Analysis of flat terrain for the Atlas robot

Maarten de Waard*, Maarten Inja*, and Arnoud Visser*
*Intelligent Systems Laboratory Amsterdam
Universiteit van Amsterdam

***Abstract*—This paper gives a description of an approach to analyze the sensor information of the surroundings to select places where the foot of a humanoid can be placed. This will allow apply such robot in a rescue scenario, as foreseen in the DARPA Robotics Challenge, where a robot is forced to traverse difficult terrain.**

I. Introduction

On October 24th 2012, the DARPA Robotics Challenge (DRC) kicked off[1]. To quote their website:

> "The primary technical goal of the DRC is to develop ground robots capable of executing complex tasks in dangerous, degraded, human-engineered environments. Competitors in the DRC are expected to focus on robots that can use standard tools and equipment commonly available in human environments, ranging from hand tools to vehicles, with an emphasis on adaptability to tools with diverse specifications."

The robot that is used in this challenge is the Atlas, a bipedal human sized robot, shown in figure 1. The Atlas robot is a continuation of the anthropomorphic robot developed in the Petman project. The Petman project was intended to reproduce the human movements. The Atlas robot is intended to be able to traverse difficult terrain. The Atlas has 28 degrees of freedom (DoF); the skeleton is shown in figure 2. At this moment the robot is only available in the DRC Simulator based on the Gazebo environment (see Fig. 3). The first real prototypes will be delivered later this year. In simulation the Atlas is equipped with a laser scanner and two cameras located in the head.

Figure 1. The Atlas is a humanoid robot will be tailored for rescue operations (Courtesy Boston Dynamics).

[1]http://www.darpa.mil/Our_Work/TTO/Programs/DARPA_Robotics_Challenge.aspx

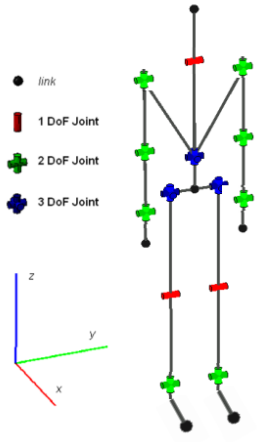

Figure 2. Skeleton of the Atlas showing the degrees of freedom.

Three tasks are part of the virtual DARPA challenge:

- Climb into a utility vehicle, drive along a roadway at a speed no greater than 16 kph (10 mph), and climb out of the utility vehicle.
- Walk across progressively more difficult terrain, for example, progressing from parking lot to short grass to tall grass to tall grass on slope to ditch to rock field. In the earlier terrain, the default balancing and walking behaviors of Atlas will suffice. In the later terrain, DARPA expects perception and footstep planning will

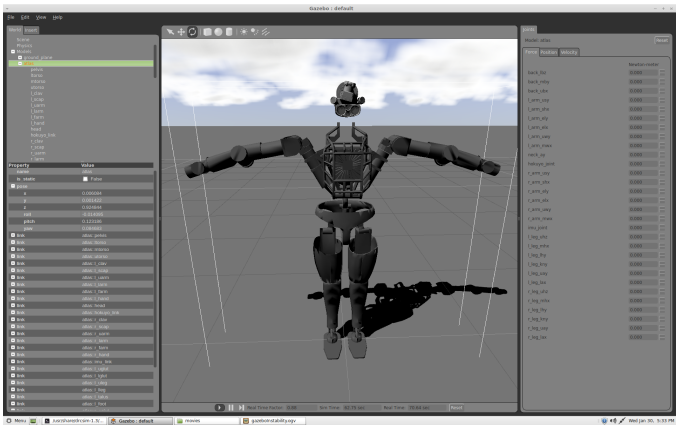

Figure 3. Screenshot of the GUI of Gazebo showing the rendering of the Atlas URDF model.

978-1-4673-6315-0/13/$31.00 ©2013 IEEE

be needed.

- Connect hose to spigot. This is purely a manipulation task, that is, the robot starts with everything within reach and so does not need to travel to the work site.

For this paper we will try to tackle the second task in de virtual DARPA challenge. This virtual challenge entails using the Gazebo[2] simulator together with the Robot Operating System[3] (ROS) to make a model of the robot perform the task. In the Technical Guide of the Virtual Robotic Challenge [1] this task is described in more detail. In Fig 4 the arena for the walking test is displayed. In the front is the starting pen, the final gate is in the hills at the back. In between those gates the robot has to walk across flat pavement, cross a mud pit, climb a gentle incline and traverse a rubble pile. To complete such challenge, a robot has to interpret the terrain ahead with its sensors, aggregate the measurements into a world model and perform footstep planning based on this model. This is the subject of this paper.

