IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 20, 2019, accepted July 17, 2019, date of publication August 5, 2019, date of current version August 19, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2933178

Efficient Modeling and Evaluation of Constraints in Path Planning for Multi-Legged Walking Robots

DOMINIK BELTER, (Member, IEEE)
Institute of Control, Robotics and Information Engineering, Poznań University of Technology, 60-965 Poznan, Poland
e-mail: dominik.belter@put.poznan.pl

This work was supported in part by the DS-BP at the Poznan University of Technology (article processing charge and research on path planning) under Grant 04/45/DSPB/0196, in part by the EU Horizon 2020 project THING (general constraint modeling), and in part by the Czech Science Foundation (GA.R) under Grant 15-09600Y.

ABSTRACT In this article, we propose a new method for constraints evaluation during the path planning of a multi-legged walking robot. We propose the application of Gaussian Mixtures (GM) to build constraint models. With the proposed analytical constraint function we can compute the gradient and move the robot out of the bound configuration. The proposed method allows checking self-collisions, and a workspace of the robot in a few microseconds and efficiently plan the motion of the robot. The proposed path planning method uses RRT-Connect framework. We show how to efficiently apply the constraints evaluation methods to optimize the posture of the robot, optimize the position of the feet during the swing phase and efficiently explore the search space. We compare the proposed method with the standard constraints evaluation algorithms. Finally, we present the results of path planning for the six-legged robot in various scenarios to show properties of the proposed motion planning algorithm with GM-based constraints evaluation.

INDEX TERMS Autonomous systems, legged locomotion, path planning, robot motion.

I. INTRODUCTION

Six-legged robots mimic insects which have enormous locomotion capabilities on rough terrains. Even simple insects can deal with obstacles and find a path to the goal position. They can select stable footholds and avoid larger obstacles. They are also aware of their capabilities. Animals can avoid self-collisions and collisions with the ground, preserve stability, and move their legs within the maximal range given by the leg's kinematic.

Legged robots still have difficulties with walking on rough terrains. The problem of rough terrain locomotion becomes more challenging when we consider path planning in advance. The goal of path planning is to find a feasible motion of the robot to the goal position with the given model of the environment. Each position of the robot on the path to the goal should be fully determined. The configuration of the robot allows for constraints checking and determining the feasible motion of the robot for the given terrain model.

The associate editor coordinating the review of this manuscript and approving it for publication was Luigi Biagiotti.

In this article, we consider six-legged locomotion. The six-legged robots can walk statically stable and relatively fast on rough terrain. However, we identified two main difficulties related to the path planning problem. The first challenge is related to the size of the search space. A state of six-legged robot with three degrees of freedom for each leg is defined in 24-dimensional space. If we determine the sequence of the robot's states between initial and goal position, the problem becomes even more challenging. The second difficulty is related to the constraint checking. Along the path, each state of the robot should be collision-free C_{free}, statically stable C_{stab}, and inside the workspace C_{work}. The constraints checking is time-consuming and it is difficult to handle them online during path planning. Thus, the popular methods for path planning of walking robot use simplified transition costs [1], [34] instead of full constraint checking.

Constraints limit the motion of the robot, but with motion planning, we can avoid deadlocks – states of the robot when the robot cannot perform a defined motion. The motion of the robot may be impossible because of the lack of static stability, self-collisions, collisions with the obstacles and limited kinematic range of the robot. Motion planning, in contrast to

the reactive control strategy, allows moving the robot through a feasible region in the high dimensional space. We use RRT-Connect [30], an efficient randomized sampling-based method to find accessible states of the robot and connect them to create a path from the initial to the goal pose.

A. PROBLEM STATEMENT

We consider the problem of constraints evaluation during path planning for a multi-legged walking robot. We assume that the robot is equipped with a perception system which allows building the model of the environment and localize the robot. Our robot uses an elevation map to model the terrain and the utilized mapping method is described in our previous work [7]. The map is updated once per two seconds. The maximal range of the map is set to 10 m. The robot is also capable to localize itself during walking in the previously unknown environment [5]. Our robot uses optimization-based legged odometry to utilize data from the interoceptive sensors. Then, the data from legged odometry are integrated with the data obtained from the visual-based localization system. We use graph-based optimization to integrate data from legged odometry and Visual SLAM (ORB-SLAM2). The localization system continuously provides data about the state of the robot and provides globally consistent trajectories enabled by the loop closure mechanism [5].

In this article, we deal with the problem of path planning and constraints evaluation for walking robots. The path planning method uses the elevation map and the model of the robot as an input. The algorithm utilizes the kinematic and CAD model of the robot and searches for the feasible path of the robot in the multi-dimensional (24-dimensional) search space. We propose efficient methods which allow exploring the search space. More importantly, we focus on the efficient and accurate constraints evaluation which allows finding a collision-free and stable path from the initial to goal state in real-time.

B. APPROACH AND CONTRIBUTION

The main contribution of this article is the Gaussian Mixture-based method used for modeling constraints of the walking robot. We utilize the Gaussian Mixture regression to create the constraints model (collision checking, and workspace evaluation) of the robot. With the proposed model, the constraints evaluation is significantly speeded up and can be used for real-time path planning of the legged robot. We show how the proposed constraints evaluation methods can be used in the motion planning framework to return a kinematically feasible path from the initial to the goal pose of the robot. The proposed motion planning method can find the path which allows climbing the obstacles. The method also can find the path around the obstacle if the obstacle is too high for the robot.

We also contribute to the problem of general path planning of legged robots. In this article, we extend our previous work related to path planning [6], [9], [12]. The proposed method is modular and utilizes foothold selection, posture optimization, and feet trajectory planning during the swing phase to boost path planning methods and explore efficiently the search space. With the proposed strategy the path of the robot can be found in a time which makes the method suitable for the real robots. Finally, we provide the experimental evaluation of the proposed method with the real six-legged walking robot. We also compare the various configurations of the planner and constraint evaluation to demonstrate the properties of the proposed system.

II. RELATED WORK

A. LEGGED LOCOMOTION

We can identify two main approaches to legged locomotion: planning-based and reactive-based. The first approach is based on path planning where each action of the robot is planned in advance using the environment model. This is the opposite method to reactive behavior, for example, the approach where the robot classifies the terrain using onboard sensors and changes the direction of walking to follow the preferred type of the terrain [42].

The example path planner for the quadruped robot presented in [26] decomposes the problem into footstep planning, pose finding, body trajectory generator, and foot trajectory planner. Similar hierarchical architecture for the LittleDog robot is proposed in [51]. We can recognize similar modules but the main difference is in the path-planning meta-algorithm. In [26], the algorithm generates a cost map according to the assessment from the foothold selection module. Then, the sequence of the robot's postures is planned between the initial and goal pose of the robot. Both methods [26], [51] use Anytime Repairing A* (ARA*) algorithm [31] that finds a suboptimal path very quickly using a loose bound and then tightens the bound to improve the results. Finally, the Zero-Moment Point stability criterion and Covariant Hamiltonian Optimization for motion planning [39] are used to generate the robot trajectory. In contrast, our method plans full-body motion of the robot for each step and evaluates all constraints as quickly as possible. We found this approach more efficient because we avoid exploring regions which are promising from the beginning (good footholds) but in fact are not traversable by the robot (transition between stable poses is impossible because of high obstacles).

The motion planning architecture designed for the LittleDog robot is divided into two layers: body path planning and footstep planning [29]. In the proposed approach, the single-stage architecture is considered to define the full state of the robot during planning each elementary motion (step). Our goal is to avoid the disadvantages of the two-stage planners which explore regions which are later rejected in the second phase of the path planning.

The second approach to legged locomotion is based on dynamic walking. In this case, the trajectory of the robot is planned but very often without information about the 3D model of the terrain (map). This approach is popular on the

biped or quadruped robots which have limited capability of statically stable walking. The example of dynamic motion planning is based on trajectory optimization through constraints forces [38]. The complex motion of such robots can be obtained by optimizing simultaneously the contact and behavior of the robot [32]. However, the kino-dynamic planning is very often considered in the configuration space [37] and rarely the model of the rough terrain during motion planning is taken into account [49]. In the proposed research, we are interested in the path planning method which precisely determines the configuration of the robot on rough terrain. In such a case, the robot walks relatively slowly thus we can ignore the dynamic properties of the robot but we take into account the shape of the obstacles.

B. CONSTRAINTS EVALUATION

When the motion of the robot is determined the planning algorithm has to check if the planned action can be executed by the robot. The constraints can be evaluated using Convolutional Neural Networks (CNN). The HyQ [47] and ANYmal [4] robots use CNN to select footholds and evaluate the constraints at the same time. The CNNs have some advantages over our method because they can handle multi-dimensional problems and can work directly with elevation maps (images). However, in this article, we solve low-dimensional constraints checking (despite the fact that the state of the robot is defined in the multi-dimensional space) and the proposed methodology (Gaussian Mixtures) is well fitted to the problem. Thus, our method is faster and allows to easily compute the gradient of the output function, which is not straightforward with the existing CNN-based methods.

The capability of the robot to change the state over the terrain is also called traversability. The robot model, terrain model, kinematic and stability constraints should be taken into account to determine traversability. An extended literature review which evaluates various approaches is presented in [34]. Rarely, motion planning methods evaluate all possible constraints. The constraints checking is oversimplified by introducing coefficients which describe the traversability cost. Such an approach is justified when the robot has a simplified representation of the terrain only. However, the currently available sensors allow to precisely model the terrain in the neighborhood of the robot, and thus the planner should precisely check if the robot can safely change the configuration to ensure the feasibility of the motion.

Most of the path planning methods for legged robots compute simplified transition cost. On the HyQ walking robot, the cost of transition depends on the footholds and the projection of the center of the gravity of the robot [1]. Stelzer *et al.* compute the danger factor which depends on the slope inclination, terrain roughness and local height difference of the elevation map [43]. The obtained factor is used as a traversability cost in the A* planner applied to path planning of the six-legged walking robot. Also, the BigDog robot navigation system uses 2D cost map which allows planning the motion of the robot on the horizontal plane [50]. The system avoids obstacles like trees and the robot deals with small terrain irregularities using a reactive controller.

Tonneau et al. approximate the workspace of the robot using simplified convex hull [45]. The convex hull is obtained by sampling valid joint configurations and generating a point cloud from the obtained end-effector poses. Then, the obtained convex hull is simplified using Blender software [45]. In our work, we create the model of the workspace using Gaussian Mixture (GM). Our methods allow checking constraint but also allow finding the gradient to increase the safety margin. The constraints checking depends also on the representation of the environment. On the six-legged robot Weaver, the posture of the robot is optimized to avoid collisions with the obstacles [14]. The obstacles are represented by the multi-level elevation map but during posture optimization, the signed distance field is used to represent obstacles around the robot.

C. PATH PLANNING

Each of the path planning methods for the legged robot can be assigned to two groups: graph search-based (A*, ARA* [31], D* [44]) and sampling-based methods like RRT [30] and Probabilistic Roadmaps [27]. The first group of methods divides the search space into cells which are later used to construct a graph and search for the path which minimizes the defined cost. The precision of the planned path depends on the discretization of the search space. The main disadvantage of these methods comes from the fact that they require the value which defines the cost of transition. Such values can be defined taking into account hand tunned heuristics and it represents a simplified solution to path planning in multi-dimensional space.

The sampling-based path planning algorithms sample the continuous search space and try to connect the sampled nodes. This type of planning method might be time-consuming if the search space is multi-dimensional and is limited by multiple constraints. However, we chose this type of motion planning method because, in contrast to graph search-based methods, we can directly determine if the robot can traverse between neighboring nodes. Sometimes both types of path planning methods are mixed together to create more robust path planning algorithm such as [6], [48].

Sampling-based methods are prone to get stuck in environments with narrow passages. This problem can be handled by algorithms that guide the sampling-based methods through regions along the initial approximate solution [6], [48]. The hybrid method of motion planning for a hexapod robot is also presented in [15], where the randomly sampled robot configurations are guided by the optimal path found using A* planner.

