High-level motion planning for CPG-driven modular robots

Vojtěch Vonášek, Martin Saska, Lutz Winkler, Libor Preučil

Czech Technical University in Prague, Faculty of Electrical Engineering, Technická 2, Prague 6, Czech Republic
Karlsruhe Institute of Technology, Institute for Process Control and Robotics, Engler-Bunte-Ring 8, D-76131 Karlsruhe, Germany

Abstract

Modular robots may become candidates for search and rescue operations or even for future space missions, as they can change their structure to adapt to terrain conditions and to better fulfill a given task. A core problem in such missions is the ability to visit distant places in rough terrain. Traditionally, the motion of modular robots is modeled using locomotion generators that can provide various gaits, e.g., crawling or walking. However, pure locomotion generation cannot ensure that desired places in a complex environment with obstacles will in fact be reached. These cases require several locomotion generators providing motion primitives that are switched using a planning process that takes the obstacles into account. In this paper, we present a novel motion planning method for modular robots equipped with elementary motion primitives. The utilization of primitives significantly reduces the complexity of the motion planning, which enables plans to be created for robots of arbitrary shapes. The primitives used here do not need to cope with environmental changes, which can therefore be realized using simple locomotion generators that are scalable, i.e., the primitives can provide motion for robots with many modules. As the motion primitives are realized using locomotion generators, no reconfiguration is required and the proposed approach can thus be used even for modular robots without self-reconfiguration capabilities. The performance of the proposed algorithm has been experimentally verified in various environments, in physical simulations and also in hardware experiments.

Keywords: Motion Planning, Modular Robots, Rapidly Exploring Random Trees, Central Pattern Generator

1. Introduction

Modular robots consist of many autonomous modules, which can be connected to form bodies of various shapes. Modular robots have attracted research interest in recent years, which has resulted in numerous platforms such as PolyBot [1], CONRO [2], M-TRAN [3], ATRON [4] and SuperBot [5], Molecubes [6], ICubes [7] and Symbrion/Replicator [8]. We refer to [9] for a comprehensive survey of existing platforms. Modular robots can be utilized in challenging applications like space exploration [10], search & rescue missions [11] and object manipulation [12].

The main advantage of modular robots lies in the possibility of forming versatile organisms that can move in diverse terrains. The modules can form a lizard-like robot to crawl on an undulating terrain or they can build a spider-like structure to traverse a rough terrain. Due to the many possible configurations of modular robots, their behavior cannot be preprogrammed. It has to be designed for the actual structure when needed.

Studies of modular robot motion are usually focused on locomotion generation. Widely-used locomotion generators are based on Central Pattern Generators (CPGs), which produce periodical signals for the actuators [13]. Locomotion generators can provide an efficient way to move a robot in a desired manner or in a given direction. However, a single locomotion generator is not sufficient to reach a distant goal in a complex environment. Similarly, a single locomotion generator may become useless if the environment changes, e.g., if the terrain changes from gravel to sand. In such cases, the robot should be equipped with several locomotion generators, which are switched by a high-level algorithm to achieve the desired goal. For example, a robot equipped with ‘walk’ and ‘rotate-left’ actions can visit many places in an environment, but these actions need to be combined to reach a goal and to avoid obstacles. This can be achieved by using motion planning.

In this work, we present a novel high-level motion planning technique for modular robots operating in complex environments. The proposed planner finds a sequence of predefined motion primitives to reach a desired goal. The motion primitives are the vocabulary of the basic skills of the robot, such as ‘walk-forward’ and ‘rotate-left’. The robot can thus be equipped with several primitives optimized for a given terrain, rather than having only a single locomotion system that can be adapted. Such primitives can be realized using simple methods, e.g., well-known CPGs, which can be optimized for robots of arbitrary structure.

The text is organized as follows. Related work is summarized in the next section. The hardware architecture of the robotic modules developed in the Symbrion/Replicator...
projects is described in Section 3. A novel approach for global motion planning with motion primitives is described in Section 4. The results of the simulations are discussed in Section 5 and the results of the hardware experiments are discussed in Section 6.

2. Related work

According to the hardware design of the robotic modules, modular robots can move using three concepts: a) wheeled locomotion, b) joint control locomotion and c) locomotion through self-reconfiguration. For wheeled locomotion, some modules need to be equipped with wheels [14], tracks [15], omnidirectional wheels [15] or even screw-drives [16]. Locomotion using joint-control is achieved by changing the angles between the connected modules similarly to movements of the arms or legs of living creatures. Joint-control requires the connected modules to be able to change angles between them. This type of motion does not require any disconnection and reconnection. Therefore, it can also be used for modules without self-reconfiguration capabilities. In the concept of self-reconfiguration, the motion of an organism is achieved by repeated disconnection of the modules, moving them to a new position and reconnecting them back to the robot [17, 18, 19, 20, 21, 22].

In this paper, the concept of joint-control locomotion is considered. This type of locomotion can be modeled using Central Pattern Generators (CPG) that provide rhythmic signals for the actuators. This is inspired by evidence from nature, that motion is generated by coupled neuro-oscillators providing rhythmic signals [23]. CPGs have been used to generate locomotion of snake-like robots, to control swimming robots, to generate biped locomotion of humanoid robots and also to control modular robots [24, 25, 26]. A comprehensive review of CPGs and their usage in robotics can be found in [13].

The CPGs distribute the control signals to each actuator from a central unit, which can be prone to communication failures. Another approach for joint-control locomotion is to run a cyclic function on each module separately. To synchronize actions of the modules, each module sends a message to its neighbors after a fraction of the period is completed [27]. A similar approach is used in role-based control in [28]. Synchronization can also be achieved using artificial hormones [29, 30], which flow through the modules and trigger their actions. Another type of triggered synchronization was proposed in [31]. More general types of motion can be achieved using non-cyclic functions, that can be derived automatically using the concept of Genetic Programming [32, 33].

