Fast Object Approximation for Real-Time 3D Obstacle Avoidance with Biped Robots

Daniel Wahrmann, Arne-Christoph Hildebrandt, Robert Wittmann,
Felix Sygulla, Daniel Rixen and Thomas Buschmann*

***Abstract*— In order to achieve fully autonomous humanoid navigation, environment perception must be both fast enough for real-time planning in dynamic environments and robust against previously unknown scenarios. We present an open source, flexible and efficient vision system that represents dynamic environments using simple geometries. Based only on on-board sensing and 3D point cloud processing, it approximates objects using swept-sphere-volumes while the robot is moving. It does not rely on color or any previous models or information. We demonstrate the viability of our approach by testing it on our human-sized biped robot *Lola*, which is able to avoid moving obstacles in real-time while walking at a set speed of 0.4m/s and performing whole-body collision avoidance.**

I. Introduction

One of the main incentives behind the development of humanoid robots is the natural advantage of these machines over wheeled platforms in uneven terrain. Having a machine that can walk autonomously in a cluttered environment becomes even more important in hazardous areas (e.g. a nuclear power plant after an accident) that were designed for humans but cannot be accessed by them.

One of the challenges to achieve real autonomy in unknown dynamic scenarios is to recognize and model the environment in a certain way that the robot can adapt its movements in real-time. Stereo-solving algorithms and 3D computer vision methods are usually computationally very expensive (see [1]), which complicates the application on mobile robots due to the dynamics of these fast machines. In recent years small and inexpensive RGB-D sensors, which provide direct 3D information, became available (notably the systems developed by PrimeSense, e.g. Kinect [2]). Although not exempt from certain flaws (e.g. a low precision and poor outdoors performance [3]), they facilitate the implementation of complex navigation tasks.

Our work focuses on the autonomous real-time navigation of biped robots in unknown environments. As a step towards that goal, we present an approach for 3D object recognition and modelling that is both efficient enough to work in real-time during walking and sufficiently detailed for complex obstacle avoidance strategies (walking speed is set at 0.4m/s, which is relatively fast for a biped robot). It is based only on on-board sensing, providing a high degree of autonomy. Our system allows our full-sized humanoid robot *Lola* to avoid previously unknown dynamic obstacles during walking, using a standard RGB-D sensor. Our obstacle avoidance strategy was initially presented in [4], where obstacle approximations were manually sent to the system and no perception was involved. It was extended in [5] to use a more general A*-based solution in a framework that admits perception information. It made use of a simple vision system based on height-maps, which was not presented. In this work we present a new, more complex vision system which admits a full 3D representation of the environment, generates better and more complex approximations and can track moving objects. It is integrated with our planning system [5] to autonomously navigate in cluttered, dynamic scenarios. Its main characteristics are:

- Based on point cloud processing, easily adaptable to other sensors and even other robots.
- No external sensors. All sensing and processing is performed on-board.
- No reliance on identifiers such as color, form or texture.
- Objects are approximated using simple geometries for fast distance calculation.
- Highly efficient. Can be used in real-time applications (each frame is processed in 20-120ms).
- Tracks moving objects.
- Designed to operate while the robot is moving. It enables *Lola* to perceive, model and avoid unknown objects during continuous walking.
- Open source[1].

This paper is organized as follows: in sec. II we give an overview of related work. In sec. III we present our humanoid robot *Lola* and hardware. Our vision system is presented in detail in sec. IV, while in sec. V we briefly describe our control system and our motion planning and collision avoidance system. Sec. VI includes our results and experimental validation of our system. The conclusion and outline of future work is presented in sec. VII.

II. Related Work

In this section we review related work on the approximation of obstacles for biped navigation. As we assume no information about the environment, we do not focus on work based on techniques which involve some kind of learning or previous assumptions about the objects.

One of the most common environment modelling strategies for autonomous navigation consists of a two-dimensional height-map or 2.5D map, as introduced by Movarec [6].

*Institute of Applied Mechanics, Technische Universität München, 85748 Garching, Germany. Email: daniel.wahrmann@tum.de

This compact representation of a non-horizontal terrain is computationally very efficient, therefore it was used in most early works on autonomous navigation, as well as in our previous implementation [5].

In 2003 Kagami et al. [7] used stereo cameras to create a height-map of the terrain, from which they could also extract walkable planar surfaces. Assuming a sufficiently structured environment, their *H7* robot was able to plan and navigate collision-free trajectories. In 2004 Cupec and Schmidt [8] used a 2D classification of the environment to allow the robot *Johnnie* to walk on different surfaces and over obstacles, which had previously defined geometries. Gutmann et al. [9], [10] used a combination between a 3D occupancy grid and height-map to navigate between obstacles and climb stairs with Sony's *QRIO* robot. The vision system, based on stereo cameras, classified the environment into obstacles and walkable surfaces but relied on textured surfaces and assumed quasi-static obstacles.

