Regulated Walking for Multipod Robots

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

Abstract— **Following a computed path is a fundamental task for robot motion. The goal is to compensate error effects, such as slippage and create a 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 an approach to regulate paths for multipods. It is based on explicit slippage detection and re-uses a trajectory planning component to compute regulation trajectories. The approach is implemented and tested on the Bugbot hexapod robot.**

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

I. INTRODUCTION

Multipod robots are robots with multiple legs, inspired by insects. Their main advantage is that they can walk over rough terrain or go over small obstacles. In contrast to bipedal robots, multipods have a stable footprint, thus the control does not have to consider dynamics or balancing issues.

In the context of task execution, a robot must follow a planned path to a target. Walking usually is slower than driving, but we have to face new problems: walking is much more imprecise. In addition, we have the concept of *gaits* – we first have to plan sequences of leg movements (i.e., servo commands) to move the whole robot. Even though we may consider multipods as holonomic vehicles, we get non-holonomic constraints as a result of gait execution capabilities. Certain sensor configurations may cause further restrictions. For example, a sensor that prevents falling downstairs ideally points in front, thus we may prefer forward walking.

Our approach is 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 compute regulation trajectories to a pose ahead on the formerly planned path.

To compute regulation trajectories we use the same approach as for the global path planning. However, as the regulation trajectories are much shorter, the planning is much faster.

In section 2, we present related work. In section 3, we present our regulation approach. Experiments are presented in section 4. Section 5 concludes the paper.

II. RELATED WORK

Research on path following and trajectory tracking has a long tradition in control theory [4][11][18]. The basic goal is to provide a formal representation of the so-called *control law* [1]. Both, the vehicle and trajectories are strongly formalized in order to derive quality statements, in particular regarding the controller's stability [2].

Model Predictive Control (*MPC*) [9][12] 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 [15][20]. Once the sliding surface is reached, sliding mode control keeps the states on the close neighborhood 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 [6] or underwater vehicles [19]. The approaches differ how to reach the point ahead. Examples are arcs, straight lines or Dubins paths.

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 [10], effects on motors are measured for this. [16] 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 behavior can formalized easily. Multipods are only rarely taken into account. [5] 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 to reach this position (e.g., an arc). Early work about Pure Pursuit is [17]. The basic version only tries to reach a position ahead without considering the robot's orientation [3]. Improvements dynamically adapt the look-ahead distance [8].

 ISBN: 978-1-61208-706-1

III. THE REGULATION APPROACH

In contrast to easy to formalize wheeled robots, we have to face issues that make it difficult to apply a traditional approach based on control theory. First, walking is in general more error-prone than driving. As a result, we cannot execute regulation trajectories as precisely as expected. Second, as multipods may be different in the capabilities to execute certain gaits, we want to consider the set of possible walking commands as black box. As a consequence, it is not useful to integrate kinematic properties into the model. Finally, our regulation mechanism should directly consider obstacles and an arbitrary cost function, again given as black boxes. A certain regulation trajectory may not only be based on regulation parameters, but also on the environment.

Our approach was inspired by the pure pursuit idea. We project the current position ahead to the target and try to get there. We extend the basic idea in two ways:

- We try to reach a planned *configuration*, i.e., position *and* orientation.
- We are not restricted to a certain primitive trajectory (e.g., arc) to reach the configuration ahead, but execute a full trajectory planning step.

We re-use the trajectory planning both to compute a full plan to the final target, as well as for the regulation approach. As a benefit, both components produce output that can directly be used as walking command by the motion system. In particular, the motion capabilities are modeled in one place in the system. But we have to face two issues:

- 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 [13].
- As we do not explicitly model a control law, we have to consider sources of disturbance, foremost the slippage effect.

The regulation is embedded into a data flow as presented in Figure 1. We have the following major components:

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. It computes a line string of minimal costs that in particular avoids obstacles. This component is useful to segment the overall path planning task.

Trajectory Planning computes a walkable sequence of trajectories and considers non-holonomic constraints.

Trajectory Regulation permanently tries to hold the planned trajectories, even if the position drifts off.