The methods mentioned here can provide efficient locomotion for various types of modular robots. Several systems (e.g. [28, 29, 30]) are well scalable for robots with many modules. However, the performance of the locomotion may vary as the robots move through environments with changing terrain. In these cases, the parameters of the locomotion generator need to be adapted, or another generator needs to be utilized. Moreover, a single locomotion generator may not be suitable if obstacles need to be avoided. An approach utilizing several gaits that are switched according to the type of terrain was proposed in [34]. This kind of model contains many parameters that need to be tuned in advance. This is not suitable for modular robots, which can reconfigure to many shapes. Another approach for adapting the locomotion utilizes sensory information in the feedback [25, 35, 36, 37]. However, like the previous approach, it is difficult to optimize the required parameters.

Another approach to ensure motion of a robot through complex environments with obstacles employs a motion planner. The motion planning problem has been studied well in the literature; we refer to [38] for a detailed survey. In the case of modular robots with many DOFs and kinematic constraints, sampling-based methods can be utilized [39]. Sampling-based methods solve the task by random sampling of the robot’s configuration space. This allows the creation of a roadmap of free configurations, in which a feasible trajectory can be found. Widely-used sampling-based planners are Probabilistic Roadmaps [40] and Rapidly Exploring Random Trees (RRT) [41] and their variants. These methods have been utilized in many applications [42, 43, 44], including modular robotics [45, 46, 47] and reconfiguration planning [48].

The above-mentioned locomotion generators as well as motion planning methods can provide motion for modular robots. Locomotion generators are efficient in providing basic motions, but they do not consider complex situations in the environment. Although generators can be extended to cope with altering terrain, finding suitable parameters for such models is time consuming. However, motion planning techniques are well fitted to solve the global planning problem, as they cope with obstacles and consider the situation on a higher level. However, their performance can degrade when they need to derive control signals for many actuators, which is the case of modular robots.

In this paper, we propose the novel RRT-MP planner (Rapidly Exploring Random Tree with Motion Primitives), that combines locomotion generation and motion planning. To fulfill a high-level goal, sampling-based motion planning is utilized, as it considers the overall situation in the environment and it can avoid collisions with obstacles or other robots. To speed up the planning process, the problem of finding control inputs is solved by utilizing locomotion generators that provide basic skills — motion primitives. The result of RRT-MP is a sequence of motion primitives, which can easily be executed by the robot. Whenever the situation in the environment changes, a new plan can easily be generated without the necessity to adapt the individual motion primitives.

The main contribution of our approach is the novel motion planner, which can create plans for modular robots moving in complex environments. Our approach can be combined with various locomotion generators, which can
realize different gaits. The motion planner allows the use of simple motion primitives without the need to introduce sensory feedback to detect obstacles or other difficult areas. The proposed system allows the robot to visit distant places in complex environments, where obstacles need to be avoided. It is assumed, that the dynamics of the robots is not crucial here, as the studied robots move mostly using crawling and therefore the impact of dynamics is negligible and it can be handled on the CPG level. This allows us to focus this paper only to the motion planning problem.

3. Robotic platform and its configuration space

In this paper, the Cosmo modules [16] developed within the EU Symbion/Replicator projects [8], are utilized. Each module is equipped with basic sensors (camera, IR distance sensor, accelerometers), a Blackfin processor, two screw-drives for 2D locomotion and the main hinge for 3D locomotion. Power is provided by an internal battery or can be shared within the organism through a power bus. The modules can communicate via the ZigBee interface and, if connected, also using Ethernet. The modules are depicted in Fig. 1.

In the first step of the robot control using motion primitive \( p \), the settings \( x^p \) are retrieved from a look-up table. Then, the joint signals \( a_i(t) \) are generated to form the control signal \( u(t) \). These signals are distributed to individual modules.

4. Sampling-based motion planning for modular robots

The task of motion planning is to generate a feasible trajectory between an initial configuration \( q_{init} \in C_{free} \) and a goal configuration \( q_{goal} \in C_{free} \). Motion planning for modular robots with many degrees of freedom and many kinematic constraints can be solved using sampling-based methods. These methods solve the planning problem by creating a roadmap of free configurations in the configuration space \( C \). Due to the high dimension of the configuration space, which is equal to the DOFs of the robot, the space cannot be represented as a grid and standard state-search techniques cannot be applied. Instead, a roadmap is built by random sampling of the configuration space. The sampled configurations are classified as free (i.e., \( q \in C_{free} \)) or non-free using collision detection between the modules and the obstacles. The free samples are stored in the roadmap, and close configurations are connected by oriented edges using a local planner. The roadmap is represented as a graph, in which a path between two nodes describes a motion in the workspace. Sampling-based methods can find the motion of robots with many DOFs and arbitrary shape. Moreover, the kinematic and dynamic constraints can be taken into account by the methods [49].

The crucial part of the sampling-based planner is the local planner. Its task is to determine the control inputs that have to be applied to the actuators to reach the end-configuration of the edge. For simple robots, a line planner that connects two configurations by a line can be used. However, this cannot be applied to complex systems, as the line planner cannot ensure that two subsequent configurations are reachable by the real system. Moreover, this planner cannot provide control inputs to drive the individual actuators of a modular robot. To connect the nodes representing configurations of a modular robot and to obtain the control inputs, a motion model of the robot needs to be utilized.

In the Rapidly Exploring Random Tree (RRT) [41] algorithm, the motion model is used to create the roadmap. The RRT represents the roadmap as a tree \( T \) rooted in \( q_{init} \). The tree is grown as follows. In each iteration, a random sample \( q_{rand} \in C \) is generated. The nearest node \( q_{near} \in T \) in the tree is found and is expanded to obtain a set \( S \) of free configurations reachable from \( q_{near} \). From this set, a configuration \( q_{near} \) which is nearest to \( q_{rand} \) is selected and stored into the tree. The \( q_{near} \) is marked as its parent node. The algorithm terminates if the goal region defined by radius \( d_{goal} \) is approached. The basic RRT algorithm is listed in Alg. 1. The RRT algorithm has been employed for motion planning of various robotic
Algorithm 1: RRT algorithm.