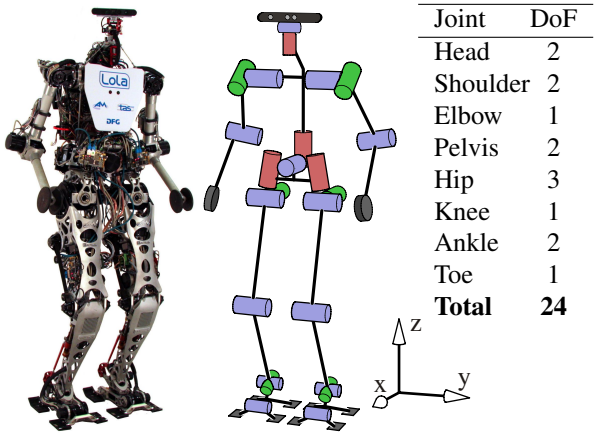
In one of the few works dealing with dynamic environments, Chestnutt et al. [11] managed to navigate between previously known, moving obstacles. In [12] they used a height-map representation of a previously unknown environment using an on-board laser scanner. By extracting planes and labeling other regions as obstacles their biped robot was able to walk over obstacles and onto platforms. The pivoting laser scanner on the waist provided very accurate distance information but could not be easily applied in dynamic environments as it had a scanning time of 1s [13]. More recently, Maier et al. [14], [15] used first an RGB-D sensor and then a laser range finder to create a 3D representation of the environment using *octrees*, which was classified according to texture and color information for collision-free navigation.

The representation of the environment via height-maps has certain limitations, and prevents the exploitation of the robot's capabilities in complex environments. In all works mentioned above the obstacles are either approximated using certain assumptions and previously known information or simply represented by a grid-based map. This results in numerous distance calculations, which complicates online collision checking if the robot moves relatively fast. Therefore they only avoid collisions by checking safe footstep positions and without taking the whole body motion into account.

A different approach was taken by Buschmann et al. [16], who used a stereo camera to test proposed trajectories for viability in unknown environments checking collisions in 2D. The method allowed fast movements and collision-free trajectories in non-static environments, but prevented stepping over obstacles.

In the recent DARPA Robotics Challenge[2], different robots tried to solve tasks inspired by a nuclear plant disaster. The completion of the tasks, however, relied strongly on teleoperation. To the author's knowledge, none of the participating teams performed completely autonomous environment modeling and obstacle avoidance.

[2]The DARPA Robotics Challenge (DRC): `http://www.theroboticschallenge.org/`



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

Fig. 1. Photo and kinematic structure of the humanoid robot *Lola* with the RGB-D sensor mounted on top. The right side shows the joint distribution and the used world coordinate system.

In contrast to the methods presented above, we propose a strategy for the efficient detection and representation of obstacles which greatly reduces the cost of collision checks and doesn't depend on any previous information or identifiers. Instead of using a reduced, height-map representation of the environment we perform direct 3D point cloud processing, which is computationally more challenging but allows for more complex scenarios. By using the same representation for the environment and the robot, we are able to perform autonomous whole-body collision avoidance online during walking. A full 3D representation of the environment allows for complex obstacle avoidance 3D motions like stepping over or swinging sideways (see [4]). Additionally, this representation does not assume a static environment and our robot is able to step over previously unknown moving obstacles.

In a related subject, different approaches have been taken for autonomous navigation of vehicles, where conditions and collision avoidance strategies greatly differ from the ones considered here. A thorough review of the subject is outside the scope of this paper and most results belong to private companies and are not openly published. Still, some examples show the detection and tracking of dynamic objects (see [17], [18]), but are usually based on learned features of the environment and determined kinds of objects (e.g. cars or pedestrians), which are approximated with simple bounding boxes. These methods are not adequate for our application due to the complex motions of humanoid robots.

III. Humanoid Robot Lola

The presented vision system has been tested on the robot *Lola*. It is an electrically actuated humanoid robot which weights approximately 60 kg and is 180 cm tall. *Lola* has 24 position controlled joints. The left hand side of Fig.1 shows

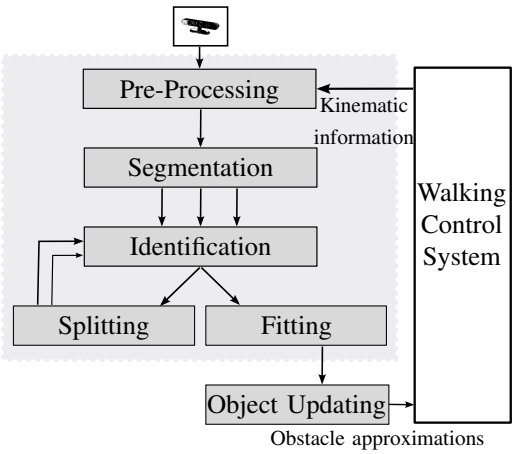


Fig. 2. Structure of our Vision System. Operations inside the blue area are performed independently for every frame.

a photo of *Lola* and the right hand side of Fig.1 gives a detailed view of the kinematic configuration. A more precise introduction to the hardware can be found in [19]. Located on *Lola*'s head, our RGB-D sensor Asus Xtion PRO LIVE[3] can be seen. It was calibrated using a simplified method based on the works of [20], [21] (a detailed description is outside the scope of this paper). The vision system runs on a parallel on–board computer with an Intel Core i7-4770S@3.1 GHz (4x) processor and 8GB RAM and communicates with our control computer via Ethernet using UDP.

IV. Vision System

Our Vision System is based mainly on the open source Point Cloud Library (PCL[4]). For further information about its basic principles and algorithms, see [22], [23].

