

Planning the Sequencing of Movement Primitives

Marcelo Kallmann¹, Robert Bargmann^{2*} and Maja Mataric³

^{1,3} Interaction Lab/Robotics Research Lab
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781, USA
{kallmann,mataric}@usc.edu

² Computer Graphics Group
Max-Plank-Institut für Informatik
66123 Saarbrücken, Germany
bargmann@mpi-sb.mpg.de

(*Work done while at EPFL-VRlab)

Abstract

Neuroscience evidence supports the idea that biological adaptive behavior may utilize combination and sequences of movement primitives, allowing the motor system to reduce the dimensionality of movement control.

We present a framework, using sampling-based motion planning, that is able to automatically determine the sequencing of parametric movement primitives needed to execute a given motion task.

Our approach builds a search tree in which nodes are configurations reachable with one or more movement primitives, and edges represent valid paths connecting parent and child nodes. The paths are determined by a motion planner that operates in the parameter space of a single movement primitive. The search tree is expanded with A*-like best-first search using greedy problem-specific heuristics.

The benefits of our approach are twofold: 1) planning complex motions becomes more efficient in the reduced dimensionality of each movement primitive, and 2) the ability to plan entire motions containing heterogeneous types of constraints, such as collision-free, balanced, alternating support contacts, etc.

We present a general framework and several simulation results of statically stable biped walking motions among obstacles. The presented planning capabilities enable robots to better handle unpredicted situations, and can be used as a method of self-organization of higher-level primitives.

1. Introduction

The control of complex robot motions remains a key challenge in robotics. While the number of degrees of freedom (DOF) can characterize the complexity of a robot, the complexity of its motions is further influenced by the constraints they are subjected to.

Evidence from neuroscience supports the idea that complex and adaptive motor behavior might be obtained through the combination of motor primitives [TS00] [Ma02].

In the frog and rat, for instance, the presence of spinal force fields, when appropriately combined through supra-spinal inputs, results in the entire repertoire of observed movement [Gi⁺93]. Studies of human movement [TS00] also provide evidence towards an encoding of primitives.

In robotics, most work in this domain has focused on the design, learning, and combination of different kinds of motor primitives [SS98] [AM02] [Wi96]. In contrast, our work focuses on the problem of automatic adaptation and sequencing of movement primitives in order to satisfy a given motion task in an unknown environment.

We consider that a movement primitive affects the configuration of a robot through a proper parameterization, respecting a set of motion constraints. Our method is then able to plan motions traversing different configuration subspaces, each being covered by a single primitive. As a result, we are able to plan entire motions respecting heterogeneous types of constraints, such as: collision free, in balance, alternating support contacts, etc (see Figure 1).

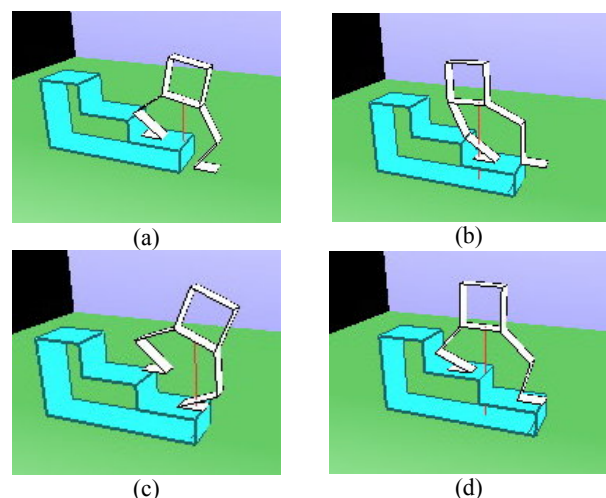


Figure 1: Climbing motion obtained with movement primitives designed to operate in the different support modes.

Our method finds the sequencing of primitives by means of a search tree in which nodes are configurations reachable by more than one movement primitive, and edges represent valid paths connecting parent and child nodes. A sampling-

based motion planner operating in the parameter space of a single movement primitive determines each valid path. Therefore, edges represent primitive motions leading to nodes that serve as connection points allowing primitive change. The tree is expanded with A*-like best-first search using greedy problem-specific heuristics [RN95], until the desired task is satisfied.

The given vocabulary of movement primitives is responsible for reducing the planning complexity, which is related both to the number of DOF to be controlled and to the diversity of motion constraints to be satisfied. If the vocabulary of movement primitives is able to express the motions required to satisfy the task, the planner will easily find connection points between primitives, allowing the search tree to approach a solution. The chosen planner dictates the strategy of how movement primitives are adapted to overcome obstacles towards connection points.

Our method focuses on the generation of kinematic trajectories, assuming that a PD controller is available in order to transfer motions to the real robot. This choice is motivated by three main reasons:

- Kinematic plans are independent of motor commands, thus more suited to support the idea that complex motions are structured through combination and sequencing of movement primitives.
- There is neuroscience evidence of representation of kinematic trajectory plans in the parietal cortex and inverse dynamics models in the cerebellum [SS98].
- We are able to plan motions respecting more difficult constraints, such as maintaining support contacts during locomotion or object manipulation.