Simultaneous Localization and Mapping (*SLAM*) constantly observes the environment and computes the most probable own location and location of obstacles by motion feedback and sensors (e.g., Lidar or camera). 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, expected energy consumption or the amount of turn-in-place operations. Also, the distance to obstacles could be considered, if, e.g., we want the robot to keep a safety distance where possible.

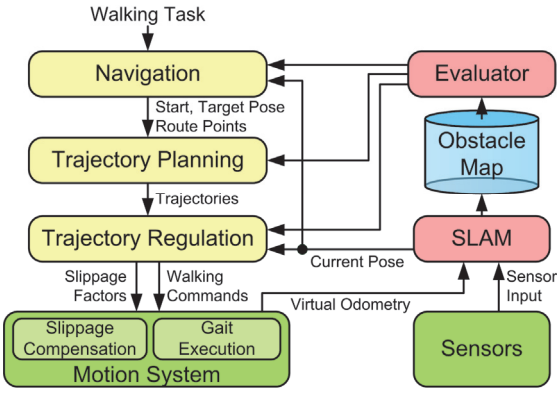

Figure 1. Data flow to execute walking tasks

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

In this paper, we assume *Navigation*, *Trajectory Planning* and *SLAM* already exist. We may use an approach as described in [13] for this. We here focus on *Trajectory Regulation*.

A. Gaits

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 legs on the ground (*stance phase*) or swing in moving direction (*swing phase*). An important observation: we can deal with trajectory and time sequence independently. This means, the respective trajectory shape is *not* influenced by the sequence pattern.

Figure 2 shows the two phases for a specific leg. Let (f_{xi},f_{yi}) denote neutral foot position. It marks the center of a stance movement from $(f_{xi},f_{yi})+(s_{xi},s_{yi})$ to $(f_{xi},f_{yi})-(s_{xi},s_{yi})$ 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. The gaits define the cooperation of legs in the respective phases. Many gaits are known, e.g., *Tripod*, *Wave*, *Ripple* that differ in stability and propulsion [14]. We assume gait execution and the choice for a certain gait is encapsulated in the Motion System component.

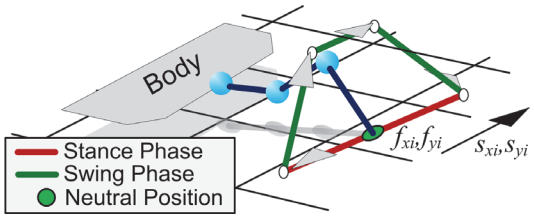

Figure 2. Gait movement phases

B. 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. This leads to the idea of *virtual odometry*: We transform walking to corresponding wheeled movement. We could think of roller-skates attached to the multipod's legs while the legs remain in neutral position. To fully describe gait movement with the help of virtual odometry we need

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

Note that t_{st} depends on the respective gait (e.g., Tripod or Wave). For a small time Δt, a foot of leg i moves along the vector

$$2 \cdot \frac{\Delta t}{t_{st}} \begin{pmatrix} s_{xi} \\ s_{yi} \end{pmatrix} \tag{1}$$

in local robot coordinates. We derivate over time. Not necessarily all legs move along the same vector, thus the robot's pose changes over time by

$$\begin{pmatrix} \Delta_x \\ \Delta_y \\ \Delta_\theta \end{pmatrix} = \Psi\left(\begin{pmatrix} f_{xi} \\ f_{yi} \end{pmatrix}, \begin{pmatrix} f_{xi} \\ f_{yi} \end{pmatrix} + \frac{2}{t_{st}} \begin{pmatrix} s_{xi} \\ s_{yi} \end{pmatrix} \right) \tag{2}$$

Here, Ψ denotes a function that computes a roto-translation which maps all positions of the first list to positions of a second list, meanwhile minimizing the mean square error. There exists an approach based on Gibbs vectors [7] to set up and solve a linear equation system for Ψ.