As explained before, the vision system runs parallel to the main control on our robot's on-board computer. It constantly receives updated kinematic data from the walking control system and sends back updated obstacle information. In Fig. 2 the main structure is shown.

Our program is not based on any kind of environment map. Instead, at each cycle, it takes the latest frame from the sensor and the kinematic information as input to create a homogeneous point cloud (PC). This PC is then segmented to extract objects from the environment. Each of these objects (or sub-PCs) is approximated by one or more swept-sphere-volumes (SSVs) (in order to have an accurate approximation of complex objects, their corresponding PCs are split into more sub-PCs). These geometries, previously introduced for self-collision avoidance [24], have the main advantage of making distance calculations simple and efficient. By using the same representation for the robot and the environment we can integrate both self-collision and obstacle avoidance in one module running in real-time. It enables a more complex representation of the environment than height maps which is used for real-time 3D collision avoidance (see [4]). The environment is approximated using point- and line- SSVs, which can be seen in Fig.3. In this paper we don't take into account planar surfaces as ramps or stairs where the robot could step on, but avoid collision with every detected object. We are already working on an extension to include these cases as well and perform more complex navigation strategies.

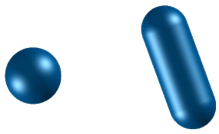


Fig. 3. Geometries used by the vision system for the approximation of objects: point-SSVs (left) and line-SSVs (right)

The *Approximation* procedure consists of the *Identification*, *Splitting* and *Fitting* steps and creates a list of SSVs which is matched to a pool of previously detected objects. In the following, each of these steps is explained in detail.

A. Pre-Processing

On the walking control system we run a service that continuously updates the kinematics transformation from the camera to a reference *world coordinate system* (W), fixed to the ground at the starting position of the robot. By transforming the incoming point cloud (PC) from the camera coordinate system into W, we can successfully track objects despite the robot's motion. It is worth noting that this is not intended as a SLAM strategy, as we don't update a map of the environment but always analyze each incoming PC separately. W will be affected by inaccuracies in the robot's motion (e.g. sliding on the ground) and will change slowly with time, but that is not relevant for our application as W is only used to match objects between consecutive frames (see IV-F). As the point cloud is obtained with the on-board camera, updated obstacle locations are always correct with respect to the robot, regardless of W.

We also apply a simple voxelization of the PC into a 3D grid with 1cm size to have a homogeneous PC in the following steps. Note also that this is the only process that depends on our particular robot and sensor, and it can be easily adapted to other systems just by keeping a similar coordinate system and density.

B. Segmentation

Several standard segmentation techniques were tested with different scenarios and obstacles. In our implementation, planes are removed using PCL's *Sample Consensus Segmentation* (based on RANSAC) and remaining PCs are clustered using PCL's *Euclidean Cluster Extraction*[5]. This proved to be fast enough (see sec.VI) for our purposes and robust against

[3]See `https://www.asus.com/de/Multimedia/Xtion_PRO_LIVE/`

[4]See `http://pointclouds.org/`

[5]See `http://docs.pointclouds.org/`

different scenarios. Other algorithms tested (i.e. *Depth-map-based*, *Normal-based*, *Global-plane*) were either slower or failed to extract some of the objects tested.

C. Identification

Obstacles are approximated with one or more point- or line-SSVs in order to ensure efficiency in distance calculations [24]. This approach does not create the best possible approximation but one that runs in real-time during walking while being safe and detailed enough.

In the *Identification* step, each incoming PC is analyzed to determine if it will be approximated by a single SSV (and in that case, by which), or if and where it is necessary to *split* it into smaller PCs which will be analyzed in turn. Naturally, the splitting process usually generates better approximations but at an additional computational cost. In our procedure, a PC is split unless it meets one of the following conditions (cond.):

1) It "resembles" a perfect point-SSV.
2) It "resembles" a perfect line-SSV.
3) The distance to the robot is bigger than a certain threshold d_{min}.
4) Its volume is smaller than a certain threshold V_{min}.
5) It is the result of a certain number of "splitting steps" n_{steps}.

These conditions have the purpose of using more detailed representations of obstacles only when they are needed. They keep the necessary number of SSVs to represent the environment at a minimum, thus reducing the computational cost of the step-planner and collision avoidance modules. This hierarchical strategy only improves the approximation of objects that are relevant to the robot's navigation. Objects that resemble ideal geometries don't need a more detailed approximation. In fact, in these cases the splitting operation can be counterproductive, as shown in Fig.4. Objects that are too far away don't have a big influence in the planning and with objects that are too small compared to our robot a detailed approximation is irrelevant. Cond. 5 limits the number of splitting steps (therefore the total cycle time).

If the PC meets one of these conditions it is sent to the *Fitting* step to be approximated by a line-SSV in case of cond. 2 or by a point-SSV in every other case, because of the smaller computational cost.

