Trajectory Regulation for Walking Multipod Robots

Jörg Roth
Nuremberg Institute of Technology
Faculty of Computer Science
Nuremberg, Germany
e-mail: Joerg.Roth@th-nuernberg.de

Abstract— **Walking on a computed path is a fundamental task for mobile robots. Due to error effects such as slippage or imprecise mechanics, motion commands usually cannot exactly be executed. Thus, resulting positions and orientations may differ from the expectations. A robot has to compensate motion errors and should create a continuous movement that minimizes the difference between planned and real positions. This problem becomes even more difficult in case we have legged mobile robots instead of wheeled robots. In this paper, we present different mechanisms to regulate walking for multipod robots such as hexapods. They are based on virtual odometry, slippage detection and compensation as well as different types of regulation trajectories. These mechanisms are implemented and tested on the Bugbot hexapod robot.**

Keywords – Multipods; Hexapod; Autonomous Walking; Path-Following; Trajectory Regulation.

I. INTRODUCTION

A significant task of mobile robots is to navigate and move to a target position. The navigation in partly unknown environments is well-understood – we know numerous approaches to generate paths based on environment maps, created from sensor input. For the actual moving task, however, there is the problem of imprecise execution of movement commands. The problem is even more difficult, if we have walking robots such as hexapods as the movement consists of several phases with different motor actions. This is a major difference to driving robots, as their wheel-based movement typically can be fully controlled by steering angles or motor revolutions per time.

Figure 1 illustrates the problem. A hexapod should walk on a straight line. A slippery area on the ground (not known by the robot) has a lower traction because of, e.g., ice. If the robot walks without considering this area, the right feet provide smaller propulsion. As a result, the real trajectory is a curve to the right, not straight as intended.

As a desired behaviour, the robot should automatically adapt to the respective ground and execute a left curve to move back to the planned trajectory. On the first view, this problem is a typical regulation problem addressed by control theory. We could consider the joint angles (e.g., 18 for a 6-legged hexapod with 3 servos per leg) as the system output and want to minimize the difference of real and desired location and orientation. However, we have two significant differences to the classical problem.

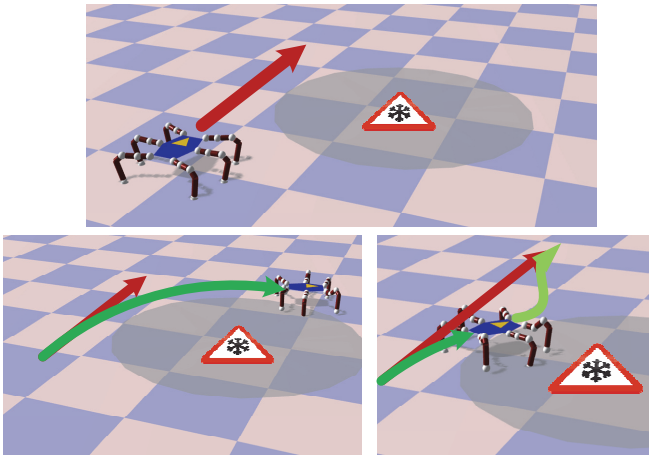

Figure 1. Illustration of the regulation problem

First, the regulation task is heavily influenced by non-holonomic walking constraints and obstacles in the environment. Second, the control output is not a set of joint angles, but a *sequence* of different joint angles, i.e., a function of angles over time (i.e., the *gait*). As a result, traditional tools to relate the joints speed and the resulting robot location (e.g., Jacobian matrices) are not suitable.

This paper presents a regulation approach based on the following ideas:

- We introduce the concept of *virtual odometry* to abstract from complex walking gaits.
- We measure and compensate *slippage* as main source of disturbance.
- We periodically compute *regulation trajectories* to a pose ahead on the formerly planned path.
- As the regulation trajectories are not computed in zero-time, we introduce *micro regulation* to compensate the delay.

This paper is an extended version of a shorter publication presented at the ADAPTIVE 2019 [1]. It extends the original paper in the following ways: First, we demonstrate the approach in more depth, in particular the mathematics behind it. Second, we introduce the concept of micro regulation to address the delay problem. Third, we provide a more detailed evaluation of the overall approach.

In Section II, we present related work. Section III describes our regulation approach. Experiments are presented in Section IV. Section V concludes the paper.

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

II. Related Work

Research on path following and trajectory tracking has a long tradition in control theory [2][3][4]. The basic goal is to provide a formal representation of the so-called *control law* [5]. Both, the vehicle and trajectories are strongly formalized in order to derive quality statements, in particular regarding the controller's stability [6].

Model Predictive Control (*MPC*) [7][8] is based on a finite-horizon continuous time minimization of predicted tracking errors. At each sampling time, the controller generates an optimal control sequence by solving an optimization problem. The first element of this sequence is applied to the system. The problem is solved again at the next sampling time using the updated process measurements and a shifted horizon.

Sliding Mode Control (*SMC*) is a nonlinear controller that drives system states onto a sliding surface in the state space [9][10]. Once the sliding surface is reached, sliding mode control keeps the states on the close neighbourhood of the sliding surface. Its benefits are accuracy, robustness, easy tuning and easy implementation.

The *Line-of-Sight* path following principle leads a robot towards a point ahead on the desired path. It is often used for vessels [11] or underwater vehicles [12]. The approaches differ how to reach the point ahead. Examples are arcs, straight lines or Dubins paths [13].

Another approach explicitly measures and predicts slippage, in particular of wheeled robots. As this is often a main source of disturbance to follow a path, it is reasonable to model it explicitly. In [14], effects on motors are measured for this. [15] uses GPS and inertial sensors and applies a Kalman filter to estimate slippage.

The majority of vehicles that are considered for the path following problem are wheeled robots, because their behaviour can be formalized easily. Multipods are only rarely taken into account. [16] presents trajectory planning and control for a hexapod that mainly keeps the robot balanced in rough terrain.

Pure Pursuit describes a class of algorithms that project a position ahead on the planned trajectory and create a regulation path (e.g., an arc) to reach this position. Early work about Pure Pursuit is [17]. The basic version only tries to reach a position ahead without considering the robot's orientation [18]. Improvements dynamically adapt the look-ahead distance [19].

Only rare approaches directly address the regulation problem of multipods. [20][21] consider the influence of disturbances such as grip on a climbing surface. The regulation is more fine-grained, i.e., on motor-level. As a result, the joint angles are controlled to execute a desired gait. The approach does not intend to regulate the robot movement to hold a certain trajectory or pose.

III. The Regulation Approach

Our regulation approach is embedded into the larger *Bugbot* project [22][23] (Figure 2). Bugbot is a hexapod robot, created to explore motion, navigation, world modelling and action planning for walking robots.

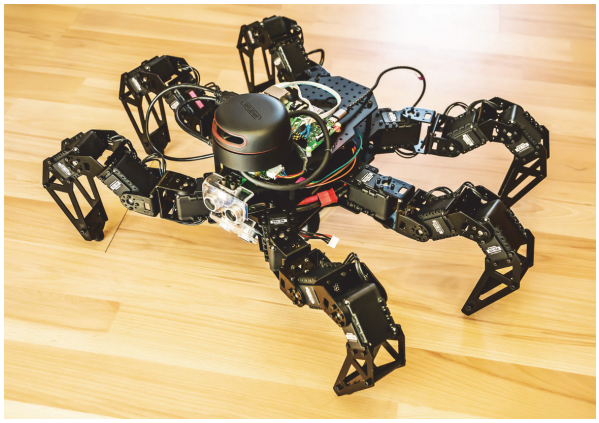

Figure 2. The Bugbot

The platform is an 18-DOF hexapod based on Trossen PhantomX Mark III. We added a Lidar device (light detection and ranging) and further sensors for collision detection and falling prevention. A Raspberry PI 3B is used for main computations, e.g., route and trajectory planning, SLAM and trajectory regulation.

Even though the Bugbot is fixed to six legs, the regulation approach is suitable for all multipods with *statically stable gaits* [24], e.g., spider-like octopods, but also robots with odd leg numbers, maybe mounted circular around the centre.

The Bugbot also comes along with a software stack (Figure 3). We have the following major components:

The *Robot Application* contains the actual task code for the robot's mission, e.g., to explore the environment, to carry things or to move to a target location.

Navigation provides a point-to-point route planning in the workspace (i.e., without dealing with the robot's orientation). This component does *not* consider non-holonomic constraints – these are shifted to lower components. It computes a line string of minimal costs that avoids obstacles. This component is useful to segment the overall path planning task into subtasks with lower complexity.