We present a general framework, applicable to different problems and several simulation results of a statically stable biped walking among obstacles.

2. Related work

Very few works have attempted to plan complex robot motions by sequencing movement primitives. Typically, research has independently addressed the problems of designing and learning control policies, or the general problem of motion planning in configuration spaces. We review the main works addressing these issues.

Motor primitives Most previous work in robotics related to motor primitives has focused on the design and learning of control policies, based on the combination of different kinds of motor primitives [SS98] [AM02] [Wi96]. Note that, although motor, movement, and motion primitives are commonly interchangeable terms, the term motor primitive usually deals with motor commands. Motor primitives are often divided in two classes: oscillatory and postural (discrete). Being influenced by neuroscience, these works mainly concentrate on learning or skill acquisition, in dynamically simulated or real humanoid robots.

Motion planning Several algorithms are available for the general problem of robot motion planning [La91]. In particular, sampling-based methods provide general

algorithms applicable to problems of arbitrary dimensionality of control. These methods can be divided into two main categories: multi-query and single-query.

Multi-query methods build a Probabilistic Roadmap (PRM) [Ka⁺96] that can be used for several different queries in a single static environment. The basic procedure consists of randomly sampling the configuration space, creating nodes when samples are valid, and connecting pairs of nodes each time the connection is tested to be valid and the nodes are considered to be close enough. Several variations to the basic PRM approach have been proposed [Be03] [Si⁺00] [BK00]. A good overview and comparison is given in [GO02].

Single-query methods are used when the environment is not static. Roadmaps are built specifically for each query, but, for better efficiency, trees are used instead of graphs. The Rapidly-exploring Random Tree (RRT) [KV00] [Va98] is a popular single-query method. Its basic idea is to expand nodes of the tree toward random samples until reaching the goal configuration. Another efficient method is based on Expansive Spaces Trees [Hs⁺99], where nodes in low-density locations are locally expanded. An efficient bi-directional version [SL01] incorporating lazy collision detection [BK00] is also available.

We make use of probabilistic roadmaps constructed in the valid portion of the configuration space covered by one movement primitive. Nodes of the roadmap are candidate configurations to serve as connection points to another movement primitive. Although the choice of a planner should take into account problem-specific issues, we use in this work an RRT [KV00] [Va98] approach to create roadmaps in the parametric space of movement primitives.

Legged locomotion Several different approaches have been presented for the control of legged locomotion. Genetic algorithms [RN95] have been used to adjust synapse weights of neural networks in central pattern generators [Ij01], and to evolve developmental programs based on pre-designed specific grammars [Fi⁺99]. Dynamic biped locomotion is usually achieved through the design of specific control policies [Zo⁺02], or based on the Zero Moment Point [Mi⁺95].

Our work is limited to statically balanced motions. Our approach can be seen as a complementary way to plan different patterns of motions required to overcome unanticipated situations.

Some works in the virtual reality and computer games domain have also addressed the locomotion problem of legged characters. In particular, motion planners were proposed for the multi-modal locomotion of a 2D character [KP01], and for planning footsteps locations, which can be connected with warped motion-captured sequences applied to virtual characters [Ch03].

The work presented by Kuffner et al. [Ku⁺01] [Ku⁺02] also relies on a search procedure in order to sequence pre-designed leg motions of a simulated humanoid robot. Their approach is more efficient than ours, however is limited to pre-designed motions. In our approach new types of motions

are generated, for instance to avoid obstacles of any shape or use them as support.

3. Definitions and notations

Let d be the number of degrees of freedom (DOF) of a given robot and let C be the d -dimensional configuration space of the robot. A configuration in C is said to be valid if it satisfies problem-specific validity requirements. Examples of validity requirements are: collision-free, balanced, etc. The subset of all valid configurations in C is denoted as C_{free} .

Movement primitive As d might be too high and C might be too complex due to motion and environmental constraints, we are limited to manipulating the robot only by means of a finite set of movement primitives. Each primitive defines a control policy that locally alters a given configuration, according to a proper parameterization space. Therefore, each movement primitive \mathcal{P}_i , $i \in \{1, \dots, n\}$, when instantiated at a given configuration $s \in C$, becomes a function of the type:

$$\mathcal{P}_i^s: S_i^s \rightarrow C \quad (1)$$

S_i^s is the parameter space of primitive i , instantiated at configuration s . Configuration s is also said to be the starting point of movement primitive \mathcal{P}_i^s . There are two main reasons for having the parameterization space dependable on s : first, parameters are often considered in relation to a local frame relative to s , and second, s might imply specific limits on the parameterization. We also allow that the dimension of the parameterization space change according to the instantiation.

Each primitive \mathcal{P}_i has instantiation conditions to satisfy. If, for a given $s \in C$, the instantiation conditions of \mathcal{P}_i cannot be verified, we say that \mathcal{P}_i cannot start at configuration s . For example, a movement primitive for balanced knee flexion might be designed to start only when both feet are in contact with a proper support.

