International Journal of Humanoid Robotics
PREPRINT

Vision-Based 3D Modeling of Unknown Dynamic Environments for Real-Time Humanoid Navigation

Daniel Wahrmann,* Arne-Christoph Hildebrandt, Tamas Bates, Robert Wittmann, Felix Sygulla, Philipp Seiwald and Daniel Rixen

Chair of Applied Mechanics, Technical University of Munich
Boltzmannstr. 15, 85478 Garching, Germany
{*daniel.wahrmann, arne.hildebrandt, tamas.bates, robert.wittmann, felix.sygulla, philipp.seiwald, rixen*}*@tum.de*

Received Day Month Year
Revised Day Month Year
Accepted Day Month Year

In order to achieve real autonomy, robots have to be able to navigate in completely unknown environments. Due to the complexity of computer vision algorithms, almost every approach for robotic navigation is either based on previous knowledge of the environment, such as markers or as resulting from learning methods, or makes strong simplifying assumptions about it (height-map representations, static scenarios). While showing impressive success in certain applications, these approaches limit the potential of legged robots to achieve the amazing flexibility of humans in all kind of terrains. In this work, we present a strategy for full 3D vision processing that does not assume previous knowledge about the surroundings and is able to handle changing, dynamic environments. These are modeled using simple geometries, which are processed in real-time by the motion planner of our biped robot *Lola* for avoiding moving obstacles and walking over unexpected platforms. In order to allow for a more intuitive development of such systems in the future, we present tools for visualization including two mixed reality applications using both an external camera and Microsoft's HoloLens. We validate our system in simulations and experiments with our full-size humanoid robot *Lola* and publish our framework open source for the benefit of the community.

Keywords: Environment modeling; 3D Image Processing; Autonomous Robotics; Legged Robots; Navigation.

1. Introduction

Two main components of an autonomous navigation system are the perception and motion planning modules. In order for a robot to navigate in a real environment, the perception module needs to acquire an abstract representation (or "model") of it, which is then used by the planning module to find a feasible path while avoiding collisions [1]. The modeling strategy restricts the kind of environments the robot can

*Corresponding author.

navigate through and determines the capabilities of the motion planner [2].

In what is usually considered to be the first example of autonomous robotic navigation, 2D maps were used to represent the environment and the A* algorithm was presented to navigate an unknown scenario avoiding static walls [1]. Later works included height information in a 2D grid to generate "height-maps" and navigate in uncluttered, horizontal terrain [3]. Using this approach, wheeled robots have been able to achieve impressive feats [4].

Since the Fukushima nuclear disaster, however, the mobility limitations of these robotic systems became manifest and there was an unprecedented interest in anthropomorphic robots. Legged robots are better suited for navigating through cluttered environments and could potentially be used in such hazardous areas. A few years ago, the DARPA Robotics Challenge (*DRC*) [5] was organized, in which several teams with different robots tried to solve several tasks inspired by those scenarios. Still, the completion of the tasks relied strongly on teleoperation. To the authors' knowledge, none of the participating teams performed completely autonomous navigation.

The complexity of this problem lies, on the one hand, in the particular dynamics of biped robots. Besides being naturally underactuated, their high number of degrees of freedom makes it computationally challenging to perform real-time motion planning and control. In order to solve this, reduced models are used for approximating the robot's dynamics [6, 7, 8, 9, 10].

On the other hand, a detailed model of the environment is not suited for real-time navigation due to its computational costs, both on the computer vision and the motion planning side [11, 12]. Therefore, real-time humanoid navigation has only been achieved using simplified environment models on relatively uncomplicated, static scenarios. In the first application of height-maps to biped navigation, static scenes could be traversed in real-time due to its simplified representation [13]. An extension to this representation, consisting of the segmentation and classification of these 2.5D maps, is a very popular approach in humanoid research as it permits a real-time solution for the motion planning problem [14, 15, 16, 17, 18]. However, it still limits the complexity of applicable scenarios. One possible example of a more detailed representation are the *Octomaps* [19], which are fully 3D and can be used for more complex environments but assume them to be static. However, collision avoidance algorithms take significantly longer time with these representations, even for standard robotic manipulators [12] and cannot currently be applied in real-time (i.e. during walking).

In this work we try to bridge this gap by introducing a full 3D environment representation which can represent complex, dynamic scenarios and can be processed in real-time by the motion planner. Together with an efficient vision system, we present our strategy for biped navigation in 3D environments. We validate it with our full-size humanoid robot *Lola*, which can autonomously avoid dynamic objects, traverse unknown obstacles and walk over unexpected platforms using a standard

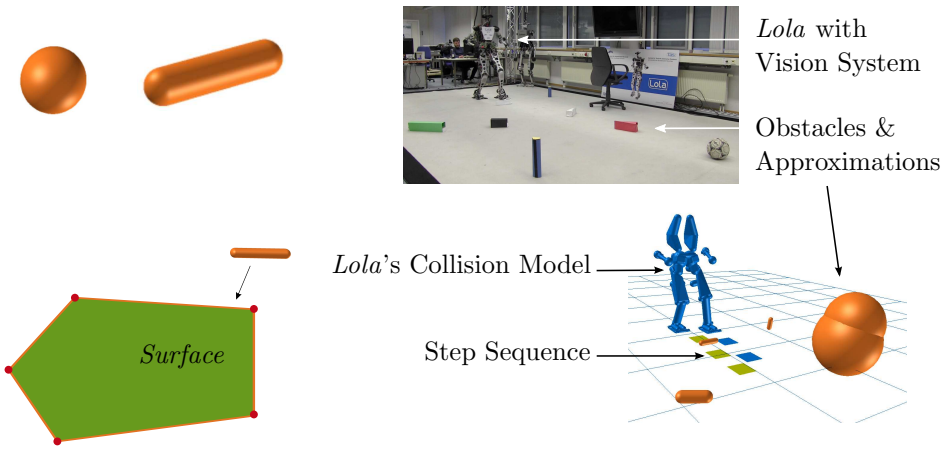


Fig. 1. Environment modeling. Left: obstacles are represented as point- and line-SSVs; surfaces as polygons with line-SSVs at the edges. Right: environment representation enables real-time 3D planning and collision avoidance.

RGB-D sensor. To the authors' knowledge, this is the fastest demonstration of autonomous walking in uneven terrain so far, as well as the first to handle unknown, dynamic environments in real-time.

In our approach, we use swept-sphere-volumes (SSVs) for efficient collision checking [20], which consist of extended volumes (with constant radius) of points, segments or triangles. As shown in previous works [18, 21], this representation of objects allows for fast planning and obstacle avoidance in cluttered environments (see Fig. 1). Collisions -both internal and external- are checked in 3D in real-time [22]. In a previous version of our vision system [23], we showed the approximation of obstacles with these geometries for real-time collision avoidance; our robot *Lola* was able to avoid previously unknown obstacles by performing complex motions such as stepping over or walking sideways [21, 24].

We extend our system and give it a general structure that handles more complex scenarios. Our previous vision system [23] could only detect obstacles on the ground which were considered static. In this work, instead of assuming a flat ground we detect all kinds of walkable surfaces. Additionally, we improve the detection and tracking of moving obstacles by explicitly estimating their velocity with a Kalman filter combined with a more robust clustering algorithm. Surfaces on which the robot can walk (e.g. platforms, ramps, stairs) are represented using polygons; obstacles which the robot needs to avoid are represented with one or more SSVs. In contrast to other works, this full 3D representation does not require previous information and is able to react to dynamic environments during walking. Additionally, we only rely on on-board sensing and processing. We combine new and state-of-the-art algorithms to make our system as efficient and robust as possible and design it in a

modular fashion so that it can be easily integrated as a mix of vision processing and visualization libraries into other systems. These libraries include tools for local visualization and mixed reality with either an external camera or Microsoft's HoloLens[25]. Our code is available open source[a]. As it processes 3D point clouds directly and does not depend on identifiers such as form, color or texture, it is directly compatible with other robots and sensors. Furthermore, it can be combined with classic computer vision algorithms for scene or object recognition.
This paper is organized as follows. In Section 2, we give an overview of other representation models, focusing on applications to biped locomotion. Our vision system is explained and evaluated in detail in Section 3. In Section 4, we present the integration with the walking controller and our strategy for real-time motion planning as well as external mixed reality applications. Experimental results are shown in Section 5, while the system's capabilities and limitations, as well as future work are discussed in Section 6.

2. Related Work

In this paper, we deal with the environment modeling of completely unknown scenarios for humanoid navigation. This is in contrast to the object recognition methods (based on e.g. geometric descriptors or learning algorithms), which are an active area of research in the field of computer vision [12]. As they cannot identify all possible situations, we consider that a modeling strategy for unknown environments will always continue to be relevant; that is the focus of this section.
As mentioned before, 2.5D maps were used in 2003 to represent a sufficiently structured terrain [13]. The authors could generate collision-free trajectories for their *H7* robot. Other early approaches include a 2D classification of the environment [26] in which the biped robot *Johnnie* could navigate over obstacles and surfaces which were previously known. In order to deal with more complex scenarios, a 3D occupancy grid was added to the 2.5D map in order to recognize obstacles [14, 15], but relied on textured surfaces to climb stairs with the *QRIO* robot. In one of the few works dealing with dynamic environments [27], the *Asimo* robot managed to safely navigate between 2D, moving obstacles; these were previously known and moved only with constant speed in the lateral direction. An impressive degree of autonomy was shown in [16, 17] with the *HRP-2* robot. Out of a structured environment, it could extract planes and label other regions as obstacles for walking over them and onto platforms. It relied on a pivoting laser scanner for which static scenarios were assumed. In 2010, a 2D-based occupancy approach was used for quickly generating collision-free trajectories in non-static environments [28], but it didn't consider complex obstacle avoidance motions such as stepping over. A different representation of the environment using *octrees* was presented later [29, 30]. The authors achieved collision-free navigation with the *Nao* robot but used texture and color for

[a]The different tools can be found in our repository: `https://github.com/am-lola`

classification. Motivated by the *DRC*, some authors [31, 32] presented autonomous navigation results of the Atlas robot; they do not consider dynamic environments. In one work, a height-map of relatively simple scenarios was generated while the robot was standing [31] and simple collision checking was done via *Octomaps*. In another one [32], accurate and dense 3D maps of the environment were used to extract walkable surfaces (but not obstacles) from static terrains. The robot is able to plan future footsteps during walking.

The works mentioned above present several limitations. In most cases, the scenarios considered are relatively simple, such that they can be represented using height-maps. Additionally, collision checking is based only on footstep locations and heuristics. Most importantly, either the vision system, motion planner or both take too long to be applied to unknown dynamic environments in real-time. Our work overcomes all of these issues with an efficient environment representation based on direct point cloud processing. On the vision side, it is fast enough to be generated online while the robot is moving[b]. On the planning side, it enables reactive footstep planning and real-time 3D collision avoidance.