Our work is most similar to the work on the ANYmal robot presented in [16]. The planner for the quadruped ANYmal robot finds footholds, optimizes the posture of the robot and finds collision-free trajectories of the feet above the obstacles. The planner also uses elevation map on the input. However, the planner presented in [16] and [17] uses a traditional

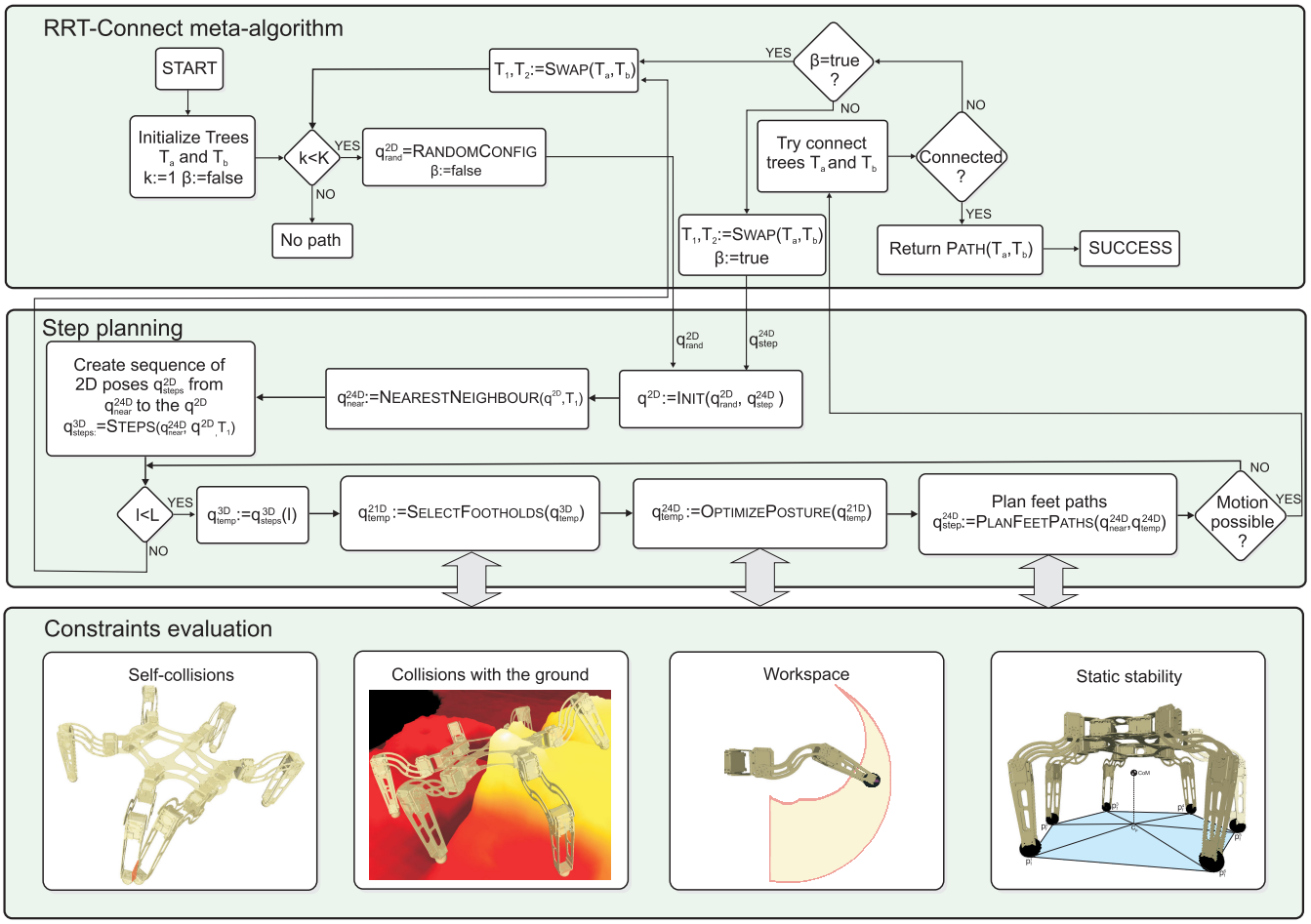

FIGURE 1. Block diagram of the whole motion planning system.

way of constraints evaluation. Moreover, the planner does not utilize any meta-planning algorithm which allows finding the path around the obstacle if the local planner fails. Havoutis et al. use sensory feedback and the elevation map to define the motion of the robot in advance [20]. In this case, the locomotion considers dynamic walking on rough terrain. Tonneau et al. proposed a multi-contact planner for multi-legged robots [45]. They define a geometric approximation of the contact manifold and sample the obtained search space using the RRT algorithm to find the contact points. Then, the sequence of the full-body configurations is generated.

III. PATH PLANNING

A. PATH PLANNING ARCHITECTURE

The block diagram of the proposed motion planning algorithm is presented in Fig. 1. We divided the proposed algorithm into three main blocks:

- RRT-Connect meta-algorithm,
- step planning,
- constraints evaluation.

The structure of the first block is based on the RRT-Connect algorithm proposed by Kuffner and LaValle [30]. The meta-algorithm is responsible for sampling and exploring the search space using two tree-like structures. The second block (called step planning) plans a single step (gait cycle) of the robot. For each step, the algorithm defines the full transition between the current and goal position of the robot at the end of the gait cycle. The last block is responsible for constraints checking.

The RRT-Connect meta-algorithm is summarized in Alg. 1. The algorithm extends consecutively two trees T_a and T_b. The algorithm samples from the two-dimensional space (x and y coordinates) only. The full state of the robot is defined by the dedicated procedures in the step planning procedure. At the beginning of the algorithm, we initialize the T_a and T_b trees. The root of the T_a tree is located at the current position of the robot. The root of the T_b tree is located in the goal position. In the first iteration of the algorithm ($k = 1$), we try to extend T_a ($T_1 := T_a$) and if the procedure is successful we extend the second tree ($T_2 := T_b$). In the second iteration ($k = 2$), the sequence is reversed – we extend T_b first ($T_1, T_2 := \texttt{Swap}(T_a, T_b)$).

The selected tree T_1 is extended in the random direction q^{2D}_{rand} sampled uniformly from the 2D xy search space using

Algorithm 1 RRT-Connect

Initialize trees T_a and T_b, k:=1, T_1:=T_a, T_2:=T_b
while $k < K$ **do**
  q_{rand}^{2D} := RandomConfig
  q_{step}^{24D} := StepPlanning(T_1, q_{rand}^{2D})
  if execution of q_{step}^{24D} is possible **then**
    if ConnectTrees(T_a,T_b)=SUCCESS **then**
      path := Path(T_a,T_b)
      break
    end if
    q_{step}^{24D} := StepPlanning(T_2, q_{step}^{24D})
  end if
  T_1, T_2 := Swap(T_a,T_b)
end while
return path or ∅ if not successful

RandomConfig method. Then, we plan the single step of the robot in the direction given by the sample q_{rand}^{2D} using StepPlanning(T_1, q_{rand}^{2D}) procedure. If the planned step can be executed, which means that we can avoid constraints when moving in the given direction, the algorithm tries to connect two trees in ConnectTrees(T_a,T_b). If T_a and T_b can be connected, the path is computed by the Path(T_a, T_b) procedure. The Path(T_a, T_b) procedure computes the final path by iterating over each tree from the node which is common for two trees to the root node. Note, that the path returned for the T_a tree has to be reversed to compute the forward motion of the robot (the procedures goes from the leaf to the root which is the current pose of the robot). If the connection of two trees is not possible, the algorithm tries to extend the second tree T_2 in the direction given by the recently added node q_{step}^{24D} of the first tree T_1.

The crucial procedure of our path planning framework is related to the step planning that is summarized in Alg. 2. The step planner tries to find the sequence of motions in the direction to q^{2D} given by the random node q_{rand}^{2D} or the node recently added q_{step}^{24D} to the first tree T_1. In the first step of the algorithm, we initialize q^{2D} according to the state of the RRT-Connect meta-algorithm. Then, we find the nearest node q_{near}^{24D} to q^{2D} in the tree T_1 using the NearestNeighbour procedure. In the next step, the Steps procedure finds the sequence of the body positions (x, y, and yaw angle) q_{steps}^{3D} in the direction from q_{near}^{24D} to q^{2D}. The number of considered body positions is set to L. The obtained positions correspond to the positions of the robot's body after the execution of the single step. The length of the step is limited by the maximal kinematic range of the robot in the given direction.

The planner checks the potential poses of the robot q_{steps}^{3D} in the direction to the goal node q^{2D} to find the motion which can be executed by the robot. We start from the position that corresponds to the longest step of the robot ($l = 1$) and then we gradually reduce the length of the step and the first successfully found path is accepted to support fast locomotion.

Algorithm 2 Step Planning

q^{2D} := Init(q_{rand}^{2D}, q_{step}^{24D})
q_{near}^{24D} := NearestNeighbour(q^{2D}, T_1)
q_{steps}^{3D} := Steps(q_{near}^{24D}, q^{2D})
l:=1
while $l < L$ **do**
  q_{temp}^{3D} := $q_{steps}^{3D}(l)$
  q_{temp}^{21D} := SelectFootholds(q_{temp}^{3D})
  if NOT MotionPossible(q_{temp}^{21D}) **then**
    l:=l+1
  else
    q_{temp}^{24D} := OptimizePosture(q_{temp}^{21D})
    if NOT MotionPossible(q_{temp}^{24D}) **then**
      l:=l+1
    else
      q_{step}^{24D} := PlanFeetPaths(q_{near}^{24D}, q_{temp}^{24D})
      if NOT MotionPossible(q_{step}^{24D}) **then**
        l:=l+1
      else
        return q_{step}^{24D}
      end if
    end if
  end if
end while
return No possible step

For each potential position of the robot, we select footholds using SelectFootholds procedure. Then, we optimize the posture of the robot using OptimizePosture to find the optimal inclination and the clearance between the ground and the robot's body. Finally, we plan the paths of the feet from the initial pose to the goal pose of the robot (PlanFeetPaths). In all experiments presented in the paper, the robot uses tripod gait but our controller also allows to define wave and free gaits. During the foothold selection, posture optimization and feet path planning we check all constraints to detect the motion which can't be executed. This strategy significantly reduces the execution time because we avoid exploring areas which are inaccessible to the robot.

B. SUPPORTING MODULES

In this section, we describe the modules used to support path planning: foothold selection module, posture optimization, and feet path planning procedure.

1) FOOTHOLD SELECTION

The SelectFootholds procedure determines the acceptable positions of the robot's feet on the ground. We use the method which was trained originally on the Ragno and Messor robots [11]. In this research, the procedure is adopted to the Messor II robot. The footholds are selected for the given 3D position q_{temp}^{3D} of the robot's body and the elevation map. At the beginning of the procedure, we compute the distance of the robot's body to the ground. At the beginning, we assume

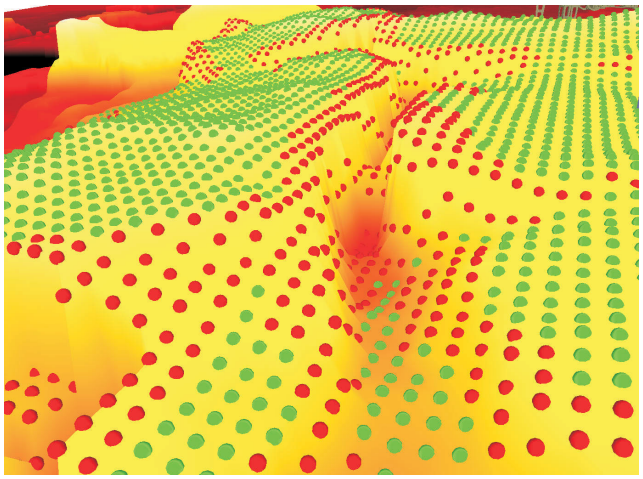

FIGURE 2. Example results from the foothold selection module. Green points represent acceptable footholds that are mainly located on flat terrain or small concavities while unacceptable footholds (red) are located on the edges and slopes.