Each movement primitive has to be constructed in such a way as to allow the implementation of some required operators. These operators vary according to the motion planner selected to operate on the primitive parameter space. Typical motion planners considered in our framework require, for each instantiated primitive \mathcal{P}_i^s , the existence of a distance function and an interpolation function. The motion planner is responsible for building a roadmap graph connecting s to other configurations in the free portion of the configuration space covered by primitive \mathcal{P}_i^s .

Roadmap Let $C_i^s \subset C$ denote the image of movement primitive \mathcal{P}_i^s , i.e., for any $s \in C$, $C_i^s = \mathcal{P}_i^s(S_i^s)$ if \mathcal{P}_i^s can be instantiated at s , and $C_i^s \equiv \emptyset$ otherwise. C_i^s represents the subspace of C covered by primitive \mathcal{P}_i^s .

We define the roadmap of a movement primitive instance as a single connected graph $R(\mathcal{P}_i^s)$. Nodes in the graph are valid configurations belonging to $C_i^s \cap C_{free}$, and edges in the graph represent valid paths in $C_i^s \cap C_{free}$ joining the edges endpoints. The starting node s of the primitive is required to

be a node of $R(\mathcal{P}_i^s)$. For simplicity of notation, we may also refer to the nodes of a roadmap as the configurations of the roadmap.

Usually, for computational efficiency, roadmap edges are determined by checking the validity of the interpolation between the edge endpoints, and thus it is assumed that primitive \mathcal{P}_i^s has a proper efficient interpolation function $interp_i^s$ of the type:

$$interp_i^s: S_i^s \times S_i^s \times [0,1] \rightarrow S_i^s \quad (2)$$

As usual, the interpolation function is parameterized over the interval $[0,1]$, so that $interp_i^s(p_1, p_2, 0) = p_1$, and $interp_i^s(p_1, p_2, 1) = p_2$, for a given pair of points $\{p_1, p_2\} \in S_i^s$. We say that the interpolation of a pair of points $\{p_1, p_2\}$ is valid if, for all $t \in [0,1]$, $\mathcal{P}_i^s(interp_i^s(p_1, p_2, t))$ is a valid configuration, i.e., belongs to C_{free} .

Analogously, we say that there is a valid path joining $q_1 \in C_{free}$ and $q_2 \in C_{free}$ if there is an instantiated primitive \mathcal{P}_i^s and two points $\{p_1, p_2\} \in S_i^s$, such that $q_1 = \mathcal{P}_i^s(p_1)$, $q_2 = \mathcal{P}_i^s(p_2)$, and the interpolation between p_1 and p_2 is valid.

Note that the interpolation function is defined in parameter space, meaning that paths are first generated in parameter space and then transformed into the configuration space for validity testing. With this formulation all the roadmap computation is done locally to the movement primitive, and does not require the computation of the inverse of the movement primitive function.

Besides the interpolation function, a distance metric is often required during the roadmap construction process. In Section 4 we describe an algorithm that constructs roadmaps following the RRT [KV00] [Va98] expansion strategy.

Roadmaps are used as a sampling strategy to transform the continuous parameterization of a movement primitive into a discrete set of configurations (the nodes of the roadmap), which are suitable for inclusion in a search tree.

Problem definition Let \mathcal{P}_i , $i \in \{1, \dots, n\}$ be a given set of movement primitives manipulating a robot as defined above. Consider that the task to be accomplished is defined as a function that returns 1 if a given configuration q satisfies the task, and zero otherwise, i.e.:

$$task(q) : C \rightarrow \{0,1\} \quad (3)$$

The problem we want to solve is determining a sequence of configurations $(q_j)_{j=1,m}$, such that:

- The current robot configuration is equal to q_1
- $task(q_m) = 1$
- For each pair of configurations q_k, q_{k+1} , $1 \leq k < m$, there is a valid path connecting q_k and q_{k+1} .

Note that the determination of each valid path requires the determination of the instantiated primitive that generates it (through the primitive's interpolation function).

The solution $(q_j)_{j=1,m}$ implies the determination of a sequence of paths joining q_1 to q_m . For simplicity of notation, we also say that a valid path between two configurations exists when in fact there is a sequence of paths joining them.

When needed, we distinguish these two cases with the terms: composed path and direct path.

4. General method

Our approach is based on a search tree where nodes represent reachable valid configurations, and edges contain paths between parent and child nodes. At any point, a partial solution can be constructed by concatenating the paths in the unique sequence of edges joining the root node to a given node.

The algorithm starts by initializing the root of the tree with the current robot configuration, and then an expansion process adds nodes to the tree until the task function is satisfied. A cost is associated with each node and represents the cost of the path constructed so far. The root node is initialized with cost 0. A priority queue is used to efficiently store the leaves of the tree according to their priority of expansion.

Generally, an A* expansion [RN95] is followed, where the highest priority is given to the leaves with less heuristic cost. The simplest heuristic cost for a given node n sums the cost of n with an estimate of the distance to achieve the goal task from n . Several problem-specific heuristics can be added to this basic formula, and they are important in ensuring that the search tree grows toward the solution.