Another related research area worth mentioning in this section is the autonomous vehicles industry, where dynamic scenarios are taken into account for real-time navigation. However, conditions differ greatly from the ones considered here. On the one hand, the environment is not entirely unknown: streets can be previously scanned and localized with positioning methods; usual components of the scene, such as other vehicles, bicycles and pedestrians can be recognized using learning methods. On the other hand, objects can be modeled with simple bounding boxes and avoided using 2D planning, without considering the complex motions of humanoid robots [33, 34].

3. Vision System

3.1. *Structure*

Our concept for a vision system for autonomous robots can be seen in Fig. 2. Object recognition techniques, which have lately become more robust with the use of deep learning methods [35], enable all kinds of applications such as manipulation and interaction. Still, there will always be sections of the environment (or even complete scenarios) that will not be recognized by the most comprehensive database; these can be modeled in parallel using basic geometries, which the robot's motion planner can handle. Furthermore, for obstacle avoidance a rough approximation of an object is not only sufficient but desirable for computational reasons. In this section, we will present our strategy for environment modeling which is available open

[b]Our sensor generates point clouds with 30 Hz and our vision system can process them at 10 Hz or faster, as will be shown later.

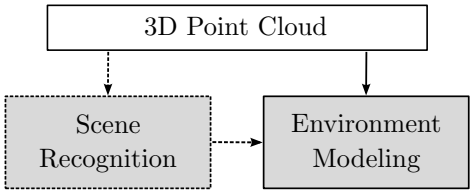


Fig. 2. Concept for a Vision System for Autonomous Navigation: sections of the environment which are not recognized by learning methods (*Scene Recognition*) are modeled by an approximation strategy (*Environment Modeling*) with a set of pre-defined geometries. This work deals with the latter.

source[c] together with our visualization library[d]. In the future, we plan to combine it with object recognition methods as shown in Fig. 2 to extend the manipulation capabilities of our humanoid robot or determine objects to lean on while performing autonomous navigation.

In the last few years, 3D sensors became widely available [36], which in turn motivated the research on efficient 3D computer vision algorithms [37]. In order to make our system as general as possible, we base our vision algorithm on direct 3D point cloud processing; we make use of the *Point Cloud Library* (*PCL*) [37] and a completely modular design to make it compatible with other kinds of robots and sensors. For our experiments, we use a standard RGB-D sensor that serves as a robustness assessment: if the system is robust against this sensor's high levels of noise it will perform even better with other, more accurate sensors. Additionally, the sensor's noise can be interpreted as simulating the result of a more accurate sensor over more irregular terrain, testing the robustness of the system against real-world scenarios.

In Fig. 3, the structure of our *Environment Modeling* system can be seen. A parallel structure allows to speed up the whole system and make use of multi-threading computation: each process runs with its own cycle time and can access the latest scene information regardless of the other parallel processes. In order to do that, classification criteria have to be defined. In this work, we classify the environment according to whether it is walkable or not. For other robotic systems, different criteria could be applied, making use of the same approximation algorithms.

As shown in Fig. 3, one process (*Plane Segmentation*) finds points in the scene (planes) which can potentially be walked over by the robot. Using the latest plane coefficients obtained, both the *Surface* and *Obstacle Approximation* filter the new incoming point cloud (the *Surface Approximation* keeps points belonging to the planes and the *Obstacle Approximation* discards them) for generating the corresponding basic geometries. Note that, even though the coefficients are taken from previous frames (less than 50 ms old, see Section 4), they should still be valid in our

[c] `https://github.com/am-lola/lepp3`
[d] `https://github.com/am-lola/ARVisualizer`

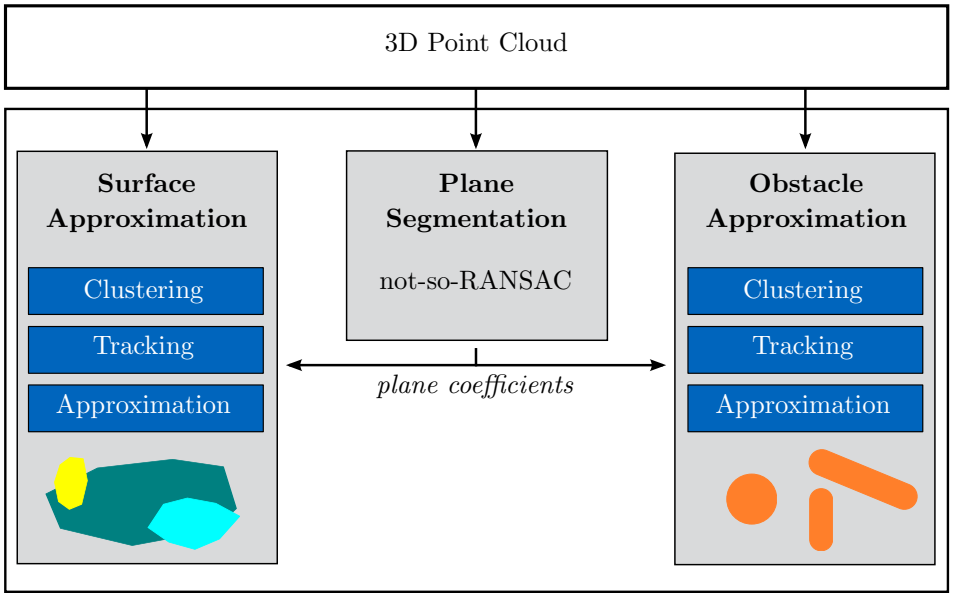


Fig. 3. Environment Modeling. The *Plane Segmentation* process classifies the incoming point cloud into walkable surfaces (for the *Surface Approximation*) and collision objects (for the *Obstacle Approximation*).

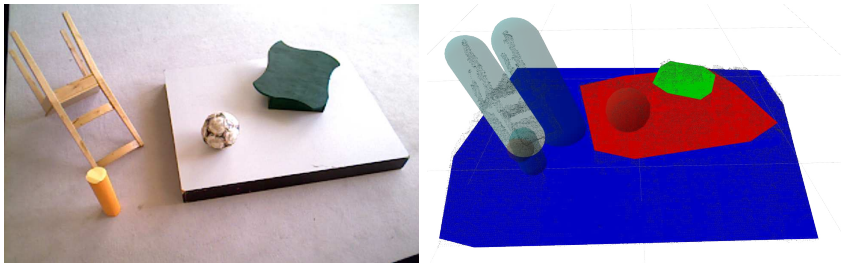


Fig. 4. Result of the vision system. Left: rgb image of an example scene (for reference). Right: surfaces (including the ground) are modeled with polygons and obstacles with SSVs.

application scenarios. Fast moving surfaces (e.g. escalators) are therefore treated as obstacles because their points do not correspond with earlier plane parameters (which is fine, as they cannot be handled by our planner yet). In Fig. 4 the approximation of an example scene using polygons and SSVs can be seen.
Afterwards, both processes follow a similar strategy for the geometric approximation which consists of the following steps:

- *clustering*, where points are grouped into separate "clusters",

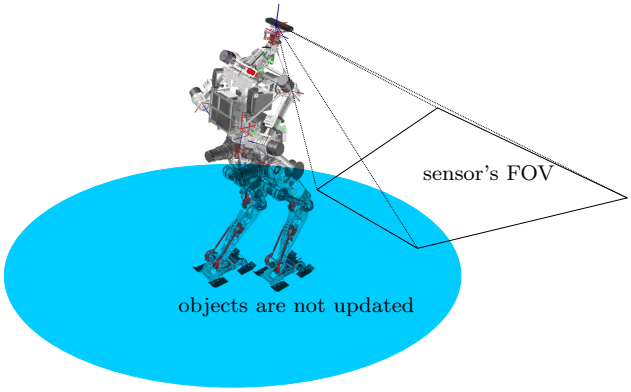


Fig. 5. The sensor's field of view is limited. Objects that enter the skyblue region stop being updated but are not removed. Objects outside both the FOV and skyblue region are removed.

- *tracking*, where each surface or obstacle is tracked and filtered across frames and
- *approximation*, where each cluster of points is assigned one or more approximating geometries.

In the *tracking* step the data is additionally checked for consistency. As explained in our previous paper [23], sensor noise can cause false detection of non-existing objects. Therefore, surfaces or obstacles need to appear in several consecutive frames before they are considered *real* and are sent to the planner. Moreover, as the sensor's field of view (FOV) does not include the section of the ground nearest the robot, objects stop being updated once they get close to the robot and before they leave the FOV (see Fig. 5).
In order to maintain consistency during *tracking* regardless of the robot's motion, the point cloud is transformed to a fixed coordinate system using the robot's odometry. Even though a *Simultaneous Localization and Mapping* (*SLAM*) method could be applied to reduce odometry errors, these are sufficiently small for our application [23]. Moreover, as we consider only on-board sensing, measurements of the environment are continuously updated and correct with respect to the robot. In the following, we explain the different processes of Fig. 3 in more detail. As stated above, a real-time application is the main motivation behind our proposed system. Throughout our implementation, we chose strategies by prioritizing robustness and runtime over optimality. The following algorithms satisfy these criteria and are the result of extensive testing and development.

3.2. *Plane Segmentation*

Our segmentation strategy is based on RANSAC [38]. Compared to *Depth-map-based* and *Normal-based* methods, it proved to be faster, more robust and transferable

throughout different frames (key for our parallel approach). *PCL*'s *Sample Consensus Segmentation* [39] provides a RANSAC implementation which randomly selects 3 points of a point cloud to get a plane's coefficients $((a, b, c, d) \mid ax + by + cz + d = 0)$ and test them against the remaining points. If enough points belong to the plane (within a certain threshold), these are removed and the process starts over. As the algorithm finds different planes' sections separately, these are joined together and clustered in the end to obtain clean surfaces. We select only surfaces that are sufficiently large to fit the robot's foot and which are nearly horizontal (up to 20° slope in our configuration) and, thus, walkable. Additionally, we introduce the following two modifications:

i) Not-so-RANSAC. In order to speed up the process, old plane parameters are tested against the incoming point cloud before starting RANSAC, so that previously existing surfaces can be quickly identified. This can speed up the segmentation routine up to seven times.
ii) Classification. The standard segmentation process is not robust against intersecting surfaces with similar inclinations (such as the ground and a ramp); when joining sections of planes together, both surfaces may be joined into one single plane. Therefore, before this step, planes' sections are classified according to their inclination so that all different plane parameters are correctly identified and adjacent surfaces separately approximated. As will be seen in the following, this simple modification also helps speed up the surface approximation routine.