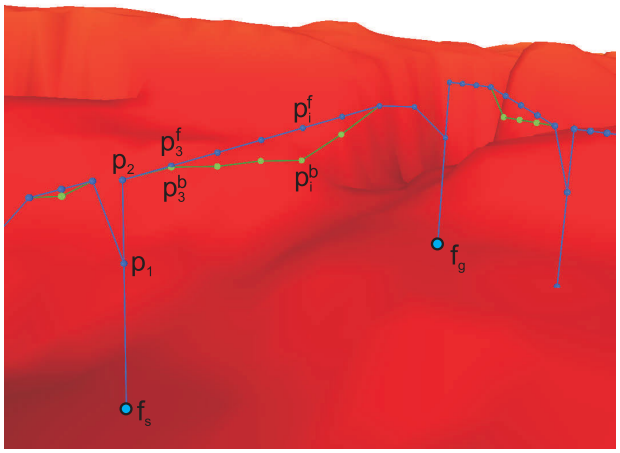

FIGURE 3. Example results from the foot planning module. Green lines represent the initial path planned between initial f_s and goal foothold f_g. The obtained final path is represented by the blue line.

the minimal distance of the robot's platform to the ground d_{min}. Then, we use the Gaussian mixture model from [11] to assess potential footholds for each leg.

The example foothold selection results are presented in Fig. 2, where the footholds are divided into two groups: acceptable and unacceptable footholds. In practice, we use the continuous output from the Gaussian mixture to select the best foothold. We also evaluate constraints for each leg: self-collision, leg's workspace, and kinematic margin. In contrast to [11] all constraints are represented by Gaussian Mixtures.

2) POSTURE OPTIMIZATION

During the foothold selection, the orientation of the robot's body is horizontal. Also, the distance to the ground is initialized using a simple heuristic. In the posture optimization procedure, we determine the vector $\mathbf{p}_R = [roll_R, pitch_R, z_R]^T$ which defines the optimal inclination of the robot's body (roll and pitch angles) and distance to the ground z_R:

$$\underset{\mathbf{p}_R}{\operatorname{argmax}}\, d^{KM}(q^{24D}),\ q^{24D} \in C_{free} \cap C_{stab} \cap C_{work} \qquad (1)$$

where $d^{KM}(q^{24D})$ is the kinematic margin for the given configuration of the robot, C_{free} is the collision-free space, C_{work} is the workspace of the robot, and C_{stab} is the statically stable configuration.

The PSO algorithm is employed to solve (1). The procedure is described originally in [10] and in this work, we use the same procedure with the improved constraints checking presented in section IV.

3) FOOT PATH PLANNING

After posture optimization, we plan the transition between q_{near}^{24D} and the pose determined by the foothold selection and posture optimization strategy q_{temp}^{24D} implemented in the `PlanFeetPaths`$(q_{near}^{24D}, q_{temp}^{24D})$ procedure. The goal is to find the constraint-free path of each robot foot between initial f_s and goal foothold f_g. The found path should be collision-free (self-collisions and collisions with obstacles), each foot position defined inside the leg's workspace, and the robot should be statically stable.

First, we plan the initial path for each foot. The planned path is set to avoid collisions of the foot with obstacles (green line in Fig. 3). The minimal distance of the foot to the terrain is set to d_{min}^f. The path of the foot is located on the plane aligned with the gravity vector and connects the initial f_s and goal foothold f_g.

The path which follows the shape of the obstacles is not the shortest possible and collision-free path. Thus, we simplify the initial path by computing the convex hull path over the initial path. To this end, we iterate over each point of the initial path and ray-trace between the considered point and the following points of the initial path. Then, we accept "shortcuts" which create a path for which the distance between the robot's foot and the obstacles is larger than d_{min}^f. The example result is presented in Fig. 3.

4) OPTIMIZATION DURING THE SWING PHASE

The transition planning between two stable postures may produce postures of the robot which can't be executed. If the robot is unstable, we move the robot's body in the direction given by the center of the support polygon (c.f. Fig. 9). This procedure is simple and sufficient to keep the robot stable. On the other hand, preventing deadlocks caused by the lack of kinematic margin for legs or collisions is a more difficult task. We plan the motion of each swing leg above obstacles in the 3D space. It is easier to avoid obstacles in the 3D space but this procedure does not guarantee that the path is continuous in the configuration space. As a result, some foot positions may be outside the workspace. We propose the optimization procedure to find the path of the swing leg which is executable by the robot.

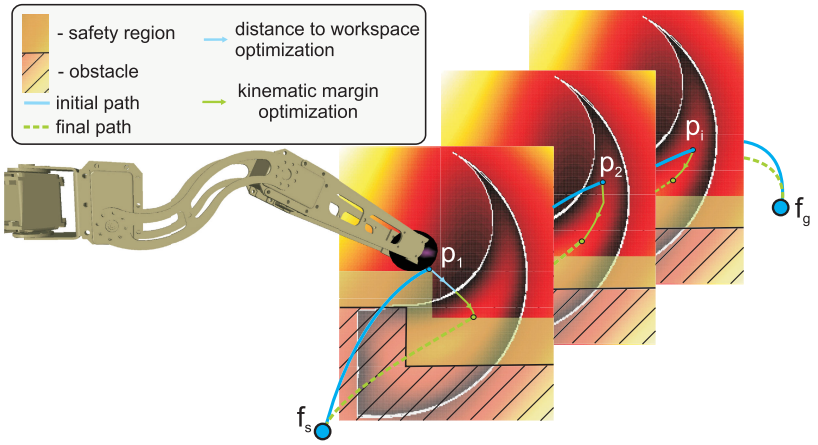

FIGURE 4. Two-stage optimization of the foot position. If the initial position of the foot is outside the workspace, we use the gradient of (5) to move the foot into the workspace. The position of the foot is later optimized to increase the kinematic margin of the leg.

The illustration of the foot position optimization procedure is presented in Fig. 4. For each point of the path obtained from the procedure presented in Section III-B.3, we check the value of the kinematic margin. If the planned foot position is outside the workspace, we use the gradient descent optimization and the gradient of (5) to move the foot to the region inside the workspace (blue arrow in Fig. 4). Then, we optimize the kinematic margin using gradient of (4) (green arrow in Fig. 4). The computation of the kinematic margin is described in details in section IV-B. Our goal is to find positions of the foot that are inside the workspace. The value of kinematic margin for the obtained foot position should be larger than the threshold $d_{l_{min}}^{KM}$.

During optimization, we change the position of the considered foot in the direction normal to the direction of the motion. Thus, we use the cross-section over the workspace to find the optimal position of the considered foot. The procedure is repeated for each position of the initially planned path shown in Fig. 4. We start the procedure only if the initial position of the foot is outside the workspace or the kinematic margin is below the threshold (3 cm in the experiments presented in the article).

IV. CONSTRAINTS EVALUATION

In this section, we introduce the constraints evaluation problem during path planning of a legged robot. We show the standard methods used for constraints checking. Then, we introduce the Gaussian Mixture-based method used to create constraints model which significantly speeds up constraints evaluation.

A. WORKSPACE OF THE ROBOT

The workspace of the robot is a set of poses which can be reached by the robot. We distinguish between the workspace of each leg and workspace of the whole robot because walking robots have multiple independent kinematic chains (legs). The workspace of the robot depends on the mechanical design

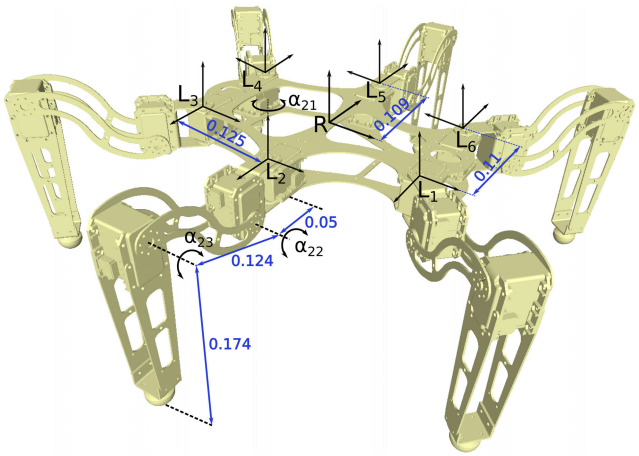

FIGURE 5. Messor II robot used in this research (all dimensions shown in blue are in [m]).

and kinematic structure of the robot. In this research, we show results on the six-legged Messor II robot [8]. The kinematic structure of the Messor II robot is presented in Fig. 5. The robot has six identical anthropomorphic legs. The kinematic structure of the robot is symmetrical but because the robot has visual sensors mounted on the "front" side we consider the L_1 and L_6 legs as the front legs of the robot.

The motion of the robot is planned in the 3D space. The planner has to determine the pose of the robot's body and positions of the robot's feet. The planner has to also check if the planned positions of the feet are accessible by the robot. Thus, we need a fast and efficient procedure which checks if the defined position of the foot is inside the workspace of the robot's leg. The visualization of the workspace of the robot's right front leg is presented in Fig. 6a. All foot positions which are inside this region are reachable by the robot.

The 3D shape of the leg's workspace is complex (Fig. 6). Also checking if a 3D point is inside the 3D hull is

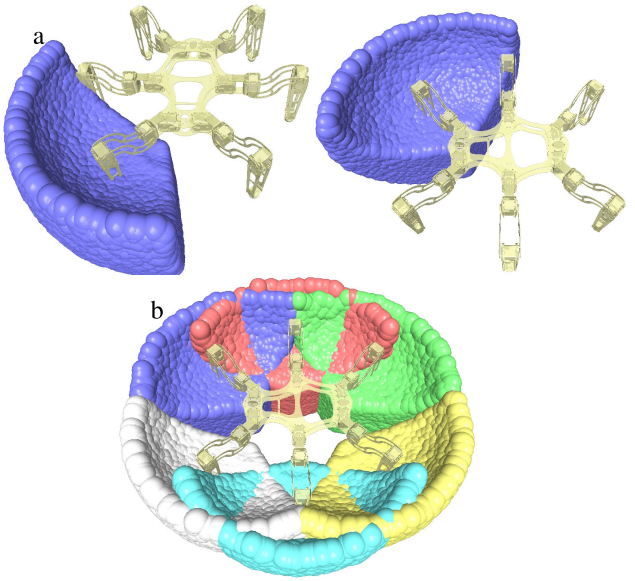

FIGURE 6. Visualization of the single leg workspace (a), and the overlapping workspaces of all robot's legs (b). The shape of the workspace is visualized by a set of spheres which centers are located in the workspace of the robot.

