Free Gait — An architecture for the versatile control of legged robots

Conference Paper · November 2016

DOI: 10.1109/HUMANOIDS.2016.7803401

CITATIONS
6

READS
2,074

6 authors, including:

Péter Fankhauser
ANYbotics
46 PUBLICATIONS **1,183** CITATIONS
SEE PROFILE

Dario Bellicoso
Boston Dynamics
34 PUBLICATIONS **996** CITATIONS
SEE PROFILE

Christian Gehring
ANYbotics
39 PUBLICATIONS **1,388** CITATIONS
SEE PROFILE

Renaud Dube
ETH Zurich
40 PUBLICATIONS **411** CITATIONS
SEE PROFILE

Free Gait – An Architecture for the Versatile Control of Legged Robots

Péter Fankhauser[1], C. Dario Bellicoso[1], Christian Gehring[2], Renaud Dubé[2], Abel Gawel[2], Marco Hutter[1]

***Abstract*— This paper introduces *Free Gait*, a software framework for the control of robust, versatile, and task-oriented control of legged robots. In contrast to common hardware abstraction layers, this work focusses on the description and execution of generic whole-body motions (whole-body abstraction layer). The motion generation and motion execution algorithms are connected through the Free Gait API. This facilitates the development and execution of higher level behaviors and motion planning algorithms. The API is structured to accommodate a variety of task-space control commands. With these, the framework is applicable to intuitive tele-operation of the robot, scripting of user defined behaviors, and fully autonomous operation with motion planners. The defined motion plans are tracked with a feedback whole-body controller to ensure accurate and robust motion execution. We use Free Gait with our quadrupedal robot ANYmal and present results for rough terrain climbing, whole-body stair scaling, and special motions such as push-ups and squad jumps. Free Gait is available open-source and compatible with any type of legged robot, independent of the number of legs and joints.**

I. Introduction

Controlling robots with a high number of degrees of freedom (DOF) is a complex task and focus of active research. A major challenge for legged robots is the generation of movements through intermittent contact while ensuring stability and avoiding collision with the environment and with the robot itself. In addition, when implementing controllers for real robots, one has to account for real-time performance, safety, and robustness against uncertainties in actuation and sensing. For these reasons, developing software for legged robots is complex and coupled with significant development overhead and learning curve for new developers.

Abstraction layers provide a tool to hide complexity when systems become too difficult to efficiently work with. One example are computer systems where hardware and applications are separated by a series of abstraction layers such as firmware, kernel, libraries and more. Based on this idea, we have developed *Free Gait*, a framework for the versatile control of legged robots. The goal of Free Gait is to reduce the complexity of developing software for the motion generation of legged robots. Free Gait consists of a *whole-body abstraction layer* (Free Gait API) and several tools that are designed to interface higher level motion goals with the lower level tracking and stabilizing controllers. The separation between *motion generation* and *motion execution* through the Free Gait API enhances the ease of use, interoperability, and platform independence. The application of our framework ranges from intuitive tele-operation of the robot, scripting of user defined behaviors, to fully autonomous operation with the help of motion planners. Figure 1 provides an overview of possible motions which can be executed with the building blocks of the Free Gait API. Our Free Gait implementation is available open-source as C++ library with interfaces to the Robot Operating System (ROS).[1] It is designed with general legged robots in mind and compatible with existing whole-body controllers.

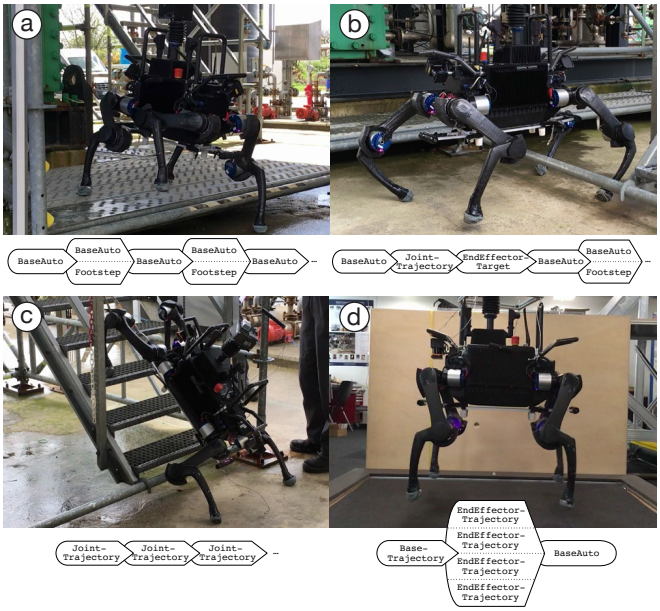

Fig. 1: Free Gait provides the command structure to control legged robots in various ways. a) The quadrupedal robot ANYmal [1] localizes with laser range sensors and climbs up a step based on a sequence of footholds. b) While walking, the robot switches the leg configuration (from X- to O-configuration) with help of a joint position trajectory. c) A whole-body motion planner generates joint trajectories to climb steep stairs. d) Free Gait is also applicable for dynamic motions such as a squad jump, which we synthesize with base and feet trajectories. A video of these maneuvers is available at https://youtu.be/EI1zBTYpXW0.