The final list of plane coefficients $\{a_i, b_i, c_i, d_i\}, 1 \leq i \leq n_{\text{planes}}$ is sent to both the *Surface* and the *Obstacle Approximation* process.

3.3. *Surface Approximation*

3.3.1. *Clustering*

After filtering surface-points with the plane coefficients, these must be clustered into separate surface objects. Even points with the same coefficients might belong to separate surfaces (e.g. two separated platforms with equal height). Standard clustering algorithms (e.g. *PCL*'s *Euclidean Clustering* [39]) are computationally expensive, partly because they are implemented for 3D point clouds. However, we can take advantage of our modified RANSAC implementation to reduce the clustering step to a 2D problem: as planes are already classified according to their inclination and position, only points belonging to the same plane need to be clustered further. Thus, points can be projected into their corresponding plane and clustered using local 2D coordinates.

For our application, clustering criteria depend on the robot's feet. A cluster can be defined such that the distance between neighboring points is considerably smaller than the foot size. For clustering a given plane, we use a simple grid discretization: we group points according to a grid with a relatively small unit length (5 cm in our implementation) and define connectivity based on neighboring occupied cells

(note that the grid is only used for clustering while the original points are passed along to further stages). We tested our implementation against different clustering algorithms using varied scenes with multiple separated platforms of equal height. Algorithms tested include: *PCL*'s 3D Euclidean Clustering [39], *OpenCV*'s 2D Euclidean Clustering [40] and a local implementation of *DBSCAN* [41]. The difference in clustering results was always less than 1% of the number of points while the difference in runtime was significant: the grid strategy (runtime of less than 2 ms in the most complicated scenarios) performed more than 50 times faster than every other algorithm.

3.3.2. *Approximation*

Surfaces are approximated by simple polygons for the reasons mentioned before. At present, concave or incomplete surfaces are not considered due to the way footstep locations are optimized for collision avoidance (Section 4); we plan to include them in the future. We start our approximation by projecting the clustered point clouds to the corresponding ideal plane and applying the popular *QuickHull* algorithm [42, 43] (chosen for its runtime and easy parallelization). It iteratively expands a polygon until it contains all points. The result is a group of polygons with varying number of vertices. In order to facilitate the integration with the motion planner these are reduced to a maximum number of vertices n_{vertices} ($n_{\text{vertices}} = 8$ in our experiments) by iteratively removing those vertices which subtract the smallest area from the polygon (see Algorithm 1).

Algorithm 1 Reduction of convex polygons

1: Polygon P with adjacent vertices $P[i]$
2: **repeat**
3: **for all** modified vertices j **do**
4: Update area $\triangle P[j]$ of the triangle $\langle P[j-1], P[j], P[j+1] \rangle$
5: **end for**
6: Remove vertex corresponding to $\triangle P[m] = \min(\triangle P[i])$
7: Update P
8: **until** Polygon contains desired number of vertices or less

3.3.3. *Tracking*

Due to sensor noise and a limited FOV, the approximation process results in different polygon approximations from frame to frame. We match surfaces between frames by comparing plane coefficients and positions. If a corresponding previous polygon is found, it is updated (at the *approximation* step) by averaging both polygons. For this purpose, we propose a geometric interpretation of the classic low-pass filter, or "geometric low-pass filter". The main problem when averaging two polygons is that there is no clear correspondence between both sets of parameters (vertices).

Methods which simplify filtering using the polygon's center or area often result in inaccurate approximations of the surface's margins, which are extremely relevant to our application. Instead of matching vertices from both polygons with one another we match each polygon vertex with the closest point (or projection) in the other polygon. Then we average the positions of each vertex and its corresponding projection with a factor α (this can be interpreted as a low-pass filter with inverted factors for both polygons for consistency, as seen in Fig. 6). The set of average points results in a polygon with double as many vertices as the original that can be reduced using Algorithm 1 (see Algorithm 2).

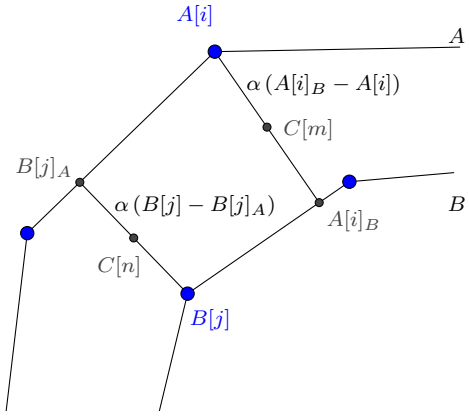


Fig. 6. "Geometric low-pass filter" for filtering successive polygon approximations of surfaces (see Algorithm 2). Note that the value of α determines the weight between both polygons. By shifting it closer to the old or new polygons, the algorithm becomes either more or less damping, respectively ($\alpha = 0.5$ in the experiments).

Algorithm 2 Geometric low-pass filter

1: Polygons A, B, with $n_{\text{vertices}_{A,B}}$ vertices $A[i], B[j]$
2: **for all** i, j **do**
3: Compute projection of $A[i]$ in B, $A[i]_B$
4: Compute projection of $B[j]$ in A, $B[j]_A$
5: **end for**
6: Compute pre-filtered polygon $C =$
$\{\alpha A[i] + (1-\alpha)\, A[i]_B\} \cup \{(1-\alpha)\, B[j] + \alpha B[j]_A\}\, \forall i, j$
with $0 \leq \alpha \leq 1$
7: Perform Algorithm 1 on C

3.4. *Obstacle Approximation*

An early version of our obstacle approximation strategy was published previously [23]. One of the main limitations was that due to the chosen *euclidean clustering* method, runtime would depend on the complexity of the scene (effectively restricting the amount and velocity of objects). Here we overcome that problem by using new clustering and tracking methods which are both more robust and dynamic, handling extremely complex scenarios faster than the sensor's frame rate (30 Hz). The SSV approximation method hasn't been changed and is therefore only briefly described here.

3.4.1. *Clustering*

In machine learning theory, data clustering is a fundamental problem of unsupervised learning. The objective is to determine correlation relationships between variables out of training data sets, without any additional information. A common approach [44] is the *Gaussian Mixture Model* (GMM), which consists of a set of probabilistic Gaussian distributions (or Gaussians) with the form:

$$p\left(x_i|\theta\right) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(x_i|\mu_k, \Sigma_k\right) \tag{1}$$

where K is the number of Gaussians and $\mathcal{N}\left(x_i|\mu_k, \Sigma_k\right)$ is the normal distribution of Gaussian k with mean μ_k, covariance Σ_k and mixing weight π_k. These weights satisfy $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$, to ensure a correct probability distribution $p\left(x_i|\theta\right)$ of the n points x_i (in this case, x_i has three coordinates) with the list of parameters θ, which consist of $\{\pi_k, \mu_k, \Sigma_k\}\ \forall k$ in this case [44].

These models are potentially well suited for our application as they don't need many assumptions on the data and their probabilistic nature make them robust against sensor noise. Additionally, the implicit principal axis decomposition analysis can be directly used for the SSV approximation, as explained below. However, classical implementations are based on static data sets and require an iteration procedure which converges to a local maximum likelihood estimate (MLE). Even though other authors have recently shown applications to object tracking [45], these are still too complex for real-time applications. For our *clustering* and *tracking* application, we propose the following adaptation of the *Expectation Maximization* (EM) algorithm which, combined with a Kalman Filter, can be successfully applied for online tracking of unknown, dynamic objects.

In order to improve runtime, only one EM-iteration is performed for each new point cloud. They classically consist of:

- Expectation (E) step: compute an auxiliary *responsibility* function for each point x_i and Gaussian k.

$$r_{ik} = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} \tag{2}$$

- Maximization (M) step: for each Gaussian k, compute new estimates for the weights π_k^{new} and parameters $\{\mu_k^{\text{new}}, \Sigma_k^{\text{new}}\}$.

$$\pi_k^{\text{new}} = \frac{1}{n}\sum_{i=1}^{n} r_{ik} = \frac{R_k}{n} \tag{3}$$

$$\mu_k^{\text{new}} = \frac{1}{R_k}\sum_{i=1}^{n} r_{ik} x_i \tag{4}$$

$$\Sigma_k^{\text{new}} = \left(\frac{1}{R_k}\sum_{n=1}^{n} r_{ik} x_i \left(x_i\right)^{\mathsf{T}}\right) - \mu_k^{\text{new}} \left(\mu_k^{\text{new}}\right)^{\mathsf{T}} \tag{5}$$

Even though an MLE is not guaranteed for dynamic data, our experiments in real environments show that it typically converges within a few frames and effectively tracks existing objects afterwards. Nevertheless, (4) and (5) do not take into account dynamic scenarios (or changing data). In order to deal with these scenarios, we propose a few modifications to the EM algorithm that take objects' dynamics directly into account and are explained in the following.

3.4.2. *Tracking*

It is interesting to note that the use of GMMs makes tracking much easier. Compared to the euclidean segmentation [23], where new objects need to be matched against old ones, the iterative nature the EM algorithm means that cluster identities and parameters are kept from frame to frame. However, data is supposed to be static, meaning that new values of μ_k^{new} do not take into account the object's velocities and are always "lagging behind" the motions of the objects. For this reason, we apply a Kalman filter to the values of μ_k^{new} which results in a more dynamic update procedure and better tracking[e]. We use a linear model with constant velocity and Gaussian noise in both position and velocity values and replace (4) with the new estimated value $\hat{\mu}_k^{\text{new}}$. This combination of the EM and Kalman algorithms[46] results in the effective tracking of multiple dynamic elements, as can be seen in Fig. 7. Furthermore, the estimate of the velocity can be passed along to the robot in order to directly consider objects' motions in the motion planning module.
For what concerns Σ, the estimation in (5) depends strongly on the present data and thus can vary from frame to frame. In our application, it means that separate dynamic objects are quickly joined once they are close, preventing correct tracking and velocity estimation. Therefore, we replace it with a *maximum a posteriori* estimation, which can be interpreted as a low-pass filter that keeps separate obstacles separated (see Fig. 8):

$$\hat{\Sigma}_k^{\text{new}} = \lambda\Sigma_k + (1-\lambda)\Sigma_k^{\text{new}} \text{ with } 0 \leq \lambda \leq 1 \tag{6}$$

[e]Even though the obstacles might be correctly approximated after some frames without the Kalman filter if the update rate is fast enough, it is still useful as a velocity estimator.