Input: initial configuration \( q_{init} \), goal configuration \( q_{goal} \), goal region \( d_{goal} \), maximum number of iterations \( k \)

Output: path to the goal configuration or failure

1. \( T \).initialize(); // create empty configuration tree
2. \( T \).add\((q_{init})\); // add root node
3. for \( iteration = 1 \ldots k \) do
4.     \( q_{rand} \) = generate random sample in \( C \);
5.     \( q_{near} \) = nearestConfiguration(\( T \), \( q_{rand} \));
6.     \( q_{new} \) = expandConfiguration(\( q_{near} \), \( q_{rand} \));
7.     if \( q_{new} \) != NULL then
8.         \( T \).addNode(\( q_{new} \));
9.         \( T \).addEdge(\( q_{near} \), \( q_{new} \));
10. if \( \rho(\( q_{new} \), \( q_{goal} \)) < d_{goal} \) then
11.     return path in the tree from \( q_{init} \) to \( q_{goal} \);
12. end
13. end
14. end
15. return failure; // \( q_{goal} \) is not approached in \( k \) iterations

Algorithm 2: expandConfiguration() of basic RRT

Input: random configuration \( q_{rand} \), configuration to expand \( q_{near} \)

Output: configuration reachable from \( q_{near} \) or NULL

1. \( S = \emptyset \);
2. \( U \) = set of robot’s possible control inputs
3. for \( u \in U \) do
4.     \( q \) = apply input \( u \) to the motion model \( f(\( q_{near} \), \( u \)) \) over time \( \Delta t \);
5.     if collisionFree(\( q \)) then
6.         \( S = S \cup \{ q \} \);
7. end
8. return closest config. from \( S \) to \( q_{rand} \) or NULL if \( S = \emptyset \);

systems, and many modifications have been proposed; we refer to \([38, 50]\) for a survey of the modifications of this algorithm.

RRT quickly explores the configuration space, because the boundary nodes of the tree are more likely selected for the expansion. The probability that a node will be selected for the expansion is given by the area of its Voronoi cell (this is called the Voronoi bias). The nodes located on the boundary of the tree have larger Voronoi cells, hence they are chosen for the expansion more frequently than the nodes located inside the tree \([41]\).

In the expansion step (line 3 in Alg. 2), a motion model of the robot is used to obtain new configurations reachable from \( q_{near} \). A closed-form motion model of a modular robot cannot be easily derived due to complex kinematics and interactions with the terrain \([51]\). The motion model is therefore implemented using a physical simulation. Unlike a closed-form solution, a physical simulation allows us to model various types of robots taking into account the influence of environment. Before the simulation is executed, the objects representing the modules are placed into the desired initial positions described by \( q_{near} \). The simulation is run over time interval \( \Delta t \), and the resulting configuration \( q \) is constructed on the basis of their final positions. We refer to \([47]\) for implementation details of utilizing a physical simulation in sampling-based motion planning.

To allow the configuration space to be explored, suitable control inputs \( U \) have to be applied to \( q_{near} \) to obtain new configurations. Specifically in the case of mobile robots with only few control inputs (e.g., a differential drive robot), the set \( U \) can be made of all combinations of discretized input values. This approach however cannot be applied to modular robots due to the large number of control inputs, which would lead to large \( U \). In \([47]\), the authors suggest generating the inputs randomly. This leads to clumsy and inefficient motions, because randomly generated inputs move the robot slowly and chaotically. This requires many planning iterations, which increases planning time.

4.1. RRT-MP: Using motion primitives for the tree expansion

In this paper, we present a novel approach for motion planning of modular robots using predefined control inputs — motion primitives. Motion primitives provide short control signals \( u(t) \) for the robot. Each primitive is responsible for moving the robot in a different direction or in a different manner. For example, a crawling-like motion can be achieved using a CPG to move the robot in a given direction; another CPG can provide walking. To find an overall feasible trajectory towards a position in a complex environment with obstacles, a plan for switching these primitives needs to be determined by motion planning.

The idea of RRT-MP is to use the input signals \( u(t) \) of the motion primitives in the expansion step. To obtain new configurations reachable from \( q_{near} \), input signals \( u(t) \) are generated from the settings \( \tau^p \) for each of the employed motion primitives. The new configurations are obtained using a physical simulation, which is run over time intervals \( \tau^p \). The built tree consists of the primitives is shown in Fig. 2. The expansion step of the modified RRT-MP algorithm is listed in Alg. 3 and an example of the built tree is shown in Fig. 3.
The utilization of motion primitives brings significant advantages.

- We can assume that each primitive generates the desired motion effectively. Such primitives hence move the robot faster than randomly generated inputs. Planning with motion primitives is thus faster and less memory consuming than naive sampling-based planning with random control inputs.

- The input signals of the motion primitives do not need to be stored in the tree, because they can be generated on demand by the utilized CPG. Therefore, the edges in RRT-MP keep only information on the utilized primitive, which further decreases the memory requirements.

- The complexity of the expansion step is not influenced by the number of actuators, but only by the number of available motion primitives (size of the set \( \text{motion-Primitives} \)). Therefore, the RRT-MP is scalable and can also be used for systems with many actuators. As will be shown in Section 5, only a few motion primitives are needed in order to be able to cover large environments.

- The utilized primitives can be sub-optimal, and they can therefore be derived using standard optimization methods.