Previous work on abstraction layers in robotics has often focused on Hardware Abstraction Layers (HAL) [2, 3]. The goal here is to separate the communication with sensors and actuators from the perception and motion algorithms. From the motion generation perspective, the HAL provides the interface between the commanded joint states (position, torque, etc. commands) and the underlying actuator controller for electric, hydraulic, or other actuators. They are an important tool in robotics and many robots such as the LittleDog [4] provide such an interface. However, a HAL only captures

[1]Robotic Systems Lab, ETH Zurich, Switzerland (pfankhauser@ethz.ch)
[2]Autonomous Systems Lab, ETH Zurich, Switzerland
This work was supported in part by the Swiss National Science Foundation (SNF) through project 200021_149427 / 1 and the National Centre of Competence in Research Robotics.

[1]Available at http://github.com/leggedrobotics/free_gait

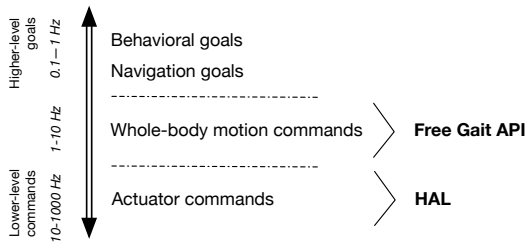

Fig. 2: The Free Gait API is designed for motion commands involving the entire structure of the robot (whole-body abstraction layer). This differentiates it from more abstract task-level goals and from the more local Hardware Abstraction Layer (HAL) on actuator level.

the local control on joint (or limb) level, whereas Free Gait is designed as an interface for whole-body motions. As illustrated in Fig. 2, the Free Gait API can be understood as intermediate layer between a HAL and more abstract navigation goals such as platform velocity commands as used in [5, 6].

Several software packages exist for the intuitive generation of robot motions, for example for the NAO robot [7, 8]. These tools provide complete development environments and build up on pre-defined sequences for behaviors such as 'stand up' or 'walk forward'. With these tools, whole-body motions are executed as interpolation between robot pose 'snapshots', which are defined by the user as joint position configurations. The designed motions are executed as open-loop joint position trajectories, an approach that requires highly controlled environments for successful motion execution.

A concept similar to our work has been discussed in [9] and [10]. In the former the HAL has been extended (and is limited to) to more abstract input types such as feet and base velocities relative to the ground. The velocity output of the different modules (gait generation, stability, obstacle avoidance, body posture, etc.) was then combined as weighted sum to command the robot. In the more recent work [10], a structure for the intuitive remote control of a Humanoid is proposed. It is based on the definition of desired joint angles, Cartesian goals for the end-effector, and pelvis poses. These commands are either directly executed on the robot or used as target for the joint or footstep planner. This work extends over these concepts by generalizing to the more versatile definition of commands including position, velocity, and force command types. Furthermore, Free Gait also includes the definition of targets (time-independent), entire trajectories (time-dependent), and automatized commands such as footsteps and pose adaptation commands.

In this paper, we introduce the Free Gait architecture (Section II), address several issues that arise when working with real robots (Section III), and describe three applications of the Free Gait framework (Section IV) before concluding our work (Section V).

II. Free Gait API

The Free Gait API connects the *motion generation* algorithms with the *motion execution* control (Fig. 3). The motion generation modules define the desired motion of the robot

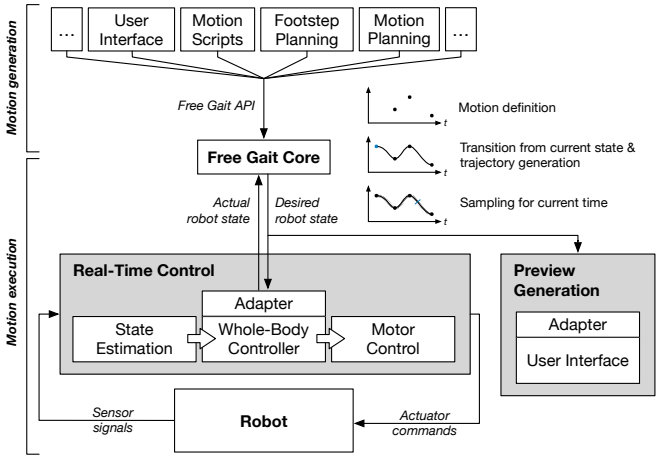

Fig. 3: Motion goals are commanded through the Free Gait API to the whole-body motion controller. The API is suitable to be interfaced with many motion generation sources such as user interfaces, motion behavior scripts, or footstep and motion planners. During motion execution, the desired robot state is sampled and tracked by the real-time whole-body controller.