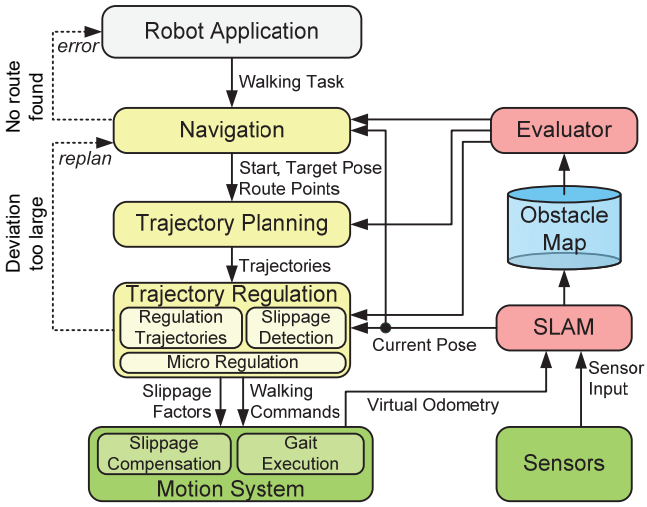

Figure 3. Data flow to execute walking tasks

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Trajectory Planning computes a walkable sequence of trajectories according to the formerly computed route points and now considers non-holonomic constraints. The trajectories are taken from a set of *primitive trajectories* such as straight forward or arc. We consider primitive trajectories as directly executable by the Motion System.

Trajectory Regulation permanently tries to hold the planned trajectories, even if the position drifts off. It contains the sub-components *Regulation Trajectories*, *Slippage Detection* and *Micro Regulation* that are described in the next sections.

Simultaneous Localization and Mapping (*SLAM*) constantly observes the environment and computes the most probable own location and locations of obstacles with the help of, e.g., ultrasonic sensors or Lidar. The current error-corrected configuration is passed to all planning components. Observed and error-corrected obstacle positions are stored in an *Obstacle Map* for further planning tasks.

The *Evaluator* computes costs of routes and trajectories based on the obstacle map and the desired properties. Cost values may take into account the path length, walking time, or expected energy consumption. In addition, the distance to obstacles could be considered, if, e.g., we want the robot to keep a safety distance whenever possible.

The *Motion System* is able to execute and supervise walking commands by formalized gaits. It considers slippage and provides *Virtual Odometry*. These concepts are also described in the next sections.

In this paper, we assume that *Navigation*, *Trajectory Planning* and *SLAM* already exist. We may use an approach as described in [22] for this. We here focus on the component *Trajectory Regulation*.

When starting with the Bugbot project in 2017, we already had long-term experience with a former robot, the wheeled *Carbot* [25]. Carbot also provided navigation, trajectory planning and motion components. However, the legged Bugbot was not as easy to formalize as a wheeled robot. Moreover, we had to face issues that made it difficult to apply traditional approaches based on control theory. Walking is in general more error-prone than driving. As a result, we cannot execute regulation trajectories as precisely as expected. As different multipods may be different in the capabilities to execute certain gaits, our goal was to consider the set of possible walking commands as black box. In particular, we did not want to restrict our general regulation mechanism by specific kinematic properties, e.g., by certain gaits or leg configurations.

Our approach should directly consider obstacles and arbitrary cost functions, again given as black boxes. A certain regulation trajectory may not only be based on regulation parameters, but also on the environment.

The resulting approach was inspired by the pure pursuit idea. We project the current position ahead to the target and try to get there. But we extended the basic idea in two ways:

- We try to reach a planned *configuration*, i.e., position *and* orientation. This is much more difficult than only to reach a planned position, but as a benefit, future configurations are much closer to the intended path.
- We are not restricted to a certain primitive trajectory (e.g., single arc) to reach the configuration ahead, but execute a full trajectory planning step that may result in multiple primitive trajectories.

We use the trajectory planning both to compute a full path to the final target, as well as for the regulation approach. As a benefit, both components produce output that the Motion System directly accepts as walking commands. Moreover, the motion capabilities are modeled in one place in the system. However, we have to face the following issues:

- As we do not explicitly model a control law, we have to consider sources of disturbance, foremost the slippage effect.
- As the regulation component permanently calls a trajectory planning, we have to consider execution time. In our approach, we thus apply an efficient trajectory planning approach.
- Even though executed fast, the trajectory planning is not performed in zero time. Thus, the robot slightly has moved, before the next trajectory is computed. We need a further mechanism, we call *micro regulation*, to compensate this effect.

In Figure 3, we also see a facility to report errors to the components above. If planned and real pose deviate too much, we consider the regulation as failed and restart navigation. With a newly planned route, the first deviations are small. With this facility, actually even naïve implementations of the Trajectory Regulation would work in theory, but may have bad performance in reality. In particular, we then had to face the *cascading failure problem* as described in Section III.E.

Before we describe the regulation approach in detail, we start with the motion model and mathematical foundations.

A. The Motion Model

Multipod robots often have legs as shown in Figure 4. Legs must have at last 3 degrees of freedom to freely place and move the foot during gait execution. The leg segments usually are called *Coxa*, *Femur* and *Tibia* based on insect anatomy naming. Robot legs with more degrees may provide redundancy in leg positioning, but are not generally capable to execute more gaits. In this paper, we abstract from inverse kinematics questions and assume the controlling mechanism is capable to place the feet as required by a movement.

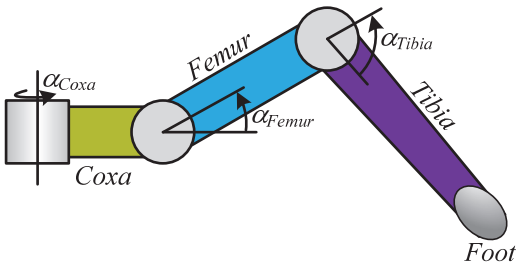

Figure 4. Typical construction of a multipod leg

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

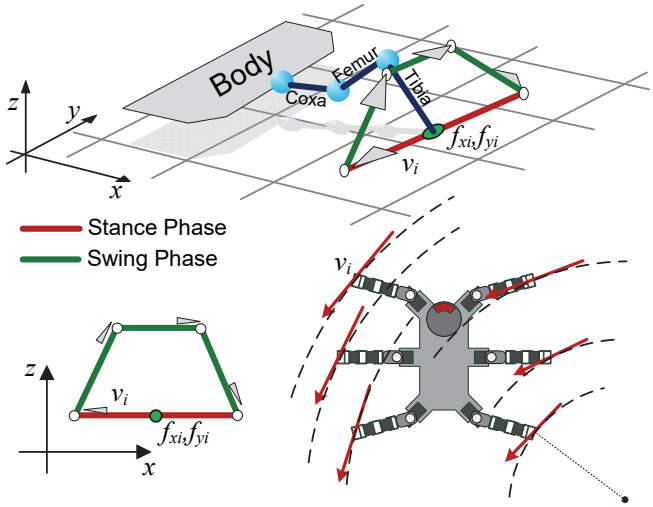

Figure 5. Structure of a multipod gait (top and left), v_i and arc trajectory (bottom right)

Multipods can walk in different ways. First, we can look at the actual trajectory, e.g., straight forward, sideways (i.e., crab gait), arc or turn in place. Second, we can distinguish the *gait* that defines the time sequence of moving legs.

We start with the trajectory. Each leg moves through two phases when walking:

- In the *stance phase,* the foot touches the ground. This phase is important for the robot's static stability. Always at least three feet must be in the stance phase, whereas the feet's convex hull must enclose the point below the centre of gravity.
- In the *swing phase,* the foot is lifted and moved to the start of the stance phase. During the swing phase, the leg can move over small obstacles.

Figure 5 shows the two phases for a specific leg. Let (f_{xi},f_{yi}) denote the neutral foot position of leg i. It marks the centre of a stance movement vector v_i in local robot coordinates. In world coordinates, the foot remains on the ground at the same position (in the absence of slippage). We assume the stance movement is linear or can at least be linearly approximated.

In the swing phase, the leg is lifted and moved in walking direction, whereas the ground projection may reside on v_i. The swing phase has a polygonal representation in three dimensions, but can be defined by a 2D polygon roto-translated to be aligned to v_i.

The second facet of multipod walking is the *gait*. Gaits define the cooperation of legs in the respective phases. Figure 6 shows the example of the *Ripple* gait, a gait that always has two lifted legs. Many further gaits are known, e.g., *Tripod* or *Wave* that differ in stability and propulsion [24]. A gait is fully described by a *gait matrix* that specifies per leg (row) and step (column) whether a leg is in swing phase (1) or stance phase (0). For the Ripple gait, we get the matrix

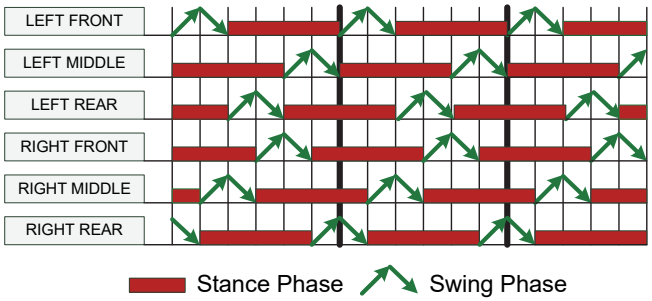

Figure 6. Gait pattern for the Ripple gait of hexapods

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (1)$$

We assume that gait execution and the choice for a certain gait are encapsulated in the Motion System component. For this, it receives the gait matrix and stance vectors v_i and is able to autonomously execute a gait as long as required by the respective trajectory.