For cond. 1 and 2 we use invariants similar to those used by Ditrich et al. to fit PCs using prisms or superellipsoids [25], [26]. Our heuristic approach is based on the inertia matrix of the incoming PC $I^{(PC_i)}$. Let I_{max}, I_{mid}, I_{min} be the eigenvalues of $I^{(PC_i)} \mid I_{max} \geq I_{mid} \geq I_{min}$ (principal moments of inertia). The quotients $\xi_1 = \frac{I_{min}}{I_{max}}$ and $\xi_2 = \frac{I_{mid}}{I_{max}}$ are invariants, as they don't depend on the scale of PC_i. If we consider ideal PCs of our geometric approximations:

- A point-SSV has $\xi_1 = \xi_2 = 1$
- A line-SSV has $\xi_1 < \xi_2 = 1$

In our application, however, PCs are noisy, hollow and incomplete as we have only one sensor. Our final cond. 1 and 2 differ from the ideal ones as they were established empirically by testing several objects and adjusting the range values of ξ_1 and ξ_2 (see sec.VI).

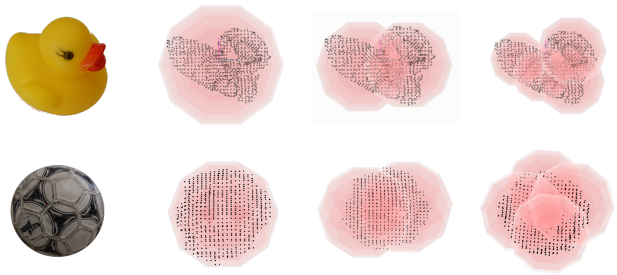


Fig. 4. Approximation of point clouds. From left to right: a picture of the object, the point cloud (black) and SSV approximation (pink) without splitting, after 1 splitting step, after 2 splitting steps. In the second example, it can be seen that the splitting operation can result in worse approximations for some objects.

The distance of the PC to the robot in cond. 3 is estimated as the projected ground distance between the PC's centroid and the origin of the robot's local coordinate system, located at the "toe" of the current stance foot. The PC's volume in cond. 4 is estimated as the volume of the PC's bounding box in W. Our final values for all condition parameters are shown in sec.VI.

D. Splitting

We split PC_i using a plane ψ that goes through the PC's centroid (or center of mass) $\overline{p}^{(PC_i)}$. Thus, each of the resulting sub-PCs PC_{i_1} and PC_{i_2} will have approximately the same number of points. In order to choose ψ it is useful to look at the following example:

Let PC_s be a PC point symmetric to $\overline{p}$, i.e. $\forall$ point $p_n \in PC_s$ $\exists$ $p_m \in PC_s \mid$

$$\left(p_n - \overline{p}^{(PC_s)}\right) = -\left(p_m - \overline{p}^{(PC_s)}\right) \tag{1}$$

We define the coordinate system $S(x,y,z)$ coincident with the eigenvectors of $I^{(PC_s)}$ (principal axes of inertia) and its origin in $\overline{p}^{(PC_s)}$. If we choose ψ as the $x-y$ plane we can split PC_s into two *identical* sub-PCs PC_{s_1} and PC_{s_2} (if there were any points $\in \psi$ we don't take them into account). PC_{s_1} and PC_{s_2} are symmetrical with respect to the planes $x-z$ and $y-z$, so their principal axes of inertia are parallel to the axes of $S(x,y,z)$. Applying Steiner's Parallel Axis Theorem,

$$I_{ij}^{(PC_s)} = 2\left[I_{ij}^{(PC_{s_1})} + m^{(PC_{s_1})}\left(\sum_k a_k^2 \delta_{ij} - a_i a_j\right)\right] \tag{2}$$

where

- $I_{ij}^{(PC_x)}$ is the point cloud's PC_x inertia tensor
- $i,j,k \in \{1,2,3\}$
- $m^{(PC_{s_1})}$ is the number of points (mass) of PC_{s_1}
- δ_{ij} is the Kronecker-delta
- The vector $a = (a_1,a_2,a_3)^\intercal = \overline{p}^{(PC_s)} - \overline{p}^{(PC_{s_1})}$

Because of our choice of ψ, $a_1 = a_2 = 0$.

$$\therefore I_{ij}^{(PC_{s_1})} = \begin{cases} \frac{1}{2}\left(I_{ij}^{(PC_s)} - m^{(PC_s)} a_z^2\right) & : i = j = 1,2 \\ \frac{1}{2}\left(I_{ij}^{(PC_s)}\right) & : i = j = 3 \\ 0 & : i \neq j \end{cases} \quad (3)$$

If we suppose that PC_s doesn't resemble a point- or line-SSV and the corresponding axes to I_{max}, I_{mid}, I_{min} are the same for PC_{s_1} as for PC_s, we obtain the relationships shown in Table I. Out of these relationships we can conclude that, in this example, splitting a PC through a plane perpendicular to the axis corresponding to I_{min} or I_{mid} results in sub-PCs having a closer resemblance to point-SSVs or line-SSVs respectively. That means, these splitting strategies would favor reaching cond. 1 or 2 respectively. Although PCs are rarely symmetrical, experiments with several objects and scenarios showed a similar trend. Regardless of the chosen strategy, a safe approximation is assured later by the *Fitting* step.

TABLE I
CHOOSING THE SPLITTING PLANE

z corresponds to I_{min}	z corresponds to I_{mid}
$\xi_1^{(PC_{s_1})} > \xi_1^{(PC_s)}$	$\xi_1^{(PC_{s_1})} < \xi_1^{(PC_s)}$
$\xi_2^{(PC_{s_1})} < \xi_2^{(PC_s)}$	$\xi_2^{(PC_{s_1})} > \xi_2^{(PC_s)}$

E. Fitting

In this step we fit the corresponding SSV to each incoming point cloud PC_i based on its inertial parameters. We cover the potential collision region by including all points in the SSV. This is especially relevant as it guarantees a safe approximation, regardless of the identification and splitting parameters.