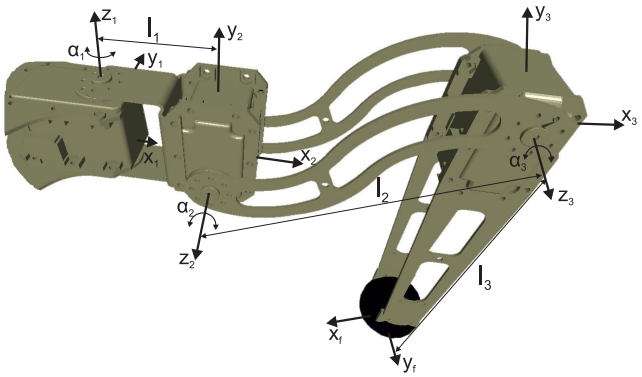

FIGURE 7. Kinematic model of the robot's leg.

a time-consuming task. Therefore, instead of checking the workspace in the 3D space, we use the inverse kinematic model of the robot's leg and configuration space to check if the given position is inside the workspace. The kinematic model of the leg is presented in Fig. 7. The leg of the robot is represented by the standard anthropomorphic kinematic chain. We use the inverse model of the anthropomorphic arm to compute the configuration of the leg $[\alpha_1, \alpha_2, \alpha_3]^T$ for the given position of the foot $\mathbf{p}_f = [x_f, y_f, z_f]^T$ that can be expressed as:

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} \text{atan2}(y_f, x_f)) \\ \text{atan2}(z_f, x_p - l_1) + \arccos(\frac{l_2^2 + d_{23}^2}{2 l_2 d_{23}}) \\ -\arccos(\frac{d_{23}^2 - l_2^2 - l_3^2}{2 l_2 l_3}) \end{bmatrix}, \quad (2)$$

where $d_{23} = \sqrt{(x_f - l_1)^2 + z^2}$ and $x_p = x_f \sin(\alpha_1) + y_f \cos(\alpha_1)$.

In general, the inverse kinematics for the anthropomorphic kinematic chain returns two solutions for the given position of the foot. In the first solution, the angle in the knee joint is negative and this joint position is above the foot. In the second solution, the angle in the knee joint is positive and the joint position is lower than the foot. The second solution can be used when the robot is capable of walking upside down. Our robot has cameras mounted on the platform and is not capable of walking upside-down thus, we are interested in the first solution only.

We check the value of inverse cosine functions in (2) to determine if the foot position is inside the workspace of the robot. If the foot position is outside the workspace the distance between the reference foot position p_f and the joint α_2 is larger than $l_2 + l_3$. In this case, the argument of the inverse cosine function is outside the domain $-1 \leq x \leq 1$ of the $acos(x)$ function. We detect such situations to check if the foot is inside the workspace of the leg. Additionally, we check the joint limits defined for each joint of the leg. This procedure is extremely fast and takes about 1 us on the standard x86_64 architecture processor. However, the procedure accepts positions of the foot which are not collision-free and we later show how to reject these positions.

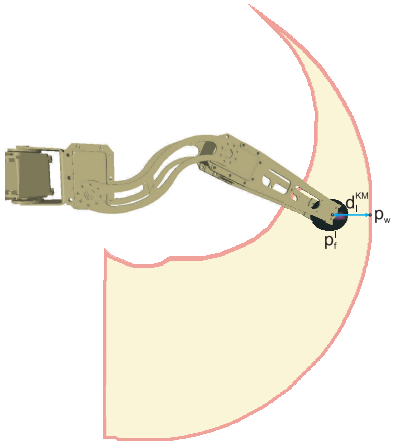

FIGURE 8. Illustration of the kinematic margin d_l^{KM} as the minimal distance between the foot position $\mathbf{p}_f^l$ and the closest point on the surface of the robot's workspace $\mathbf{p}_w$.

B. KINEMATIC MARGIN

The illustration of the kinematic margin is presented in Fig. 8. We define the kinematic margin of the l-th leg d_l^{KM} as the minimal distance from the current position of the foot $\mathbf{p}_f^l$ to the border of the workspace:

$$d_l^{KM} = \min_{\mathbf{p}_w} ||\mathbf{p}_f^l - \mathbf{p}_w||, \quad (3)$$

where $\mathbf{p}_w$ is the point located on the surface of the leg's workspace. Because we do not have the analytical model of the leg's workspace, we solve the optimization problem (3) iteratively. To this end, we ray-trace from the foot position $\mathbf{p}_f^l$ to the seven cardinal directions in the 3D space. We iteratively

move from the given foot position along three main axis (x, y, z) and axis defined in the spherical coordinate system by the vector $[r, \Theta, \phi]^T = [1, n \cdot \Pi/4, m \cdot \Pi/4]^T$, where $n = \{-1, 1\}$ and $m = \{-1, 1\}$. At each direction, we find the first position which is outside the workspace and use them to find the minimal value in (3). The largest kinematic margin is defined for the pose which is at the center of the leg's workspace. On the surface of the leg's workspace, the kinematic margin is 0.

We also define the kinematic margin of the robot d_R^{KM} which is defined by the minimal value of the leg's kinematic margin:

$$d_R^{KM} = \underset{d_l^{KM}}{arg\ min}\{d_1^{KM}, \ldots, d_6^{KM}\}. \tag{4}$$

The proposed path planner plans the positions of the robot's feet in the Cartesian coordinate system. The planer may set the foot positions outside the robot's workspace to o avoid collisions. As a result, the planned path can't be executed by the robot. We also compute the distance to the workspace of the l-th leg to deal with this problem:

$$d_l^W = \underset{\mathbf{p}_w}{min}||\mathbf{p} - \mathbf{p}_w||, \tag{5}$$

where $\mathbf{p}$ is the position in the Cartesian coordinate system, and $\mathbf{p}_w$ is the point located on the surface of the leg's workspace. Similarly to the computation of the kinematic margin, we solve the optimization problem (5) by iterative computation of the distance between the considered point $\mathbf{p}$ and the closest point on the surface of the leg's workspace.

C. SELF-COLLISIONS

The workspace of each robot's leg overlaps with at least one workspace of the neighboring leg (Fig. 6b). The motion of the legs can be limited by adding a joint's range to avoid collisions. In this case, the locomotion capabilities of the robot are also significantly limited. We suggest to dynamically check self-collisions of the robot [3] instead of checking static joint limits.

We use the CAD model of the robot to check self-collisions. Each part of the robot is represented by the triangle mesh surfaces. we use triangle-to-triangle intersection test with oriented bounding box (OBB) [46]. The OBB is used to initially check the collision between the pair of objects. The triangle-to-triangle intersection test is applied if the bounding boxes collide to precisely check the collision constraint.

In this research, we compared two implementations of collision detection methods for triangle mesh models: the Coldet library [18] and Flexible Collision Library (FCL) [36]. In our case, the Coldet library is about ten times faster than the FCL library but has problems with objects which are very close to each other. In this case, the Coldet library returns information about collision even if the meshes do not collide. The FCL is slower but much more precise and therefore we use it as the reference collision checker.

We additionally exclude collision checking between parts which never collide to speed up the collision checking. We also checked the influence of the number of triangles in the model on the collision checking time. Collision checking with the full mesh model of the object is time-consuming. The required time can be reduced by using a GPGPU unit [21], [22] but this type of the hardware is not available on all types of the robot because of the energy consumption and maximum payload of small walking machines. Therefore, the first idea to speed-up collision checking is to simplify the mesh model of the robot. To this end, we reduced the number of triangles of the model and evaluated the collision checking time for the given models. The results are presented in Tab. 1.

TABLE 1. Collision detection time [ms] for various complexity models and different number of colliding parts.

collisions	0	3	7	9	13	16	19
$model_{100\%}$	3.95	4.13	4.50	4.58	5.04	6.37	7.18
$model_{70\%}$	2.46	2.94	3.13	3.18	3.62	4.94	5.28
$model_{40\%}$	1.92	2.13	2.27	2.33	2.75	3.44	3.97
$model_{2\%}$	1.35	1.41	1.75	1.74	1.95	2.08	2.22

Even if the model contains 98% fewer triangles than the initial model, the collision checking is only three times faster. This is also because the collision checking procedure verifies the bounding boxes of the objects. In practice, we are unable to reduce the collision checking time below milliseconds.

D. COLLISIONS WITH THE TERRAIN

Collision checking with the environment can be solved by using iterative libraries like FCL [36]. The most popular method is to represent the environment as a mesh and check collisions between the robot and the mesh representing terrain and obstacles. The method is similar to self-collision checking. However, in this research, we assume that the robot walks in the previously unknown environment. In this case, the model of the environment has to be updated whenever the robot acquires new information about the obstacles [7]. Thus, we avoid this approach because updating the collision model is time-consuming and very often takes more time than collision checking.

Instead, we exploit the properties of the elevation map to deal with the problem of real-time collision checking between the robot and the terrain. The elevation map provides fast and random-access to each cell. We iterate over cells below the foot for foot-terrain collision checking to check collision between the leg of the robot and the terrain. We increase the elevation of the map by the safety margin (0.03 m).

We use a similar procedure for the robot's body. We check the cells of the elevation map below the robot's body (we approximate the robot's body by the rectangular shape). If the cells of the map are lower than the points of the robot body above the cells, the robot's trunk does not collide with the terrain. This procedure is accurate and computationally efficient.

E. STATIC STABILITY

The static stability constraint checking method considers the projection of the robot's center of mass (CoM) on the support

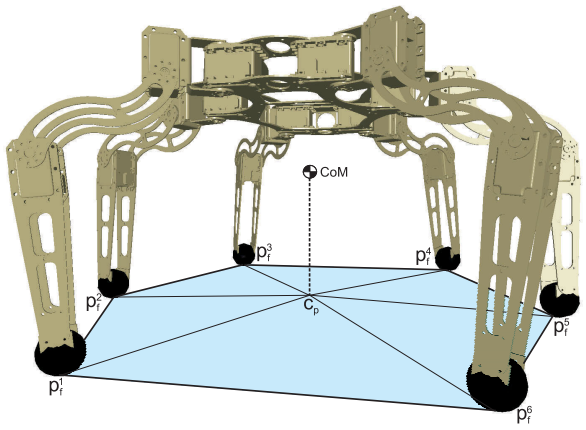

FIGURE 9. Computation of the static stability constraint. The robot is statically stable if the projection of the center of mass (CoM) is located inside the support polygon.

polygon [23]. We compute the CoM taking into account the masses of the robots links and the current configuration of the robot. If the projection of the CoM is inside the support polygon the robot is statically stable. The support polygon is created by the projections of the supporting feet on the horizontal plane $\mathbf{p}_f^s$ (note that the index s corresponds to the number of the support foot and this may differ to the number of the leg l). The visualization of the support polygon is presented in Fig. 9. We compute the area of the support polygon A to check whether the robot is statically stable:

$$A = \frac{1}{2}\sum_{s=0}^{S} x_s \cdot y_{s+1} - x_{s+1} \cdot y_s, \tag{6}$$

where (x_s, y_s) are the coordinates of the supporting foot and the list of supporting points is extended by the $\mathbf{p}_f^0$ $\left(\mathbf{p}_f^{S+1} = (x_{S+1}, y_{S+1}) = \mathbf{p}_f^0 = (x_0, y_0)\right)$. Then, we compute the area of triangles T_s created by the projection of the center of mass (C_p) and neighboring supporting feet $\mathbf{p}_f^s$ and $\mathbf{p}_f^{s+1}$. If $\sum_{s=0}^{S} T_s > A$ the projection of the CoM is outside the support polygon and the robot is not statically stable. This procedure is computationally efficient and we can perform it multiple times during path planning.

F. CONSTRAINT MODELING WITH GAUSSIAN MIXTURE

The computation of kinematic margin, distance to the workspace, and collision checking is time-consuming and very often is mitigated during path planning of legged robot. Sampling-based motion planning methods, like Rapidly-exploring Random Trees [30] or Probabilistic Roadmaps [27] sample from the multi-dimensional space. The constraints checking slows down motion planning significantly. For the walking robots, which operate in the high-dimensional space, the constraints checking with standard methods make the online motion planning impossible. Thus, we propose to represent the constraints by the analytical functions (Gaussian Mixtures). The value of the function represented by the GM can be computed quickly on the computer with the standard CPU and makes the online motion planning possible. Moreover, the analytical representation of the constraints allows us to compute the gradient of the "constraint function" and find the acceptable configuration of the robot.

Sums of radial basis functions (RBF) are widely applied to approximate unknown nonlinearities [33]. The summation of RBF with the same variance component $\sigma^2_{i=1\ldots N}$ is a universal approximator [35]. Gaussian Mixtures are used in this research to model constraints in the legged locomotion. The GM is applied to define the relation between the state of the robot and a constraint value (kinematic margin, collision). The GM represents constraints in an analytical form which allows also computing the derivative of the constraint function. In this case, the motion planning strategy can find the way to move the robot out of the "forbidden" space.