An important observation: we can deal with movement trajectory and gait independently. This means that the respective trajectory shape is *not* influenced by the gait sequence pattern. The gait only affects the movement speed and stability.

B. Mathematical Foundations

For the intended approach, we need mathematical answers for three questions:

(A) Given a primitive trajectory, what are the stance vectors v_i to move along the required trajectory?

(B) Given two robot poses, what is the primitive trajectory that connects these poses?

(C) Given stance vectors v_i, what is the primitive trajectory the robot walks?

(A) and (C) are reverse questions. (B) and (C) are similar, but base on different input variables. We additionally could ask for the pose after walking a primitive trajectory. This would be the reverse of (B). However, this usually has a simple solution.

In the following, we consider two primitive trajectories: moving straight to target (t_x, t_y) and moving along an arc with centre c_x, c_y and curve radius r. More trajectories are conceivable, e.g., moving along clothoids. However, these have certain properties that are more suitable for driving robots. We also consider turning in place as primitive, but subsume it under the arc trajectory (whereas the arc centre is the robot centre).

We further assume that there exists a maximum stance vector length v_{max}. The maximum length is a result of the respective leg mechanics, e.g., leg segment lengths, servo angle limits and collision areas between the legs.

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

The solution (A) for straight trajectories is

$$v_i = -\begin{pmatrix} t_x \\ t_y \end{pmatrix} \frac{v_{\max}}{\left\| \begin{pmatrix} t_x \\ t_y \end{pmatrix} \right\|} \tag{2}$$

For arcs, we have the following constraints. First, the stance vector must be orthogonal to the line between neutral position and arc centre. Second, the ratio of stance vector lengths of two legs must be equal to the ratio of the distances between the respective neutral positions and arc centre. Third: the largest stance vector must have length v_{max}. More formally:

$$v_i \times \begin{pmatrix} f_{ix} - c_x \\ f_{iy} - c_y \end{pmatrix} = 0 \,, \frac{\|v_i\|}{\|v_j\|} = \frac{\left\| \begin{pmatrix} f_{ix} - c_x \\ f_{iy} - c_y \end{pmatrix} \right\|}{\left\| \begin{pmatrix} f_{jx} - c_x \\ f_{jy} - c_y \end{pmatrix} \right\|} \tag{3}$$
$$\max(\|v_i\|) = v_{\max}$$

With these equations, we can easily construct the v_i: we turn the line between neutral position and arc centre by 90°. Then we identify leg i that has the largest distance to the arc centre – this must receive the stance vector length v_{max}. Finally, we rescale the stance vector lengths according to the required relation, i.e.,

$$v_i' = \begin{pmatrix} c_y - f_{iy} \\ f_{ix} - c_x \end{pmatrix}, v_i = v_i' \cdot \frac{v_{\max}}{\max(v_i')} \tag{4}$$

For (B) we want to connect the current pose with a target pose (t_x, t_y, t_θ). For t_θ=0 we get a straight trajectory to (t_x, t_y). For $t_\theta \neq 0$ we get an arc with angle t_θ. Figure 7 shows the details.

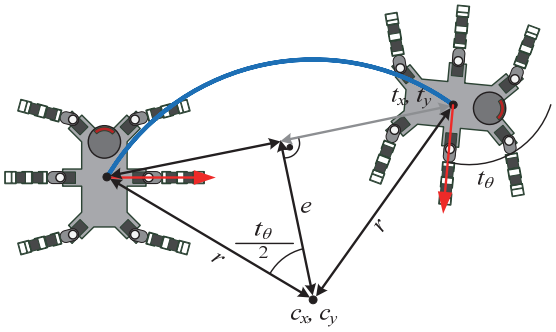

Figure 7. Construction of the arc geometry from two poses

We get the arc centre, if we add vectors $(t_x, t_y)/2$ and e, whereas e has the length

$$\|e\| = \left\| \begin{pmatrix} t_x \\ t_y \end{pmatrix} \right\| \frac{1}{2 \cdot \tan(t_\theta / 2)} \tag{5}$$

for f=1/(2tan(t_α/2)) we thus get

$$\begin{pmatrix} c_x \\ c_y \end{pmatrix} = \begin{pmatrix} t_x / 2 \\ t_y / 2 \end{pmatrix} + \begin{pmatrix} -t_y \cdot f \\ t_x \cdot f \end{pmatrix} \tag{6}$$

or

$$\begin{pmatrix} c_x \\ c_y \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & -1/\tan(t_\theta/2) \\ 1/\tan(t_\theta/2) & 1 \end{pmatrix} \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{7}$$

From (c_x, c_y), we finally get r.

Problem (C) is the hardest to solve. We first compute the amount of propulsion in a small time interval Δt. Even though a foot in stance phase remains on the ground, in local robot coordinates it moves along v_i. In turn, the neutral foot position (f_{ix}, f_{iy}) moves along $-v_i$ in world coordinates. When a foot remains in the stance phase during the time t_{st}, the foot moves during Δt

$$-\frac{\Delta t}{t_{st}} v_i \tag{8}$$

The time t_{st} depends on the respective gait (e.g., Tripod or Wave) and can be derived from the gait matrix: it is the ratio of zeros in a row, multiplied by total cycle time. Note: even though a foot in swing phase provides no actual propulsion to the robot, the respective (f_{ix}, f_{iy}) virtually move continuously, also for $t > t_{st}$. As a result, we have a constant speed over all steps.

Viewed from the first position, the feet virtually move from $f_i=(f_{ix}, f_{iy})$ to $f'_i=(f'_{ix}, f'_{iy})$ whereas

$$\begin{pmatrix} f'_{ix} \\ f'_{iy} \end{pmatrix} = \begin{pmatrix} f_{ix} \\ f_{iy} \end{pmatrix} - \frac{v_i}{t_{st}} \tag{9}$$

To find the respective primitive trajectory, we need a function Ψ that computes

$$\begin{pmatrix} t_x \\ t_y \\ \alpha \end{pmatrix} = \Psi\left((f_1 \ldots f_n), (f'_1 \ldots f'_n) \right) \tag{10}$$

Ψ denotes a function to compute a roto-translation, which maps all positions of the first list to positions of a second list, meanwhile minimizing the mean square error. We apply an approach based on Gibbs vectors [26] for Ψ: We look for a rotation matrix R and translation t with

$$f'_i = R \cdot f_i + t \tag{11}$$

According to the Cayley transform [27], we are able to replace the rotation matrix as follows

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

$$f_i' = (I+Q)^{-1}(I-Q)\cdot f_i + t \tag{12}$$

where I is the unity matrix and $Q=[q]_\times$ with q the vector of Rodriguez parameters (Gibbs vector), where $[]_\times$ denotes the cross product operation by a matrix, i.e., $a\times b=[a]_\times \cdot b$.

We can rewrite this to

$$\begin{aligned} f_i'(I+Q) &= (I-Q)\cdot f_i + t(I+Q) \\ &= (I-Q)\cdot f_i + t^* \end{aligned} \tag{13}$$

where $t^*=t(I+Q)$. We get

$$f_i' - f_i = -Q(f_i + f)_i + t^* \tag{14}$$

For two dimensions we get

$$\begin{aligned} y = \begin{pmatrix} f_{1x}' - f_{1x} \\ y_{1y} - f_{1y} \\ \dots \\ f_{nx}'' - f_{nx} \\ f_{ny}' - f_{ny} \end{pmatrix} &= \begin{pmatrix} f_{1y} + f_{1y}' & 1 & 0 \\ -(f_{nx} + f_{nx}') & 0 & 1 \\ & \dots & \\ f_{1y} + f_{1y}' & 1 & 0 \\ -(f_{nx} + f_{nx}') & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} g_\alpha \\ t_x^* \\ t_y^* \end{pmatrix} \\ &= H \cdot \begin{pmatrix} g_\alpha \\ t_x^* \\ t_y^* \end{pmatrix} \end{aligned} \tag{15}$$

This is an overdetermined linear equation system. We minimize least squares with

$$\begin{pmatrix} g_\alpha \\ t_x^* \\ t_y^* \end{pmatrix} = (H^T H)^{-1} H^T y \tag{16}$$

We finally get the roto-translation (t_x, t_y, α) with

$$\begin{aligned} \alpha &= 2\cdot \arctan(g_\alpha) \\ \begin{pmatrix} t_x \\ t_y \end{pmatrix} &= \begin{pmatrix} \frac{1}{1+\tau^2} & \frac{\tau}{1+\tau^2} \\ \frac{-\tau}{1+\tau^2} & \frac{1}{1+\tau^2} \end{pmatrix} \cdot \begin{pmatrix} t_x^* \\ t_y^* \end{pmatrix} \end{aligned} \tag{17}$$

where $\tau = \tan(\alpha/2)$.