1) Fitting a point-SSV with center o and radius r:

$$o = \overline{p}^{(PC_i)} \quad (4)$$

$$r = max\left(\left|p - \overline{p}^{(PC_i)}\right|\right) \quad (5)$$

2) Fitting a line-SSV with centers o_1 and o_2 and radius r:

$$o_1 = \overline{p}^{(PC_i)} + \lambda_1 \mathbf{w} \quad (6)$$

$$o_2 = \overline{p}^{(PC_i)} + \lambda_2 \mathbf{w} \quad (7)$$

$$r = max\left(\left|\left(p - \overline{p}^{(PC_i)}\right) \times \mathbf{w}\right|\right) \quad (8)$$

where

- $\mathbf{w}$ is the directional vector of the principal axis of inertia corresponding to I_{min}
- $\lambda_1 = max\left(\left(p - \overline{p}^{(PC_i)}\right) \cdot \mathbf{w}\right)$
- $\lambda_2 = min\left(\left(p - \overline{p}^{(PC_i)}\right) \cdot \mathbf{w}\right)$
- *max* and *min* functions are evaluated $\forall point\, p \in PC_i$

F. Obstacle Updating

As stated before, objects are tracked and filtered before being sent to the motion planning system. We create a local pool of objects that not only helps reducing noise and inaccuracies in the objects' location, but also provides coherent information to the step-planner by keeping track of objects between frames. The pool (set *old*) comprises of the following sub-sets:

- *Phantom*: Each detected object is first defined as phantom. If a phantom object fails to be detected in a future frame (before becoming real), it is removed from the sub-set as it could be caused by noise.
- *Real*: If a phantom object has been detected more than a certain number of cycles noc_{real} in a row, it becomes real. These are sent to the motion planning system, becoming the robot's environment model. If a real object hasn't been seen for a number of cycles noc_{die} it is removed from the whole set.

At each cycle, the detected set of objects *new* is matched against the *old* set. Each object in the new set $new[i]$ is recognized as an old object $old[k]$ if the distance between them is minimal and smaller than a certain threshold ε_{max}:

$$\forall new[i] \in new, \;\; match(i) = k \mid \begin{cases} dist(i,k) = min(dist(i,j)) & \forall old[j] \in old \\ dist(i,k) < \varepsilon_{max} \end{cases} \quad (9)$$

where $dist(i,j) = \left|\overline{p}^{(new[i])} - \overline{p}^{(old[j])}\right|$. This matching algorithm results in an effective tracking of moving objects in the scene. Note that, as W is "fixed" to the ground, objects with a speed smaller than

$$v_{obj,max} \simeq \frac{\varepsilon_{max}}{max\left(t_{frame-acquisition}, t_{frame-processing}\right)} \quad (10)$$

will be successfully tracked, independently of the robot's motion.

The algorithm may, in theory, match two different objects (in *new*) against the same object (in *old*), if both their distances to the *old* object are smaller than ε_{max}. This, however, is rare in practice as ε_{max} is chosen relatively small respect to the object's dimensions and they would have to overlap.

After matching, old objects are updated to the new approximation by low-pass filtering their coordinates. The new approximation is displaced by the vector $r_{disp} = \rho\left(\overline{p}^{(old[k])} - \overline{p}^{(new[i])}\right)$, with $0 \leq \rho \leq 1$. This results in an improved estimation of the object's location and better results from the real-time planning system. As a consequence, $v_{obj,max}$ gets multiplied by $(1-\rho)$ (by varying ρ, $v_{obj,max}$ can be improved at the cost of lower precision). The main objective of this post-processing step is to reduce noise and compensate various sources of errors to get more stable and accurate approximations. Thus, $v_{obj,max}$ is limited in our implementation (see VI). In the future, we plan to integrate a state estimator to filter both the object's position and velocity and handle faster moving objects.

Due to the limited bandwidth, every 3 cycles the *Real* subset of objects is sent to the walking control system, updating its environment model.

V. Integration in Walking Controller

The walking control system of the robot *Lola* follows a hierarchical approach, which is outlined in Fig. 5. It is presented in more detail in [27]. Once before each step cycle a planning unit is called. It generates a desired walking pattern based on joystick or GUI input, taking into account an advanced three-mass-model. The desired walking pattern is used as set-points for feedback control which is called in a cycle time of $\Delta t = 2\,\text{ms}$. The feedback control stabilizes the robot using a hybrid force/position control approach, which modifies the ideal walking pattern. Joint trajectories are calculated based on the modified walking pattern. It solves the redundancy problem while tracking the robot's center of mass trajectory and while minimizing an optimization criteria dedicated to self-collision-avoidance, joint-limit-avoidance and angular-momentum-minimization [24], [28]. The key idea for collision avoidance in real-time is the use of a collision model based on SSVs representing the robot as well as the environment throughout the walking controller. The integration of collision avoidance in the walking controller follows the hierarchical approach: A step-planner based on an advanced A*–Algorithm reacts to high-level user commands like desired velocity and direction within less than a step. Instead of investigating only the footholds, an articulated SSV-based 3D approximation of the lower leg and the foot is considered to find feasible and optimal footstep locations. Additionally, it provides an initial solution for the swing-foot movement. It is denoted as *Step planning and trajectory adaptation* in Fig.5. Based on the initial solution the *Trajectory optimization* unit produces optimized collision-free trajectories using a full-body collision model of the robot. In the feedback control layer the *Trajectory optimization* takes into account collisions as well as self-collisions by optimizing all the foot's degrees of freedom. Details are presented in [4], [5].