The Gaussian Mixture is defined as a sum of multivariate normal distributions

$$P(x_1, \ldots, x_N) = \sum_{k=1}^{K} c_k \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)} \tag{7}$$

where N is the number of the dimensions, μ is the mean value, Σ is the covariance matrix, K is the number of Gaussians, and c_k is the constant coefficient. We assume that the covariance matrix Σ is diagonal and the covariance related to each dimension is independent [13], [25]. We rewrite (7):

$$P(x_1, \ldots, x_N) = \sum_{k=0}^{K} c_k \cdot \exp\left(\sum_{n=1}^{N} \lambda_{n,k}(x_n - \mu_{n,k})^2\right), \tag{8}$$

where $\mu_{n,k}$ is the mean value for the n dimension of the k-th Gaussian, and $\lambda_{n,k}$ is the constant coefficient. The analytical form of the constraints gives the possibility to compute the gradient:

$$\nabla P(x_1, \ldots, x_N) = \begin{bmatrix} \sum_{k=0}^{K} c_k \cdot P(x_1, \ldots, x_N) \cdot \left(-2\frac{\lambda_{1,k}}{N}(x_1 - \mu_{1,k})\right) \\ \cdots \\ \sum_{k=0}^{K} c_k \cdot P(x_1, \ldots, x_N) \cdot \left(-2\frac{\lambda_{N,k}}{N}(x_N - \mu_{N,k})\right) \end{bmatrix}, \tag{9}$$

which can be later used to quickly find the solution outside the constrained region and avoid constraints during motion planning.

One has to find the centroid μ_n for each N-dimensional Gaussian function and the parameter λ_n of each kernel which depends on the variance σ_n to determine (8). We use (8) to solve the regression problem. We explore multi-dimensional search space using optimization techniques to find the optimal values of $\lambda_{n,k}$ and $\mu_{n,k}$. Among many techniques (Expectation-maximization, gradient-based methods) we identified population-based black-box optimization methods as the most efficient. We applied three black-box optimization methods: Evolutionary Strategy (ES) [2], Particle Swarm Optimization (PSO) [28], and Covariance Matrix

Adaptation Evolution Strategy (CMA-ES) [24]. Among the selected methods the CMA-ES can find the best solution but the optimization is time-consuming thus we use PSO in the experiments to find the parameters of the Gaussian Mixture (8).[1] Each individual in the optimization method represents the solution (vector of $\lambda_{n,k}$ and $\mu_{n,k}$ values) to the regression problem.

The regression function $P(\mathbf{x})$ is found using training samples collected on the reference constraint-checking method. The input training vector $\mathbf{x}_t$ has the corresponding output value in the output training vector $\mathbf{y}_t$ of the length M. During the optimization, the method moves N-dimensional Gaussian functions through the search space to find the best-fitted set of the parameters $\lambda_{n,k}$ and $\mu_{n,k}$. The weights c_k related to each N-dimensional Gaussian function do not have to be found by the optimization method. We can compute these values directly using the Least-Squares Fitting method.

First, we compute the Gram matrix $\mathbf{G}$:

$$\mathbf{G} = \mathbf{V}^T \cdot \mathbf{V} \tag{10}$$

where

$$\mathbf{V} = \begin{bmatrix} \phi_1(\mathbf{x}_{t_1}) & \phi_2(\mathbf{x}_{t_1}) & \dots & \phi_K(\mathbf{x}_{t_1}) \\ \dots & \dots & \dots & \dots \\ \phi_1(\mathbf{x}_{t_M}) & \phi_2(\mathbf{x}_{t_M}) & \dots & \phi_K(\mathbf{x}_{t_M}) \end{bmatrix} \tag{11}$$

is a Vandermonde matrix, ϕ_k is the k-th multivariate Gaussian function in the sum (8), $\mathbf{x}_{t_m}$ is the m-th training input sample learning vector. Then, we compute the vector $\mathbf{c}$ of coefficients c_k

$$\mathbf{c} = [c_1, c_2, \dots, c_K]^T, \tag{12}$$

using Gram and Vandermonde matrix and the vector of training output values $\mathbf{y}_t$:

$$\mathbf{c} = \mathbf{G}^{-1} \cdot \mathbf{V}^T \cdot \mathbf{y}_t. \tag{13}$$

The length of the vector c is K. We solve (13) using standard Cholesky decomposition which works on the positive definite matrices.

The number of elements K in the polynomial (8) which defines the number of Gaussians in the mixture is a design parameter. The fitness function f_r in the population-based optimization algorithm is defined as a sum of squared distances between the trained $\mathbf{y}_t$ points and the corresponding points obtained from the regression function:

$$f_r = \sum_{m=1}^{M} (y_{t_m} - P(x_{t_m}))^2. \tag{14}$$

We only define the boundaries for $\lambda_{n,k}$, $\mu_{n,k}$, and the number of elements K to start the optimization. We normalize the domain and the output of the regression function (8) so the boundaries are defined in the range [0, 1].

[1]The C++ implementation of the proposed method is available at https://github.com/LRMPUT/constraintsGM

V. RESULTS

A. REGRESSION OF THE KINEMATIC MARGIN

Computation of the kinematic margin with the iterative method is time-consuming and cannot be used in the real-time path planning. Thus, we create the Gaussian Mixture model which computes the kinematic margin $\hat{d}_l^{KM}$ for the given configuration of the l-th leg $[\alpha_1^l, \alpha_2^l, \alpha_3^l]^T$

$$\hat{d}_l^{KM} = P(\alpha_1^l, \alpha_2^l, \alpha_3^l). \tag{15}$$

We train GM kinematic margin model $\hat{d}_l^{KM}$ for each leg separately. We use 10000 training samples obtained from the iterative model of the kinematic margin (3). The number of Gaussian functions is set to 200.

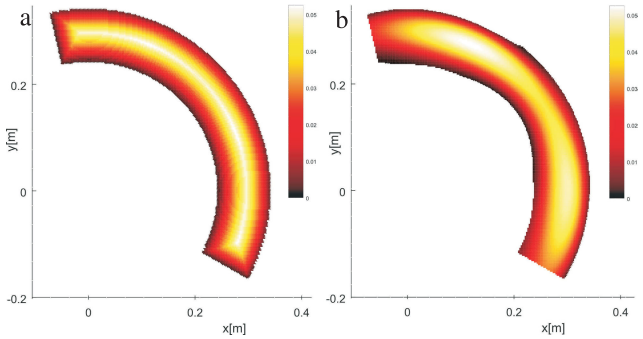

FIGURE 10. Horizontal cross-section over the workspace of the single leg at z=0: Kinematic margin computed using iterative method (a) and obtained from the Gaussian Mixture model (b).

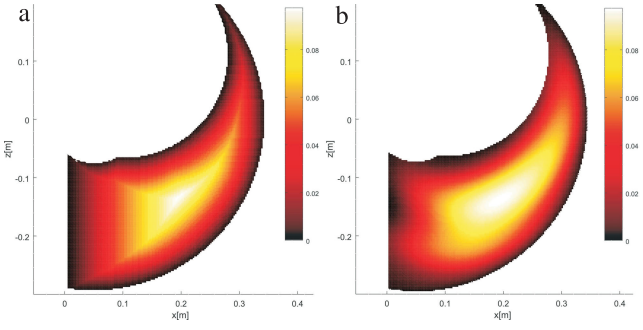

FIGURE 11. Vertical cross-section over the workspace of the single leg at y=0: Kinematic margin computed using iterative method (a) and obtained from the Gaussian Mixture model (b).

TABLE 2. Average error of the kinematic margin model.

Leg no.	1	2	3	4	5	6
$e_{train}^{av}[mm]$	3.4	3.6	3.7	3.1	3.2	3.4
$e_{test}^{av}[mm]$	3.4	3.6	3.6	3.2	3.3	3.5

The visualization of the obtained kinematic margin model of the single leg is presented in Fig. 10 and Fig. 11. We show the horizontal and vertical cross-section over the workspace of the leg to visualize the obtained model. We compare the iterative model used for training and the obtained GM model. The detailed results obtained for kinematic margin modeling for each leg of the robot are presented in Tab. 2.

The average error between the reference and estimated value of the kinematic margin is 3.42 mm.

B. REGRESSION OF THE DISTANCE TO THE WORKSPACE

We create the Gaussian Mixture model which computes the distance to the l-th leg workspace $\hat{d}_l^W$ for the given point $\mathbf{p} = [p_x, p_y, p_z]^T$:

$$\hat{d}_l^W = P(p_x, p_y, p_z). \tag{16}$$

The model is trained for each leg separately. We use 10000 training samples obtained from the iterative model of the distance to the workspace (5). The number of Gaussian functions is set to 200.

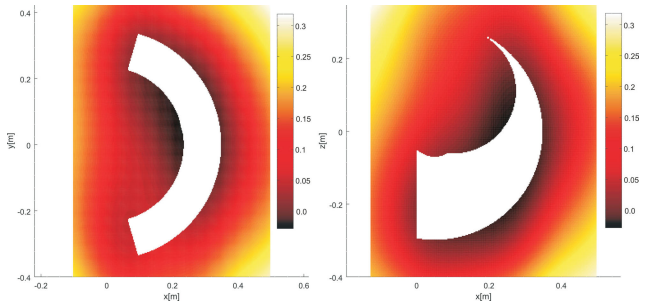

FIGURE 12. Horizontal (a) and vertical (b) cross-section over space around the leg's workspace. The color represents the estimated distance to the workspace of the leg.

The visualization of the approximated distance to the workspace is presented in Fig. 12. The obtained model is used to find the direction to the workspace of the leg. The accuracy of the estimated value of the distance to the workspace is not significant here. More important for the planner is the direction which allows finding the position of the foot which is accessible by the robot. Thus, whenever the planner gives the position of the foot which is outside of the leg's workspace we compute the gradient $\nabla\hat{d}_l^W$ and use it in the gradient descent method [41]. The gradient descent optimization allows us to find the closest point which is inside the workspace of the robot's leg.

C. REGRESSION OF SELF-COLLISIONS CHECKING

The full self-collision model of the robot requires 18 values on the input. The input vector corresponds to the joint configuration of the robot. The problem is difficult for the proposed Gaussian Mixture because the regression model has to learn binary representation on the output. This requires the high number of Gaussians in the mixture K to represent properly the approximated function. Instead of dealing with a single high-dimensional problem, we train multiple models which detect self-collisions and collisions between neighboring legs. We train the Gaussian Mixture to check if the given configuration of the leg $[\alpha_1^l, \alpha_2^l, \alpha_3^l]^T$ is collision-free:

$$\hat{c}_{self}^l = P(\alpha_1^l, \alpha_2^l, \alpha_3^l). \tag{17}$$

TABLE 3. Accuracy of the single leg collision model.

Leg no.	1	2	3	4	5	6
$acc_{train}[\%]$	99.8	99.2	99.8	99.7	99.1	99.7
$acc_{test}[\%]$	99.5	98.8	99.6	99.6	98.9	99.5

TABLE 4. Accuracy of the neighboring leg collision model.

Neighboring legs. (l,l')	1,2	2,2	3,4	4,5
$acc_{train}[\%]$	93.0	93.0	93.5	93.2
$acc_{test}[\%]$	92.3	92.5	93.0	92.3

The results of the training are presented in Tab. 3. The accuracy is computed from the number of correctly recognized collision divided by the total number of samples. The accuracy for the training acc_{train} and test dataset acc_{test} is around 99%.