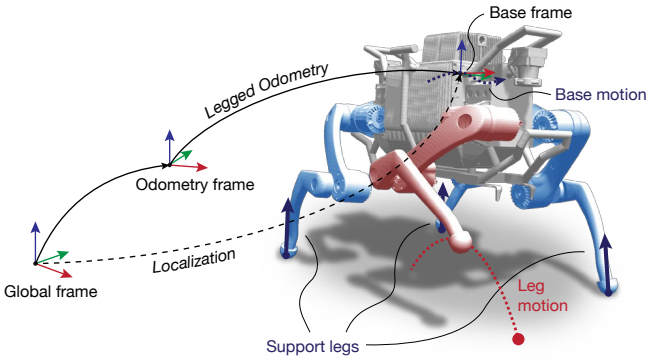

Fig. 4: Free Gait motions are based on a combination of (possibly multiple) leg motions and a base motion per command (step). The base motion defines indirectly the motion of the support legs. Both leg and base motions can be defined a reference frame suitable to the task.

in terms of various base and leg motion commands (Free Gait API). In the *Free Gait Core* module, these commands are transformed to a *robot desired state*, which describes the desired base pose and joint states for each time step of the motion execution. The motion is typically executed in a real-time controller that implements the tracking of the desired robot state with help of the state estimation and motor controllers.

In the following, we describe the structure and the elements of the Free Gait API.

A. Command Structure

A Free Gait command (or action) is composed of a list of *steps* (of type `Step`), which are executed successively in the specified order. During execution, continuous feedback on the progress of the execution is provided to the sender of the action. Once the command is successfully finished (or in case it could not be executed or was intentionally stopped), the according result is returned. At any time, the sender has the possibility to stop the execution and to replace it with new commands. In this case, the remaining steps are

discarded and the new commands executed. This enables the implementation of receding horizon controllers with the Free Gait API.

For the command definition, we differentiate between *leg motion* and *base motion* commands (Fig. 4). Leg motions describe the desired motion of the legs that do not contribute to the stabilization of the base (non-support legs). Depending on the application, leg motions can be expressed as joint space commands or as Cartesian space commands of the end-effector (foot). Base motions describe the desired motion of the base, which will be executed with help of the legs in contact with the environment (support legs). Both leg and base motions can be defined in three different ways:

Target	a single target state of a leg or the base,
Trajectory	a series of target states each associated with a timestamp,
Automatic	a command to use online motion generation/optimization for the legs (footsteps) or the base (pose adaptation).

An overview of the Free Gait commands is given in Table I. Note that each step in the command's list of steps can contain a combination of different leg and base motion definitions. This allows to express each command with the most suitable motion definition for the task at hand.

In the following, we discuss the different types of motion definitions of the Free Gait API in more detail.

B. Leg Motions in Joint Space

The joints of the robot's non-support legs can be controlled with the leg motion commands in joint space $\boldsymbol{q}(t) \in \mathbb{R}^n$ with n the number of joints per leg. In this case, a combination of desired joint position, velocity, acceleration, and torque can be defined for each joint of the leg. Combining different modes of control allows to determine the desired behavior of the leg. For example, specifying desired joint positions and velocities enables smooth motions while the combination of the joint position with a feed-forward torque is suited for interacting with the environment.

In the simplest case, a target joint state is defined as a `JointTarget` command type. Free Gait determines the duration of the motion based on the current state the joints, the desired state, and (optional) information about the desired average rate of change (e.g. average joint velocity in case of joint position commands). Building up on a rate definition (instead of direct duration information) decouples the resulting velocities and accelerations from the current state of the robot. This makes the target definition in practice more convenient and safer to work with.

For more complex joint motions, a full trajectory can be specified as type `JointTrajectory`. This command consists of a list of joint states each associated with a time and supports variable durations between the states. Upon start of the motion, Free Gait computes a spline that connects the current state of the joints and the desired joint state knot points (see Fig. 3). For each joint, we parameterize the trajectory as a quintic polynomial spline. Similarly, the `JointTarget` command is a joint trajectory with a start and end knot point.

For convenience, the `LegMode` command is used to determine if a leg should be used as support leg or not while keeping the current control mode and set points.