After initialization, the highest priority leaf q is removed from the priority queue (not the tree), and a roadmap is constructed from q . Nodes in the roadmap that allow primitive change are added to the priority queue as new leaves, and added to the search tree as children of q .

The following algorithm summarizes this expansion step:

```

expand ( tree, queue )
1.  $q$  = remove higher priority leaf from queue;
2. if  $task(q)=1$ , do:
   return composed path from root(tree) to  $q$ ;
3. for  $i = 1$  to  $n$ , do:
4.   if  $\mathcal{P}_i$  can be instantiated at  $q$ , and
    $\mathcal{P}_i$  was not instantiated by  $q$  parent, do:
5.      $R(\mathcal{P}_i^q) = build\_roadmap ( i, q );$ 
6.     for all configurations  $q_r \neq q$  in  $R(\mathcal{P}_i^q)$ , do:
7.       if  $q_r$  can be instantiated by a different
       movement primitive than  $\mathcal{P}_i$ , do:
8.          $edge = add\_child ( tree, q, q_r );$ 
9.         store in  $edge$  the path in  $R(\mathcal{P}_i^q)$ 
           joining  $(q, q_r)$ ;
10.        add leaf  $q_r$  to  $queue$ ;
11. return null path;

build_roadmap ( i, s )
1.  $R(\mathcal{P}_i^s) = init$  roadmap with node  $s$ 
2.  $failures = 0$ 
3. while  $failures < MaxTries$ , do:
4.    $p_{rand} = random$  point in  $S_i^s$ ;
5.    $p_{near} = nearest$  node to  $p_{rand}$  in  $R(\mathcal{P}_i^s)$ ;
6.    $dist = distance ( p_{near}, p_{rand} );$ 
7.    $p_{new} = interp_{i^s} ( p_{near}, p_{rand}, IncrementalStep );$ 
8.   if (  $p_{near}$  and  $p_{new}$  interpolation valid ), do:
9.     add node  $p_{new}$  and edge  $\{p_{near}, p_{new}\}$  to  $R(\mathcal{P}_i^s)$ ;
10.     $failures = 0$ ;
   else
11.     $failures++$ ;
12. return  $R(\mathcal{P}_i^s)$ ;

```

Procedure `expand` is called until a non-null path sequence (the solution) is returned. Each time a node is expanded, procedure `roadmap` returns a single connected roadmap graph. The nodes of the graph are candidate to become new leaves in the search tree.

Procedure `build_roadmap` implements a general roadmap construction method following the RRT [KV00] [Va98] expansion strategy. It requires the definition of two parameters. Parameter `MaxTries` specifies the number of failures that are needed in order to decide that the roadmap cannot grow any more. Parameter `IncrementalStep` provides control over the length of the edges in the graph. For more uniform resolution control, a length step measured in configuration space should be used whenever possible.

Several problem-specific issues can be addressed during roadmap construction. For instance, in the biped robot case presented in the next section, whenever a node in the roadmap is detected to be close to a configuration serving as a connection point, we move the node to that connection point.

Note that a random generator routine is used in procedure `roadmap` in order to generate the points in parameter space that guide the roadmap expansion. There is no problem if the free portion of the parametric space is much smaller than the whole parametric space. The RRT expansion strategy provides a suitable gradual exploration of the free space, and its implementation is simple and efficient. However, whenever possible, the choice of the roadmap construction strategy should be problem-(and primitive-)-specific.

As roadmaps represent a discretization of a continuous space, it is not possible to guarantee that the search tree will find the optimal solution. However, usually we are not interested in finding the optimal solution as the search tree easily becomes prohibitively large and greedy heuristics are always preferred. In addition, problem-specific optimizations are likely to be crucial for having acceptable running times for complex problems.

In the next section we demonstrate how our framework can be applied to generate motions for the control of a statically stable biped robot.

5. Biped robot example

We have implemented and tested the algorithm described in the previous section on planning statically stable walking motions for a biped robot moving in a planar environment containing polygonal obstacles. Obstacles are avoided during motion and are also used as support, allowing the generation of climbing sequences (see Figure 1).

The designed biped robot has a total of 9 DOF: the first two specify the position of the body center in the Cartesian plane. The remaining DOF are rotation angles: one to specify the orientation of the body, and three for the articulations of each leg (see Figure 2).

Each rotational DOF of the robot has specific lower and upper articulation limits. The two positional DOF have no limits, however when they are controlled by a movement

primitive, they have limits imposed by the instantiated parameterization space of the primitive.

Configuration validity Let C be the 9-dimensional configuration space of the biped robot. We define a configuration $q \in C$ to be valid if q :

- Satisfies the articulation limits of the robot.
- Is collision-free, i.e., does not intersect with obstacles.
- Is in balance, i.e., its center of mass projects inside the support segment of the robot.

As the simulated robot is constructed in a planar environment, the validity tests have a straightforward implementation. However, some special care is required in geometric tests, such as deciding support contacts. In our implementation, all geometric tests are based on an epsilon precision distance.