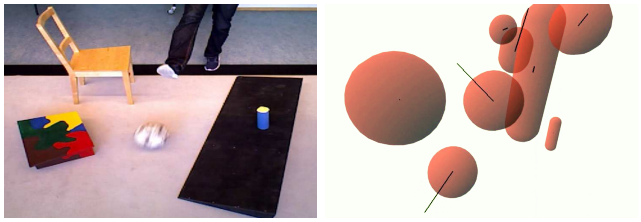


Fig. 7. GMM clustering with Kalman filter for obstacle tracking. Left: in the reference rgb figure it can be seen how a person suddenly kicks a ball. Right: several SSVs can be tracked simultaneously (surfaces are discarded for this example); velocity vectors are shown as black segments.

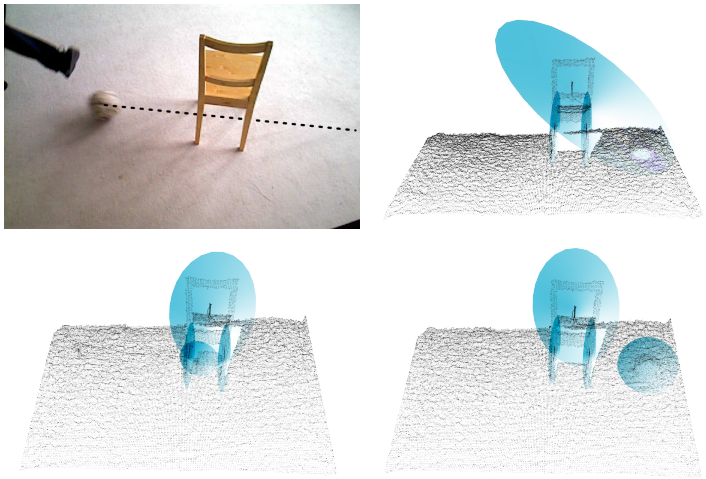


Fig. 8. Keeping separate obstacles separated. In this example, a ball rolls under a chair (top left). The standard application of the EM algorithm doesn't take into account dynamic scenes and obstacles are joined together (top right, $\lambda = 0$). A low-pass filter on Σ (see (6)) keeps a better track of the ball (bottom, $\lambda = 0.95$). Only the Gaussians are shown here for simplicity.

An intrinsic problem of such a probabilistic method consists of determining the correct number of Gaussians (or obstacles) in the scene, especially when handling dynamic scenarios. In order to solve it, we group the points according to a simple voxel grid with a coarse resolution (see Fig. 9) such as the one used for surface clustering (Subsection 3.3) but in 3D (note again that the 3D grid is only used for clustering while the original points are passed along to further stages). Again we obtain a quick clustering of the scene, consisting of several "qclusters". These are used to split Gaussians that contain a considerable amount of points in more than one qcluster (see middle image in Fig. 9). Additionally, new obstacles are created when a qcluster is found on which most points don't belong to previous Gaussians.

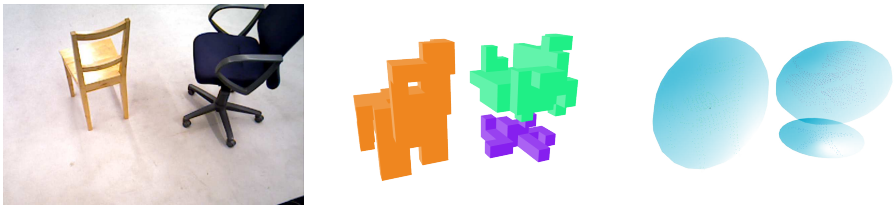


Fig. 9. Determination of the correct number of Gaussians. From left to right: two chairs, voxel grid for quick clustering and point cloud with approximating Gaussians (the ground is hidden).

Obstacles are removed when they present low weight values π_k, as they are not needed to represent existing points. Using these criteria to add, split and remove Gaussians from the scene, we initialize the distribution with one Gaussian for the entire point cloud and iterate further. Even in complex scenarios, initialization time is fast enough for our application. Note that each obstacle can be approximated with more than one SSV later [23].

3.4.3. *Approximation*

The SSV approximation is based on the maximum, middle and minimum principal moments of inertia $I_{\max}$, I_{mid} and $I_{\min}$ of every obstacle point cloud (a.k.a. principal axis decomposition). The quotients $\xi_1 = \frac{I_{\min}}{I_{\max}}$ and $\xi_2 = \frac{I_{\text{mid}}}{I_{\max}}$ are geometric invariants, as they don't depend on the scale of the point cloud. Our approximation uses only point- and line-SSVs (see Fig. 10), which satisfy:

- Ideal point-SSVs have $\xi_1 = \xi_2 = 1$
- Ideal line-SSVs have $\xi_1 < \xi_2 = 1$

However, as detected point clouds are incomplete, the identification criteria have to be adapted to experimental values. When neither criterion is satisfied, a more detailed approximation can be achieved by iteratively splitting the point cloud (again) and assigning more than one SSV object to every obstacle (e.g. the chair approximation in Fig. 4). The SSV parameters (fitting) are then heuristically determined the following way [23]: centers along the $I_{\min}$-corresponding axis for line-SSVs and at the point cloud centroid for point-SSVs; radius corresponding to the point with the maximum distance from the axis or centroid, respectively.
These algorithms are implemented to run on-board a humanoid robot. In the following section, their integration with the walking controller is explained. The performance of the vision system and its validation in experiments are shown in Section 5.

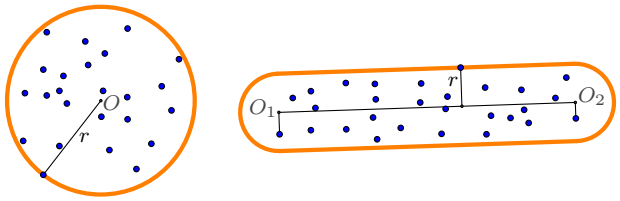


Fig. 10. SSV approximation of point clouds. After clustering, identification and splitting, point-SSVs (left) and line-SSVs (right) are fitted using the centroid O and projections O_1 and O_2 to the $I_{\min}$-corresponding axis, respectively. The radius r corresponds to the farthest-away-point in each case. On the right, the axis of inertia and projections on it are represented with black lines.

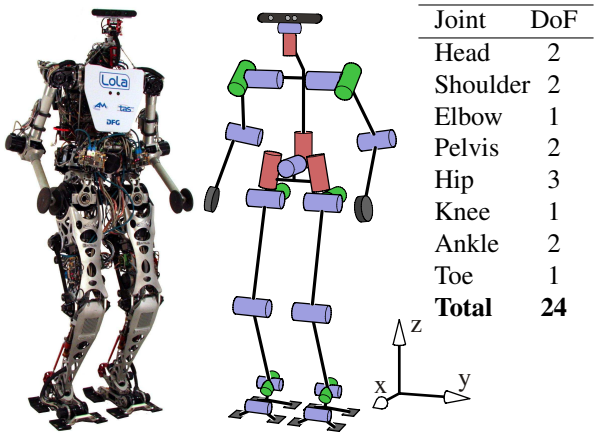
4. Integration with the Walking Controller

4.1. *The Lola Robot*

The humanoid platform for testing these algorithms is the biped robot *Lola*. It is an electrically actuated robot with 24 degrees of freedom which weighs approximately 60 kg and is 1.8 m tall. In Fig. 11, a photo and the kinematic configuration of the robot can be seen. A more precise hardware description can be found in our previous work[47]. For 3D sensing, we use a standard RGB-D sensor, the Asus Xtion PRO LIVE [48] (30 Hz) mounted on top. These low-cost sensors come with relatively high noise. As our motivation is to make a system capable of walking in all kinds of environments (e.g. gravel, grass) where surface detection can never be accurate, this choice serves as robustness testbench.
The walking controller and the vision system run on two parallel on–board computers with Intel Core i7-4770S@3.1 GHz (4x) processors and 8GB RAM and communicate via Ethernet using UDP and TCP/IP.

4.2. *Motion Planning*

Our walking control system as depicted in Fig. 12 follows a hierarchical approach. It is divided into a *Planning Unit* and a *Feedback Control*. Both receive the environment approximation of the vision system and use it internally to allow for collision-free motion generation. The planning unit calculates an *Ideal Walking Pattern* over a time horizon of multiple walking steps. It gets desired walking step parameters, like the walking step length, desired goal positions or a desired velocity vector as an input from the user. Thanks to a cycle time of T_{step}, it is very responsive to changes in the environment or in user input.
Based on the environment representation and the user's input, the *Navigation* module calculates a sequence of parameter sets describing the walking pattern based on an A^* search implementation [18]. Here, we present an extension to take stairs and platforms into account.
Using the output of the navigation module as an initial solution to the motion planning problem, the *Predictive Kinematic Evaluation & Optimization* module



Joint	DoF
Head	2
Shoulder	2
Elbow	1
Pelvis	2
Hip	3
Knee	1
Ankle	2
Toe	1
Total	**24**

Fig. 11. Photo and kinematic structure of the humanoid robot *Lola* with an RGB-D sensor mounted on top.

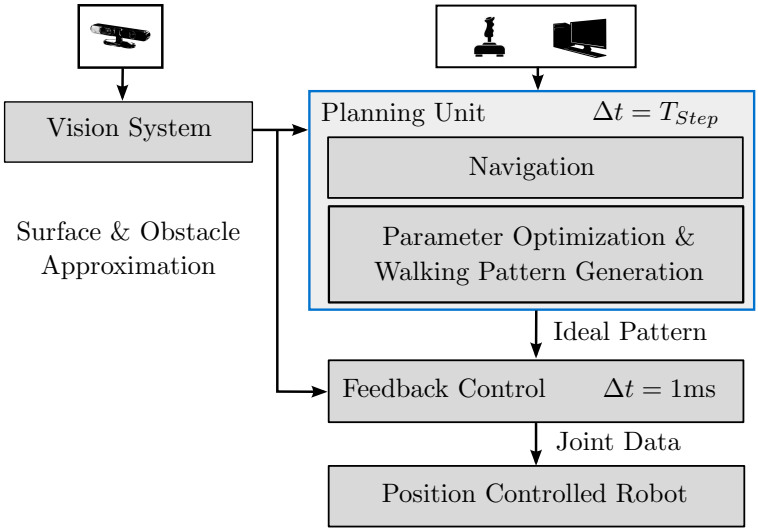


Fig. 12. *Lola*'s real-time walking control system.

optimizes the parameter set and calculates kinematically feasible, collision-free and dynamically executable trajectories taking the environment representation and the full robot approximation into account [21].

These trajectories are the input to the *Feedback Control* unit. It is called in a cycle time of 1 ms and adapts the ideal planned trajectories locally according to sensor

feedback and taking the full approximation of the robot and the environment for collision avoidance into account [22].