C. Leg Motions in Cartesian Space

As an alternative to the joint space commands, the desired foot motion of the non-support legs can be defined in Cartesian space. In this case, one specifies a combination of the desired position, velocity, acceleration, and force of the end-effector (foot) in Cartesian space as $\boldsymbol{f}(t) \in \mathbb{R}^3$.

Conceptually similar to the joint space target definition, the `EndEffectorTarget` command specifies a target state of the foot together with (optional) information about the desired average rate of change. Analogously, the `EndEffectorTrajectory` type is used to command a sequence of timed foot states. Importantly, both end-effector target and trajectory are expressed in a coordinate frame that can be chosen to suit the task at hand. On one side, this makes it easy to specify leg configurations relative to the base (base frame) which are independent of the robot's pose. On the other side, it is often useful to interact with the environment and to specify the desired foot state relative in a global frame. In the latter case, Free Gait tracks the state in the specified frame during execution, enabling robust and accurate motion execution even under external disturbances (slipping, pushes etc.).

As a specialized version of a foot trajectory, Free Gait implements a `footstep` command type for stepping to desired footholds. A footstep is determined by the target position (x-, y-, and z-coordinate) and the height and profile form of the swing motion. Currently, we have implemented triangle, square, and straight line swing profiles, which internally use three, four, and two spline knot points, respectively.

D. Base Motions

Base motion commands determine the motion of the robot's base/torso. The desired base motion is tracked with help of the support legs (Fig. 4). Similar to leg motions, a combination of the desired pose in $\mathbb{R}^3 \times \mathrm{SO}(3)$, twist and acceleration in $\mathbb{R}^6$, and net feed-forward force and torque at the base in $\mathbb{R}^6$ can be commanded. The base motion definition is independent of the number of support legs and the position of the feet. Again, both single targets as `BaseTarget` (timing determined through desired average rate of change) and full trajectories as `BaseTrajectory` are available. Internally, the base motions are parametrized as cubic Hermite splines [11]. As with leg motions in Cartesian space, both base targets and trajectories are associated with a reference frame.

Base motions are practical for the general locomotion of the robot where the center of mass has to be shifted to take steps. Additionally, we have found great use of them for stand up/lie down maneuvers and for positioning onboard equipment (e.g. cameras) at specific locations. In these cases, the desired base motions are intuitively defined and robustly

	Target	Trajectory	Automatic
Leg motion in joint space	`JointTarget`$^{+}$	`JointTrajectory`	`LegMode`
Leg motion in Cartesian space	`EndEffectorTarget`$^{*+}$	`EndEffectorTrajectory`*	`Footstep`$^{*+}$
Base motion	`BaseTarget`$^{*+}$	`BaseTrajectory`*	`BaseAuto`$^{+}$

* With frame definition $^{+}$ With average rate of change definition

TABLE I: Free Gait commands are expressed as a combination of leg (in joint or end-effector Cartesian space) and base motions with definition of position, velocity, and/or force/torque targets or trajectories. Some command types (*) are associated with a reference frame to match the task frame. Certain command types ($^{+}$) are based on the desired average rate of change (instead of duration) for convenience.

executed even from very different feet positions and leg configurations. It is important to note that for the definition of base target and trajectory commands, the motion generation has to ensure stability and compatibility with the current position of the feet.

Online motion generation for the base motion is provided by the `BaseAuto` command type. It computes a base target pose based on the current and, if available, next contact situation. The computation bases on the notion of *feet/legs in support* (given by current stance) and the *feet/contacts to reach* (defined be the motion generation). While the underlying pose adaptation algorithm should be adapted for different types of robots, we have found the following approach to work well with quadrupedal systems. For our implementation, the pose optimization finds a target base pose that minimizes the error to the nominal/'default' joint positions for all support feet and feet to reach under a set of constraints. The constraints include the stability constraint (defined by the support legs), kinematic limits and optionally collision constraints. The `BaseAuto` command is useful for using before, together with, and after leg motion commands. When using it as a preparation step before a leg motion command, it moves the base to a position from which the feet can be unloaded safely in the next step. The usage in parallel with leg motions moves the base in the direction of the anticipated target foothold, supporting the reachability of the robot. Using the pose adaptation after a contact change moves the base taking the new contacts into account, distributing the load to all legs and hence increasing the stability margin.

E. Output and Integration of Free Gait

While the Free Gait API is designed to support various motions commands, the output consists of a simple description of the desired robot state such that it can be easily integrated with existing robot control software (Fig. 3). The desired robot state consists for the non-support legs of the control mode (position, velocity, acceleration, and/or torque) and the desired set points for each joint. Similarly, the state also contains the control mode and set points for the base motion. The set points of the base motion are converted by the whole-body controller to joint commands of the support legs. Additionally, the surface normal for each contact is also contained in the robot state if it is given by the motion generation.