C. Computing Trajectories

A basic building block for the trajectory regulation is to compute regulation trajectories, i.e., such trajectories that bring the robot back to a planned path. As a basic idea, we use for regulation trajectories and long-range paths the same approach [22]. We adapted the original approach that was optimized for a wheeled robot [25] to a walking robot. It is based on the following ideas:

- The navigation component (Figure 3) solely operates on workspace $\mathcal{W}$ and computes a sequence of collision-free lines of sight (with respect to the robot's width) that minimize the costs.
- As the navigation only computes route points in $\mathcal{W}$, we have to specify additional variables in $\mathcal{C}$ (here orientation θ). From the infinite assignments, we only consider a small finite set.
- From the infinite set of trajectories between two route points, we only consider a finite set of *maneuvers*. Maneuvers are sequences of primitive trajectories, for which we know formulas that derive the respective parameters (e.g., curve radii) from start and target configurations.
- Even though these concepts reduce the problem space to a finite set of variations, this set would by far be too large for complete checks. We thus apply a Viterbi-like approach that significantly reduces the number of checked variations to find an optimum.

We carefully separated the cost function (component *Evaluator*, Figure 3) from planning components. We assume that there is a mapping from a route or trajectory sequence to a cost value according to two rules: first, we have to assign a single, scalar value to a trajectory sequence that indicates its costs. If costs cover multiple attributes (e.g., walking time *and* battery consumption), the cost function has to weight these attributes and create a single cost value. Second, a collision with obstacles has to result in infinite costs.

The basic capabilities of movement are defined by the supported set of primitive trajectories. The respective set can vary between different robots. A walking robot should support:

- $L(\ell)$: linear (straight) walking over a distance of ℓ;
- $T(\Delta\theta)$: turn in place over $\Delta\theta$;
- $A(\ell, r)$: move a circular arc with radius r (sign distinguishes left/right) over a distance of ℓ

We are able to map primitive trajectories directly to walking commands that are natively executed by the robot's Motion System (Figure 3).

A certain multipod may also support *holonomic* locomotion, e.g., walk straight and turn the viewing direction during a single motion command. In this case, however, we have changing stance vectors over time, what complicates the gait formalism. Moreover, for a certain walking scenario not all trajectories may be reasonable. E.g., we may expect a camera, an ultrasonic sensor or the sensors that prevent from falling downstairs to always point in walking direction. Thus, we require identical viewing and walking directions for all trajectories of type *A* or *L*, i.e., the robot only moves in the direction, it also views. Considering these constraints, it is not possible to reach a certain pose with a single primitive trajectory. At this point, we introduce *maneuvers*. Maneuvers are small sequences of primitive trajectories (usually 2-3 elements) that are able to map given start and target configurations c_s, $c_t \in \mathcal{C}$. More specifically:

- A maneuver is defined by a sequence of primitive trajectories (e.g., denoted *ALA* or *AA*) and further

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

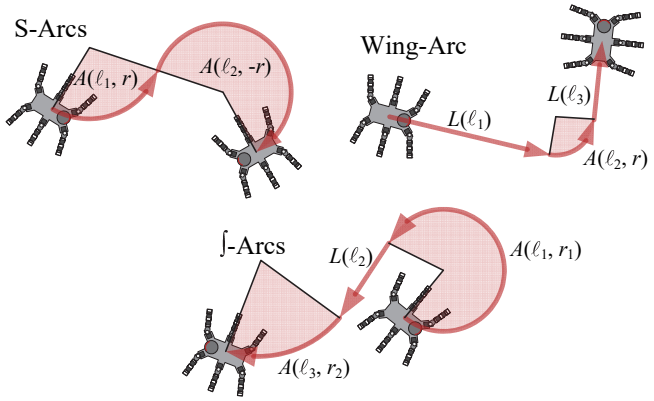

Figure 8. Example maneuvers

constraints. Constraints may relate or restrict the respective primitive trajectory parameters.

- For given c_s, $c_t \in \mathcal{C}$ there exist formulas that specify the parameters of the involved primitive trajectories, e.g., ℓ for L, A and ℓ, r for A.
- Sometimes, the respective equations are underdetermined. As a result, multiple maneuvers of a certain type (sometimes an infinite number) map c_s to c_t. Thus, we need further parameters, we call *free parameters* to get a unique maneuver.

Figure 8 shows three example maneuvers: S-Arcs (*AA*), Wing-Arc (*LAL*) and ∫-Arcs (*ALA*). For, e.g., S-Arcs we have to set up equations to get the respective free parameters ℓ_1, ℓ_2, r. To simplify the computation, we first roto-translate start and target to move the start to (0, 0, 0) and target to (x_t', y_t', θ_t'). We get

$$r = \frac{y_t'(1+ct) - x_t' st - \sqrt{x_t'^2 (st^2 - 2ct + 2) + y_t'^2 (3 + ct^2) - 2x_t' y_t' st(1+ct)}}{2(ct-1)}$$

$$\ell_1 = r \cdot \arccos\left(\frac{(1+ct)}{2} - \frac{y_t'}{2r}\right) \qquad (18)$$

$$\ell_2 = \ell_1 - r \cdot \theta_t'$$

where st=sin(θ_t'), ct=cos(θ_t'). We identified a total of 8 maneuvers so far (Table I). We assigned names that illustrate the maneuver's shape, e.g., the J-Bow goes through a path that looks like the letter 'J'. The Dubins-Arcs correspond to the combination with three arcs of Dubins original approach [13]. From all maneuvers, ∫-Arcs can be considered as a 'Swiss knife': it allows reaching any target configuration without a turn in place whereas the middle linear trajectory spans a reasonable distance to the target.

We also may invent new maneuvers to increase the overall walking capabilities. For a new maneuver, we only have to set up equations that derive the respective trajectory parameters from start and target configuration.

To find a trajectory sequence of maneuvers is an optimization problem. For a given start and target pose in $\mathcal{C}$ and a list of route points in $\mathcal{W}$ we have to find a sequence of maneuvers (and thus primitive trajectories) that

TABLE I. AVAILABLE MANEUVER TYPES

Maneuver	Pattern	Free Parameters
1-Turn	*LTL*	no
2-Turns	*TLT*	no
J-Bow	*LA*	arc radius
J-Bow2	*AL*	arc radius
∫-Arcs	*ALA*	two arc radii
S-Arcs	*AA*	no
Wing-Arc	*LAL*	arc radius
Dubins-Arcs	*AAA*	(same) arc radius for all three arcs

- connect start pose, route points and target pose,
- minimizes the costs, computed by the Evaluator.

The controllable variables are: the maneuver types, their free parameters and the orientation angles at the route points. From the infinite set of the respective variations, we choose a finite promising set of candidates. Even though finite, the number of variations still is by far too large for a complete check. To give an impression: for 5 route points we get a total number of 20 million, for 20 route points $2 \cdot 10^{37}$ permutations. Obviously, we need an approach that computes an appropriate result without iterating through all permutations.

Our approach is inspired by the Viterbi algorithm that tries to find the most likely path through hidden states. To make use of this approach, we replace 'most likely' by 'least costs', and 'hidden states' by 'unknown parameters'. We thus look for a sequence of maneuvers/orientations/free parameters that connect them with minimal costs. Details of the underlying algorithm can be found in [22].

This approach is suitable, because optimal paths have a property: the interference between two primitive trajectories in that path depends on their distance. If they are close, a change of one usually also causes a change of the other, in particular, if they are connected. If they are far, we may change one trajectory of the sequence, without affecting the other. Viterbi reflects these characteristics, as it checks all combinations of neighbouring (i.e., close) maneuvers to get the optimum. We can reduce the number of variations to check for a complete route to some thousands.

As a further benefit of the approach: The first primitive trajectory of the final path converges very fast. If fixed, the robot can start walking, while further trajectories are computed during the movement. This property makes our trajectory planning as an ideal candidate for regulation trajectories: The regulation trajectory is frequently computed in the background, each of it only, until the first primitive trajectory is fixed. During walking, the next primitive trajectory can be computed by following iterations.