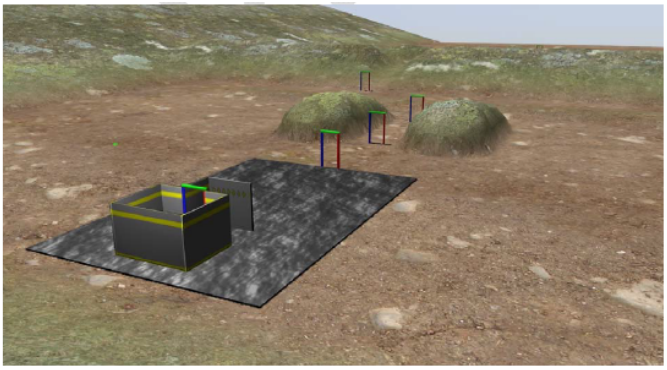

Figure 4. Overview of the arena for the walking test (Courtesy E. Krotkov *et al.*[1]).

This task was split into two parts. The first part consists of making a model of the surface in front of the robot and selecting candidate surfaces for foot placement. The second part consists of computing a path to reach this location with the foot in a safe way. The candidate surfaces will serve as input for the walking engine, where it first performs a stable leg lifting motion, and then put the foot down in the desired location while keeping the robot balanced.

This paper is organized as follows: in section 2 the related work that has been done on both subjects will be discussed; in sections 3 the footstep planning will be described, followed by results, conclusion and future work.

II. Related Work

A. Footstep Planning

There are several approaches to the planning of footsteps.

To prefer a straight surface over a surface with slopes one can use A* and include the slope of the ground plane into the heuristic function [2]. This works well for their application (crossing an uneven hill). In their algorithm the terrain is categorized in four categories (flat ground, tilted ground, stairs, holes), none of which deals with obstacles which could be encountered at the rubble pile. The A* path planning is divided into two parts, first a trajectory is planned from the current position to the goal, then the actual footsteps are planned to follow the trajectory.

By modeling the robots valid (foot) configurations, estimating the bounding box for the shape of the robot, and modeling the environment (obstacles) one can represent the robot and the world in a mathematical context. This representation can be used to determine illegal and legal positions and configurations, or states, for the robot. The states can be seen as nodes in a graph, in which the edges are the transitions from one configuration, or pose, to another. This results in a searchable graph that can be searched with a modified A* algorithm without having to be built entirely (the states do not all have to be calculated) [3].

A single rock can be seen as an obstacle over which the robot could step entirely, but if the terrain consists of many rocks it becomes 'rough' terrain. The difference with the previous methods is that there is no flat surface at all; the robot should attempt to find the best spot for its foot that would minimize falling or slipping risk.

Dealing with rough terrain can successfully be learned through reinforcement learning [4]. The observed terrain is modeled and matched to models, called templates, which are enriched with the positions that experts deem the best position for foot placement.

Ideally, low ceilings should also be dealt with by the robot by either avoiding such an area or by crouching. However, this is considered not part of our objective.

III. Method

A. Theory

Section II-A describes all the necessities we have to implement for complete footstep planning. We focused on the first step by investigating point cloud data and see if we could extract surfaces and make the distinction between those that will support the robots foot and those who do not.

A point cloud is a collection of points, in which a point is defined as an x, y, z value, which might be collected by a sensor, such as a Kinect sensor or laser range scanner.

The Point Cloud library[4] which already is integrated into ROS environment coupled to the DRC Simulator. ROS offers two methods to do plane segmentation: region growing and plane segmentation using RANSAC.

1) Region Growing: Region growing segments points based on their curvature and surface normals, which are both local features based on the nearest neighbors of the points [5][5]

A region is subset of a cloud of points which are classified as belonging to the region, in our case we mean with a 'region' a plane, or surface. So any points that make up a plane should be considered a region. For different purposes one could

[2]http://gazebosim.org/
[3]http://www.ros.org/wiki/
[4]http://www.pointclouds.org/
[5]as implemented by Sergey Ushakov, see http://www.pointclouds.org/blog/trcs/velizhev/.

for example want to find the regions that make up spheres. Segmentation is the process of dividing, or segmenting, the point cloud data in to different subsets, or regions.

