection between them.

\textit{MPC-RRT} was proposed by the author of this paper.\textsuperscript{11,46} It consists in changing the
Algorithm 3 RRT-Connect
CONNECT \((T,q)\)
1: repeat
2: \(S \leftarrow \text{EXTEND} \,(T,q)\)
3: until not \((S = \text{Advanced})\)
4: Return \(S\)
RRT_BIDIRECTIONAL \((q_{\text{init}}, q_{\text{goal}})\)
1: \(T_a.\text{init}(q_{\text{init}}), T_b.\text{init}(q_{\text{goal}})\)
2: for \(k = 1\) to \(K\) do
3: \(q_{\text{rand}} \leftarrow \text{RANDOM\_CONFIG}()\)
4: if not \((\text{EXTEND} \,(T_a,q_{\text{rand}}) = \text{Trapped})\) then
5: if \((\text{CONNECT} \,(T_a,q_{\text{new}}) = \text{Reached})\) then
6: Return PATH \((T_a,T_b)\)
7: end if
8: SWAP \((T_a,T_b)\)
9: end if
10: Return Failure
11: end for

CONNECT heuristic by a path-space iteration method.\(^{37-39,52}\) This method iterate on the control along a trajectory until a feasible trajectory is found, incorporating the inequality constraints by using exterior penalty functions. At each iteration a performance index that reflects the feasibility of the path itself is minimized, i.e., the error between the end-point map of the system and the desired final configuration is minimized. The convergence of the algorithm requires that the brackets of the Control Lie Algebra (CLA) satisfy a linear growth condition. For the exterior penalty functions a generalized potential function\(^{10,11}\) is computed. This functions assume that the generalized potential of a differential mass \(dm\) is given by \(dm/r^m\). Then it can be shown that the generalized potential field of a polyhedral body of mass \(M\) and volume \(V\) can be expressed as:

\[
U(q) \equiv \int \int \int_M \frac{dm}{r^m} = k \int \int \int_V \frac{dV}{r^m} = \sum_{e \in E} U_e(q),
\]

where \(U_e(q)\) can be computed in closed form if \(m = 1\) or \(m > 3\).\(^{10,11}\)

4 MPSLAB: MOTION PLANNING AND SIMULATION IN 3D ENVIRONMENTS

The previous sections have introduced some of the tools needed for motion planning in virtual environments. In order to try different planners a flexible platform is needed: MPSLab is a \(C++\) generic software package intended for motion planning and simulation in virtual environments. It can represent multiple robots with high degrees of freedom.
using polyhedra, Fig. 2. Generality and easy implementation are the purposes of this software. To this end, it is used a language similar to VRML as language programming, and were added three collision packages/distance computation libraries, three differential equation solvers, and an optimization package. Figure 3 shows a typical simulation using MPSLab.

In the following paragraphs the main components of MPSLab are resumed.

**Collision Detection/distance computation** Constructing an explicit representation of the obstacle region, $CB$, is not an easy task, and is nor always required. It is often preferable to simply build a logical predicate that serves as a probe that tests whether a configuration lies in $CB$. This is referred to as collision detection. Distance computation refers to the computation of the minimum distance of any object to any body of the robot.

The packages used by MPSLab for these purposes are: PQP, a library for collision detection, distance computation, and tolerance verification on a pair of geometric models.\textsuperscript{29} V-Collide, a package for collision detection between arbitrary polygonal objects in large environments\textsuperscript{30}. SWIFT++, a library for intersection detection, tolerance verification, approximate or exact distance computation of polyhedral models.\textsuperscript{26}

**Differential Equation Solvers** MPSLab supports resolution of ordinary differential equation (ODE) using RKSUITE\textsuperscript{53} and a library written by the author of this paper, and resolution of differential algebraic equation (DAE) using DASPK.\textsuperscript{54}

**Optimization** Nonlinearly constrained large-scale optimization can be solved with the interface with Huge Quadratic Programming (HQP).\textsuperscript{55}
5 DISCUSSION

Motion planning algorithms born with the main idea of solving the problem of finding a collision-free path for a robot in a given environment. However, several exciting application fields have been appeared in last few years and new ones present challenges for researches. For instance:

**Medical Surgery** Imaging techniques are widely available to produce detailed and precise computer representations of 3D tissue structures. It is desirable to precompute minimally invasive paths of surgical tools among soft tissue structures having different elastic properties. Motions must be computed for objects sliding in contact with one another, which requires dealing with friction models.

**Planning for sensing** Sensors can be used to acquire information about an environment, like building a 3D model or localizing objects of interest. *Active sensing* deals with the problem of finding the next placement (and the corresponding motion) where the acquisition of new information is a maximum to avoid redundancy and data explosion.

**Graphic Animation** During the last few years, animation have incorporated dynamic modeling to create physically realistic animation, vision sensing to automatically acquire 3D models, haptic interaction to *feel* virtual objects, and motion planning to create collision-free motions. Fast planner can help to generate realistic-looking motions.
6 CONCLUSIONS AND FUTURE WORK

In this work a novel software package combining randomized motion planning algorithms, collision detection/distance computation, optimization, and differential equation resolution have been presented. There exist a lot of applications where this software can be used, and more research is being done on the integration of this simulation platform with real systems. Moreover, the author is adding collision response capability to finish the simulation power of the software. There is also work on programming more complex examples, i.e., robots with several degrees of freedom (more than 70) in order to test the real capacity of the planner.

REFERENCES


[41] C.I. Connolly, R.A. Gruppen, and K.X. Souccar. A Hamiltonian approach to kinody-