D. Virtual Odometry

Leg movement with a complex timing pattern is difficult to handle in the context of trajectory planning and regulation. For geometric computations the model of turning wheels is more convenient, because we have a simple relation between motor revolutions and odometry. This leads to the idea of *virtual odometry*: We transform walking to corresponding wheeled movement. We could think of roller skates attached

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

to the multipod's legs while the legs remain in neutral position (f_{xi}, f_{yi}).

To compute virtual odometry, we intercept the formal gait description that is passed to the Motion System, namely:

- the neutral position (f_{xi}, f_{yi}) for each leg i,
- the stance vector v_i for each leg i,
- the time t_{st} that the gait resides in stance phase for a complete cycle of one stance and one swing phase.

According to formulas (10)-(17) we are able to compute a roto-translation (t_x, t_y, α) that maps the neutral leg positions to the moved neutral positions meanwhile minimizing mean squares. We further get (c_x, c_y) in case of an arc from formula (7).

We now assume in a small time interval the robot either only moves an arc (respectively turn in place) or linear trajectory. For small intervals and thus small running lengths, this is a reasonable approximation. We further consider the angular velocity and absolute speed as constant in a small time interval. If we consider both arc and straight movement, we get the moving distance for a leg i over time Δt as

$$\ell_i(\Delta t) = \Delta t \cdot \begin{cases} |\alpha| \sqrt{(f_{xi} - c_x)^2 + (f_{yi} - c_y)^2} & \text{if } \alpha \neq 0 \\ \sqrt{t_x{}^2 + t_y{}^2} & \text{if } \alpha = 0 \end{cases} \quad (19)$$

We call $\ell_i(\Delta t)$ the *virtual odometry*. It represents the *expected* portion of the overall moving distance of each foot when walking.

E. Slippage Detection and Compensation

With the help of virtual odometry we now are able to compute the expected run length and expected location. If they deviate from the planned trajectory, we try to walk back, using regulation trajectories (see below).

If we did not take into account slippage, the regulation trajectories will not be executed as expected, thus the deviation may increase (Figure 9). Even though the robot permanently tries to go back on the planned path, the distance gets larger, because the regulation trajectories were also executed with a drift (here, to the left). In this case, we get what we call the *cascading failure problem*: at a certain point, the regulation fails and triggers a new route planning by the navigation component. This however, can only fix the problem for a certain time, as also further routes are not executed as expected. Finally, the robot, e.g., moves too close to a wall and the entire movement is stopped with an error.

In addition to the expected walking distances, we now need the real distances. We assume that the robot owns sensors (e.g., Lidar) and respective SLAM mechanisms that permanently estimate the robot's real position. We consider these mechanisms as black box, but expect, they detect the real pose change (t_x', t_y', α') after walking a time Δt. We again assume that during Δt, only a single movement pattern is executed. This obviously is wrong, if there is a change in the trajectory (e.g., changing from arc to straight). However, for small Δt, we can model both patterns by a single ('average') pattern, thus receive only small errors.

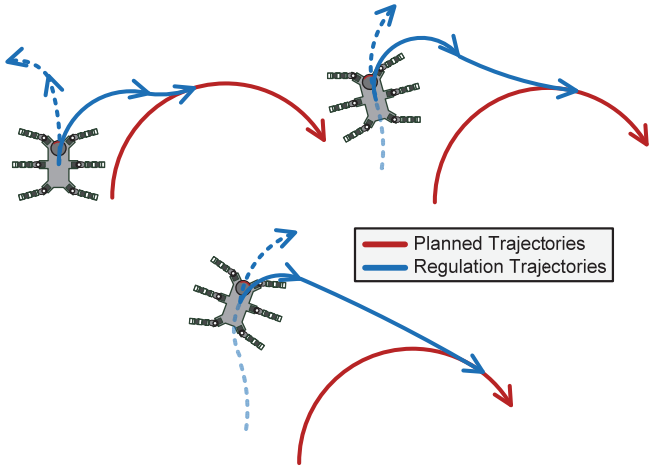

Figure 9. Problem of missing slippage compensation

For given (t_x', t_y', α') we can apply formulas (7) and (19) to get the *real* walking distances $\ell_i'(\Delta t)$. We define

$$S_i = \frac{\ell_i(\Delta t)}{\ell_i'(\Delta t)} \quad (20)$$

as the *leg-specific slippage factor* and

$$S = \frac{1}{L} \sum_i S_i \quad (21)$$

as the *general slippage factor*, where L is the number of legs. Obviously $S \geq 1$ in reality. S describes the slippage property of the current bottom's pavement. E.g., $S=2$ means that the robot walks half as far as expected when executing a certain trajectory. The S_i describe slippage *per leg* and could indicate malfunctioning leg servos or feet that do not properly touch the ground.

We are able to compensate slippage in two ways:

- only compensate the general slippage;
- also compensate leg-specific slippage.

The assumption is: what we measured recently is a good estimation for the nearer future. If, e.g., we walk on a slippery floor, we can consider the respective slippage factor when executing next trajectories, because it is likely to reside on the same floor for a certain time. To consider the general slippage factor, we have to extend the respective trajectory by the discovered factor:

- Walking straight over a certain distance, we have to multiply the planned distance by S.
- Walking on an arc, we have to multiply the planned arc angle by S.

To consider the leg-specific slippage is more difficult. The problem: these factors do not only affect the trajectory length, but also its shape. E.g., if we want to walk straight

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

with different factors S_i for left and right legs, the robot effectively walks on an arc instead. A first approach would be to extend the respective stance vectors. E.g., if we got S_i=2 for a specific leg (i.e., the leg produces only half of the expected propulsion), we could enlarge v_i by 2 to compensate this effect. However, this is not always possible, because the stance vector lengths are limited to v_{max} – either by the mechanics, or because neighbour legs should not collide during walking. Thus, we usually are only able to shorten the stance vectors. Our approach is to compute

$$S_{\max} = \max(S_i)\,,\ \widetilde{S}_i = S_i / S_{\max}\,,\ \widetilde{v}_i = v_i \cdot \widetilde{S}_i \qquad (22)$$

We use S_{max} as the general factor to extend the trajectory and multiply each leg's stance vector by $\widetilde{S}_i$. Note that $\widetilde{S}_i \leq 1$, thus a stance vector only can get smaller.

It depends on the respective scenario, whether the compensation only should consider the general slippage or should also apply a leg-specific compensation. The latter is only reasonable, if we expect a leg-specific slippage that may be result of malfunctioned legs or different pavement for different legs.

F. Regulation Trajectories

The task to compensate the drift during walking and to meet the planned trajectories is related to control theory, where a system tries to produce a desired output with the help of controllable input values. In the case of trajectory regulation, however, the desired output is a pose that usually cannot directly be achieved by adapting current joint angles or by a primitive walking operation. Due to non-holonomic constraints, we usually require a *sequence* of trajectories, i.e., our maneuvers.

To explain our approach, we need some definitions. First, we need a function *TP* that provides a trajectory planning from start pose *s* to target pose *t* based on Section III.C.

$$(T_i) = TP\big((s_x, s_y, s_\theta), (t_x, t_y, t_\theta)\big) \qquad (23)$$

The (T_i) is a sequence of primitive trajectories. We further need to identify an *expected* pose *e* of a current pose *c*.

$$(e_x, e_y, e_\theta) = E\big((T_i), (c_x, c_y, c_\theta)\big) \qquad (24)$$

Expected means: the intended pose on the planned path for a given pose. If the multipod remains on the planned path, *c* and *e* are identical. If the pose leaves the planned path, we have to introduce a notion of '*nearest pose on the trajectory*', whereas we may have different definitions for this. The function *E* may be *stateful* or *stateless*. A stateful implementation observes the current walking task and identifies the expected pose based on walking time or virtual odometry. As an example: we could measure the walking distance from the start of walking on (T_i) and identify the pose that has the same distance from the start. A stateless implementation only identifies the nearest trajectory point based on geometric distances. In our experiments (Section IV), we implemented the stateless version.

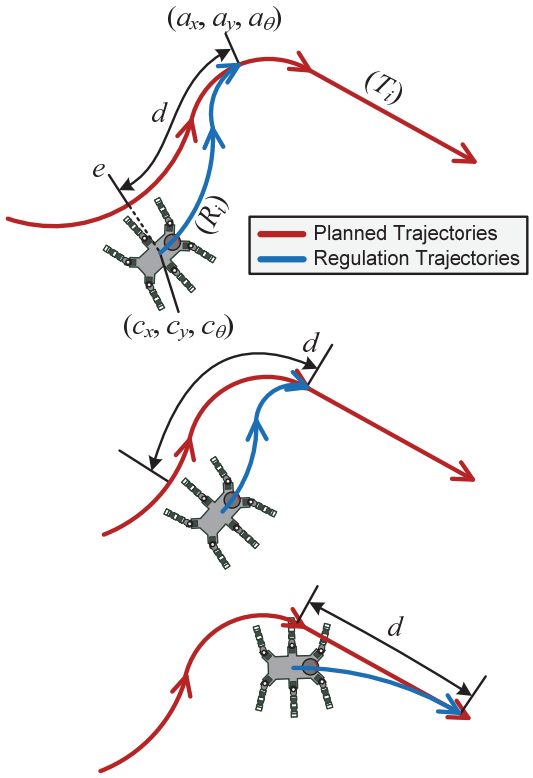

Figure 10. Idea of regulation-ahead

We finally need a function *A* that projects the current expected pose *ahead*.

$$(a_x, a_y, a_\theta) = A\big((T_i), e, d\big) \qquad (25)$$

Here, *d* describes how much the current expected pose is projected ahead in target direction. We assume *d* to be constant. Figure 10 illustrates the idea.