The second model detects collisions between neighboring legs. In this case, the dimensionality of the problem is reduced to 6. The input is represented by six joint values. The GM which represents the collision model between neighboring legs is defined as follows

$$\hat{c}_{neigh}^{l,l'} = P(\alpha_1^l, \alpha_2^l, \alpha_3^l, \alpha_1^{l'}, \alpha_2^{l'}, \alpha_3^{l'}), \tag{18}$$

where l and l' represent the identifiers of neighbouring legs, $[\alpha_1^l, \alpha_2^l, \alpha_3^l]^T$ and $[\alpha_1^{l'}, \alpha_2^{l'}, \alpha_3^{l'}]^T$ represent the configuration of the l-th and l'-th leg, respectively. We train this model only for the legs which may collide. The front and rear legs may collide with only one neighboring leg while the middle legs have two collision models for their neighboring legs.

The collision model returns binary value which represents the information about the collision between two legs. In practice, the GM produces continuous values on the input but we added additional binarization layer. The threshold is set to 0.5. All values below the threshold are considered as 0 (lack of collisions) and all values above the threshold are considered as 1 (leg collides).

The leg collision model is trained using 20000 samples. The samples are obtained from the mesh-based collision model and the FCL library [36]. The number of Gaussian functions in the mixture K is set to 600. The results of the training are presented in Tab. 3. The number of correctly classified collisions for the testing dataset is 92.13%. We expect false positive and false negative collision detections. The false positive detections cause the robot to avoid regions which are collisions-free. We accept this behavior because the positions which are classified improperly are close to the configurations which cause collisions and we treat it as a safety margin.

We also accept false negative detection. They are more problematic because the robot may plan its trajectory through regions which are not collision-free. To deal with this problem, we check collisions using precise model and FCL library before trajectory execution and we can easily avoid these situations. The collision checking procedure, which uses GM, on average takes 5 μs. It means that the approximated

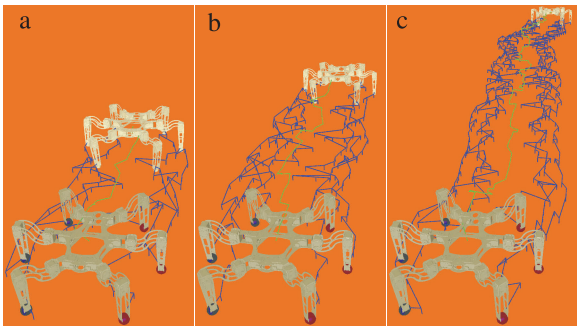

FIGURE 13. Example planned paths on flat terrain. The distance between initial and goal pose of the robot is 1 m (a), 2 m (b) and 5 m (c).

collision model is 40 times faster than the module which uses triangle meshes. We can use this module efficiently during motion planning with sampling-based methods.

D. SUMMARY OF THE CONSTRAINTS MODELING

The main advantage of using the proposed constraint regression method is the reduced computation time which allows the planner to efficiently sample the search space. We summarize the computation time for each regression method in Tab. 5.

TABLE 5. Average computation time of the constraint checking procedures.

Constraint	iterative [μs]	GM [μs]
kinematic margin	414356	6
dist. to workspace	115487	6
self-collision	7180	5

The results presented in Tab. 5 are obtained on the computer with Intel i5-8250U processor. The iterative computation of the kinematic margin and distance to the workspace depends on the estimated value. The higher the estimated value the longer computation time. For the results presented in the table, the value of the kinematic margin is 0.085 m and the distance to the workspace is 0.0825 m. The constraints checking with GM model takes 5-6 μs and depends only on the complexity of the GM (dimensionality of the problem and the number of Gaussians). The proposed computation of the kinematic margin is more than 69 thousand times faster than the iterative method. The computation of the distance to the workspace and collision checking is 19 thousand and 1436 times faster than the reference iterative method, respectively.

E. COMPUTATIONAL EFFICIENCY

We planned the path of the robot on flat terrain to show the computational efficiency of the proposed method. The planned distance changes from 1 m to 5 m. The obtained paths for three different distances are presented in Fig. 13. The planning time is summarized in Tab. 6. We compare two collision checking methods: Gaussian-based collision checking and standard approach with FCL library [36].

TABLE 6. Relation between average planning time (all values in [s]) and planning distance on flat terrain when FCL and Gaussian Mixture are used for collision checking.

	1 [m]	2 [m]	3 [m]	4 [m]	5 [m]
FCL	293±127	500±106	850±101	870±52	1394±132
GM	4.5±0.5	7.1±0.6	10.7±0.6	15.4±1.7	22.1±1.3

The average planning time is more than 70 times slower when the FCL library for collision checking is used. This experiment also shows that the computation time increases linearly with the distance between the initial and goal pose of the robot.

TABLE 7. Relation between average posture optimization time (all values in [s]) and various methods for constraints checking (explanation in the text).

	GM	FCL	d_l^{KM}	FCL+d_l^{KM}
time [s]	0.1±0.007	11.0±0.71	2.0e3±430	2.2e3±261

It is difficult to show the influence of the regression of the kinematic margin d_l^{KM} on the planning time because the planning takes hours when the iterative method is used. Thus, we show the influence of the various regression methods on the execution time of the posture optimization procedure. The results are presented in Tab. 7. The mean values and standard deviations in the table are obtained for 10 trials for each configuration of the posture optimization method. We set our approach where all constraints are approximated by the Gaussian Mixtures (GM) as a reference method. Then, we replace our collision checking method with the FCL library. We also replaced the GM-based computation of the kinematic margin with the iterative method (d_l^{KM}). Finally, we use both iterative methods at the same time (FCL + d_l^{KM}).

The optimization of the robot's posture is 100 times slower when the FCL library is used instead of the proposed GM-based model. However, the iterative computation of the kinematic margin is the most time-consuming. The posture optimization is more than 10000 times slower when we use the iterative method instead of the proposed GM-based regression. Finally, we optimize the posture of the robot using FCL for collision checking and iterative version of d_l^{KM}. In this case, a single posture optimization takes on average 37 minutes and cannot be used to plan the motion of the real robot. In contrast, the proposed constraint checking methods are extremely fast and the posture optimization takes only 0.1 s.

We also performed verification experiments on rough terrain. We checked the influence of the roughness of the terrain on the planning time. We also registered the success ratio for the series of 10 trials. We set the maximal allowed planning time to 120 s. If the planning takes longer than the threshold we set this trial as unsuccessful. The planner can provide the path to the goal pose but it also can give information that the goal position is beyond the range of the robot.

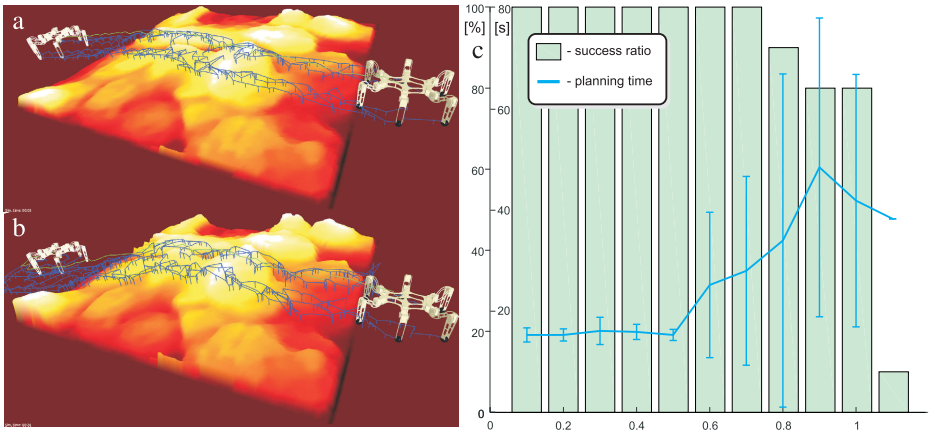

FIGURE 14. Influence of the terrain roughness on the planning time: Rough terrain mockup scaled by 0.5 (a) and 0.9 (b), and the relation between the roughness of the terrain and planning time and the success ratio (c).

TABLE 8. Properties of the planning algorithm (roughness scale, success ratio, average planning time, standard deviation of the planning time, average number of self collision checking, average number of checking collisions with the ground, , average number of workspace checking, average number of kinematic margin computations, average number of stability checking, average number of foothold selections and posture optimizations) as a function of the terrain roughness (scale).

scale	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1
success ratio [%]	100	100	100	100	100	100	100	90	80	80	10
av. time [s]	15.3	15.3	16.1	15.9	15.3	25.2	28.0	34.0	48.5	41.9	38.3
std. time [s]	1.4	1.2	2.7	1.5	1.1	14.4	18.7	33.0	29.6	25.0	0
self-col.	108e3	111e3	116e3	114e3	111e3	168e3	226e3	261e3	416e3	350e3	558e3
col. with ground	25e3	26e3	27e3	27e3	26e3	40e3	53e3	62e3	99e3	83e3	131e3
select foot.	421	439	463	460	453	716	1096	1210	2352	1804	3802
workspace	331e3	342e3	355e3	349e3	340e3	519e3	703e3	810e3	1299e3	1090e3	1754e3
kinem marg.	331e3	341e3	355e3	349e3	339e3	517e3	699e3	805e3	1289e3	1083e3	1742e3
posture opt.	70	73	77	76	75	119	178	199	353	275	438
stability	54e3	56e3	59e3	58e3	57e3	85e3	120e3	133e3	211e3	176e3	271e3

The experimental setup is presented in Fig. 14. The planning distance is 4 m. We scale the rough terrain mockup by the scale factor changing between 0.1 and 1.2. We multiply the height of each cell of the elevation map by the scaling factor and thus the terrain is flat when the scale is set to 0.0 and the mockup does not change when the scale factor is set to 1.0. If the scale factor is set to 1.0 the highest peak of the mockup is 0.34 m high (the maximum clearance between the robot body and the terrain in the most upright position is 0.298 m). The terrain mockup scaled by 0.5 and 0.9 are presented in Fig. 14a and Fig. 14b, respectively.

The results presented in Fig. 14c show that the planning time does not change significantly when the roughness is below 0.5. It increases from 15 s to 50 s when the roughness of the terrain increases to 1.0. The success ratio is 100% for the moderately rough terrain (scale factor is below 0.8). The success ratio drops gradually with the increase of the scale factor and reaches 10% for the rough terrain obtained for the scale factor equal to 1.1.

We also show the significance of the constraints evaluation by providing the average numbers of constraints checking during path planning. The results are provided in Tab. 8. We show how the properties of the planning method change with the roughness of the terrain (scale). We take into account the success ratio, average planning time, standard deviation of the planning time, average number of self collision checking, average number of checking collisions with the ground, average number of workspace checking, average number of kinematic margin computations, average number of stability checking, average number of foothold selections and posture optimizations.

The results presented in Tab. 8 show that even if the terrain is almost flat the average number of constraints checking is higher than 849 thousand. All these constraints are checked in 15.3 s. This is possible because we use the Gaussian Mixture model to evaluate the most time-consuming constraints. During planning, the workspace of the robot is checked at the beginning. This constraint can be evaluated very quickly, and if the desired foot pose is outside the workspace we avoid evaluating the remaining time-consuming constraints. Thus, we evaluate simple constraints at the beginning (workspace, kinematic margin) and then, we evaluate more complex constraints (stability checking, collision with the ground) to speed up the planning time. Also, the reason is to avoid running of the time-consuming procedures like foothold selection and posture optimization. As a result, the posture optimization procedure is performed only 70 times when the

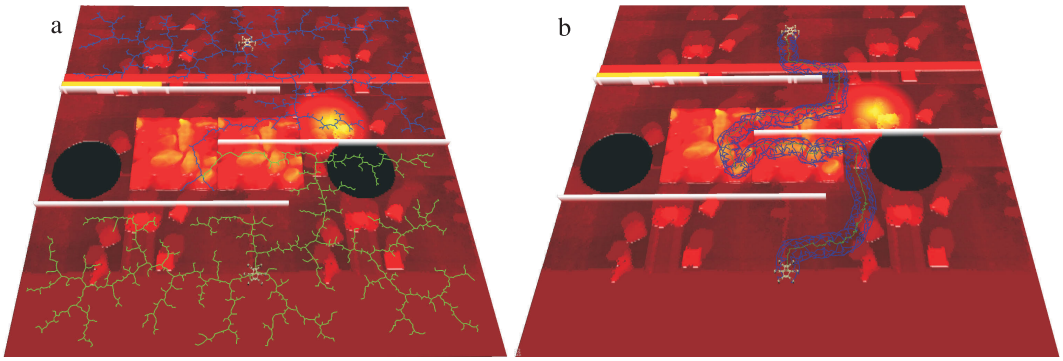

FIGURE 15. Example RRT trees (a) obtained in the first planning scenario (green – T_a starting in the initial robot pose, blue – T_b starting in the goal robot pose) and the obtained path (b).