For $\Delta_\theta \neq 0$, the robot walks along an arc with center

$$\begin{pmatrix} c_x \\ c_y \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & -1/\tan(\Delta_\theta/2) \\ 1/\tan(\Delta_\theta/2) & 1 \end{pmatrix} \begin{pmatrix} \Delta_x \\ \Delta_y \end{pmatrix} \tag{3}$$

and curve angle Δ_θ. For $\Delta_\theta = 0$, the robot walks along a straight line with direction (Δ_x, Δ_y). Considering both cases, the moving distance for a leg i over time Δt is

$$\ell_i(\Delta t) = \Delta t \cdot \begin{cases} |\Delta_\theta| \sqrt{(f_{xi} - c_x)^2 + (f_{yi} - c_y)^2} & \text{if } \Delta_\theta \neq 0 \\ \sqrt{\Delta_x{}^2 + \Delta_y{}^2} & \text{if } \Delta_\theta = 0 \end{cases} \tag{4}$$

Note that a turn in place is considered as arc movement with arc center in the robot's center.

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

C. Slippage Detection and Compensation

In contrast to the expected walking distances, we now compute the real distances. We assume sensors (e.g., Lidar) and respective SLAM mechanisms that permanently measure the robot's real position. We consider these mechanisms as black box, but expect they detect the real pose change (Δ_x', Δ_y', Δ_α') after walking a time Δt. We assume 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 expect only small errors.

For given (Δ_x', Δ_y', Δ_α') we can apply formulas (3) and (4) to get the *real* walking distances $\ell_i'(\Delta t)$. We define

$$S_i = \frac{\ell_i(\Delta t)}{\ell_i'(\Delta t)} \tag{5}$$

as the *leg-specific slippage factors* and

$$S = \frac{1}{L} \sum_i S_i \tag{6}$$

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, the robot walks half as far as expected when executing a certain trajectory. The S_i describe slippage *per leg* and could indicate malfunctions in 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 consider leg-specific slippage.

The assumption is: what we measured recently is a good estimation for the nearer future. E.g., if we walk on a slippery floor, we can consider the respective slippage factor to execute 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, e.g.:

- 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 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 for a specific leg we get $S_i=2$ (i.e., leg produces only half of the expected propulsion), we could multiply (s_{xi}, s_{yi}) by 2 to compensate this effect. This however is not always possible because the stance vector length is limited – either by the me-

chanics, or because neighbor legs should not collide during walking. We usually are only able to shorten the stance vectors. Our approach is thus to compute

$$S_{\max} = \max(S_i)\ ,\ \widetilde{S}_i = S_i / S_{\max} \qquad (7)$$

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 apply a leg-specific compensation. The latter is only reasonable, if we actually expect a leg-specific slippage that may be result of malfunctioned legs.

D. Regulation Trajectories

When walking on a planned trajectory, the real position differs from the planned position. This is because the execution of walking commands is never absolutely accurate due to slippage and minimal mechanical impreciseness. The task is to compensate the differences during walking and to meet the planned trajectories. This problem 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 single values or by a primitive walking operation. Due to non-holonomic constraints a *sequence* of trajectories usually is required.

Even though a certain multipod may support holonomic locomotion, not all trajectories may be suitable to get to the planned trajectory. E.g., we may have a cost function that considers a safety distance to obstacles, or we expect an ultrasonic sensor to always point in walking direction. To compute a regulation trajectory we thus need an additional planning step that (similar to the navigation and trajectory planning) minimizes a certain cost function.

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*.

$$(T_i) = TP\big((s_x, s_y, s_\theta), (t_x, t_y, t_\theta)\big) \qquad (8)$$

The result (T_i) is a sequence of primitive trajectories, i.e., expressible by simple walking commands (e.g., arc or straight). *TP* takes into account the route points from the navigation, the cost function and the obstacle map. We can consider *TP* as black box, but solutions are widely known. We, e.g., may use *Rapidly-exploring Random Trees* (RRT). Our implementation is based on sets of *maneuvers* that are combined using a Viterbi approach [13].

We further need to identify an *expected* pose *e* of a current pose *c*.

$$(e_x, e_y, e_\theta) = E\big((T_i), (c_x, c_y, c_\theta)\big) \qquad (9)$$

Expected means: the intended pose for a pose that is *not* on the planned path. If the multipod remains on the trajectory sequence, *c* and *e* are equal. But if the current pose leaves the planned trajectory, 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 since the start of walking on (T_i) and identify the pose that has the respective distance from the start. A stateless implementation only identifies the nearest trajectory point based geometric distance computation.