We now compute a trajectory sequence (R_i) that brings the robot back to the originally planned trajectory. Our approach is to compute

$$(R_i) = TP\big((c_x, c_y, c_\theta), A\big((T_i), E\big((T_i), (c_x, c_y, c_\theta)\big), d\big)\big) \qquad (26)$$

The major benefit: we do not have to introduce a new mechanism to plan regulation trajectories, but re-use the function *TP*. One could suggest to bypass regulation trajectories and directly compute *TP*(*c*, *t*). However, the pose ahead is much closer to the current pose, thus a planning is much more efficient. Furthermore, we do not expect obstacles between current and ahead pose, as the original path already is planned to be obstacle-free.

We finally have to think about *d*:

- For a small *d*, we force the robot to walk on sharp turns to restore the planned trajectory sequence.

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

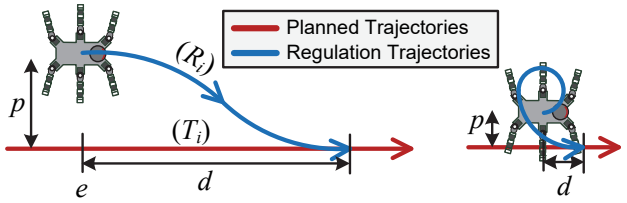

Figure 11. Effects of large and small d, p

- For a large d, the robot walks a long time parallel to the planned trajectory before it reaches the meeting point.

Both effects lead to higher costs – either because the path gets significantly longer or because the robot walks on positions with higher costs, only nearby the planned trajectory.

Figure 11 illustrates these effects. In this example, we planned a linear trajectory and the real position is besides the linear trajectory with distance p, but with correct orientation angle.

If both p and d are large, usual regulation trajectories contain two arcs. If p and d are small (Figure 11 right), the regulation trajectory may be a spiral that starts in opposite direction. This situation is unwanted, as the regulation first enlarges the distance to the planned trajectory.

We want to investigate this effect. As a first observation, it heavily depends on the walking capabilities, in particular the set of primitive trajectories and minimal arc radii, in addition the cost function. We thus cannot give a general specification of a 'good' d. However, we can provide an idea to discover d for a respective scenario.

Let $|(R_i)|$ be the length of the regulation trajectories. We define

$$q = \frac{|(R_i)|}{d} \tag{27}$$

as the *stretch factor*. It specifies how much longer the regulation path is compared to the way on the planned path. Figure 12 shows typical curves of q.

Due to the effect presented in Figure 11 (right), small d result in high q. At a certain point (here at d=40 cm) q is close to 1.0. For d>40 cm, we get only minor improvements

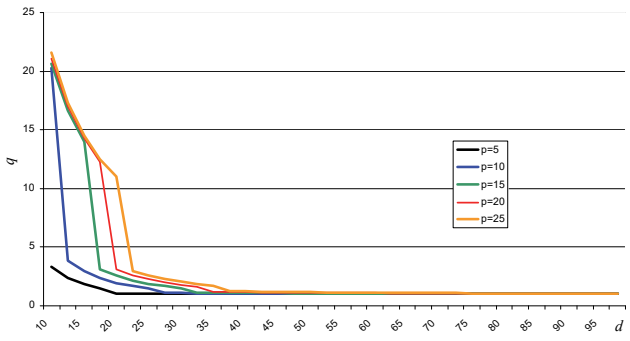

Figure 12. Typical stretch factors (d, p in cm)

of q. As a result, d=40 cm is a good choice for our scenario.

This is only an example for a certain scenario. If we want to discover an appropriate d for other scenarios, we have to consider the range of expected position errors (here p), but also the expected orientation errors.

G. Micro Regulation

We can compute (R_i) with the function *TP* periodically, e.g., every few seconds without considerable stressing the CPU. However, on very small mobile platforms that also execute additional tasks (e.g., image processing), the corresponding planning process may be delayed. This may cause a problem: *TP* is executed for a specific pose, but when *TP* finished, the robot has slightly moved to another pose. If the computed regulation trajectory then is applied to the new position, the endpoint does not reside on the originally planned trajectory. This in particular is a problem, if the orientation angle differs from the expectation.

It is not reasonable to stop the movement during *TP* computation as this would seriously disturb the continuous movement of legs. This forms the idea of another type of regulation – the *micro regulation*: We compute a short path that moves back to the regulation trajectory with a *single primitive trajectory*. Usually, it is not possible to reach a position *and* orientation with a single trajectory without to relax some constraints.

The idea of maneuvers (i.e., *multiple* primitive trajectories) respects the requirement to always walk in forward-direction and considers the non-holonomic constraint not to move side-ways. However, a walking robot *is* able to move sideways (like a crab). The idea of micro regulation is to create arc trajectories that cautiously make use of this possibility. To respect sensors that only scan in forward-direction, the amount of side-ways movement should be very small compared to forward movement.

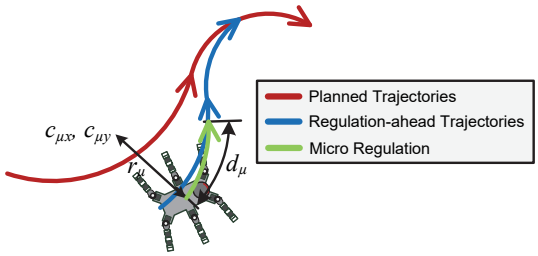

Figure 13. Idea of combined micro regulation and regulation-ahead

Figure 13 shows the idea: according to the solution of problem (B) (Section III.B), we compute a single trajectory (usually an arc) that brings the robot back to the regulation trajectory. This arc is not like the *A* trajectory (Section III.C) as the corresponding arc centre does not necessarily resides ±90° to the viewing angle. As a result, the walking direction slightly goes sideways.

Micro regulation requires fewer computation power as *TP*, as only a single trajectory has to be computed. Because we only slightly leave the regulation trajectory, we additionally may ignore obstacles. As a result, we have two background loops with different cycle times: one loop that com-

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

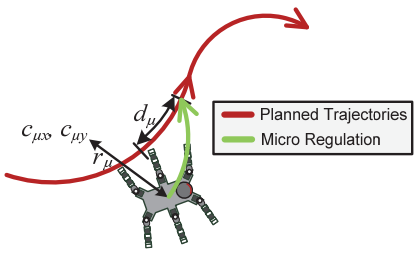

Figure 14. Idea of pure micro regulation

putes a regulation trajectory (e.g., every 4 s) and one loop that computes a micro regulation trajectory (e.g., every 2 s). In addition, the micro regulation loop has a smaller ahead distance d_μ, e.g., 20 cm. For a certain robot, the cycle time and ahead distance again must be subject to experimental optimization.

We can even go one step further (Figure 14): We do not necessarily have to compute a micro regulation trajectory to the regulation trajectory (R_i). We instead could try to reach to planned trajectory (T_i). We call the approach in Figure 14 *pure micro* regulation, in contrast to *ahead micro* (Figure 13). In case of pure micro, however, we expect a larger amount of sideways walking, as there is a greater distance between real pose and desired target pose. This effect is investigated with experiments in the next section.

IV. EXPERIMENTS

We implemented our trajectory regulation approach on the *Bugbot* platform. Even though we fully tested the approach on this platform, it was difficult to create a huge number of different experiments in reality. E.g., it is costly to test the slippage detection for different floors and different slippage factors. It is even a problem to create pavements with a very specific constant slippage factor. It is also a problem to adjust leg-specific slippage in reality in a fine-granular manner. Moreover, it is very difficult to create reproducible test runs as slippage and traction vary over time, even for the same pavement. This makes it difficult to compare results. We thus created a simulation environment that simulates the Bugbot on hardware- and physical level. A physical simulation component is able to compute gravitation, any form of slippage and collision effects. The control software is the same as on the real hardware, i.e., the simulator's Bugbot model is able to create sensor values and carries out native servo commands.