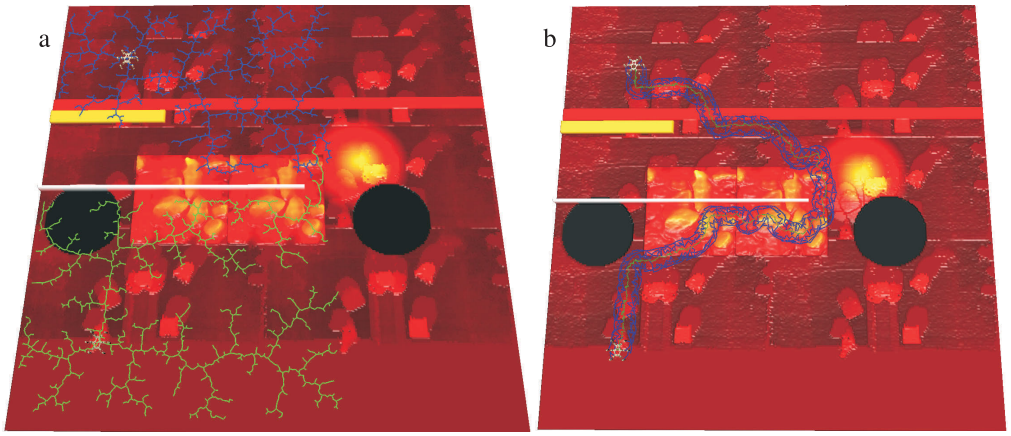

FIGURE 16. Example RRT trees (a) obtained in the second planning scenario (green – T_a starting in the initial robot pose, blue – T_b starting in the goal robot pose) and the obtained path (b).

terrain mockup is scaled by 0.1. This number increases to 438 when the scale is set to 1.1.

F. PATH PLANNING

We also performed experiments with path planning on the environment with obstacles which are not traversable. The experimental sets are presented in Fig. 15 and Fig. 16. The goal of this experiment is to show how the planner deals with obstacles. The results presented in Fig. 15a and Fig. 16a show how the trees explore the search space. The algorithms find the path around the obstacles which are not traversable by the robot. The proposed method also inherits the properties of the general RRT-Connect algorithm. It has problems with narrow passages. The planning time increases in the presence of narrow passages in the environment. This problem can be solved by guiding the RRT-Connect algorithm through narrow passages [6]. The final paths obtained from the proposed planner are presented in Fig. 15b and Fig. 16b.

Finally, we verified the motion planning algorithm on the real robot.[2] We selected three types of obstacles: a bump, a step, and a concavity. In general, we use tactile feedback during execution of the planned path [6] because the terrain map is not accurate and the selected foothold may be over or under the terrain. However, during these experiments, the robot knows the environment (elevation map) in advance and the robot does not use tactile feedback during path execution.

The results of the experiments on the bump are presented in Fig. 17. The visualization of the planned path is presented in the bottom row. In Fig. 17, we also visualize the state of the real robot during the execution of the planned path. The inclination of the robot's platform changes when the robot walks over the obstacle. The robot changes the orientation of the robot's body when at least one leg is placed on the obstacle. This is not the result of the strategy which defines the orientation of the robot's body according to the average inclination of the terrain like in [19], [40]. The inclination of the robot's body results from the posture optimization procedure which optimizes the kinematic margin d_{KM}. The same behavior can be observed when the robot climbs a step (Fig. 18) and deals with a concavity (Fig. 19).

The influence of the foothold selection method on the planned path is also visible in Fig. 17–19. The robot avoids placing its feet on the edges of the obstacles. The robot

[2]short video from experiments is available at https://youtu.be/FIamuQ4zqL0

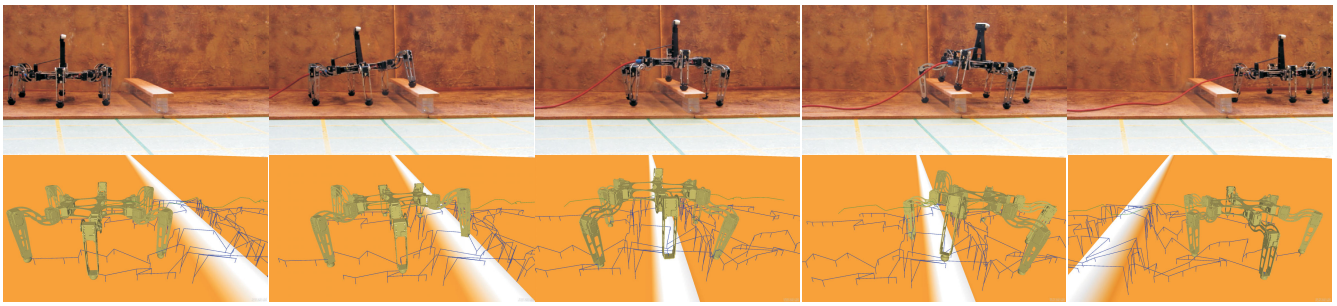

FIGURE 17. Snapshots from the experiment on the real robot while climbing a 0.085 m bump. Top row presents the selected frames from the video and the bottom row shows the corresponding visualization of the planned robot's pose.

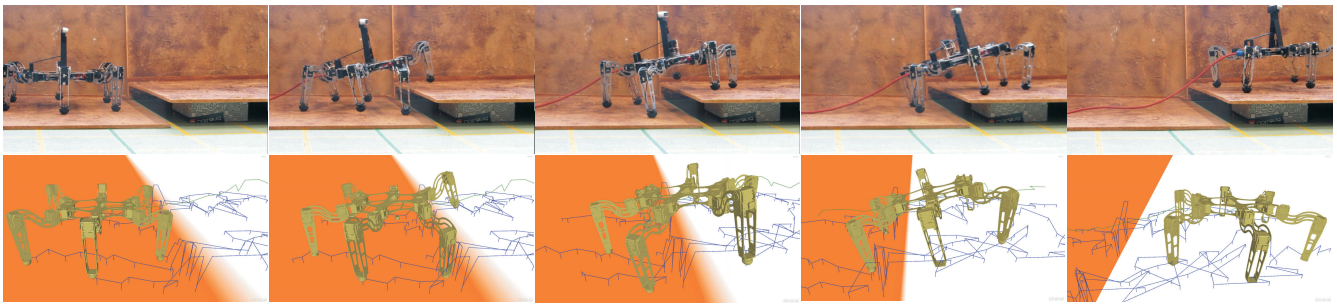

FIGURE 18. Snapshots from the experiment on the real robot while climbing a 0.105 m step. Top row presents the selected frames from the video and the bottom row shows the corresponding visualization of the planned robot's pose.

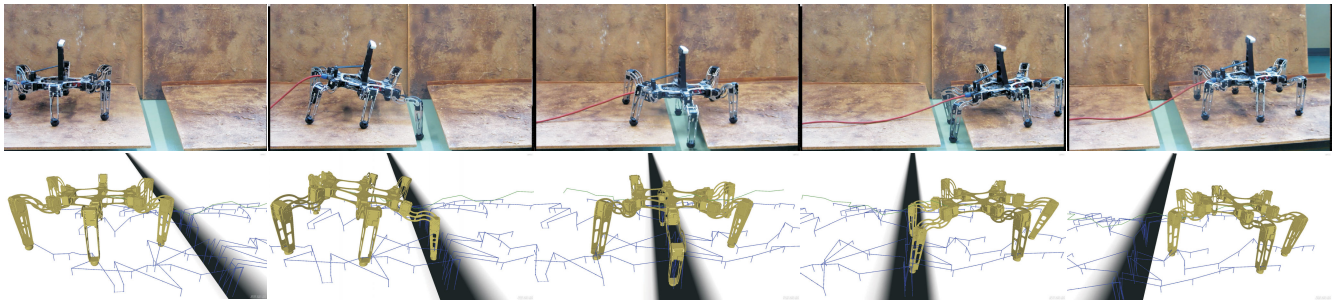

FIGURE 19. Snapshots from the experiment on the real robot while climbing a -0.1 m concavity. Top row presents the selected frames from the video and the bottom row shows the corresponding visualization of the planned robot's pose.

also plans the feet trajectories above obstacles which are collision-free and inside the workspace of the robot. The robot also moves its body up to deal with a bump (middle subfigure in Fig. 17). The robot also moves the body down to reach stable footholds which are located on the bottom of the concavity (middle subfigure in Fig. 19). In all cases, the robot was capable to execute the planned path.

VI. CONCLUSIONS

In this article, we show how to efficiently deal with constraint path planning for six-legged walking robots. Our motion planning framework deals with constraint path planning problem in multi-dimensional search space. In contrast to the most popular approach in the literature [1], [43] we avoid computing simplified transition cost to find the path of the robot. With the proposed constraints evaluation method we can find the precise and executable motion of the robot in real-time. This is due to the following contribution of the presented work.

- A new path planning framework for multi-legged walking robots. The proposed method is based on the RRT-Connect algorithm but the main contribution is related to the step planning method which precisely plans each step of the robot. The proposed strategy utilizes the foothold selection, posture optimization, and feet trajectory planning methods to efficiently explore the search space.
- Improved method for constraints evaluation. We propose the application of Gaussian Mixtures to generate self-collision model of the robot, collision model for the neighboring legs and workspace model (kinematic margin). With the proposed method the constraints

evaluation is very fast (few μs on a standard computer) and accurate. Moreover, we compute the gradient of the constraint function and optimize the configuration of the robot to avoid constraints.

- Application of the constraints evaluation methods in the path planning framework. We show how to use the constraint checking methods in the path planning framework. We show that all constraints should be evaluated for each new motion of the robot. To this end, the collision checking should be fast which is perfect application scenario for our constraints evaluation with GM.
- Experimental verification of the proposed methods. We verified our method in various scenarios. We show the properties of the proposed constraints evaluation method (accuracy and computation time) and the properties of the path planning method. We show the importance of the constraints evaluation in the path planning method by providing the numbers of each constraints checking during planning path of the robot. Finally, we show the experimental results on the real robot.

The results show that we can create a path planning system which increases the autonomy of the walking robot. The proposed system can be used to determine the way the robot climbs the obstacles but it can also be used to find the path to avoid obstacles which are not traversable. Because we use the kinematic and collision model, we can better exploit the motion capabilities of the robot. The robot can climb obstacles and at the same time avoid obstacles and remain statically stable.

The planning experiment and experiments performed on the real robot show that our method efficiently explores the search space. The constraints are checked millions of times in a single planning query. We check simple constraints at the beginning and finally we run the most-time consuming procedures to improve the computation time. The experiments on the real robot show how the robot adapts to the inclination of the terrain without directly computing this inclination. The behavior of the robot looks natural and results from the optimization strategy and not from hand-tuned heuristics.

We can also observe some limitations of the proposed method. The planning method is designed for local path planning. It means that the algorithm can be used to climb obstacles which are close to the robot. During planning the path of the robot to the distant goal, the RRT-based framework tries to explore the whole search space and the path planning becomes time-consuming. In this case, the planner should be guided by the method which can find the auxiliary path to the goal position [6], [48]. This strategy reduces significantly the planning time. On the other hand, the method requires precise localization and mapping method which have been addressed by [5], [7].