When the two endpoints of a foot are close enough to an obstacle segment, but without crossing it, the foot is considered to be in contact with the segment and a support segment is defined. When both feet are in support, the support segment is increased to contain the support segment of the two feet.

For the computation of the center of mass we associate a mass value m_k , $k \in \{1, \dots, 7\}$, to the center of each robot part P_k , each being a limb or the body. The center of mass position is then determined by the following weighted average sum:

$$(\sum center(P_k)m_k) / 7 \quad (4)$$

The robot is considered in balance if its center of mass vertically projects inside the support polygon. Collisions are detected whenever a limb segment crosses an obstacle segment or another limb segment.

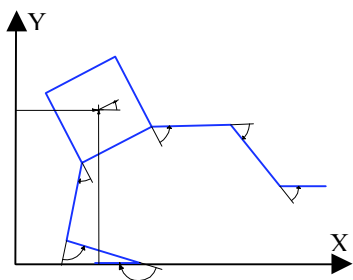


Figure 2: The planar biped robot and its nine DOF.

The defined task function checks if the center of mass of the robot is close enough to a desired location, according to a given precision.

Movement primitives Three movement primitives are defined, each specific to a support mode:

- Movement primitive \mathcal{P}_L is used to move the right leg of the robot while balance is maintained only with the support of the left foot. Therefore the instantiation condition of this movement primitive requires support on the left foot.
- Movement primitive \mathcal{P}_B was designed to move the body while the two legs remain attached to the floor. The

instantiation condition is support on both feet, and its parameterization controls the position and orientation of the robot's body, keeping the feet fixed at their support location.

- Similarly to \mathcal{P}_L , \mathcal{P}_R is designed to move the left leg while the robot keeps support on the right leg.

The parametric spaces of movement primitive \mathcal{P}_L and \mathcal{P}_R both have dimension 4. The affected rotational angles have the same range limits as those originally defined by the robot, except for body rotation, which is allowed only in the direction that favors the free leg to reach higher positions. Whenever body rotation is changed, the angles of the support leg have to be adjusted in order to maintain the support foot in the same place. This adjustment is done by employing a straightforward analytical Inverse Kinematics formulation for the foot to be fixed.

Movement primitive \mathcal{P}_B defines specific range limits for the two translational DOF it controls. Let $s \in C_{free}$ be a configuration and assume that \mathcal{P}_B can be instantiated at s , i.e., s has support in both feet. Let p be the body center point of the robot at s . The translational parameters of \mathcal{P}_B^s are limited to be inside a rectangle of center p and sides with double the length of the body sizes. In this way we have a much smaller range of motion for the motion planner to explore the free portion of the parameterization space.

An Inverse Kinematics formulation is also applied to maintain both feet fixed at their original places while the translational and rotational DOF of primitive \mathcal{P}_B are changed. Table 1 summarizes the main characteristics of the movement primitives.

Movement Primitive	Instantiation Condition	Type of Motion	Parametric Space Dimension
\mathcal{P}_L	support in left foot	moves right leg articulations and body rotation	4
\mathcal{P}_B	support in both feet	moves body, legs fixed with IK	3
\mathcal{P}_R	support in right foot	moves left leg articulations and body rotation	4

Table 1: Summary of used movement primitives.

Note that some configurations allow the instantiation of two different movement primitives. This happens when the configuration allows two different types of support, i.e., when the robot has support in both feet, and, at the same time, support in one foot alone. Such configurations serve as connection points between two different movement primitives. Figure 3 illustrates the different kinds of support modes.

Motion planner A single motion planner was implemented to operate in the parameter space of primitives \mathcal{P}_L , \mathcal{P}_B , and \mathcal{P}_R , and it closely follows the algorithm `build_roadmap` of Section 4.

The random function we used simply selects values inside the range defined for each parameter of each movement primitive. The interpolation function linearly interpolates the corresponding parameters of two given sets of parameters of a primitive.

As a distance function, instead of operating in the parameter space, we compute the mean of the Euclidian distances between the corresponding articulation points of two given configurations. The same distance function is used regardless of the movement primitive being considered.

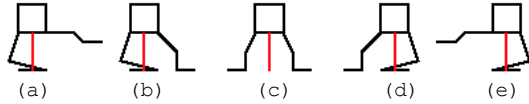


Figure 3: Example of configurations in different support modes: only in left support (a), simultaneous left and both feet support (b), only both feet support (c), simultaneous right and both feet support (d), and only in right support (e).

In order to promote the appearance of configurations in more than one type of support, we include a test during roadmap construction that detects configurations which are close to making contact with a new support. The test consists on measuring the distance between each foot of the configuration to its nearest obstacle. If one distance is smaller than the pre-specified snap distance, the configuration is adjusted with Inverse Kinematics in order to precisely place the foot in contact with the support. Such adjustment is critical, because the achievement of new supports is the only way to find configurations allowing primitive change. The specified snap distance trades the ability to find new supports with the ability to avoid obstacles, and should be set according to the environment.