The last property is crucial, as modular robots can be reconfigured to various structures and it is never possible to preprogram their behaviors in advance. Moreover, all the possible behaviors cannot be stored in a limited memory. Their motions thus need to be achieved on demand using simple approaches like CPGs. To obtain a desired locomotion, the parameters of the CPGs have to be optimized. This can be solved using genetic algorithms and other evolutionary methods [24, 25, 52]. However, optimization of the gaits using these methods may require a considerable amount of time. Utilizing motion primitives in the planning process allows us to terminate the optimization early. Possible inefficiencies, such as slow speed of the motions or unwanted rotation during the motions, can be compensated by the planner.

The nearest neighbor rule used to select the nodes for expansion ensures, that the RRT-MP can explore the configuration space. This exploration is possible even if the utilized motion primitives move the robot slowly. In such a case, the tree will grow slowly and it will take more iterations to explore the configuration space and to reach the goal configuration. Therefore, effective motion primitives are preferred as they move a robot fast, which enables rapid growth of the tree. The selection of the motion primitives is up to the user. The primitives should be selected considering capabilities of the robot as well as suitability of the primitives for the given task. For example, primitives like ‘crawl’ or ‘turn’ are enough for moving on a terrain, but ‘stand-up’ primitive may be necessary if the robot has to reach a place above the ground. As the complexity of RRT-MP is given by the amount of the motion primitives, the user should select such a finite subset of primitives, that are believed to be necessary to accomplish the task.

As the final plan is a sequence of motion primitives, it has to be ensured that a primitive can be switched to another one. The current applicability of a primitive depends on the actual state of the robot, and also on the previously used primitive. For example, the primitive ‘stand-up’ can be applied if the robot is lying on the ground. However, it may not be straightforward to determine the proper states for complex shapes and it is more comfortable to decide the applicability based on the previously used primitive. For example, we can specify, that ‘stand-up’ can be applied after ‘lie-down’ has been applied without the necessity to define all possible states in which it can be applied. Let the predicate \( \text{isApplicableAt}(p, q) \) be true if a primitive \( p \) can control the robot from state \( q \). Also, let \( \text{isApplicableAfter}(p, p_{prev}) \) be true, if a primitive \( p \) can be applied after a primitive \( p_{prev} \). The predicates are used in the RRT-MP expansion step (line 3 in Alg. 3).

![Figure 3: Example of configuration trees created by implementation [45] of RRT for modular robots (left) and by RRT-MP with four motion primitives (right). The utilized primitives can be clearly seen in the tree built by RRT-MP.](image-url)

---

**Algorithm 3: expandConfiguration() of RRT-MP**

**Input:** random configuration \( q_{rand} \), configuration to expand \( q_{near} \), previously used primitive \( p_{prev} \)

**Output:** configuration reachable from \( q_{near} \)

```
1 S = ∅;
2 foreach p ∈ motionPrimitives do
3    if \( \text{isApplicableAt}(q_{near}, i) \land \text{isApplicableAfter}(p, p_{prev}) \) then
4        \( x^p \) = parameters of the motion primitive \( p \);
5        \( \tau \) = duration of \( p \);
6        \( u(t) = (a_1(t), \ldots, a_n(t)) \) using the locomotion generator;
7        \( q \) = apply input signal \( u(t) \) to the actuators over time \( \tau \) using physical simulation;
8        \( S = S \cup \{q\} \);
9    end
10 end
11 q_{new} = closest configuration from \( S \) to \( q_{rand} \);
12 return \( q_{new} \) or NULL if \( S = ∅ \);
```
4.2. Execution of the plans

The resulting trajectory can be found in the tree as a path from \( q_{\text{init}} \) to \( q'_{\text{goal}} \), where \( q'_{\text{goal}} \in T \) is the closest node in the tree towards \( q_{\text{goal}} \). The trajectory \( P = (q_{\text{init}}, p_1, \tau_1, q_2, p_2, \tau_2, \ldots, q_{K-1}, p_{K-1}, q_{\text{goal}}) \) consists of \( K \) segments, where \( p_i \) denotes the primitive that is used in segment \( i \) and \( \tau_i \) is its duration. After executing segment \( i \) the robot should be in configuration \( q_{i+1} \).

The RRT-MP planner relies on a physical simulation that implements the motion model of the robot. To plan a feasible trajectory, the simulation should be as precise as possible. Although the precision of the simulation can be increased utilizing detailed geometric models of the robots, detailed collision detection, the precise weights and torques of the motors and realistic friction models, there will still exist a reality gap. Moreover, a precise simulation is computationally demanding and it is not suitable for real-time motion planning. Therefore, a slightly imprecise but fast physical simulation is preferred.

To cope with the possibility of imprecise navigation, the trajectory should be executed in a closed loop. Before segment \( i \) is executed, the actual position of the robot is compared with position \( q_i \), which was predicted by the planner. If the distance between these is larger than a given threshold, it is not safe to execute the actual plan, and replanning from the actual position is invoked.

5. Simulated Experiments

The simulated experiments were executed on Intel Core2@2.2 GHz with 4GB of RAM under FreeBSD 8.2. Three types of modular robots are used in the experiments: S-Bot (a six-module organism with an asymmetric one-module leg), Quadropod (nine modules with four legs) and Lizard (14 modules with four legs). The robots are depicted in Fig. 4. The simulations are based on the Robot3D simulator [53], a specialized simulator for modular robots developed by the authors within the Symbrion/Replicator projects. The time step of the physical engine is set to 10 ms, which is a compromise between speed of the simulation and its stability. For the purposes of simplicity, the size of each module is 1 map unit and other lengths are described in this unit. The nearest neighbors are searched using the KD-Tree data structure provided by the MPNN library [54]. The distances between the configurations of the robots are measured using the 6D Euclidean metric.