4.3. *Navigation*

The objective of the navigation module follows ideas from other authors [28, 17]. Its purpose is not to find long distance paths but to give the user a reactive system which is able to safely navigate in cluttered environments. This is especially important if no full map of the environment is available and the user as well as the robot depends only on the robot's limited field of view (see Fig. 5). According to the application, the user should have the possibility to guide the robot with a joystick, give it desired walking parameters, or set intermediate goal positions which the robot should reach. The robot should execute these high-level commands and take care of a safe and optimal path. In the following, this process is explained in detail.

4.3.1. *A*–search*

Based on a discrete set of footsteps an implicit A*-search is applied to solve the planning problem and to find a sequence of n_{Steps} walking steps. The robot's state s, representing the nodes of the used A*-search, is described by the current stance foot $stance = (left, right)$, the global position $\boldsymbol{r} = (x, y, z)$ and orientation $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)$ of the stance foot. It follows $s = (x, y, z, \boldsymbol{\theta}, stance)$.
Since the possible footstep locations are symmetric for the left and the right stance foot, we define one action model as $a = (\Delta x, \Delta y, \Delta\theta_z, h_{obst}, h_{Step}, c_a)$. Δx, Δy and $\Delta\theta_z$ represent the possible displacements and rotations relative to the current stance foot. Δz, $\Delta\theta_x$ and $\Delta\theta_y$ are not taken into account as part of the action model since they are directly determined by the current position (x, y) of the robot in the represented world. We augment the action model by h_{obst}, which takes into account obstacles the robot has to step over to reach the next s and by h_{step}, which denotes the height change the action can make. The cost for each action is denoted by c_a. Further details are explained in our previous publication [18].

4.3.2. *Collision Checking*

The main difference to other A*-search based footstep planners for biped locomotion [17, 49] is the collision world representation and its consistent consideration in all modules of trajectory generation – from footstep planning to reactive collision avoidance [22]. Instead of using a grid-based environment representation which is able to check collisions in a binary way for a 2.5D map, we base our planning module on the environment representation presented in this paper which allows for collision checks in full 3D. In the footstep planner, collisions are checked not by a planar rectangle but by a 3D model of the lower leg including the foot and

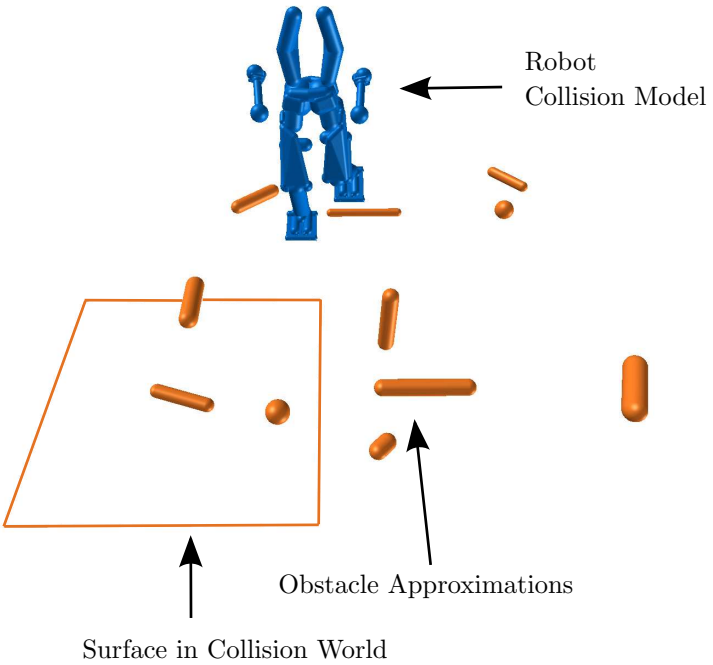


Fig. 13. Collision model of the footstep planner for *Lola* stepping over an obstacle with movable leg segments. Both the robot and obstacles are modeled with SSVs. Surfaces are represented as polygons with edges modeled as SSVs to avoid stepping on them using the collision avoidance framework.

a leg segment approximation. This gives a better approximation of the full robot movement, especially for large strides, and allows for reduced safety margins.

4.3.3. 3*D Walking*

In addition to the viability of a state s and the corresponding action model a, the footstep planner has to evaluate the 6D pose of the foothold. As introduced above, the rotation θ_y, θ_z and the height z are a direct function of the environment, x, y and θ_z. Apart from areas the robot is not able to step on (obstacles), we introduce areas the robot can step on (surfaces) to the environment representation (see Fig. 13). As presented in Subsection 3.3, surfaces are represented by convex hulls (polygons) and a normal to the surface (see Fig. 1). In order to prevent the robot from stepping onto the edges of the surfaces, these are modeled as obstacles using line-SSVs. Thus, the complete foot is in contact with the surface, which helps to maintain the robot's stability and prevents additional modifications to the walking controller. This representation has several advantages:

(1) Based on the current x and y value of s, the footstep planner is able to determine

the whole 6D pose of the foot just by checking in which polygon the current s is lying, which is computationally efficient.
(2) Surfaces are completely defined by the corner points and the normal of the surface. This is a memory-efficient representation (a more detailed representation would be unnecessary) that simplifies communication between planning modules and the vision system. In the current implementation, a maximum of eight corner points are used. Depending on the desired level of detail it can be easily extended to a higher (or lower) number of corner points.
(3) Additionally, surfaces are included consistently using SSV elements in our collision avoidance framework. That way, the motion generation modules are able to generate collision-free whole body motion [22, 21].

4.4. *Data Visualization - Augmented Reality*

When developing an autonomous navigation system, as well as when performing teleoperation with a robot, it is useful to visualize the results of the different framework modules. In order to help visualize both the results of the perception system as well as its influence on the motion planner, an augmented (or mixed) reality system is developed. It projects collision geometries, surface approximations, and footstep positions online into the scene. This is done either via an external RGB video feed from the scene[f] or using Microsoft's HoloLens[g]; both tools are available open source in the repository for the benefit of the community.
The objective of these systems is to provide immediate feedback about the perception system's accuracy and the quality of the motion planner's output in a context which can be immediately understoood at a glance. In Fig. 14, the final setup is shown. The vision system (Lola Environment Perception System, or LEPP) sends the environment approximation results to the control computer (Control) while receiving odometry information from it (State Server). Additionally, two other computers are included in the network for augmented reality. They both receive the environment approximation results from LEPP, the actual and planned footstep positions from Control and the odometry information from State Server. In the following, both implementations are described.
The results of the first system (*Lola Listener* in Fig. 14) are shown in Fig. 15. In order to render the data correctly and obtain a good registration between the virtual objects and the RGB video feed, two things need to be taken care of. First, the camera's intrinsic parameters need to be measured and used to modify the projection. Virtual objects are rendered to reproduce (as much as possible) the physical camera's characteristics (such as FoV or lens distortion) with a virtual camera. This ensures that if a virtual object and a physical object lie at the same location relative to the camera, they should cover the same pixels in the RGB image.

[f] `https://github.com/am-lola/LolAR`
[g] `https://github.com/am-lola/HoLola`

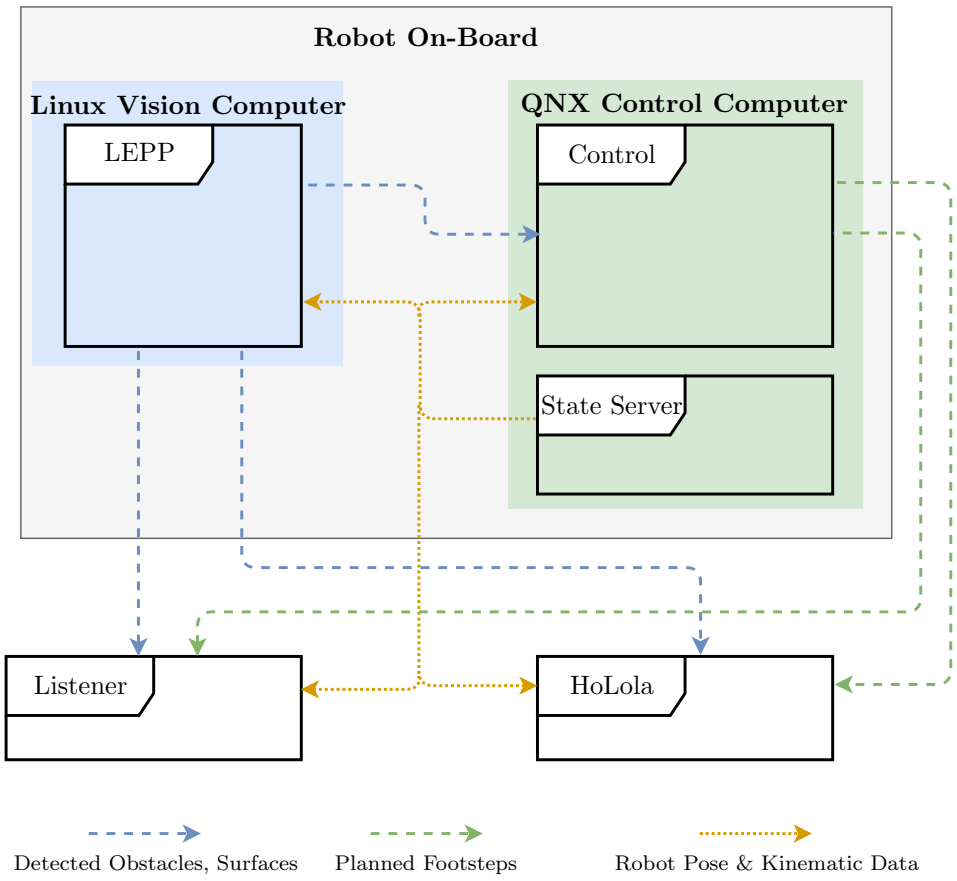


Fig. 14. Setup for autonomous walking and augmented reality. The walking controller and vision system run on two parallel computers on-board. Two external systems for augmented reality receive information from them online.