VI. Implementation & Results

In this section we present the details of our implementation and some first experimental results on *Lola*. In future work, after extending our system to include the detection and handling of walkable surfaces, we plan to evaluate the performance of the algorithm more thoroughly: against ideal scenarios (using artificial point clouds), as well as in more complex and varied experiments.

A. Parameter Definitions

The *approximation* parameters shown in Table II roughly define the size of the approximation geometries and have been adjusted based on *Lola's* size and speed. They permit adjusting the desired level of detail of the approximation geometries (at worst, they could be chosen for a conservative approximation of obstacles, which is safer for the robot's navigation). Additionally, the geometric invariants ξ_1 and ξ_2

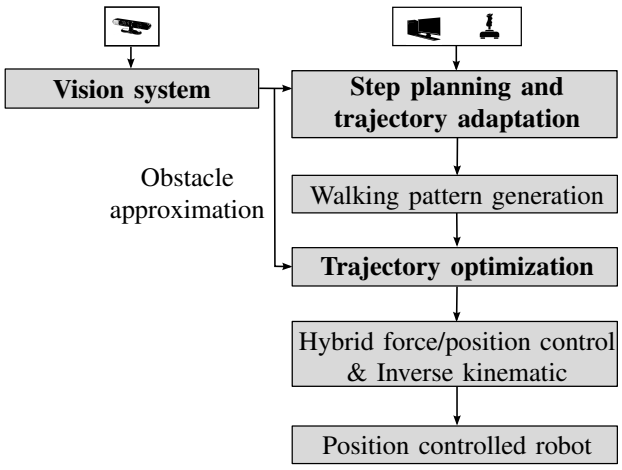


Fig. 5. Integration of Vision System in *Lola*'s real-time walking control system.

were analized for a series of objects resembling point- and line-SSVs. They differ from the ideal values due to the point clouds being hollow, incomplete and noisy. Based on results of these PCs, we defined:

- cond.1 = $(\xi_1 > 0.8) \cap (\xi_2 > 0.1)$
- cond.2 = $(\xi_2 < 0.25)$

For our *splitting* strategy, we choose the splitting plane ψ perpendicular to the principal axis of inertia corresponding to I_{mid}. Note that a safe approximation of any object is guaranteed by our *fitting* strategy, while these criteria only affect how well the approximations fit certain objects.
The *obstacle updating* parameters (also shown in Table II) define the post-processing filtering and have been chosen based on the sensor's noise. Another limitation of the sensor for this application is its relatively small field of view. In our configuration, we are only able to detect objects that are between approximately 0.8 and 3.5m in front of the robot. As the robot gets closer, objects start to leave the field of view and appear gradually smaller before finally disappearing. To prevent their removal from the robot's environment model, we stop updating them once they are less than 1m away. This configuration is used throughout all experiments.

TABLE II
Implementation Parameters

Approximation			*Obstacle Updating*			
d_{min}	V_{min}	n_{steps}	noc_{real}	noc_{die}	ε_{max}	ρ
1.5m	0.003m^3	2	5	10	0.05m	0.5

B. Experimental Results

We tested our system on our humanoid robot *Lola* by giving simple walking commands to the robot in a scene with objects of different shape, color, size and texture in front of it. The objective was to move in the desired direction avoiding collision with the objects in the scene, some of which were moving. Two experiments are described in the following (see

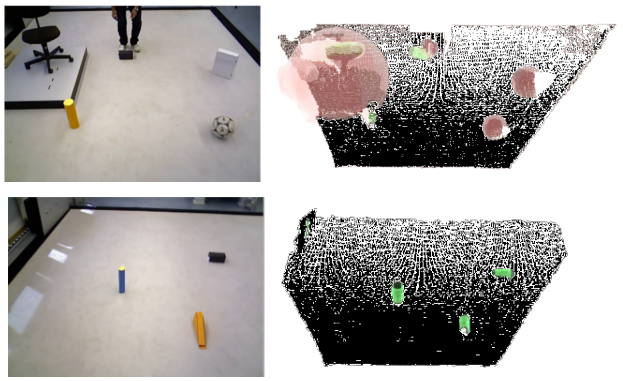


Fig. 6. Point clouds and object approximations obtained during walking in first (up) and second (bottom) experiments respectively (the RGB view from the robot is shown as reference on the left). Point-SSVs are shown in orange and line-SSVs in green

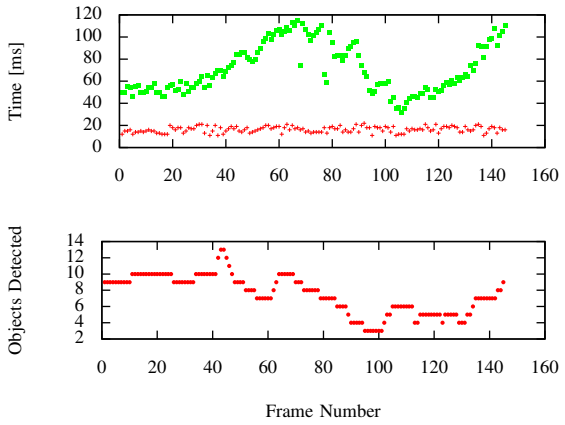


Fig. 7. First experiment. Top: calculation time of the pre-processing step (red) and total time performance of the vision system (green). Bottom: number of objects in the *real* set.