The robots are equipped with four motion primitives \( p \in \{ \text{’go-ahead’, ’go-back’, ’go-left’, ’go-right’} \} \). The primitives are modeled using sine CPG, i.e., \( a_i(t) = A_i \sin(\omega_i t + \varphi_i) + B_i \). The parameters \( x^p = (A_i, \omega_i, \varphi_i, B_i) \) of the CPGs are found using the Particle Swarm Optimization approach [55], where \( 0 < A_i < \pi/2, 0.1 < \omega_i < 5 \) and \( 0 < \varphi_i < 2\pi \). The offsets \( B_i \) are set to zero for the purposes of this work. It should be noted that these four motion primitives can be also realized using other CPGs (e.g. [56, 24]). The simple sine CPG is chosen as its parameters can be easily tuned using the PSO technique.

The performance of the CPGs is evaluated using a physical simulation, which is run over a 5 sec time interval. The fitness function is then computed as the distance between the achieved position of the pivot modules and a desired position. In the experiment, the desired position is 7 map units away from the initial position of the robot in the direction of the desired motion. For example, for optimization of the ‘go-left’ motion, the desired goal position is 7 map units to the left of the robot. The primitives found for the desired positions are depicted on Fig. 5. The basic PSO algorithm is employed with 30 particles and 200 iterations.

To verify the advantages of utilizing motion primitives in the planning process, the performance of RRT-MP is compared with the RRT-CPG [45] and RRT-K [47] methods. In RRT-CPG, the actuators are also driven by sin CPGs. However, their parameters are not predefined as in RRT-MP, but are generated randomly in each expansion step. In the RRT-K approach, the actuators are not controlled by a periodic signal. Instead, they are moved to a fixed angle \( a_i \), which is generated randomly in each expansion step within the range \((-\pi/2, \pi/2)\). To enable a time comparison of all the three planners, the complexity of their expansion steps should be equivalent. In RRT-MP, the complexity is determined by the number of utilized motion primitives. Therefore, RRT-CPG generates four random CPG settings in each iteration and RRT-K generates four random vectors of desired angles in each expansion step. The maximum number of planning iterations is \( k = 5000 \) for all algorithms.

The performance of the planners can be described by the planning time required to reach the goal region and also by the quality of the solutions that are found (e.g. the trajectory length). Another important aspect of the planners is their reliability, which is described by their success rate of finding a solution. Stochastic RRT-based algorithms may fail to reach the goal region defined by radius \( d_{\text{goal}} = 1 \) map unit around the goal configuration, even if such a solution exists. The success ratio describes the number of successful trials where the goal is approached by the tree, over number of all trials.

The performance of the motion planners is tested in several scenarios, which are chosen to simulate various situ-
ations. In the Plane scenario and in the Multi start/goal scenario, the robots move in a planar environment without obstacles. Obstacles in the form of small steps are introduced in the Step scenario. Here, the robots have to overcome the steps. As our target application is a search and rescue scenario, motion planning on a rough surface is investigated in the Outdoor-like scenario. In the Step and Outdoor-like scenarios, the influence of the motion primitives on the quality of the resulting plan is also investigated.

5.1. Plane scenario

In the first experiment, the task is to find a trajectory between two fixed positions on a plane. The distance of the goal position is 9 map units. The average results from 30 trials are shown in Tab. 1, where Path length (measured in map units) denotes the length of the found trajectory, and its time duration is denoted by Path time. Runtime is the time needed to find the solution, and the row Iterations denotes the number of iterations that are needed to find the solution.

Based on the number of iterations, we can see that almost all algorithms solve the problem in less than the allowed number of iterations ($k = 5000$), which is also indicated by success ratios close to 100%. The only exception is the RRT-K algorithm, which fails to find a solution for the Quadropod robot, and the success ratio is only 15% in this case. The RRT-MP algorithm outperforms the other two methods in all measured aspects: it solves the problem in the shortest time and it generates the fastest trajectories. RRT-MP is able to find a solution for all robots in less than 20 iterations, which indicates that its motion primitives are very efficient. Although RRT-CPG can also provide a solution, its runtime is significantly worse than the runtime of RRT-MP, especially for the Lizard and Quadropod robots. While RRT-MP is able to move the robot over long distances in each expansion step, the movements achieved by RRT-CPG are less effective. This is indicated by the highest number of iterations over all algorithms. The runtime and the number of required planning iterations of RRT-CPG increase with the size of the robot.

5.2. Step scenario

In this scenario, the task is to verify the applicability of the RRT-K and RRT-MP methods in environments with obstacles. Here, the environment has the same bounding box as in the Plane scenario, and it contains three steps which divide the environment into two areas. The task of the motion planning is to find a trajectory to overcome these steps. The influence of the vocabulary of motion primitives on the quality of the planning process is studied. Therefore, RRT-MP is tested in two variants: RRT-MP-r and RRT-MP-a. Both are used with the same primitives: ‘go-left’, ‘go-right’, ‘go-back’ and they differ in the fourth primitive. The RRT-MP-r method is equipped with a ‘raise-head’ primitive; while RRT-MP-a uses the ‘go-ahead’ primitive instead. The difference between RRT-MP-r and RRT-MP-a is depicted in Fig. 6.

The average results from 30 trials can be found in Tab. 2 and examples of solutions are shown in Fig. 7. The RRT-K and RRT-MP-r find the solution in all 30 trials, which is indicated by the 100% success ratio. Due to the large deviation of the results, the methods are compared using T-test statistics. The p-values, listed in the last column, show that the runtimes are same (under significance level $\alpha = 0.05$), however the other measures differ as the p-values are zero. We can conclude, that RRT-MP-r has better performance, because its runtime is statistically the same as the runtime of RRT-K, but it provides better trajectories than the RRT-K.

The results show that the performance of RRT-MP is determined by the capabilities of the utilized motion primitives. Therefore, RRT-MP-a fails to overcome the steps, as it is equipped only with primitives suitable for a planar environment without obstacles that require the body to be raised. By contrast, RRT-MP-r is able to reach the goal position and to overcome the steps. It should be noted, that only one primitive in RRT-MP-r can raise the head and therefore climb the step, while the other primitives can only move the robot on the ground. This shows the importance of using primitives suitable for a given situation.