The implementation of Free Gait with the robot specific software is realized through adapters. An adapter is responsible to provide the actual state of the robot and forward the desired robot state to the robot controller. Through the adapter, the execution of the Free Gait commands is controlled with a free to choose sampling time Δt.

By separating the motion generation with the motion execution, Free Gait is robot platform agnostic, enabling to easily port motion generation methods and algorithms to a different robot. As Free Gait is also independent of the underlying whole body controller, implementing different control methods for the same control tasks requires only to write a new adapter without additional overhead. Furthermore, it is often useful to simulate or preview the motion plan, which can be accomplished by writing adapters for each and viewing the Free Gait commands before the execution on the real robot.

III. Implementation Considerations

The motion generation dictates the desired motion of the robot. In reality, the actual motion is never executed perfectly as defined and we have to deal with the problems that arise among others from inaccurate tracking, simplified models, sensor noise, drifting state estimation, and external, not modeled disturbances. To this end, we describe in the following some of the strategies in Free Gait that enable a robust motion execution with a real robot.

A. State Transitions

The desired/commanded state of the robot is often offset to the actual/measured state of the robot. This tracking error can be found for both leg and base motions and can range from a few millimeters/degrees to multiple centimeter and degrees. To enable a smooth transition from the actual state to the desired motion, one can use the actual state as the first knot point when generating the spline trajectories just before they are needed for execution. However, strictly following this strategy leads to the problem that a steady state error is cancelled at the time of transition leading to a undesired jump of the actuator reference value. To this end, Free Gait employs the following strategy: In case the control mode is the same before and after the transition from one step to the next, the desired set point is used as starting point. In case the control mode changes, the actual state is used.

B. Contact Transitions

A legged robot moves by changing the contact situation at each step. Hence, it is important that a planned contact is

actually established before the leg can be loaded to support the robot. Typically, contact sensors at the feet are employed to sense the contact with the environment. In practice, a contact change happens prior or after the planned transition. In case a contact is established before the step change, our implementation checks if the leg is planned to be a support leg in the next step and allows the leg to become a support leg prior to the step change. If no contact is expected (not a support leg in the next step), the motion is continued as planned. In case a contact is not established by the end of the step, the transition to the next step is paused and a behavior is triggered to gain contact. This behavior consists of a motion of the foot from its current position towards the surface along the surface normal. If the contact can be closed during this maneuver, the transition to the next step is performed. If the maximum extension of the leg is reached without contact, the Free Gait command is stopped and the result return as cancelled.

C. Reference Frames

The Free Gait commands for leg motions in Cartesian space and the base motions are given relative to a reference frame, as illustrated in Fig. 4. The pose of the robot estimated basing on the kinematics and inertial sensors is typically provided as a smooth trajectory with a high update rate, but might accumulate drift relative to the real motion of the robot.Other estimation methods, such as GPS, vision, or laser-based localization, work often drift-free but at lower rates and might undergo sudden discrete displacements (jumps).

In Free Gait, we combine pose estimation methods with the two different characteristics, the odometry methods (smooth/drift-afflicted) and the localization methods (non-smooth/absolute). We assume that the state estimation from odometry does not undergo significant drift during one step. When commands are to be executed in a localization frame, the trajectories are transformed to the odometry frame right before the execution. Hereby, we can execute motions defined in a frame provided by localization methods, which is important for accurate interaction with the environment.

IV. Results

We have used Free Gait with the two quadrupedal robots StarlETH [12] and ANYmal [1]. Both robot's actuation is based on Series Elastic Actuators (SEA), providing accurate position and torque control. The state estimation (legged odometry) is based on a Extended Kalman Filter (EKF), fusing kinematic and inertial measurements at 400 Hz [13]. Localization is performed with rotating laser range sensors and Iterative Closest Point (ICP) scan matching against a reference map [14]. For the motion execution, we have relied on two different whole-body controllers. One controller is based on the virtual model control concept and uses an optimization based contact force distribution (actuators in torque control mode) [5]. The other controller is based on the kinematic motion tracking with gravity compensation (actuators in impedance control mode).

Figure 1 shows four different Free Gait examples and their building blocks. In the following, we discuss three motion tasks in more detail illustrating the variety of motions that can be achieved with Free Gait. All maneuvers shown can be executed with either the virtual model or the impedance-based controller by simply switching the controller with no additional change necessary.

A. Motion Scripting

The Free Gait API makes it easy to define various motions. For example, simple Python scripts can be written to control the behavior of the robot for certain tasks. To further facilitate the generation of motion sequences, Free Gait includes a framework to parse and execute motion descriptions from YAML files [15]. We have used this framework to build up a database of scripted motions such as standing up/lying down, locking into the docking station, move joints to transport configuration, recovery from a fallen state, switching leg configurations (X- and O-configurations), and several pre-defined stepping sequences and demo motions (e.g. squad jump).