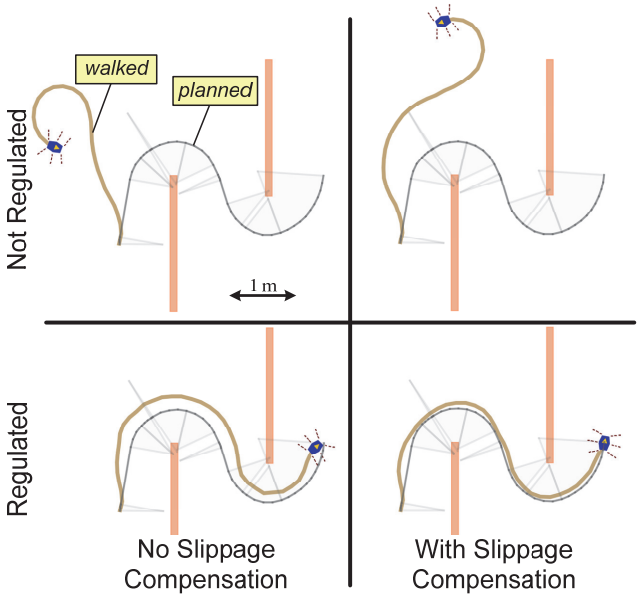

Figure 15. Simple walking scenario

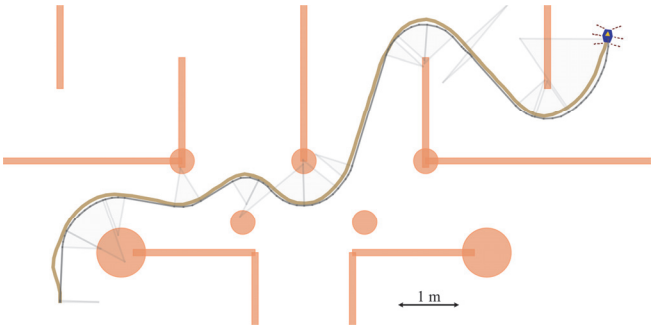

Figure 16. Complex walking scenario (regulated, slippage compensation)

We divided the experiments in two groups. The first group shows the effects of regulation vs. no regulation and slippage compensation vs. no compensation. The second group shows the effect of the different regulation approaches.

A. Regulation and Compensation

Figure 15 shows an example to illustrate the effects of slippage compensation and regulation. For regulation, we applied the regulation-ahead approach as described in Section III.C. We simulated a leg-specific slippage of 2.0 for the three left legs. This means that without any compensation, the robot walks a left arc when planned to walk right (Figure 15 top, left). With slippage detection and compensation, the shape of the planned path is mainly represented. But because the compensation is applied not before a small learning phase, the shape is rotated at the beginning (Figure 15 top, right).

Figure 15 bottom shows the regulation. On the left we see an effect when the regulation tries to meet the planned path. Because the regulation trajectories are not executed properly, we see a constant offset. On the right, we finally see both mechanisms – after a learning phase, the planned trajectory is reproduced very precisely.

Figure 16 shows a more complex example. Here, we again assigned a leg-specific slippage of 2.0 (only left legs) and in addition a general slippage of 2.0. This represents a very difficult scenario. If regulation and slippage compensation were applied, we can see a high congruence of planned and walked path.

B. Comparison of Regulation Approaches

The second group of experiments investigates the properties of *regulation-ahead*, *pure micro* and *ahead micro*. For this, we modified the complex walking scenario above and added different zones of slippage (Figure 17).

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

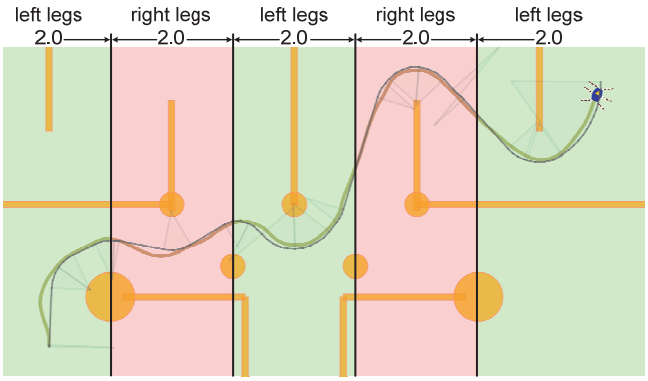

Figure 17. Walking scenario with changing slippage

We again assigned a general slippage of 2.0, but alternated the leg-specific slippage (again 2.0) every 2 m between left to right legs. This stressed the regulation mechanism: after adapting to a certain slippage, it significantly changes and the mechanism first has to learn the new situation.

We applied slippage detection and compensation and only changed to way to compute regulation trajectories. We tested the three types as shown in Table II.

TABLE II. PARAMETERS FOR THE TEST CASES

Regulation approach	d (ahead)	cycle time (ahead)	d_μ	cycle time (micro)
regulation-ahead	40 cm	4 s	n/a	n/a
micro (pure)	n/a	n/a	20 cm	2 s
micro (ahead)	40 cm	4 s	20 cm	2 s

For the ahead distance d, we selected the value from Section III.F. As we have a maximum speed of approx. 9 cm/s, a cycle time of 4 s is sufficient. For the micro regulation we choose half values, both for the ahead distance d_μ and cycle time. For the three test runs we measured three values:

- The distance d_{err} between real position and the nearest position on the planned path.
- The absolute angle error α_{err} between real orientation and the trajectory direction of the nearest planned path point.
- The absolute heading error h_{err}: it is the difference between viewing direction and walking direction. E.g., 0° means walking strictly forward, 90° means walking sideways in crab gait.

Figure 18 shows the results. We can easily see the decrease of errors d_{err} and α_{err} after adapting to the slippage. Whenever the robot enters a new area of slippage, we can see increasing errors. The errors are considerable low, even in worst case. The characteristics of h_{err} are different whether micro regulation is applied or not. If not, h_{err} is always zero. This is because TP already ensures that (R_i) only contains trajectories with forward heading. Micro regulation tries to compensate angle errors with trajectories that continuously change the heading with arcs. Not surprisingly, we thus can see a strong correlation of α_{err} and h_{err}.

Table III shows the averages of the respective values. In addition, we measured the computation time to compute regulation trajectories (R_i), micro regulation, or both in % of the overall CPU time. As we have a considerably long time between the respective computations (2 or 4 s), the total amount of time is very low.

TABLE III. TEST RESULTS

Regulation approach	CPU load in %	avg(d_{err}) in cm	avg(α_{err}) in °	avg(h_{err}) in °
regulation-ahead	0.00855	2.9	4.6	0
micro (pure)	0.00550	1.5	7.9	8.2
micro (ahead)	0.0131	1.9	4.5	3.5

Looking at the error values for all three types, the distance of real and planned position is very small. The angle errors α_{err} and h_{err} are more significant. If we have strong demands according the orientation, pure micro is not recommended as we have a maximum of 8° both for orientation and heading error. In summary, ahead micro provides the best results, but slightly requires more CPU load than regulation-ahead.

V. CONCLUSIONS

This paper presented different mechanisms to the path following problem for multipods. We formalized gaits and introduced virtual odometry to abstract from the respective leg configuration. Slippage detection and compensation is used to map planned trajectories to movement commands that are executed more precisely. We compute regulation trajectories with the help of efficient trajectory planning already used for long-range path planning to the final target. We also suggest micro regulation in case when some non-holonomic constraints can be relaxed.

The look-ahead distances currently are based on the developer's experience and experiments. Whereas small distances may lead to instabilities, larger distances increase the time to meet the planned path and moderately increase the cost value, thus are less critical. However, in the future we also want to make the ahead-distance as part of the controllable state.