5.3. Multi start/goal scenario

The proposed RRT-based planners are intended as global planners, i.e., they should provide locomotion between any two positions in an environment. The performance of the planners for single start/goal positions has been presented in previous experiments. However, this performance does not indicate average performance, when
<table>
<thead>
<tr>
<th></th>
<th>RRT-MP</th>
<th>RRT-CPG</th>
<th>RRT-K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>dev</td>
<td>mean</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dev</td>
</tr>
<tr>
<td>S-bot</td>
<td>17.20</td>
<td>4.90</td>
<td>63.13</td>
</tr>
<tr>
<td></td>
<td>54.35</td>
<td>22.87</td>
<td></td>
</tr>
<tr>
<td>Path length</td>
<td>6.60</td>
<td>4.28</td>
<td>80.23</td>
</tr>
<tr>
<td></td>
<td>1412.17</td>
<td>1174.51</td>
<td></td>
</tr>
<tr>
<td>Iterations</td>
<td>4.53</td>
<td>1.63</td>
<td>23.33</td>
</tr>
<tr>
<td></td>
<td>73.63</td>
<td>37.10</td>
<td></td>
</tr>
<tr>
<td>Path time [s]</td>
<td>1.78</td>
<td>0.96</td>
<td>44.05</td>
</tr>
<tr>
<td></td>
<td>108.30</td>
<td>93.17</td>
<td></td>
</tr>
<tr>
<td>Success ratio</td>
<td>100.00</td>
<td>100.00</td>
<td>108.30</td>
</tr>
<tr>
<td>Runtime [s]</td>
<td>4.53</td>
<td>1.63</td>
<td>23.33</td>
</tr>
<tr>
<td></td>
<td>73.63</td>
<td>37.10</td>
<td></td>
</tr>
<tr>
<td>Quadropod</td>
<td>26.54</td>
<td>6.05</td>
<td>120.95</td>
</tr>
<tr>
<td></td>
<td>186.84</td>
<td>105.73</td>
<td></td>
</tr>
<tr>
<td>Path length</td>
<td>18.85</td>
<td>25.06</td>
<td>130.30</td>
</tr>
<tr>
<td></td>
<td>4999.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>Iterations</td>
<td>6.15</td>
<td>1.53</td>
<td>50.75</td>
</tr>
<tr>
<td></td>
<td>201.40</td>
<td>109.14</td>
<td></td>
</tr>
<tr>
<td>Path time [s]</td>
<td>12.85</td>
<td>16.04</td>
<td>239.34</td>
</tr>
<tr>
<td></td>
<td>1401.15</td>
<td>530.27</td>
<td></td>
</tr>
<tr>
<td>Success ratio</td>
<td>100.00</td>
<td>100.00</td>
<td>15.00</td>
</tr>
<tr>
<td>Runtime [s]</td>
<td>9.10</td>
<td>1.40</td>
<td>211.95</td>
</tr>
<tr>
<td></td>
<td>191.05</td>
<td>46.58</td>
<td></td>
</tr>
<tr>
<td>Lizard</td>
<td>35.15</td>
<td>4.24</td>
<td>382.72</td>
</tr>
<tr>
<td></td>
<td>103.54</td>
<td>26.77</td>
<td></td>
</tr>
<tr>
<td>Path length</td>
<td>13.93</td>
<td>3.79</td>
<td>2270.29</td>
</tr>
<tr>
<td></td>
<td>943.00</td>
<td>459.32</td>
<td></td>
</tr>
<tr>
<td>Iterations</td>
<td>9.10</td>
<td>1.40</td>
<td>211.95</td>
</tr>
<tr>
<td></td>
<td>191.05</td>
<td>46.58</td>
<td></td>
</tr>
<tr>
<td>Path time [s]</td>
<td>30.31</td>
<td>7.66</td>
<td>12488.80</td>
</tr>
<tr>
<td></td>
<td>696.09</td>
<td>388.38</td>
<td></td>
</tr>
<tr>
<td>Success ratio</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Runtime [s]</td>
<td>749.52</td>
<td>623.79</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Result for the Plane scenario.

<table>
<thead>
<tr>
<th></th>
<th>RRT-MP-r</th>
<th>RRT-MP-a</th>
<th>RRT-K</th>
<th>p-val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>dev</td>
<td>mean</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dev</td>
<td></td>
</tr>
<tr>
<td>Quadropod</td>
<td>138.64</td>
<td>32.18</td>
<td>74.40</td>
<td>27.97</td>
</tr>
<tr>
<td></td>
<td>226.08</td>
<td>17.54</td>
<td></td>
<td>0.0000</td>
</tr>
<tr>
<td>Path length</td>
<td>624.93</td>
<td>517.63</td>
<td>3780.00</td>
<td>40.00</td>
</tr>
<tr>
<td></td>
<td>2985.67</td>
<td>640.10</td>
<td></td>
<td>0.0000</td>
</tr>
<tr>
<td>Iterations</td>
<td>32.33</td>
<td>7.07</td>
<td>18.20</td>
<td>6.32</td>
</tr>
<tr>
<td></td>
<td>381.00</td>
<td>29.10</td>
<td></td>
<td>0.0000</td>
</tr>
<tr>
<td>Path time [s]</td>
<td>12.85</td>
<td>16.04</td>
<td>239.34</td>
<td>138.35</td>
</tr>
<tr>
<td></td>
<td>1401.15</td>
<td>530.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Success ratio</td>
<td>100.00</td>
<td>100.00</td>
<td>15.00</td>
<td></td>
</tr>
<tr>
<td>Runtime [s]</td>
<td>749.52</td>
<td>623.79</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Results on the Steps environment. The last column is the p-value of a t-test computed between the RRT-MP-r and RRT-K columns.