As an example, we discuss the following Free Gait YAML motion definition which describes the motion of a three-legged push-up:

```
steps:
 - step:
   - base_auto:
 - step:
   - end_effector_target:
      name: RF_LEG
      ignore_contact: true
      target_position:
        frame: footprint
        position: [0.39, -0.24, 0.20]
 - step:
   - base_auto:
      height: 0.38
      ignore_timing_of_leg_motion: true
   - end_effector_target: &foot
      name: RF_LEG
      ignore_contact: true
      ignore_for_pose_adaptation: true
      target_position:
        frame: footprint
        position: [0.39, -0.24, 0.20]
 - step:
   - base_auto:
      height: 0.45
      ignore_timing_of_leg_motion: true
   - end_effector_target: *foot
 - step:
   - footstep:
      name: RF_LEG
      profile_type: straight
      target:
        frame: footprint
        position: [0.32, -0.24, 0.0]
 - step:
   - base_auto:
```

The resulting motion is illustrated in Fig. 5. In the first step, the `BaseAuto` command on Section IV-A moves the base to a position that the leg in the next step can be safely lifted off ground. In the next step on Section IV-A, the right front (RF) foot is moved to a target position with

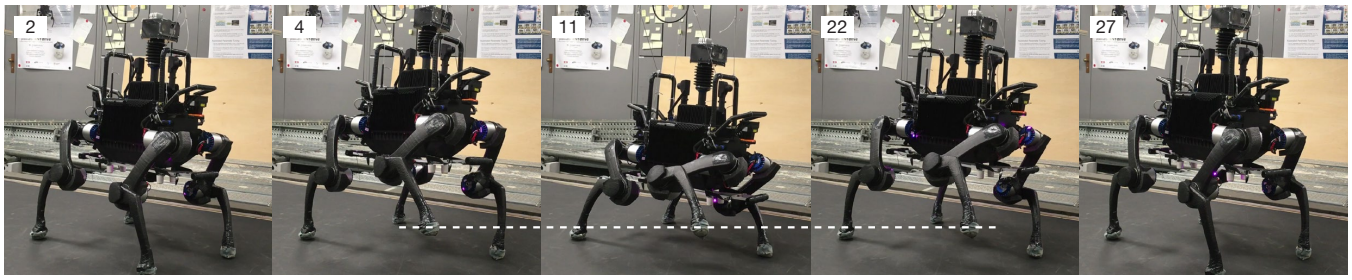

Fig. 5: A three-legged push-up motion to demonstrate the ease of use of the Free Gait API (total 35 lines of code). The numbers on the top left indicate the corresponding step command in the code example of Section IV-A

height 0.2 m defined in the footprint frame[2]. The statement `ignore_contact: true` (Section IV-A) instructs that no contact is expected at the end of the step. Section IV-A describes the step to lower the base to a height of 0.38 m (relative to the mean height of the feet) while keeping the right front foot in the air at same position.[3] The Section IV-A `ignore_timing_of_leg_motion: true` states that the timing of the `BaseAuto` command is not adapted to the leg motion, causing it the use the default average velocity value to compute the duration of the motion. The statement on Section IV-A `ignore_for_pose_adaptation: true` instructs the `BaseAuto` command to ignore the right front leg in the pose adaptation, causing the base to remain straight instead of tilting towards the elevated leg. The upward motion of the push-up to a height of 0.45 m is described in the fourth step (Section IV-A). In this step, the `EndEffectorTarget` is reused from Section IV-A with the YAML syntax for anchor `&` and reference `*`. Finally, the right front leg is lowered again (Section IV-A) and the base is moved to the center of all four legs (Section IV-A).

B. Stepping over Rough Terrain

Given a sequence of footholds from an operator or a footstep planner, a combination of the `Footstep` and `BaseAuto` commands can be used for walking over difficult terrain. In the following example, the footholds are defined in the global reference frame (given by the ICP localization) to climb up a step of 12 cm as shown in Fig. 6. The robot climbs up the step in four trials starting from different positions. During execution, we have perturbed the robot by manually pushing on the base. In one trial, we have blocked the foot from reaching the desired footstep with an obstacle, causing the foot to slip. Figure 6 shows the motion tracking of the feet during the climbing maneuver for all trials, illustrating the precision and robustness against disturbances of stepping.