REFERENCES

[1] J. Roth, “Regulated Walking for Multipod Robots”, ADAPTIVE 2019 – The Eleventh IARIA International Conference on Adaptive and Self-Adaptive Systems and Applications, May 5-9, 2019, Venice, Italy, 15-20

[2] D. Dacic, D. Nesic, and P. Kokotovic, “Path-following for nonlinear systems with unstable zero dynamics”, IEEE Trans. Autom. Control, Vol. 52, No. 3, 2007, pp. 481–487

[3] A. Morro, A. Sgorbissa, and R. Zaccaria, “Path following for unicycle robots with an arbitrary path curvature”, IEEE Trans. Robot., Vol. 27, No. 5, 2011, pp. 1016–1023

[4] P. Walters, R. Kamalapurkar, L. Andrews, and W. E. Dixon, “Online Approximate Optimal Path-Following for a Mobile Robot”, 53rd IEEE Conference on Decision and Control December 15-17, 2014. Los Angeles, California, USA

[5] S. Blažič, “A novel trajectory-tracking control law for wheeled mobile robots”, Robotics and Autonomous Systems 59, 2011, pp. 1001–1007

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

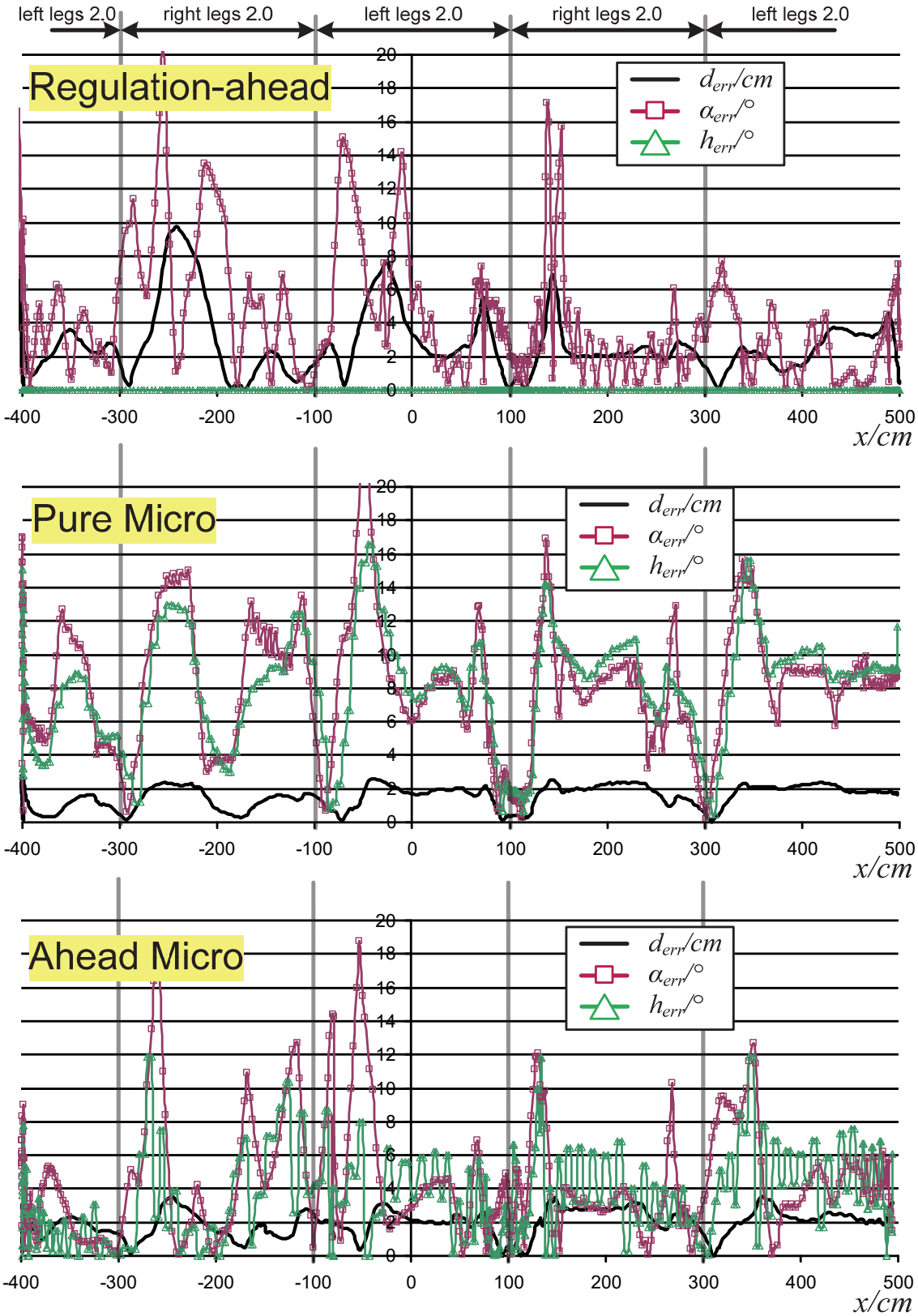

Figure 18. Detailed test run results

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

[6] R. W. Brockett, “Asymptotic stability and feedback stabilization", in R. W. Brockett, R. S. Millman, and H. J. Sussmann, (eds.), Differential geometric control theory, Birkhauser, Boston, 1983, pp. 181–191

[7] K. Kanjanawanishkul, M. Hofmeister, and A. Zell, “Path Following with an Optimal Forward Velocity for a Mobile Robot”, Elsevier IFAC Proceedings Volumes, Vol. 43, No. 16, 2010, pp. 19–24

[8] J. E. Normey-Rico, J. Gómez-Ortega, and E. F. Camacho, “A Smith-predictor-based generalised predictive controller for mobile robot path-tracking”, Control Engineering Practice 7(6), 1999, pp. 729–740

[9] J. J. E. Slotine, “Sliding controller design for nonlinear systems”, Int. J. Control, 40, 1984, pp. 421–434

[10] J.-M. Yang and J.-H. Kim, “Sliding Mode Control for Trajectory Tracking of Nonholonomic Wheeled Mobile Robots”, Proc. 1998 IEEE International Conference on Robotics and Automation

[11] T. I. Fossen, K. Y. Pettersen, and R. Galeazzi, “Line-of-Sight Path Following for Dubins Paths With Adaptive Sideslip Compensation of Drift Forces”, IEEE Trans. on Control Systems Technology, Vol. 23, No. 2, March 2015

[12] M. S. Wiig, W. Caharija, T. R. Krogstad, and K. Y. Pettersen, “Integral Line-of-Sight Guidance of Underwater Vehicles Without Neutral Buoyancy”, Elsevier, IFAC-Papers Online, Vol. 49, No. 23, 2016, pp. 590–597

[13] L. E. Dubins, “On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents”, American Journal of Mathematics, Vol. 79, No. 3, 1957, 497-516

[14] L. Ojeda, D. Cruz, G. Reina, and J. Borenstein, “Current-Based Slippage Detection and Odometry Correction for Mobile, Robots and Planetary Rovers”, IEEE Trans. on Robotics, Vol. 22, No. 2, April 2006

[15] C. C. Ward and K. Iagnemma, “Model-Based Wheel Slip Detection for Outdoor Mobile Robots”, IEEE Intern. Conf. on Robotics and Automation Rome, Italy, April 10-14 2007

[16] H. Deng, G. Xin, G. Zhong, and M. Mistry, “Gait and trajectory rolling planning and control of hexapod robots for disaster rescue applications”, Robotics and Autonomous Systems, 2017, pp. 13–24

[17] R. Wallace, A. Stentz, C. E. Thorpe, H. Moravec, W. Whittaker, and T. Kanade, “First results in robot road-following”, Proc. of the 9th Intern. Joint Confe. on Artificial Intelligence (IJCAI ’85), Vol. 1 , Los Angeles, Calif, USA, Aug. 1985, pp. 66–71

[18] S. Choi, J. Y. Lee, and W. Yu, “Comparison between Position and Posture Recovery in Path Following”, 6th Intern. Conf. on Ubiquitous Robots and Ambient Intelligence (URAI), 2009

[19] T. M. Howard, R. A. Knepper, and A. Kelly, “Constrained Optimization Path Following of Wheeled Robots in Natural Terrain”, in O. Khatib, V. Kumar, and D. Rus (eds.) Experimental Robotics. Springer Tracts in Advanced Robotics, Vol 39. Springer, 2008

[20] G. C. Haynes and A. A. Rizzi, “Gait Regulation and Feedback on a Robotic Climbing Hexapod”, Robotics: Science and Systems, August 16-19, 2006, University of Pennsylvania, Philadelphia, USA

[21] G. C. Haynes, “Gait Regulation Control Techniques for Robust Legged Locomotion”, PhD Thesis CMU-RI-TR-08-19, CMU, Pittsburgh, May 2008

[22] J. Roth, “A Viterbi-like Approach for Trajectory Planning with Different Maneuvers”, 15th International Conference on Intelligent Autonomous Systems (IAS-15), June 11-15, 2018, Baden-Baden, Germany, pp. 3–14

[23] J. Roth, “Robots in the Classroom – Mobile Robot Projects in Academic Teaching”, Innovations for Community Services: 19th International Conference, I4CS 2019, Wolfsburg, Germany, June 24-26, 2019, 39-55

[24] J. Roth, “Systematic and Complete Enumeration of Statically Stable Multipod Gaits”, Vol. 12, No. 4, 2018, 42-50, DOI: 10.14313/JAMRIS_4-2018/24

[25] J. Roth, “A Novel Development Paradigm for Event-based Applications”, Intern, Conf. on Innovations for Community Services (I4CS), Nuremberg, Germany, July 8-10, 2015, IEEE xplore, 69-75

[26] J. W. Gibbs, “Elements of Vector Analysis”, New Haven, 1884

[27] A. Cayley, “The collected mathematical papers of Arthur Cayley”, I (1841-1853), Cambridge University Press, 332-336, 1889

2019, © Copyright by authors, Published under agreement with IARIA - www.iaria.org