Search Tree Heuristics After initialization, the search tree is expanded following the general algorithm given in Section 4. The heuristic cost of a leaf in the priority queue is:

$$hc(n) = cost(n) + distance(n \text{ mass center, target point}) \quad (5)$$

Term $cost(n)$ is the cost of the path from the configuration at the root of the tree to n . The cost is defined as the length of the path, according to the same metric used for the roadmap construction: the average of the Euclidean lengths of the articulations paths. The target point is the same point considered by the task function, which tests if the center of mass is close enough to the target point.

The leaf with lowest heuristic cost has higher priority and will be removed first from the priority queue. Removed leaves are then expanded.

Figure 4 illustrates the expansion process. Figure 4a shows the roadmap constructed for the robot in a configuration with both feet in support. Each node in the roadmap represents a full configuration, but in this image, only the position of the body center is used to draw the roadmap. The roadmap in Figure 4b shows a mark (a cross) on each node allowing primitive change, which are the nodes becoming new leaves in the search tree. Figure 4c shows the robot configuration in the highest priority leaf, which is the

one selected for node expansion, now with a single leg support. The roadmap in Figure 4d shows the coverage of the free foot in free space, and shows marks in two configurations again allowing a primitive change, and thus becoming new leaves in the next expansion of the tree. In all images, the circle identifies the root of the roadmaps.

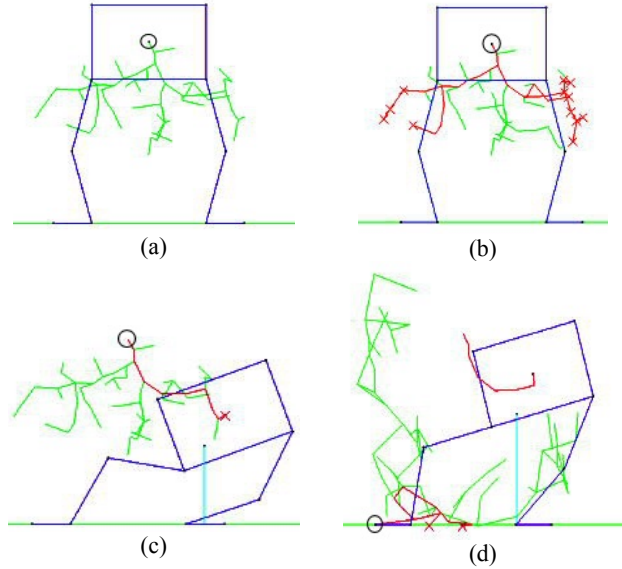


Figure 4: Node expansion example starting with both feet support.

Although the heuristic cost function is rather standard, three greedy optimizations were tested in order to reduce the branching factor and direct the search to the goal:

- The first optimization, and the most important, relies on the observation that some sequences of nodes in the search tree are not useful. This occurs when three adjacent nodes in the tree have the support modes on: left foot, both feet, and left foot, respectively; or all have support simultaneously in both feet and in the left foot alone. The same occurs to the analogous sequences in relation to the right foot. Therefore, before inserting nodes in the search tree, we ensure that such sequences are not formed by checking the parent and grandparent nodes in the tree. More complex analysis in the search tree could be investigated, as, for instance, collapsing different nodes with the supporting feet located at the same place.
- Long locomotion sequences can easily demand prohibitively large time and memory consumption. The search tree is not suitable for planning long paths around a large amount of obstacles, but for planning leg motions for nearby obstacles. Therefore, whenever the current node n being expanded is sufficiently far away from the root node of the tree, the partial path until n can be stored, the whole tree deleted, and initialized again with the new root being n .
- Another optimization can be used to limit the creation of excessive nodes in the same location of the environment. A planar grid can be easily implemented

in order to control, for each cell, a maximum number of nodes occupying that cell. This procedure can speedup the search in some situations, but has some negative impact on the quality of the generated paths.

6. Discussion and results

Our implementation is able to find simple motions in a few minutes. Complex motions, however, require several minutes or more to be computed on a 2.8GHz Pentium 4. The required computation time is related both to the difficulty to overcome obstacles and to the chosen parameters of the planner and of the implemented heuristics. As example, the motion shown in Figure 5 took 50 minutes to be computed and generated a search tree of about 600K nodes.

The branching factor of the search tree is greatly influenced by the resolution step used in the motion planner. This step affects the number of nodes computed in each roadmap, and thus also the branching factor. With a small resolution step, the solutions are likely to be closer to an optimum, but large branching factors are not efficient and can quickly exhaust memory resources.

Parameter `MaxTries` (see Section 4) can be usually set to low values, allowing fast computation of roadmaps. However, in environments with many obstacles, larger values are required. In such cases, roadmaps only generate child nodes in the search tree after many random tries.

Due to the sampling strategy of roadmaps, solutions are clearly not optimal. However, few optimal characteristics could be noted: large steps are generated towards the goal in the absence of obstacles and few finer steps are performed next to obstacles in order to determine the best position before overcoming them.