Figure 7 depicts the foot trajectories of one trial in the global frame from localization, in the odometry frame from the legged state estimation, and in the combined approach as executed by Free Gait (Section III-C). The trajectory of the ICP localization is prone to undergo discrete jumps as shown in Fig. 7A, which are problematic for control. While the odometry based trajectories are smooth, they accumulate an error of ∼5 cm over the walked distance due to drift as shown in Fig. 7B. The combined approach unifies smoothness and accuracy and has shown to work to our satisfaction in our experiments.

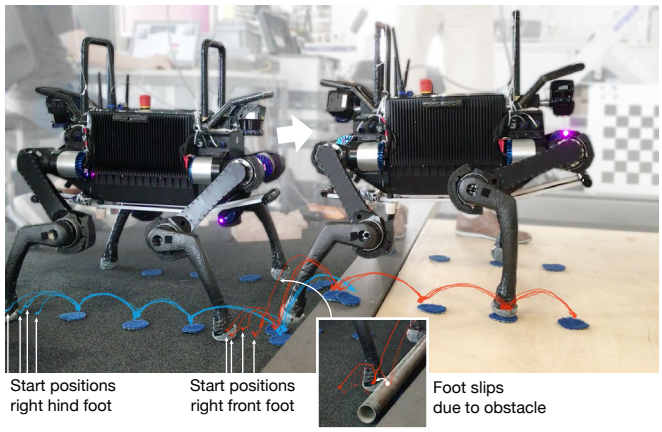

Fig. 6: To illustrate the precision and robustness of the Free Gait execution, the robot was commanded to climb a step in four trials from different starting positions. The robot localized itself with the onboard laser range sensors and the sequence of footsteps was defined in the global coordinate frame. During execution we pushed the robot several time for disturbance. In one trial, the foothold was blocked by an obstacle causing the foot to slip.

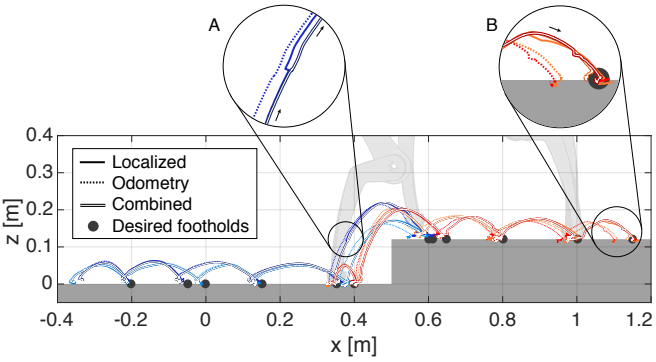

Fig. 7: Free Gait supports the motion definition in different frames. Internally, the movement is generated such that 'jumps' of low-frequency pose updates from localization are rejected (A) to allow for smooth but accurate motion tracking (combined approach) (B).

C. Whole-Body Motion Planning

For cases where stepping is not sufficient, we have developed a whole-body motion planner that takes contact of all links into account [16]. This way, climbing maneuvers can

[2]The footprint frame is defined as the geometric center of the stance feet.

[3]If the `EndEffectorTarget` command (Section IV-A) were not included, the foot would not be moved relative to the base and move down together with the base motion.

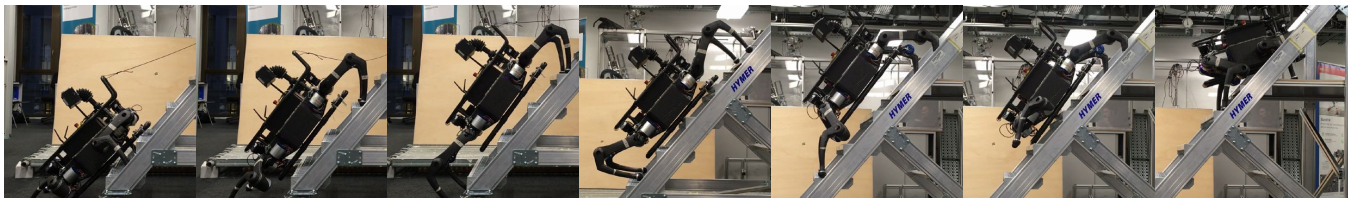

Fig. 8: ANYmal scales an industrial stair with an inclination of $> 50°$. A whole-body motion planner [16] computes the joint trajectories to be executed.

be executed that allow the robot to overcome high obstacles safely using support points on the base and knees. A high level climbing strategy is specified by the operator which consists of a sequence of contact situations. These contact situations can contain arbitrary points on the robot's body as support points. Based on information on the geometry of the environment, a motion planner finds the kinematic plan to connect the contact situations. The planner is accounts for collision with the environment, self-collision, static stability, and the kinematic limits of the robot. After the kinematic motion as been planned, gravity compensation is computed taking all contact points into account. The planner finds the motion at a time resolution of 10 Hz and transmits the motions as `JointTrajectory` with joint position and torque command to Free Gait for execution. After each contact situation, the whole-body motion planner computes the next transition to account for the current position of the robot.