Figure 7: Example of solutions found in the Steps environment by the RRT-MP-r (a) and RRT-K (b) algorithms. The configuration tree is in green, the resulting trajectory is depicted in blue. Only the position of the pivot module is visualized.

multiple starts/goals are considered. For example, let us consider an RRT-MP planner with only the ‘go-ahead’ primitive. This planner will be efficient in situations where the goal configuration can be achieved by moving forward; however, it cannot provide any solution when the goal is behind the robot. To verify the ability of the RRT-MP planner to provide global motion plans, an extensive test using multiple start/goals needs to be performed.

First, a set $T' = \{(s_i, g'_i)\}$ of $M = 120$ random start/goal pairs is generated. The configurations $s_i$ and $g'_i$ are filled with random positions $(x, y)$ and random rotations $\gamma$. The angles $a_i$ are set to zero. To ensure that each goal is reachable from its corresponding start $s_i$, motion planning using RRT-MP is performed starting from $s_i$. Then, the nearest configuration $g_i \in T'$ from the built tree towards $g'_i$ is retrieved. The final test set $T = \{(s_i, g_i)\}$, where $i = 1, \ldots, M$ is then created using $s_i$ and the configurations reached by the tree. Start/goal pairs, whose distance is less than the ending condition of the RRT ($d_{goal} < 1$ map unit) are not used in the set $T$.

For each pair $(s_i, g_i)$, the planners are run for 30 trials and the results are averaged over these trials. The resulting measurements represent average behavior in the environment and they are represented as boxplots (the average values valid for 90% of the start/goal configurations are shown by the filled boxes, and the thick black line represents the median).

The success ratio is depicted in Fig. 8. The RRT-MP planner is able to find a solution for all start/goal pairs for all three robots. The RRT-CPG fails only a few times for
the Lizard robot. Unlike these algorithms, RRT-K cannot find a solution in several cases for S-bot and Quadropod robots. Moreover, the 90 % interval of success ratio for the Quadropod robot is from 0 % to 100 % and the median is 80 %. This means that for 50 % of the start/goal pairs, the success ratio of the RRT-K planner is worse than 80 %.

The averaged planning times over all start/goal pairs are presented in Tab. 3, and the corresponding boxplots are shown in Fig. 9. Beside having the highest success ratio, the RRT-MP found the solutions in the shortest time. The obtained solutions are also better than the solutions computed by the RRT-K and RRT-CPG methods.

The performance of RRT-MP/P and RRT-MP/S is almost identical, so the T-test is used to accept or reject the hypothesis that the two algorithms behave in the same way (i.e., they have equal runtimes and produces trajectories of same quality). The mean values and the deviation of the measured indicators are given in Tab. 4 and the p-values of the t-test between the RRT-MP/P and RRT-MP/S algorithms are significantly shorter than the runtimes of the other two methods. Moreover, trajectories computed by RRT-MP/P and RRT-MP/S algorithms are significantly shorter than the trajectories computed by other methods.

The speed of the algorithms and the quality of the solutions are depicted in Fig. 11. Due to the low success ratio of RRT-MP/P and RRT-MP/S with the S-bot robot, the planning time and its deviation are higher than in the RRT-K method. The efficiency of the RRT-MP methods is significantly better for the Lizard and Quadropod robots, which is also indicated by 100 % success ratio. For the Lizard and Quadropod robots, the runtimes of the RRT-MP/P and RRT-MP/S algorithms are significantly shorter than the runtimes of the other two methods. Moreover, trajectories computed by RRT-MP/P and RRT-MP/S are significantly shorter than the trajectories computed by other methods.

The performance of RRT-MP/P and RRT-MP/S is almost identical, so the T-test is used to accept or reject the hypothesis that the two algorithms behave in the same way (i.e., they have equal runtimes and produces trajectories of same quality). The mean values and the deviation of the measured indicators are given in Tab. 4 and the p-values of the t-test between the RRT-MP/P and RRT-MP/S are given in the last column. We can conclude that both algorithms provide the same results for the Quadropod robot, as all three p-values are higher than the significance level $\alpha = 0.05$. This is caused by higher
The manipulated primitives of the Quadropod in comparison to other two robots. The Quadropod robot can move on the surface sufficiently with both types of primitives. Contrary, the quality of the plans created by RRT-MP/S and RRT-MP/P differs for the Lizard and S-bot robots. The results showed that although the planners can provide a solution with both types of primitives, the quality of the created plans is higher if suitable primitives are used, i.e., primitives learned on the surface should be used when the robot moves on the surface.

### 5.5. Discussion

The simulated experiments showed that the proposed RRT-MP motion planner significantly outperforms both RRT-CPG and the naive RRT-K in planning time as well as in the quality of the solutions. RRT-MP explores the configuration space only on the basis of the motion primitives, so the performance of RRT-MP is significantly influenced by the quality of the primitives.

This is verified in the Steps and Outdoor-like scenarios. As was shown in the Steps experiment, the robot is not able to overcome the steps if no 'raise-head' primitive is integrated into the RRT-MP algorithm. Similar results are obtained on the Surface scenario with the S-bot robot.
Table 4: Comparison between RRT-MP/P and RRT-MP/S using t-test. We conclude, that both algorithms produce statistically same results if the p-value is greater than 0.05.