Figures 1, 5 and 6 show some of the results obtained. Note that the final result may exhibit some jerky motions, due to the exploration strategy of the roadmap computation. A simple and efficient smoothing process consists of trying to replace pair of points in the path by valid straight interpolations [Sc⁺02]. However, the presented results were not smoothed. Videos showing further results can be found at: <http://robotics.usc.edu/~kallmann/biped2d/>

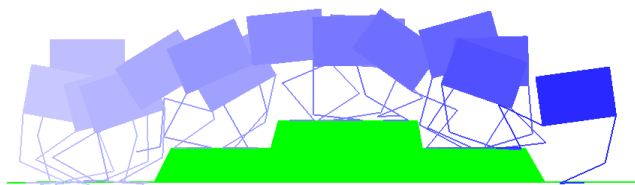


Figure 5: Example motion over an obstacle.

Although the robot design used in this work is quite simple, we note that it is rather difficult to make it overcome obstacles by manually controlling the parameters of the movement primitives by means of a user interface. The validity constraints leave a small motion range in several situations, and precise positioning prior to obstacles is often

required before overcoming them. Nevertheless, the planner successfully found several non-trivial solutions.

7. Conclusions

We have presented a search method for planning complex motions through the concatenation of paths generated by biologically-inspired movement primitives. Our approach introduces a way to apply sampling-based motion planning techniques in order to compose motions controlled by heterogeneous types of parameterizations. We are thus able to respect different types of constraints, e.g., alternating support contacts. In addition, planning complex motions usually becomes more efficient in the reduced dimensionality of each movement primitive.

Our method is able to find motions for unanticipated situations. This adaptive behavior characteristic is of key importance and is supported by neuroscience evidence showing that complex motions in animals might be structured through the use of movement primitives.

The framework presented in this paper can be applied to other kinds of problems as well. Examples include mobile manipulators equipped with different primitives for locomotion and manipulation, and the automatic generation of higher-level primitives based on motions planned with simpler ones. For instance, portions of planned motions that are detected to be useful can be directly encoded and used as a new time-parameterized primitive.

In general, any control policy with a suitable parameterization can be treated as a movement primitive. However, the success of the best-search method greatly depends on the availability of efficient problem-specific heuristics; otherwise the branching factor of the search tree can easily become prohibitively large.

Acknowledgments

The first author is grateful to Dr. Anand Panangadan for fruitful discussions. This work was supported by a DARPA MARS 2020 Program grant NAG9-1444.

References

- [AM02] R. Amit and M. J. Mataric, "Parametric Primitives for Motor Representation and Control", Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-2002), Washington, DC, May 11-15, 2002, pp. 863-868.
- [Be03] K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plakue, and L. E. Kavraki, "Multiple query probabilistic roadmap planning using single query planning primitives", Proc. of the International Conference on Intelligent Robots and Systems (IROS), Las Vegas, USA, 2003.
- [BK00] R. Bohlin and L. Kavraki, "Path planning using lazy PRM", Proc. of the International Conference on Robotics and Automation (ICRA), San Francisco, USA, 2000, pp. 521-528.
- [Ch03] M. G. Choi, J. Lee, S. Y. Shin, "Planning biped locomotion using motion capture data and probabilistic roadmaps", ACM Transactions on Graphics (TOG), 22(2), 2003, pp. 182-203.