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

$$(a_x, a_y, a_\theta) = A\big((T_i), e, d\big) \qquad (10)$$

Here, *d* describes, how much the current expected pose is projected ahead in target direction. Figure 3 illustrates the idea.

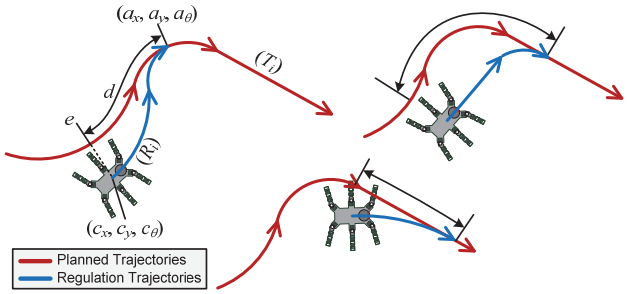

Figure 3. Idea of regulation-ahead

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

$$(R_i) = TP\big((c_x, c_y, c_\theta), A\big((T_i), E\big((T_i), (c_x, c_y, c_\theta)\big), d\big)\big) \qquad (11)$$

The major benefit: we do not have to introduce a new approach 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 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. In reality, we can compute (R_i) periodically, e.g., twice a second, without noticeable delay.

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

Both lead to higher costs – either because the path gets significantly longer or because the robot walks on positions with higher costs, besides the planned trajectory. Figure 4 illustrates these effects. In this example, we planned a linear

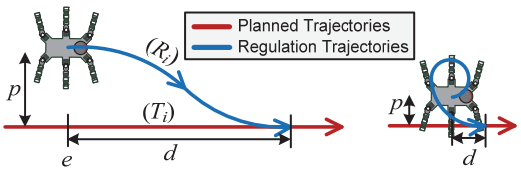

Figure 4. Effects of large and small d, p

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, the regulation trajectory may be a spiral that starts in opposite direction, because arcs cannot be unlimited tight. 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

$$q = \frac{|(R_i)|}{d} \qquad (12)$$

as the *stretch factor*. It specifies how much longer the regulation path is compared to the planned path. Figure 5 shows typical curves of q. We used the trajectory planning of the Bugbot [13] for this chart. We planned only forward walking and penalized a turn in place with high costs.

Due to the effect presented in Figure 4 (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 of q. As a result, d=40 cm is a good choice for our setting.

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.

IV. EXPERIMENTS

We implemented our trajectory regulation approach on

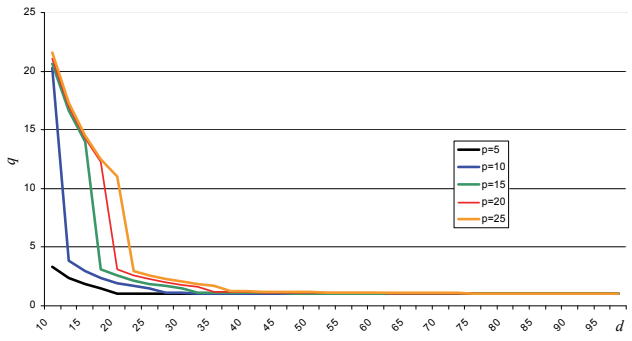

Figure 5. Typical stretch factors (d, p in cm)