<table>
<thead>
<tr>
<th></th>
<th>RRT-MP/P</th>
<th>RRT-MP/S</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-bot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime [s]</td>
<td>415.22</td>
<td>375.73</td>
<td>736.09</td>
</tr>
<tr>
<td>Path time [s]</td>
<td>26.84</td>
<td>14.06</td>
<td>30.53</td>
</tr>
<tr>
<td>Path length</td>
<td>93.42</td>
<td>47.24</td>
<td>82.98</td>
</tr>
<tr>
<td>Lizard</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime [s]</td>
<td>250.03</td>
<td>157.98</td>
<td>176.40</td>
</tr>
<tr>
<td>Path time [s]</td>
<td>16.08</td>
<td>6.48</td>
<td>13.85</td>
</tr>
<tr>
<td>Path length</td>
<td>59.11</td>
<td>24.94</td>
<td>48.46</td>
</tr>
<tr>
<td>Quad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime [s]</td>
<td>63.48</td>
<td>34.71</td>
<td>58.72</td>
</tr>
<tr>
<td>Path time [s]</td>
<td>12.01</td>
<td>5.58</td>
<td>10.52</td>
</tr>
<tr>
<td>Path length</td>
<td>52.50</td>
<td>25.08</td>
<td>51.72</td>
</tr>
</tbody>
</table>

Figure 11: Runtimes (top row) and path times (bottom row) of multiple start/goal test on surface.

where two types of motion primitives are used: primitives learned on the plane and primitives learned on the surface. As the size of the S-bot robot is comparable with the size of the hills and dips on the surface, it climbs down into a dip in certain situations. As the primitives learned on the plane are used, the robot is not able to climb up from the dip and the success ratio of the planning algorithm decreases. This situation occurs even if the surface-learned primitives are used, as these reflect only on a part of the surface and are not ready to cope with the dips. We can conclude, that the motion primitives need to be prepared carefully, taking into account the environment that the robot operates in.

The RRT-K approach is able to find solutions in the
Steps and Surface experiments, even if its success ratio is lower than the success ratio of RRT-MP. However, as the number of modules of the robots increases, the number of possible combinations of control inputs grows exponentially. As RRT-K uses only a predefined fixed number of $K$ random combinations, it has less chance to generate a good control input randomly. This slows down the exploration process and consequently increases the time required to find a solution.

While motion planning for simple robots (such as the S-bot robot) can be successfully solved by the RRT-K method (or even by the RRT-CPG method), the planning of more complex robots e.g. Lizards or Quadropods, should rather be solved using RRT-MP. The proposed RRT-MP algorithm has been evaluated as the most effective for modular robots in general, since the possibility to form complex bodies is property embodied in them.

6. Hardware verification

In the hardware experiment, a small Cross robot made of five Cosmo modules is utilized (the robot is depicted in Fig. 13). The robot moves using the same four motion primitives as in the simulated scenarios. The execution of each primitive takes 30 seconds on the real robot. The control signals $a_i(t), i = 1, \ldots, n$, where $n$ is the number of modules, define the desired position of each joint. The joints are controlled by a P regulator. The physical simulation used in the motion planner uses the same P controller which allows us to control the robot using the motion primitives obtained from the simulation. Examples of the control signals are depicted in Fig. 14.

The task of the planner is to find a path in an environment 3x5 meters in size with static obstacles (Fig. 15a). After a plan is created, the robot starts to execute it as described in Section 4.2. To allow the robot to replan when it deviates from the planned trajectory, its position is determined using a camera-based localization system [57]. Screenshots from the navigation phase are shown in Fig. 15 and the corresponding control signals are shown in Fig. 16.

Planning between two fixed positions is repeated 10 times. The average planning time in this environment is
4.5 seconds, while the time needed to execute the trajectory is of the order of minutes. The planning time is thus negligible in comparison to the execution time, thus the replanning is sufficiently fast. A video from the experiment can be found at [58].

7. Conclusion

This paper has presented a novel motion planning technique designed for modular robots. The proposed method, called RRT-MP, is based on a sampling-based principle that solves the motion planning problem by random sampling of the configuration space of the robot. Due to the high number of actuators, it became intractable to examine all possible combinations of control inputs of the modular robot. We therefore suggest using local motion primitives. This allows the RRT-MP to effectively explore the configuration space. These motion primitives should provide only short motion in the vicinity of the robot and they can be implemented e.g. using CPGs.

The performance of the planner has been verified in simulated experiments and also in real experiments. The simulated experiments showed that RRT-MP significantly outperforms other planners in runtime and also in success rate. A detailed investigation of its performance in the Step and Surface scenarios showed the influence of the primitives on the overall ability of the planner to solve the planning task.

In the experimental verification, a simpler planner was applied and the results prove, that this simplified solution...
can be efficiently applied with modular robots consisting of few modules. However, as the number of modules increases, the performance of other methods decreases significantly. RRT-MP can find a plan with a high success ratio. RRT-MP is thus an enabling technology for motion planning of complex modular robots. The resulting plans provided by RRT-MP have been executed on a real robot made of five modules.

The proposed motion planner evaluates motions of the robots using a physical simulation. Therefore, a map of the environment (described e.g. as a triangle mesh) is required to be able to compute a motion plan. The motion planner is run before the mission and its result, which is a sequence of motion primitives, is then sent to the robot for execution. During the execution of the plan, a central module of the robot distributes the control signals to all other modules. This can be realized using a simple processor and therefore the robots do not need to be equipped with powerful computers to realize the motions.

In this paper we assume, that the robots move using joint-control locomotion. The proposed planning system is therefore suitable for robots without self-reconfiguration capabilities, like swimming robots or snake-like robots. The system can be extended to use self-reconfiguration of modular robots. In such a case, some of the motion primitives realize the reconfiguration to new shapes [17, 18, 19, 20, 21, 22]. As the reconfiguration change shapes of the robots, the motion primitives realizing joint-control locomotion need to be optimized also for the newly created shapes, which can be time consuming. To be able to keep the amount of possible robotic shapes low, and therefore to be able to optimize motion primitives for each shape in a short time, only reconfiguration leading to suitable shapes should be preferred. The methods for reconfiguration planning as well as methods to determine quality of newly generated robotic shapes are beyond the scope of this paper and these will be investigated in our future work.

8. Acknowledgments

This work has been supported by MSMT grant No. 7AMB14DE007 and by Grant Agency of the Czech Republic under project No. P103-12/P756. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated. We would like to thank the anonymous reviews for their valuable comments.


J. Conradt, P. Varshavskaya, Distributed central pattern generator control for a serpentine robot, in: International Conference on Artificial Neural Networks (ICANN 2003).