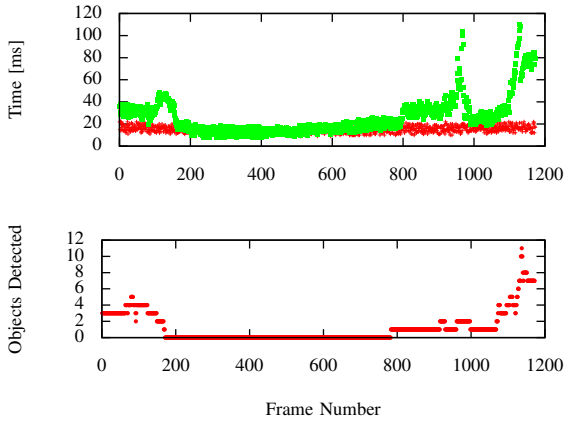


Fig. 8. Second experiment. Top: calculation time of the pre-processing step (red) and total time performance of the vision system (green). Bottom: number of objects in the *real* set.

also accompanying video), where *Lola* manages to step over obstacles that had been moving previously:

In the first experiment, we place several big objects on the sides and in front of *Lola*, including a chair on top of a platform to show different types of object approximations. After a simple command to start walking forward, *Lola* continuously adapts its trajectory to advance in the desired direction while an object is being removed from the way and a person places a different object in its path. All objects (including the person) are recognized and the approximations improve as they get nearer.

In the second experiment, a better performance of the system is shown by using smaller objects and removing the points outside the lab area. The walking command is faster (0.4m/s) and includes walking forward, turning around and walking forward again without stopping. While *Lola* advances, the objects are replaced by a chair and a new moving obstacle. *Lola* succeeds by taking steps to the side and stepping over both a static object and the obstacle that was moving.

In Fig.6 a reference view from each experiment is shown, together with the corresponding PC and SSV approximations.

In Figs. 7, 8 we show the vision system's performance during walking for both experiments. Note that the performance of the *Pre-Processing* step is approximately constant (as expected), while the total cycle time is strongly correlated with the number of objects in the scene. As predicted, the program performs best during the second experiment, due to the small number of SSVs needed to approximate the scene. While *Lola* is turning, no object is detected (due to the removal of the points outside the lab area) and the cycle time drops. In the end, although not visible in the video, the system recognizes the control desk and computers, therefore the number of SSVs (and the cycle time) increases. During both experiments, the total cycle time is always lower than 120ms and the maximum trackable object speed $v_{obj,max}$ varies between 0.2 and 0.75m/s.

The video of the described experiments is attached to the present paper. A scene of the second experiment, where *Lola* steps over an object, is shown in Fig.9.

VII. Conclusion & Future Work

An efficient vision system that identifies and approximates objects was presented. Its performance was tested using the incoming point cloud from a low-cost RGB-D sensor and integrating it into our humanoid robot *Lola* to perform fast and autonomous whole-body collision avoidance. It is robust against different kinds of obstacles and, depending on the number of objects, it can process each frame in 20-120ms in our experiments. That is sufficient to track moving objects and perform complex obstacle avoidance motions (stepping over or walking sideways) in real-time.

In the near future, we plan to extend the system to more dynamic scenarios (by estimating the objects' velocities) and add surface detection capabilities to also recognize walkable areas (e.g. stairs, ramps).

Our program is completely modular, making it very flexible

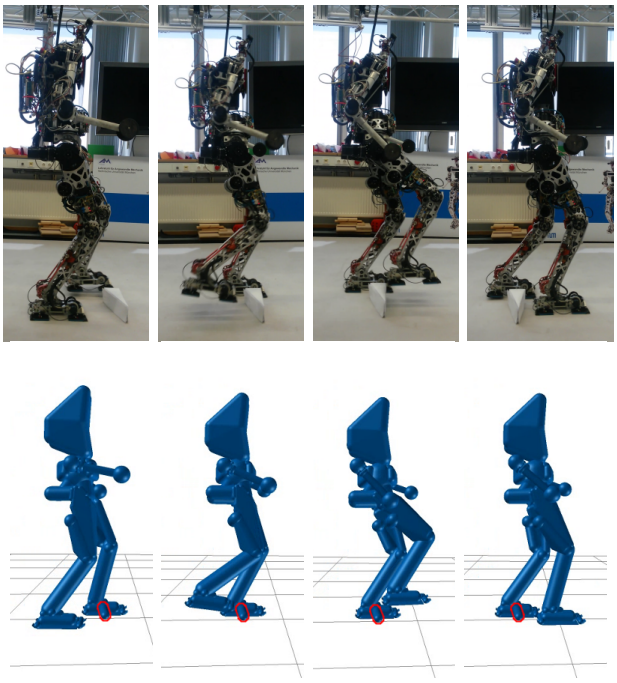


Fig. 9. Top: *Lola* steps over an obstacle with a set walking speed of 0.4m/s. Bottom: collision model of the robot (blue) and the obstacle (blue with red border)

and adaptable to other systems and applications. It can be found at `http://github.com/am-lola/lepp3`.