Fig. 8 shows the whole-body motion planner being applied to climb industrial stairs (inclination $> 50°$) with ANYmal. The robot approaches the stairs by relying on contact between the base and the stairs to move the legs to a crab-like configuration. With help of hooks at lower part of the base, ANYmal then climbs up the stairs by alternating between base contact and contact at all feet to move the base to the next step. At the upper part of the stairs, a similar maneuver is performed to get off the stairs and stand up.

V. Conclusion

With Free Gait, we have realized a tool for the versatile and platform-agnostic control of legged robots to support the development of modular and robust motion generation algorithms. The supported motion commands provide a range of command types to solve different tasks robustly, efficiently, and intuitively. In this paper, we have presented three different applications and have discussed the implementation details that allow for robust and safe motion execution in difficult environments. We have so far used Free Gait on to two different quadrupeds and found it to be a valuable tool in our research and development. Notably, we've relied on Free Gait in our participation in the Total ARGOS Challenge to climb over steps and obstacles, scale stairs, and perform various maneuvers for the industrial inspection tasks.[4]

In the future, we continue to improve Free Gait and have several features in consideration. For example, we would like to add more control over the control gains over the trajectories and work on smarter execution behavior (execution delay and cancellation criteria) in case the motion is blocked. Also, we are interested in extending Free Gait for robotic manipulators and hand motions to enable full control of humanoids.

[4]Video available at https://youtu.be/SR5OJ-vklIs.

References

[1] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger, "ANYmal - a Highly Mobile and Dynamic Quadrupedal Robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[2] T. J. Murray, B. N. Pham, and P. Pirjanian, "Hardware abstraction layer for a robot," US Patent 6,889,118, 2005.

[3] S. Jorg, J. Tully, and A. Albu-Schaffer, "The Hardware Abstraction Layer - Supporting control design by tackling the complexity of humanoid robot hardware," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6427–6433, 2014.

[4] M. P. Murphy, A. Saunders, C. Moreira, A. A. Rizzi, and M. Raibert, "The LittleDog robot," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 145–149, 2010.

[5] C. Gehring, S. Coros, M. Hutter, M. Bloesch, M. A. Hoepflinger, and R. Siegwart, "Control of Dynamic Gaits for a Quadrupedal Robot," *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[6] T. Kerscher, A. Roennau, M. Ziegenmeyer, B. Gassman, J. Zolelner, and R. Dillman, "Behaviour-based control of the six-legged walking machine Lauron IVc," in *International Conference on Climbing and Walking Robots (CLAWAR)*, pp. 736–743, 2008.

[7] E. Pot, J. Monceaux, R. Gelin, B. Maisonnier, and A. Robotics, "Choregraphe: A graphical tool for humanoid robot programming," in *IEEE International Workshop on Robot and Human Interactive Communication*, no. November, pp. 46–51, 2009.

[8] G. Pierris and M. G. Lagoudakis, "An interactive tool for designing complex robot motion patterns," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4013–4018, 2009.

[9] D. Germann, M. Hiller, and D. Schramm, "Design and Control of the Quadruped Walking Robot ALDURO," *International Symposium on Automation and Robotics in Construction (ISARC)*, 2005.

[10] S. Kohlbrecher, A. Stumpf, A. Romay, P. Schillinger, O. von Stryk, and D. C. Conner, "A Comprehensive Software Framework for Complex Locomotion and Manipulation Tasks Applicable to Different Types of Humanoid Robots," *Frontiers in Robotics and AI*, vol. 3, p. 31, 2016.

[11] M.-J. Kim, M.-S. Kim, and S. Y. Shin, "A general construction scheme for unit quaternion curves with simple high order derivatives," in *Conference on Computer Graphics and Interactive Techniques*, pp. 369–376, 1995.

[12] M. Hutter, C. Gehring, M. Bloesch, M. A. Hoepflinger, C. D. Remy, and R. Siegwart, "StarlETH: A compliant quadrupedal robot for fast, efficient, and versatile locomotion," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2012.

[13] M. Bloesch, C. Gehring, P. Fankhauser, M. Hutter, M. A. Hoepflinger, and R. Siegwart, "State Estimation for Legged Robots on Unstable and Slippery Terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[14] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, 2013.

[15] O. Ben-Kiki, C. Evans, and I. döt Net, "YAML Ain't Markup Language Version 1.2," tech. rep., 2009.

[16] G. Wiedebach, *Whole Body Climbing with Legged Robots*. Master's thesis, ETH Zurich, 2016.