Second, in order to place virtual objects correctly, the transformation between the camera's 6D pose and the robot's coordinate system needs to be found.
While the first point can be solved using standard camera calibration routines, there is no simple, adequate solution available for the second one. For this example, a manual calibration was done. Nevertheless, a module is introduced that uses markers from the ArUco library [50] to perform the calibration automatically. The location of the ArUco marker is improved using the correlated depth data. Additionally, by estimating the location of the ground from the camera, the height and two spatial orientations can be calculated. These are used to further improve the pose estimation.
A less expensive augmented reality system (*HoLola* in Fig. 14) is developed for

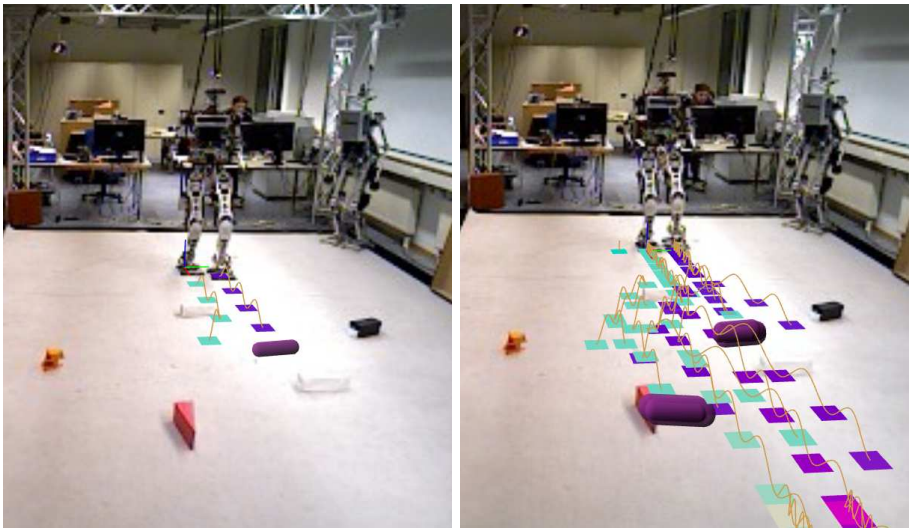


Fig. 15. Augmented reality system with a structured light sensor. In this picture, calibration is performed manually. Left: the planned walking steps and results of the perception system at one point are projected into the RGB feed. Right: the accumulated results of the perception system and the footstep planner during the complete run against the initial RGB frame.

the Microsoft HoloLens [25]. The visual feed through the HoloLens can be seen in Fig. 16. The HoloLens application is built in two layers: a simple visualization layer which draws/removes data as the robot generates new object approximations and footstep plans, and a low-level networking layer for communication with the robot's computers. Because the HoloLens localizes itself very accurately, rather than trying to identify the device in the robot's coordinate system a common reference is used. The users can place a coordinate frame anywhere in the HoloLens' map of the world, and by aligning this with the robot coordinate system, all data from the robot can easily be rendered. The location selected by users in this way is anchored to the features that the HoloLens uses for tracking and persists between executions of the application, so it only needs to be adjusted when the location of the robot coordinate system's origin changes. It can also be updated on the fly at any time. The HoloLens also provides an RGB feed from a front-facing camera on the device which could be used to locate a marker on the robot, as with the external camera application. A short video can be found online at `https://youtu.be/EeDR1UNDpIY`.

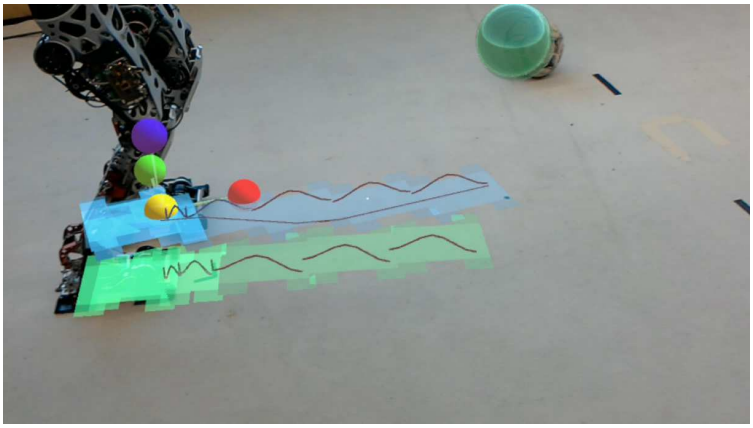


Fig. 16. Augmented reality system with the HoloLens. Both the results of the perception system and the motion planner are projected into the environment (the ground plane is left out for clarity). The executed footsteps are depicted in blue and green and the planned ones in the same, but semi-transparent colors. The calibration is performed by placing a virtual coordinate system (depicted with a yellow sphere at the origin and red, green and blue spheres corresponding to the x, y and z axes, respectively) coincident with the one of the robot at the foot. The small error in the obstacle approximation is due to occlusion.

5. Results

5.1. *Simulations*

In order to evaluate the effectiveness of our algorithm, we simulate several environments using synthetic point clouds. We first create a dataset of 3D files, including one for a large floor area from which point clouds are generated. By scaling, transforming and combining these objects, we can automatically and randomly create increasingly complex scenes. Moreover, we can simulate moving objects by applying frame-varying transformations and generating a stream of point clouds. Additionally, in order to better recreate the real scenario, the robot's field of view and its resulting occlusion effect are also taken into account by frustum culling and ray casting [39] (see Fig. 17).

With these synthetic scenarios, we evaluated the approximation of obstacles, surfaces and tracking of dynamic objects. By randomly varying scale, transformation and combinations of platforms and objects with different shapes, parameters were adjusted and the results of both the *Surface* and *Obstacle Approximation* were compared against scaled ideal values of the original object dataset. Around 100 different scenarios were tested.

For the polygon evaluation, we used inclined platforms with different sizes and the following shapes: circles, rectangles with normal/round corners, ellipses, regular and irregular convex polygons and polygons with some rounded corners. The error in inclination is negligible in all cases. While the error in area lies in the range of 0-4 %

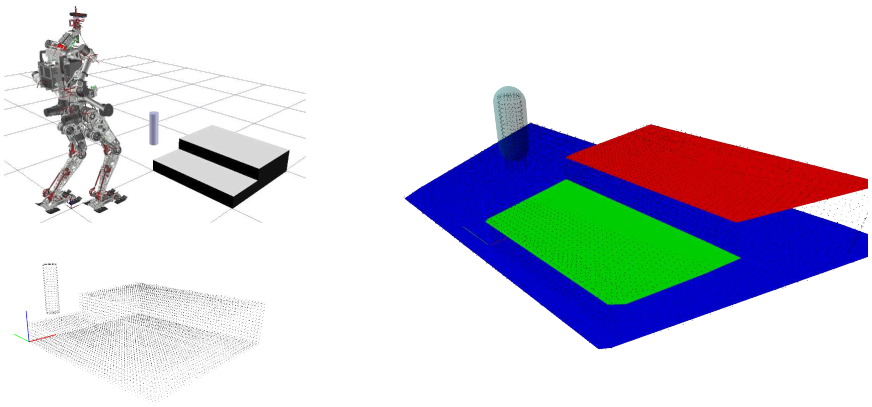


Fig. 17. Evaluation of the algorithm via synthetic point clouds: 3D objects (top left) are transformed into point clouds (bottom left) which are combined with the floor, filtered via frustum culling and ray casting, and approximated by the vision system (right).

for polygons, the error in area can be up to 10 % in the case of rounded shapes: this is mainly caused by the limited number of vertices used in the polygon approximation. However, the approximated area is always smaller than the original area, and the remaining points will be approximated with SSVs so the result is always safe for navigation.

In the case of the SSV evaluation, we used prisms, cylinders, platforms and combinations of more than one shape. The volume of the approximating SSVs varies between 100-300 % of the original shapes, with the best results corresponding to rounded shapes and higher number of splitting steps. As expected, the volume of the approximation is consistently higher than the original shapes (due to the conservative fitting strategy).

Using a stream of point clouds, we first evaluated the obstacle tracking (see Fig. 18). The obstacle tracking algorithm, running at 30 Hz, is capable of tracking objects moving with constant speeds up to 3 m/s (at higher speeds, the displacement between frames is too large to be correctly matched). We performed around $\simeq 200$ simulations of randomly-sampled velocities between 0.01 m/s and 3 m/s. Convergence of the estimated velocities to a value with less than 3 % error takes between 30-60 frames for the fastest moving objects[h].

The evaluation of synthetic point clouds is a valuable tool for development. Additionally, it helps to validate the capacities of the developed system. It is capable of correctly approximating a large variety of dynamic scenarios. Errors may become significant due to the simplifications necessary for fast processing, but they are always conservative and safe for robot navigation.

[h]The initialization time of the algorithm required when starting the robot is not taken into account.

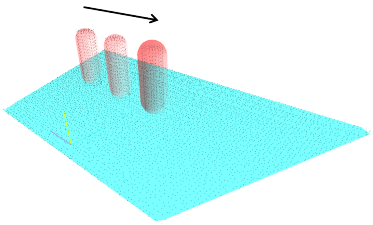


Fig. 18. Obstacle tracking performance. An object moves with constant speed and is approximated with an SSV. Objects moving with constant speeds up to 3 m/s can be successfully tracked.

Spatial Resolution	**Planes**			**Surfaces**			**Obstacles**		
	max	min	mean	max	min	mean	max	min	mean
$1\,cm^3$ ($\sim$ 65000 points)	45	6	**13**	373	1	**92**	353	8	**107**
$2\,cm^3$ ($\sim$ 21000 points)	19	2	**6**	123	1	**21**	284	3	**57**

Table 1. Performance of Vision System. Runtime (in ms) for different point cloud resolutions of the (*Plane Segmentation*), (*Surface Approximation*) and (*Obstacle Approximation*) processes of the vision system (maximum, minimum and mean values) for highly complex and dynamic scenarios spanning 500 frames approximately.

5.2. *Performance of the Vision System*

Here we evaluate the live performance of our vision system. In order to obtain a greater amount of data and validate our system in more complex environments, a series of dynamic scenes with humans, objects and platforms are recorded in front of the robot's camera. These include dynamic scenes with several objects and platforms of different shapes and sizes. Examples of these scenes can be seen throughout this paper in Figs. 4 and 7 to 9, including transitions between them. The duration of the different processes can be seen in Table 1. In the *Surface Approximation* process, higher runtimes correspond to frames that consist mostly of walkable planes, as the different algorithms need to iterate through a high number of points. In the case of the *Obstacle Approximation* process, higher runtimes are the result of sudden changes of the scene where the algorithm needs to re-converge; they improve considerably after a few frames. The final list of obstacles and surfaces are sent to the motion planning module with a set frequency of around 5 Hz, which re-plans the future 8 walking footsteps every walking step (0.8 s).