In the future, we are going to extend the proposed method by planning the path of the robot during climbing extreme obstacles. Currently, the path planning strategy is conservative and the workspace of the robot can be explored by looking for more risky configurations of the robot which allow climbing higher obstacles. We are also interested in motion planning strategy which assumes multiple contacts of the robot's body with the ground. Currently, we assume that the robot's feet can only touch the ground. When we assume that the robot body can be supported by the knees or the robot's platform, the climbing capabilities of the robot can be significantly increased.

ACKNOWLEDGMENT

The research on the general constraint modeling and evaluation was supported by EU Horizon 2020 project THING. Iterative methods on constraints evaluation were implemented during the author's research stay in Computational Robotics Laboratory of the Artificial Intelligence Center at the Faculty of Electrical Engineering, Czech Technical University in Prague.

REFERENCES

[1] A. Arain, I. Havoutis, C. Semini, J. Buchli, and D. G. Caldwell, "A comparison of search-based planners for a legged robot," in *Proc. 9th Int. Workshop Robot Motion Control*, Kuslin, Poland, Jul. 2013, pp. 104–109.

[2] M. Annunziato and S. Pizzuti, "Adaptive parameterization of evolutionary algorithms driven by reproduction and competition," in *Proc. Eur. Symp. Intell. Techn. (ESIT)*, Aachen, Germany, 2000, pp. 31–35.

[3] T. Augustyn and D. Belter, "Fast self-collision detection method for walking robots," in *Recent Advances in Automation Robotics and Measuring Techniques*, vol. 351, R. Szewczyk, Eds. 2016, pp. 549–559.

[4] D. Belter, J. Bednarek, H.-C. Lin, G. Xin, and M. Mistry, "Single-shot foothold selection and constraint evaluation for quadruped locomotion," in *Proc. IEEE/RSJ Int. Conf. Robot. Automat.*, Montreal, QC, Canada, May 2019, pp. 7441–7447.

[5] D. Belter and M. R. Nowicki, "Optimization-based legged odometry and sensor fusion for legged robot continuous localization," *Robot. Auton. Syst.*, vol. 111, pp. 110–124, Jan. 2019.

[6] D. Belter, P. Łabęcki, and P. Skrzypczyński, "Adaptive motion planning for autonomous rough terrain traversal with a walking robot," *J. Field Robot.*, vol. 33, no. 3, pp. 337–370, 2016.

[7] D. Belter, P. Łabecki, P. Fankhauser, and R. Siegwart, "RGB-D terrain perception and dense mapping for legged robots," *Int. J. Appl. Math. Comput. Sci.*, vol. 26, no. 1, pp. 81–97, 2016.

[8] D. Belter and K. Walas, "A compact walking robot—Flexible research and development platform," in *Recent Advances in Automation, Robotics and Measuring Techniques*, vol. 267, R. Szewczyk, C. Zieliński, and M. Kaliczyńska, Eds., 2014, pp. 343–352.

[9] D. Belter, "Optimization-based approach for motion planning of a robot walking on rough terrain," *J. Automat. Mobile Robot. Intell. Syst.*, vol. 7, no. 4, pp. 34–41, 2013.

[10] D. Belter and P. Skrzypczyński, "Posture optimization strategy for a statically stable robot traversing rough terrain," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Vilamoura, Portugal, Oct. 2012, pp. 2204–2209.

[11] D. Belter and P. Skrzypczyński, "Rough terrain mapping and classification for foothold selection in a walking robot," *J. Field Robot.*, vol. 28, no. 4, pp. 497–528, 2011.

[12] D. Belter and P. Skrzypczyński, "Integrated motion planning for a hexapod robot walking on rough terrain," in *Proc. 18th IFAC World Congress*, Milan, Italy, 2011, pp. 6918–6923.

[13] N. Benoudjit, C. Archambeau, A. Lendasse, J. Lee, and M. Verleysen, "Width optimization of the Gaussian kernels in radial basis function networks," in *Proc. Eur. Symp. Artif. Neural Netw.*, Bruges, Belgium, 2002, pp. 425–432.

[14] R. Buchanan, T. Bandyopadhyay, M. Bjelonic, L. Wellhausen, M. Hutter, and N. Kottege, "Walking posture adaptation for legged robot navigation in confined spaces," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 2148–2155, Apr. 2019.

[15] P. Cížek, D. Masri, and J. Faigl, "Foothold placement planning with a hexapod crawling robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Vancouver, BC, Canada, Sep. 2017, pp. 4096–4101.

[16] P. Fankhauser, M. Bjelonic, C. D. Bellicoso, T. Miki, and M. Hutter, "Robust rough-terrain locomotion with a quadrupedal robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, Brisbane, QLD, Australia, May 2018, pp. 1–8.
[17] P. Fankhauser, C. D. Bellicoso, C. Gehring, R. Dubé, A. Gawel, and M. Hutter, "Free gait—An architecture for the versatile control of legged robots," in *Proc. IEEE-RAS 16th Int. Conf. Humanoid Robots (Humanoids)*, Cancun, Mexico, Nov. 2016, pp. 1052–1058.
[18] A. Geva. (Oct. 2018). *ColDet 3D Collision Detection*. [Online]. Available: http://sourceforge.net/projects/coldet
[19] C. Gehring, C. D. Bellicoso, S. Coros, M. Bloesch, P. Fankhauser, M. Hutter, and R. Siegwart, "Dynamic trotting on slopes for quadrupedal robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Hamburg, Germany, Sep./Oct. 2015, pp. 5129–5135.
[20] I. Havoutis, J. Ortiz, S. Bazeille, V. Barasuol, C. Semini, and D. G. Caldwell, "Onboard perception-based trotting and crawling with the hydraulic quadruped robot (HyQ)," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot Syst.*, Tokyo, Japan, Nov. 2013, pp. 6052–6057.
[21] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Roennau, and R. Dillmann, "Unified GPU voxel collision detection for mobile manipulation planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Chicago, IL, USA, Sep. 2014, pp. 4154–4160.
[22] A. Hermann, F. Mauch, K. Fischnaller, S. Klemm, A. Roennau, and R. Dillmann, "Anticipate your surroundings: Predictive collision detection between dynamic obstacles and planned robot trajectories on the GPU," in *Proc. Eur. Conf. Mobile Robot.*, Lincoln, U.K., Sep. 2015, pp. 1–8.
[23] S. Hirose, H. Tsukagoshi, and K. Yoneda, "Normalized energy stability margin and its contour of walking vehicles on rough terrain," in *Proc. IEEE Int. Conf. Robot. Automat.*, Seoul, South Korea, May 2001, pp. 181–186.
[24] C. Igel, N. Hansen, and S. Roth, "Covariance matrix adaptation for multi-objective optimization,," *Evol. Comput.*, vol. 15, no. 1, pp. 1–28, 2007.
[25] R. L. Jenison and K. Fissell, "A comparison of the von Mises and Gaussian basis functions for approximating spherical acoustic scatter," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1284–1287, Sep. 1995.
[26] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Fast, robust quadruped locomotion over challenging terrain," in *Proc. IEEE Int. Conf. Robot. Automat.*, Anchorage, AK, USA, May 2010, pp. 2665–2670.
[27] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.
[28] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, Piscataway, Australia, 1995, pp. 1942–1948.
[29] J. Z. Kolter, M. P. Rodgers, and A. Y. Ng, "A control architecture for quadruped locomotion over rough terrain," in *Proc. IEEE Int. Conf. Robot. Automat.*, Pasadena, CA, USA, May 2008, pp. 811–818.
[30] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, San Francisco, CA, USA, Dec. 2000, pp. 995–1001.
[31] M. Likhachev, G. Gordon, and S. Thrun, "ARA∗: Anytime A∗ with provable bounds on sub-optimality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2003, pp. 767–774.
[32] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, p. 43, 2012.
[33] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local Gaussian process regression," *Adv. Robot.*, vol. 23, no. 15, pp. 2015–2034, 2009.
[34] P. Papadakis, "Terrain traversability analysis methods for unmanned ground vehicles: A survey," *Eng. Appl. Artif. Intell.*, vol. 26, no. 4, pp. 1373–1385, 2013.
[35] J. Park and I. W. Sandberg, "Universal approximation using radial-basis function networks," *Neural Comput.*, vol. 3, no. 2, pp. 246–257, 1991.
[36] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proc. IEEE Int. Conf. Robot. Automat.*, St. Paul, MN, USA, May 2012, pp. 3859–3866.
[37] Q. Pham, S. Caron, and Y. Nakamura, "Kinodynamic planning in the configuration space via admissible velocity propagation," in *Proc. Robot. Sci. Syst.*, Berlin, Germany, 2013, pp. 1–8.
[38] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 69–81, 2014.
[39] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, Kobe, Japan, May 2009, pp. 489–494.
[40] A. Roennau, G. Heppner, M. Nowicki, J. M. Zoellner, and R. Dillmann, "Reactive posture behaviors for stable legged locomotion over steep inclines and large obstacles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Chicago, IL, USA, Sep. 2014, pp. 4888–4894.
[41] J. A. Snyman and D. N. Wilke, *Practical Mathematical Optimization Basic Optimization Theory and Gradient-Based Algorithms*. New York, NY, USA: Springer, 2018.
[42] M. Stejskal, J. Mrva, and J. Faigl, "Road following with blind crawling robot," in *Proc. Int. Conf. Robot. Automat.*, Stockholm, Sweden, May 2016, pp. 3612–3617.
[43] A. Stelzer and H. Hirschmüller, M. Görner, "Stereo-vision-based navigation of a six-legged walking robot in unknown rough terrain," *Int. J. Robot. Res.*, vol. 31, no. 4, pp. 382–402, 2012.
[44] A. Stentz, "Optimal and efficient path planning for partially known environments," in *Proc. Int. Conf. Robot. Automat.*, San Diego, CA, USA, 1994, pp. 3310–3317.
[45] S. Tonneau, A. D. Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Trans. Robot.*, vol. 34, no. 3, pp. 586–601, Jun. 2018.
[46] O. Tropp, A. Tal, and I. Shimshoni, "A fast triangle to triangle intersection test for collision detection," *Comput. Animation Virtual Worlds*, vol. 17, no. 5, pp. 527–535, 2006.
[47] O. A. V. Magaña, V. Barasuol, M. Camurri, L. Franceschi, M. Focchi, M. Pontil, D. G. Caldwell, and C. Semini, "Fast and continuous foothold adaptation for dynamic locomotion through CNNs," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2140–2147, Apr. 2019.
[48] V. Vonásek, "Motion planning of 3D objects using rapidly exploring random tree guided by approximate solutions," in *Proc. 23rd Int. Conf. Emerg. Technol. Factory Automat.*, Turin, Italy, Sep. 2018, pp. 713–720.
[49] A. W. Winkler, C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2015, pp. 5148–5154.
[50] D. Wooden, M. Malchano, K. Blankespoor, A. Howardy, A. A. Rizzi, and M. Raibert, "Autonomous navigation for BigDog," in *Proc. IEEE Int. Conf. Robot. Automat.*, Anchorage, AK, USA, May 2010, pp. 4736–4741.
[51] M. Zucker, J. A. Bagnell, C. G. Atkeson, and J. Kuffner, "An optimization approach to rough terrain locomotion," in *Proc. IEEE Int. Conf. Robot. Automat.*, Anchorage, AK, USA, May 2010, pp. 3589–3595.

DOMINIK BELTER graduated from the Poznań University of Technology, in 2007, and received the Ph.D. degree in robotics from the Poznań University of Technology, in 2012, where he has been an Assistant Professor with the Institute of Control and Information Engineering (ICIE), since 2012, and also has been a member the Mobile and Walking Robots Team, ICIE. He has authored or coauthored more than 60 technical papers in the fields of robotics and computer science. His research interests include the control of walking robots and machine learning.