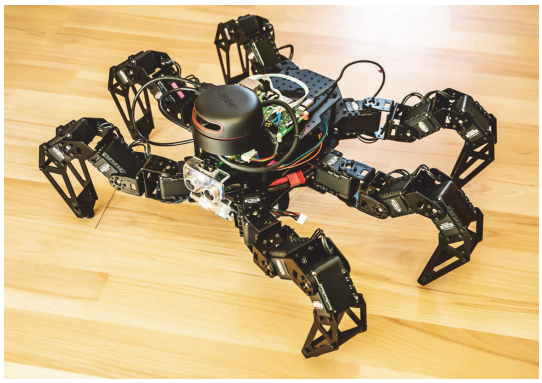

Figure 6. The Bugbot

the *Bugbot* (Figure 6). Bugbot is an 18-DOF hexapod based on the Trossen PhantomX Mark III platform. We added a Lidar device and further sensors for collision detection. A Raspberry PI 3B is used for main computations, e.g., route and trajectory planning, SLAM and trajectory regulation. As described in [13], the trajectory planning function *TP* in equation (8) can efficiently compute a first trajectory in less than 1 ms, even on the Raspberry PI. *TP* already considers obstacle avoidance and can integrate arbitrary cost functions.

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 also a problem to adjust leg-specific slippage in reality in a fine-grained manner. We thus created a simulation environment that simulates the Bugbot on hardware- and physical level. A physical simulation component is able to compute gravitation, 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 7 shows an example to illustrate the effects of

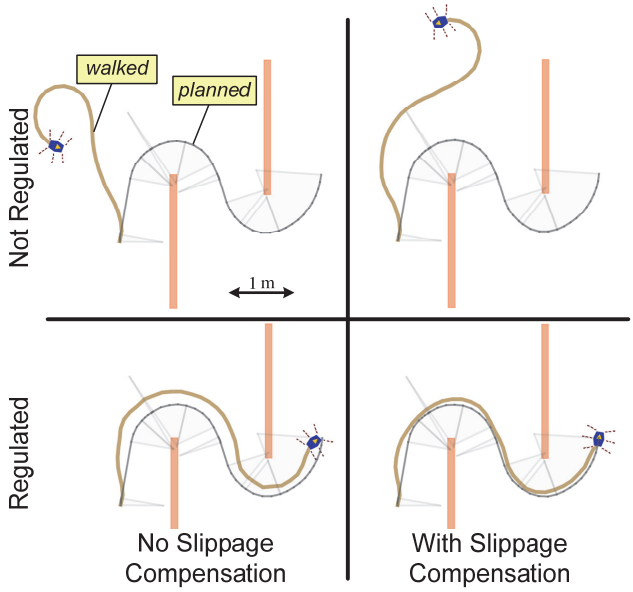

Figure 7. Simple walking scenario

 ISBN: 978-1-61208-706-1

slippage compensation and regulation. We artificially assigned a leg-specific slippage of 2.0 for the three left legs. This means, without any compensation, the robot walks a left arc when planned to walk right (Figure 7 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 7 top, right).

Figure 7 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 8 shows a more complex example. Here we assigned again a leg-specific slippage of 2.0 and in addition a general slippage of 2.0. This represents a very difficult scenario. In the execution, both mechanisms were applied. We can see a great congruence of planned and walked path.

V. CONCLUSIONS

This paper presented an approach 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.

The look-ahead distance currently is based on the developer's experience. Whereas small distances may lead to instabilities, larger distances only increase the time to meet the planned path and 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.

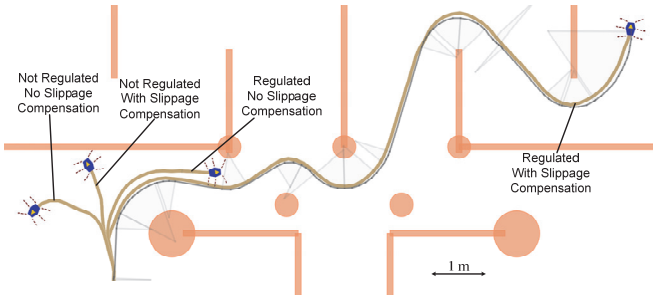

Figure 8. Complex walking scenario

REFERENCES

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

[2] 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

[3] 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

[4] 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

[5] 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

[6] 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

[7] J. W. Gibbs, "Elements of Vector Analysis", New Haven, 1884

[8] 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

[9] 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

[10] 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

[11] 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

[12] 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

[13] 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

[14] J. Roth, "Systematic and Complete Enumeration of Statically Stable Multipod Gaits", to be published

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

[16] 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

[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] 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

[19] 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

[20] 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

 ISBN: 978-1-61208-706-1