Unfortunately, open source perception systems for humanoids are extremely uncommon. Most relevant publications in the field don't release their source code and often omit thorough performance results, which makes it difficult to perform an objective comparison. Nevertheless, a few examples are mentioned for reference. The framework presented by Nishiwaki et al.[17] requires a 1 second sensor sweep with the robot still to acquire and process perception information to create 2.5D maps. Us-

ing a similar sensor and resolution as this work, the framework presented by Maier et al.[29] for the *Nao* robot runs at a frequency of 6 Hz and uses a 3D voxelization to model environments, without performing surface segmentation. More recently, Fallon et al.[32] presented an approach for sensor fusion and plane detection. It is based on 2.5D maps which are further reduced by removing points belonging to the ground. In its actual configuration, the segmentation process takes 615 ms on average. Even though the algorithm runs during motion, each walking step takes 4 s, which is five times slower than the presented experiments with *Lola*. Moreover, all these frameworks assume static environments in their application. In comparison, the framework presented in this work proved to be faster as well as more flexible and generally applicable.

5.3. *Experiments*

A video of the experiments presented in this section can be found at `https://youtu.be/VceqNJucPiw`. Videos of additional experiments, including a recorded live demo, are available on our YouTube Channel[i]. We tested our framework repeatedly by making our robot *Lola* continuously walk in different scenarios. It is important to note that, throughout all experiments performed, the robot walks continuously without stopping and both the vision system and the motion planning module react to previously unknown scenarios during walking. In order to highlight the features presented throughout in this paper, we discuss three of them[j] in the following:

Unexpected Obstacles

In this experiment we confirm the ability of the robot to avoid unexpected obstacles, using motions such as stepping over or sideways. A sequence of the video and the results of the vision system is depicted in Fig. 19. The robot needs to find a way through the room which is blocked with relatively small, previously unknown obstacles in the ground. In contrast to our previous work [23], this time the complete vision system recognizing surfaces as well as obstacles is used, and the new motion planning module [24] generates broader and more flexible paths through the room.

Platform

In these experiments we test the reaction of our framework to uneven terrain, such as platforms or stairs. We validate both the ability of the vision system to detect and model surfaces accurately and fast enough during walking and the flexibility of our motion planner to adapt the walking sequence in real-time. A sequence of the video and the results of the vision system is depicted in Fig. 20. We place a platform (12 cm high) and a few obstacles on the ground (to block alternative paths around the platform), obstructing the way through the room. The robot walks up and down the platform on its way forward (see Fig. 21).

[i] `https://www.youtube.com/appliedmechanicstum`

[j] All experiments shown here were performed with the same configuration as the one available in our repository.

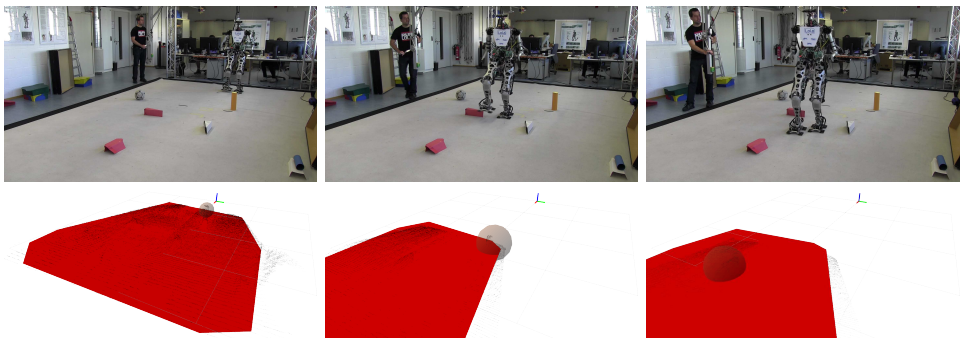


Fig. 19. Experiment with unexpected obstacles. Top: the robot tries to find a way through the room obstructed by obstacles on the ground. Bottom: the results of the vision system (the point cloud is cropped to the walking area).

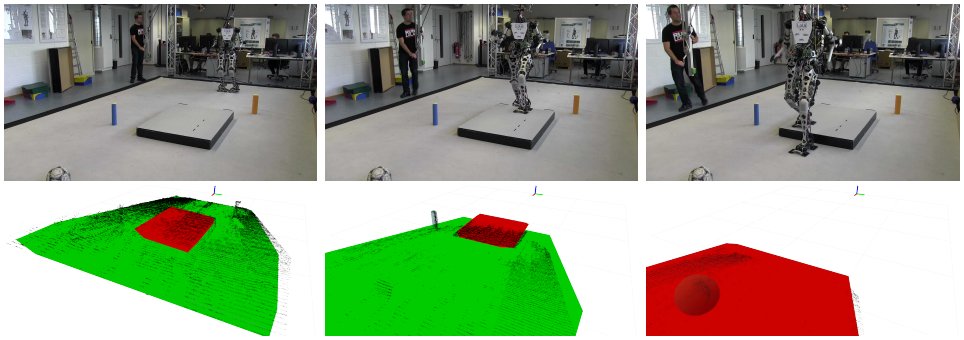


Fig. 20. Experiment with unexpected platform. Top: the robot walks up and down an unexpected platform which stands in the way to the end of the room. Bottom: the results of the vision system (the point cloud is cropped to the walking area).

Highly Dynamic Scenarios with Large Obstacles and Humans

In this experiment we test the ability of our framework to react to dynamic environments. We validate both the ability of the vision system to track large dynamic objects during walking and the fast reaction times from our motion planner [24]. A sequence of the video and the results of the vision system is depicted in Fig. 22. We give the robot a goal position in an initially empty area. When the robot starts moving, a person walks in, blocking its path. When the robot turns to avoid the person, he blocks its path again with a chair. After avoiding the chair and the person, the robot's path is blocked yet again so that it is prevented from reaching the goal position. The robot's reaction (which depends on the sensor's limited FOV) can be clearly seen in Fig. 22, emphasizing the real-time capabilities of the framework.

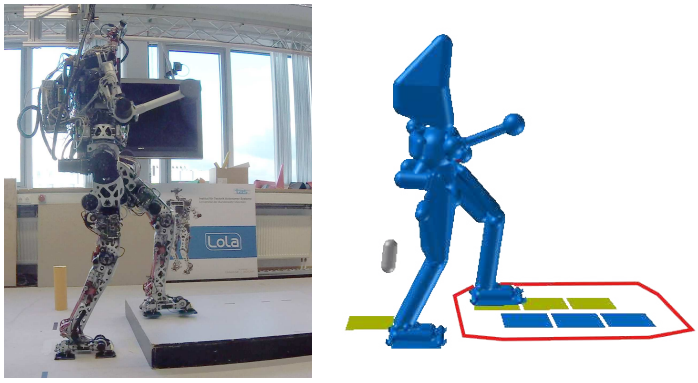


Fig. 21. Collision world. Left: the robot walks up an unexpected platform which stands in the way to the end of the room. Right: representation of the 3D collision world including the robot and external obstacles modeled as SSVs, the platform modeled as a polygon and the planned footstep locations.

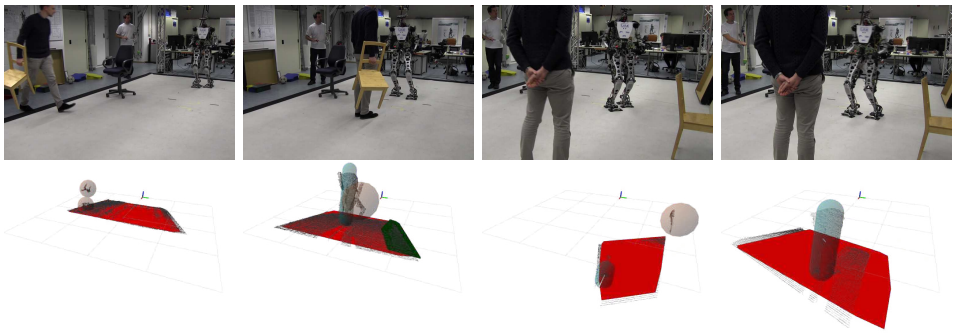


Fig. 22. Experiment in dynamic scenarios. Top: the robot tries to reach a goal position in the room despite a person repeatedly blocking its way. Bottom: the results of the vision system (the point cloud is cropped to the walking area and further-away points are removed to ensure limited reaction times).

6. Conclusion

In this work we presented a framework for autonomous walking in unknown dynamic environments. It is based on a novel 3D environment representation which presents considerable advantages for real-time motion planning and collision avoidance. We introduced an open source 3D vision system for environment modeling that represents dynamic scenarios with simple geometries. It is integrated with our motion planning system to perform autonomous navigation in unknown, dynamic scenarios with our robot *Lola*. We validated our framework via simulation and experiments. Our robot is now able to walk over platforms at around 0.4 m/s while avoiding moving obstacles in previously unknown scenarios. To facilitate our research we

developed visualization and augmented reality tools that project the results of the vision and planning systems into the scene; they are released open source as well. Future work includes the integration of environment recognition strategies to augment the capabilities of the robot. On the control side, we plan to introduce time explicitly into the motion planning strategy to coordinate the robot's motion with moving scenarios.

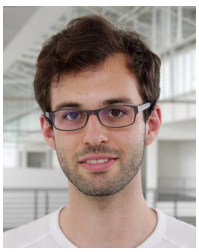
Acknowledgements

This work is supported by the DAAD and the DFG (project BU 2736/1-1). Special thanks go to our students Christian Buttner, Stefan Floeren, Dominik Gutermuth, Gregor Schwarz, Irem Uygur and Sahand Yousefpour for their help in the implementation of the ideas presented here.

Daniel Wahrmann received his Diploma degree in Mechanical Engineering from the Instituto Tecnológico de Buenos Aires, Argentina, where he worked 2 years in the development of nanosatelites. He is currently pursuing his Ph.D. at the Technical University of Munich and his research interests include biped walking, environment recognition and real-time autonomous navigation.

Arne-Christoph Hildebrandt received a Double-Diploma degree in Mechanical Engineering from the Karlsruhe Institute of Technology, Germany, and the Arts et Métiers ParisTech, France, in 2013. He is currently pursuing his Ph.D. at the Technical University of Munich. His research interests include bipedal walking, real-time motion planning and collision avoidance.

Tamas Bates received the M.Sc. degree in informatics from the Technical University of Munich, Germany, in 2017. Since then, he has been working toward the Ph.D. degree at the Technical University of Delft, the Netherlands, with a focus on cooperative human-robot interaction and communication of physical intentions between humans and robots.



Robert Wittmann received the Ph.D. degree in mechanical engineering in 2017 at the Chair of Applied Mechanics, Technical University of Munich. His research interests include bipedal walking control, state estimation and real-time trajectory generation and optimization.

Felix Sygulla received his B.Sc. (2011) and M.Sc. (2015) in mechatronics and mechanical engineering from the Technical University of Munich, Germany. He currently is a research assistant and Ph.D. student at the Chair of Applied Mechanics, Technical University of Munich. His research interests include bipedal walking control, contact force control and tactile feedback.