Acknowledgment

This work is supported by the Deutscher Akademischer Austauschdienst and the Deutsche Forschungsgemeinschaft (project BU 2736/1-1). Special thanks go to our students Fabian Bräu, Marko Lalić, Sahand Yousefpour and Wolfgang Wiedemeyer for their help in the implementation of these ideas.

References

[1] C. Wöhler, *3D Computer Vision: Efficient Methods and Applications*. Springer London, 2013.

[2] Z. Zhang, "Microsoft Kinect Sensor and Its Effect," *IEEE Multimedia*, vol. 19, no. 2, pp. 4–10, feb 2012.

[3] R. A. El-laithy, J. Huang, and M. Yeh, "Study on the use of Microsoft Kinect for robotics applications," in *IEEE/ION Position, Location and Navigation Symposium*, 2012, pp. 1280–1288.

[4] A.-C. Hildebrandt, R. Wittmann, D. Wahrmann, A. Ewald, and T. Buschmann, "Real-time 3D collision avoidance for biped robots," in *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 4184–4190.

[5] A.-C. Hildebrandt, D. Wahrmann, R. Wittmann, D. Rixen, and T. Buschmann, "Real-Time Pattern Generation Among Obstacles for Biped Robots," in *IEEE International Conference on Intelligent Robots and Systems*, 2015.

[6] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," *IEEE International Conference on Robotics and Automation*, vol. 2, 1985.

[7] S. Kagami and K. Nishiwaki, "Vision-based 2.5 D terrain modeling for humanoid locomotion," *IEEE International Conference on Robotics and Automation*, pp. 2141–2146, 2003.

[8] R. Cupec and G. Schmidt, "An approach to environment modelling for biped walking robots," in *IEEE International Conference on Intelligent Robots and Systems*, 2005, pp. 424–429.

[9] 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.

[10] J. S. Gutmann, M. Fukuchi, and M. Fujita, "3D Perception and Environment Map Generation for Humanoid Robot Navigation," *The International Journal of Robotics Research*, vol. 27, no. 10, pp. 1117–1134, 2008.

[11] J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins, and T. Kanade, "Footstep planning for the Honda ASIMO humanoid," *IEEE International Conference on Robotics and Automation*, vol. 2005, pp. 629–634, 2005.

[12] J. Chestnutt, Y. Takaoka, K. Suga, K. Nishiwaki, J. Kuffner, and S. Kagami, "Biped navigation in rough environments using on-board sensing," in *IEEE International Conference on Intelligent Robots and Systems*, 2009, pp. 3543–3548.

[13] 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*, vol. 31, no. 11, pp. 1251–1262, 2012.

[14] 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.

[15] D. Maier, C. Stachniss, and M. Bennewitz, "Vision-Based Humanoid Navigation Using Self-Supervized Obstacle Detection," *International Journal of Humanoid Robotics*, vol. 10, p. 1350016, 2013.

[16] T. Buschmann, S. Lohmeier, and H. Ulbrich, "Entwurf und Regelung des Humanoiden Laufroboters Lola," *Automatisierungstechnik*, vol. 58, no. 11, pp. 613–621, 2010.

[17] 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*, vol. 29, pp. 1707–1725, 2010.

[18] M. Bajracharya, J. Ma, A. Howard, and L. Matthies, "Real-time 3D Stereo Mapping in Complex Dynamic Environments," in *International Conference on Robotics and Automation-Semantic Mapping, Perception, and Exploration Workshop*, 2012.

[19] S. Lohmeier, T. Buschmann, and H. Ulbrich, "System design and control of anthropomorphic walking robot LOLA," *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 6, pp. 658–666, 2009.

[20] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–54, 2012.

[21] M. Jalobeanu, G. Shirakyan, G. Parent, H. Kikkeri, B. Peasley, and A. Feniello, "Reliable Kinect-based Navigation in Large Indoor Environments," in *IEEE International Conference on Robotics and Automation*, 2015.

[22] 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.

[23] R. B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," Ph.D. dissertation, 2009.

[24] 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.

[25] F. Ditrich, H. Suesse, and K. Voss, "A new efficient algorithm for fitting rectangular boxes and cubes in 3D," in *IEEE International Conference on Image Processing*, vol. 18, 2005, pp. 1–4.

[26] F. Ditrich and H. Suesse, "Robust Fitting of 3D Objects by Affinely Transformed Superellipsoids Using Normalization," *Computer Analysis of Images and Patterns*, pp. 490–497, 2007.

[27] T. Buschmann, V. Favot, S. Lohmeier, M. Schwienbacher, and H. Ulbrich, "Experiments in Fast Biped Walking," in *IEEE International Conference on Mechatronics*, 2011, pp. 863–868.

[28] M. Schwienbacher, T. Buschmann, and H. Ulbrich, "Vertical Angular Momentum Minimization for Biped Robots With Kinematically Redundant Joints," in *International Congress of Theoretical and Applied Mechanics*, no. August, 2012, pp. 8–9.