- [Fi⁺99] D. Filliat, J. Kodjabachian, and J.-A. Meyer, "Evolution of Neural Controllers for locomotion and obstacle-avoidance in a 6-legged robot", *Connection Science* 11:223-240, 1999.
- [Gi⁺93] S. F. Giszter, F. A. Mussa-Ivaldi, and E. Bizzi, "Convergent force fields organized in the frog's spinal cord", *Journal of Neuroscience* 13(2):467:491, 1993.
- [GO02] R. Geraerts and M. Overmars, "A comparative study of probabilistic roadmap planners", *Proc. of the International Workshop on Algorithmic Foundations of Robotics (WAFR)*, Nice, France, 2002.
- [Hs⁺99] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces", *International Journal of Computational Geometry and Applications*, 9(4-5):495-512, 1999.
- [Ij01] A. J. Ijspeert, "A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander", *Biological Cybernetics* 84(5): 331-348, 2001.
- [Ka⁺96] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for fast path planning in high-dimensional configuration spaces", *IEEE Transactions on Robotics and Automation*, 12(4):566-580, June 1996.
- [KP01] M. Kalisiak and M. van de Panne, "A grasp-based motion planning algorithm for character animation", *The Journal of Visualization and Computer Animation* 12(3):117-129, 2001.
- [Ku⁺01] J.J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Footstep planning among obstacles for biped robots", *Proc. of the International Conference on Intelligent Robots and Systems (IROS)*, Maui, USA, 2001.
- [Ku⁺02] J.J. Kuffner, S. Kagami, M. Inaba, and H. Inoue, "Dynamically-stable Motion Planning for Humanoid Robots", *Autonomous Robots*, 12(1): 105-118, 2002.
- [KV00] J. J. Kuffner and S. M. La Valle, "RRT-connect: an efficient approach to single-query path planning", *Proc. of the International Conference on Robotics and Automation (ICRA)*, San Francisco, USA, 2000, pp. 995-1001.
- [La91] J.-C. Latombe, "Robot motion planning", ISBN 0-7923-9206-X, Kluwer, Academic Publishers, 1991.
- [Ma02] M. J. Mataric', "Visuo-motor primitives as a basis for learning by imitation", In K. Dautenhahn and C. Nehaniv, editors, *Imitation in Animals and Artifacts*, MIT, Press, 2002, pp. 391-422.
- [Mi⁺95] K. Mitobe, N. Mori, K. Aida, and Y. Nasu, "Non-linear feedback control of a biped walking robot", *IEEE International Conference. on Robotics and Automation (ICRA)*, 1995, pp. 2865-2870.
- [RN95] S. J. Russell and P. Norvig, "Artificial intelligence: a modern approach", Prentice Hall, Englewood Cliffs, NJ, 1995.
- [Sc⁺02] F. Schwarzler, M. Saha, and J.-C. Latombe, "Exact Collision Checking of Robot Paths", In *Workshop on Algorithmic Foundations of Robotics*, Nice, France, December 2002.
- [Si⁺00] T. Simeon, J. P. Laumond, and C. Nissoux, "Visibility based probabilistic roadmaps for motion planning", *Advanced Robotics Journal*, 14(6), 2000.
- [SL01] G. Sanchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking", *Proc. of the International Symposium on Robotics Research (ISRR)*, Nagoya, Japan, 2001.
- [SS98] S. Schaal and D. Sternad, "Programmable pattern generators", In *3rd International Conference on Computational Intelligence in Neuroscience*, Research Triangle Park, NC, 1998, pp 48-51.
- [TS00] K. A. Thoroughman and R. Shadmehr, "Learning of action through combination of motor primitives", *Nature* 407:742-747, 2000.
- [Va98] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning", Technical Report TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.
- [Wi96] M. W. Williamson, "Postural primitives: interactive behavior for a humanoid robot arm", *Proceedings of the 4th International conference on Simulation of Adaptive Behavior (SAB'96)*, MIT Press, 1996, pp. 124-131.
- [Zo⁺02] F. Zonfrilli, G. Oriolo, and D. Nardi, "A biped locomotion strategy for the quadruped robot Sony ERS-210", *IEEE International Conference. on Robotics and Automation (ICRA)*, 2002, pp. 2768-2774.

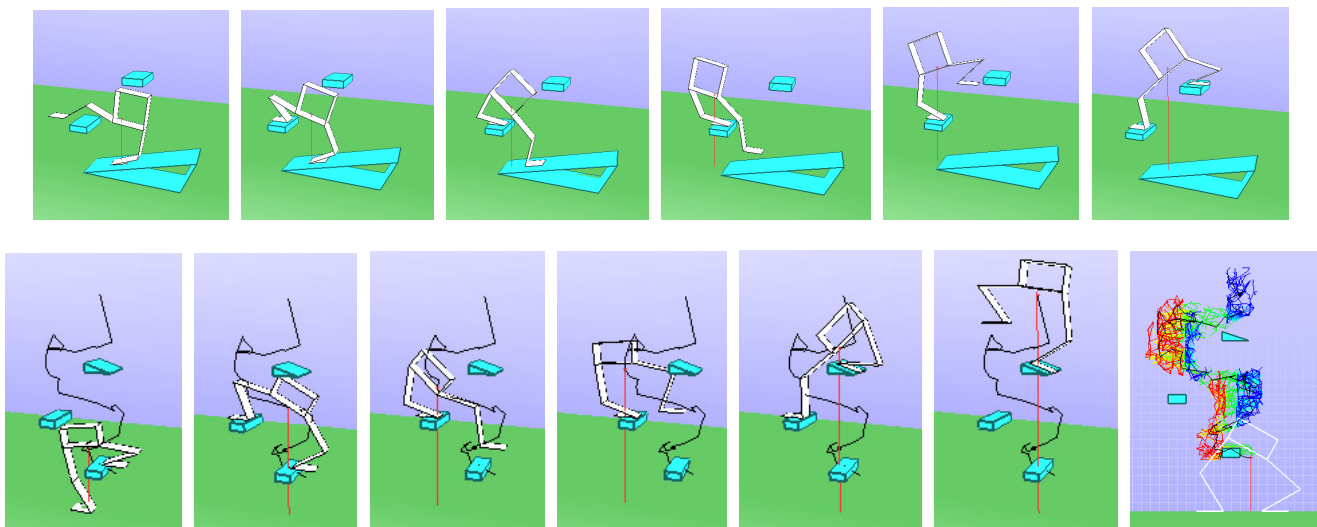


Figure 6: Two example sequences of planned motions. The vertical line illustrates the projection of the center of mass, which is always inside the support segment of the robot. In the bottom row sequence, the trajectory of the center of mass is also shown. The last image shows the center of mass of all nodes in the search tree, colored according to the support mode.