Philipp Seiwald received the M.Sc. degree in mechanical engineering from Technical University of Munich, Germany, in 2016. Since then, he has been working toward the Ph.D. degree at the Technical University of Munich, Chair of Applied Mechanics, Munich, Germany. His research interests are in the field of humanoid robots focusing on motion planning and dynamics in gaited multi-contact locomotion.

Daniel Rixen recieved his PhD from the University of Liège (Belgium) in 1997, then spent two years at the Center for Aerospace Structures of the University of Colorado. From 2000-2012, he was head of the chair for Engineering Dynamics at the Technical University of Delft (The Netherlands) and since 2012 he directs the chair for Applied Mechanics of the Technical University of Munich (Germany). His research is centered around engineering dynamics, dealing with numerical simulation methods, experimental techniques and robotics.

References

1. P. Hart, N. Nilsson and B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics* **4**(2), 100–107 (1968).
2. B. Siciliano and O. Khatib, *Springer Handbook of Robotics* (Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007).
3. H. Moravec and A. Elfes, High resolution maps from wide angle sonar, in *IEEE International Conference on Robotics and Automation* **2**1985.
4. M. Herbert, C. Caillas, E. Krotkov, I. Kweon and T. Kanade, Terrain mapping for a roving planetary explorer, in *IEEE International Conference on Robotics and Automation* 1989.
5. DARPA Robotics Challenge - `http://archive.darpa.mil/roboticschallenge/` (2017).
6. S. Kajita and K. Tani, Study of Dynamic Biped Locomotion on Rugged Terrain - Derivation and Application of the Linear Inverted Pendulum Mode, in *IEEE International Conference on Robotics and Automation* 1991, pp. 1405–1411.
7. K. Loffler, M. Gienger and F. Pfeiffer, Model Based Control of a Biped Robot, in *7th International Workshop on Advanced Motion Control* (IEEE, 2002), pp. 443–448.
8. K. Nishiwaki and S. Kagami, High Frequency Walking Pattern Generation based on Preview Control of ZMP, in *IEEE-RAS International Conference on Humanoid Robots* 2006, pp. 2667–2672.
9. T. Buschmann, S. Lohmeier, M. Bachmayer, H. Ulbrich and F. Pfeiffer, A Collocation Method for Real-Time Walking Pattern Generation, in *IEEE-RAS International Conference on Humanoid Robots* 2007, pp. 1–6.
10. T. Takenaka, T. Matsumoto and T. Yoshiike, Real Time Motion Generation and Control for Biped Robot - 1st Report: Walking Gait Pattern Generation-, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* 2009, pp. 1084–1091.
11. C. Wöhler, *3D Computer Vision: Efficient Methods and Applications* (Springer London, 2013).
12. D. Wahrmann, A.-C. Hildebrandt, C. Schuetz, R. Wittmann and D. Rixen, An Autonomous and Flexible Robotic Framework for Logistics Applications, *Journal of Intelligent & Robotic Systems* (dec 2017).
13. S. Kagami, K. Nishiwaki, J. J. Kuffner, K. Okada, M. Inaba and H. Inoue, Vision-based 2.5 D terrain modeling for humanoid locomotion, in *IEEE International Conference on Robotics and Automation* 2003, pp. 2141–2146.
14. J. Gutmann, M. Fukuchi and M. Fujita, A Floor and Obstacle Height Map for 3D Navigation of a Humanoid Robot, in *IEEE International Conference on Robotics and Automation* 2005, pp. 1066–1071.
15. J. S. Gutmann, M. Fukuchi and M. Fujita, 3D Perception and Environment Map Generation for Humanoid Robot Navigation, *The International Journal of Robotics Research* **27**(10), 1117–1134 (2008).
16. J. Chestnutt, Y. Takaoka, K. Suga, K. Nishiwaki, J. Kuffner and S. Kagami, Biped Navigation in Rough Environments using On-board Sensing, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* 2009, pp. 3543–3548.
17. K. Nishiwaki, J. Chestnutt and S. Kagami, Autonomous Navigation of a Humanoid Robot over Unknown Rough Terrain using a Laser Range Sensor, *The International Journal of Robotics Research* **31**(11), 1251–1262 (2012).
18. A.-C. Hildebrandt, D. Wahrmann, R. Wittmann, D. Rixen and T. Buschmann, Real-Time Pattern Generation Among Obstacles for Biped Robots, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* 2015, pp. 2780–2786.

19. A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees, *Autonomous Robots* **34**, 189–206 (2013).
20. M. Schwienbacher, T. Buschmann, S. Lohmeier, V. Favot and H. Ulbrich, Self-collision avoidance and angular momentum compensation for a biped humanoid robot, in *IEEE International Conference on Robotics and Automation* 2011, pp. 581–586.
21. A. C. Hildebrandt, M. Demmeler, R. Wittmann, D. Wahrmann, F. Sygulla, D. Rixen and T. Buschmann, Real-Time Predictive Kinematic Evaluation and Optimization for Biped Robots, in *IEEE International Conference on Intelligent Robots and Systems* 2016, pp. 5789–5796.
22. A.-C. Hildebrandt, R. Wittmann, D. Wahrmann, A. Ewald and T. Buschmann, Real-time 3D collision avoidance for biped robots, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* 2014, pp. 4184–4190.
23. D. Wahrmann, A.-C. Hildebrandt, R. Wittmann, F. Sygulla, D. Rixen and T. Buschmann, Fast Object Approximation for Real-Time 3D Obstacle Avoidance with Biped Robots, in *IEEE International Conference on Advanced Intelligent Mechatronics.* 2016, pp. 38–45.
24. A.-C. Hildebrandt, M. Klischat, D. Wahrmann, R. Wittmann, F. Sygulla, P. Seiwald, D. Rixen and T. Buschmann, Real-Time Path Planning in Unknown Environments for Bipedal Robots, *IEEE Robotics and Automation Letters* **2**(4), 1856—-1863 (2017).
25. Microsoft HoloLens - `https://www.microsoft.com/en-us/hololens`.
26. R. Cupec and G. Schmidt, An Approach to Environment Modelling for Biped Walking Robots, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* 2005, pp. 424–429.
27. J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins and T. Kanade, Footstep planning for the Honda ASIMO humanoid, in *IEEE International Conference on Robotics and Automation* 2005, pp. 629–634.
28. T. Buschmann, S. Lohmeier, M. Schwienbacher, V. Favot, H. Ulbrich, F. Von Hundelshausen, G. Rohe and H.-J. J. Wuensche, Walking in Unknown Environments — a Step Towards More Autonomy, in *IEEE-RAS International Conference on Humanoid Robots* 2010, pp. 237–244.
29. D. Maier, A. Hornung and M. Bennewitz, Real-time navigation in 3D environments based on depth camera data, in *IEEE International Conference on Humanoid Robots* 2012, pp. 692–697.
30. D. Maier, C. Stachniss and M. Bennewitz, Vision-Based Humanoid Navigation Using Self-Supervized Obstacle Detection, *International Journal of Humanoid Robotics* **10**, p. 1350016 (2013).
31. A. Stumpf, S. Kohlbrecher, D. C. Conner, O. von Stryk, O. V. Stryk and O. von Stryk, Supervised Footstep Planning for Humanoid Robots in Rough Terrain Tasks using a Black Box Walking Controller, in *IEEE-RAS International Conference on Humanoid Robots* 2014, pp. 287–294.
32. M. F. Fallon, P. Marion, R. Deits, T. Whelan, M. Antone, J. Mcdonald and R. Tedrake, Continuous Humanoid Locomotion over Uneven Terrain using Stereo Fusion, in *IEEE-RAS International Conference on Humanoid Robots* 2015, pp. 881–888.
33. A. Ess, K. Schindler, B. Leibe and L. Van Gool, Object Detection and Tracking for Autonomous Navigation in Dynamic Environments, *The International Journal of Robotics Research* **29**(14), 1707–1725 (2010).
34. K. Khoshelham and S. O. Elberink, Accuracy and resolution of Kinect depth data for indoor mapping applications, *Sensors* **12**(2), 1437–54 (2012).
35. W. Kehl, F. Milletari, F. Tombari, S. Ilic and N. Navab, Deep Learning of Local RGB-

D Patches for 3D Object Detection and 6D Pose Estimation, in *European Conference on Computer Vision (ECCV)* 2016.
36. Z. Zhang, Microsoft Kinect Sensor and Its Effect, *IEEE Multimedia* **19**, 4–10(feb 2012).
37. R. B. Rusu and S. Cousins, 3D is here: Point Cloud Library (PCL), in *IEEE International Conference on Robotics and Automation* 2011, pp. 1–4.
38. M. a. Fischler and R. C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, *Communications of the ACM* **24**, 381–395 (1981).
39. Point Cloud Library (PCL) Documentation - `http://docs.pointclouds.org/trunk/` (2017).
40. Open Source Computer Vision Library - `http://opencv.org/` (2017).
41. M. Ester, H. P. Kriegel, J. Sander and X. Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, in *2nd International Conference on Knowledge Discovery and Data Mining* 1996, pp. 226–231.
42. W. F. Eddy, A New Convex Hull Algorthm for Planar Sets, *ACM Transactions on Mathematical Software* **3**(4), 393–403 (1977).
43. A. Bykat, Convex hull of a finite set of points in two dimensions, *Information Processing Letters* **7**(6), 296–298 (1978).
44. K. P. Murphy, *Machine Learning: A Probabilistic Perspective* (The MIT Press, 2012).
45. S. Koo, D. Lee and D. S. Kwon, Incremental object learning and robust tracking of multiple objects from RGB-D point set data, *Journal of Visual Communication and Image Representation* **25**(1), 108–121 (2014).
46. L. Frenkel and M. Feder, Recursive Expectation-Maximization (EM) Algorithms for Time-Varying Parameters with Applications to Multiple Target Tracking, *IEEE Transactions on Signal Processing* **47**(2), 306–320 (1999).
47. F. Sygulla, R. Wittmann, P. Seiwald, T. Berninger, A.-C. Hildebrandt, D. Wahrmann and D. Rixen, An EtherCAT-Based Real-Time Control System Architecture for Humanoid Robots, *IEEE Robotics and Automation Letters (submitted)* (2018).
48. Asus Xtion PRO LIVE - `https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/` (2017).
49. A. Hornung, A. Dornbush, M. Likhachev and M. Bennewitz, Anytime search-based footstep planning with suboptimality bounds, in *IEEE-RAS International Conference on Humanoid Robots* 2012, pp. 674–679.
50. S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas and M. J. Marín-Jiménez, Automatic generation and detection of highly reliable fiducial markers under occlusion, *Pattern Recognition* **47**(6), 2280–2292 (2014).