The nearest neighbors could picked using several methods, using KD-trees or octrees, but also simply by taking the points in a radius.

It is easiest to consider the point and its nearest neighbors as a surface of which the normal is the vector perpendicular to the surface, and the curvature a scalar value indicating the curvature of the surface.

Region growing starts at the point with the lowest curvature value, this point is the start of the region, it is added to a new set called seeds. The algorithm is as followed:

- For each point in the seeds set, for each neighboring point:
  - Add neighboring point to the region if the angle between this point and the seed point is below the angle threshold θ_{th}
  - Add the neighboring point to seed set if its curvature value is below the curvature threshold c_{th}
  - Remove the seed point from the seed set.
- If the seeds set is empty, then a region has been found.

2) Plane Segmentation Using RANSAC: The second method to plane segmentation in the point cloud library is to match the model of a plane in the point cloud using RANSAC (RANdom SAmpling Consensus)[6].

This method combines fast normal computation by only considering nearby points on 6 integral images. Fast clustering of points with similar local surface normals is accomplished by first defining a voxel grid (a course discretization), followed by merging those grids. The result can be a cluster of points in the same plane but not geometrically connected. Those planes can be separated again by splitting them in a segmentation refinement step. The found planes can be smoothed by RANSAC which removes residual outliers.

The RANSAC algorithm informally goes as followed:

- Randomly select a subset of the point cloud and estimate the free model parameters
- Other data is considered, if a point fits the model a point is added (considered an inlier)
- The model is re-estimated considering all the inliers
- The model is evaluated by estimating the error relative to the model

A model is sufficient if a sufficient amount of points are considered inliers.

3) Plane Evaluation: The planes, or surfaces, that are found using the region growing or plane modeling should be evaluated to a scalar that indicates how much the robot would want to place a foot on that plane.

First the average surface normal vector for a plane is calculated, normalized. Then the Euclidean distance to the example unit vector $[0, 0, 1]$ (which is a vector pointing straight up) is calculated. This scalar should be sufficient for a path planner similar to [2] to prefer flat and straight surfaces over slopes.

Additionally we would like the size of the plane to be considered; planes smaller than the robots feet should be discarded.

B. Implementation

Some choices were made, to enable easy implementation of footstep planning. At first, the point cloud sensor was chosen as the most fitted sensor to find the environmental data that was needed for accurate footstep planning. This was because point cloud sensors are capable of collecting a lot of information about the environment in a small time frame. Using the point cloud sensor also enabled us to use the C++ Point Cloud Library (PCL), which enables a user to easily use many state-of-the-art point cloud processing algorithms [7]. Because of dependencies between the DRC simulator, ROS and PCL, in this study, the 1.5 version of PCL was used.

The current implementation[6] finds planes in the environment of the robot and gives those planes a measure as to how badly the robot should want to step on them. This works in the following three steps. These steps require a robot to have a working point cloud sensor. Alternatively, a rotary laser range scanner can be used to simulate one.

1) Find planes in the point cloud
2) For each plane, find its mean surface normal
3) Evaluate each plane and its surface normal

The following subsections will explain the methods in detail.

1) Finding planes in a point cloud: As mentioned in section III-A, planes can be located in various manners. In version 1.5 of the point cloud library, Region Growing has not been implemented yet. That is why our software solution uses plane segmentation based on Ransac. This is implemented using the PCL provided `SACSegmentation`. The exact parameters for the segmentation differ per goal and sensor.

The planes that are found using this segmentation algorithm are extracted from the cloud using `ExtractIndices`, also provided by PCL. Then the program loops through these planes, and enters the next step.

2) Finding mean surface normals: Normals of the points in a point cloud can be found using the pcl class `NormalEstimationOMP`, which uses a K-nearest neighbor search to find other close points, and estimates a normal vector using those points and, if specified, a camera position for the correct direction.

The mean surface normal is then calculated by simply adding them all and dividing by the number of points.

3) Evaluate each plane and its surface normal: The surface normal of the ideal standing surface is pointing up. That, by definition, means that a surface is placed horizontally, which is good to stand on. For that reason, the surface normal of each found surface is compared with one pointing up (vector

[6]The code is available at https://code.google.com/p/voetlos/

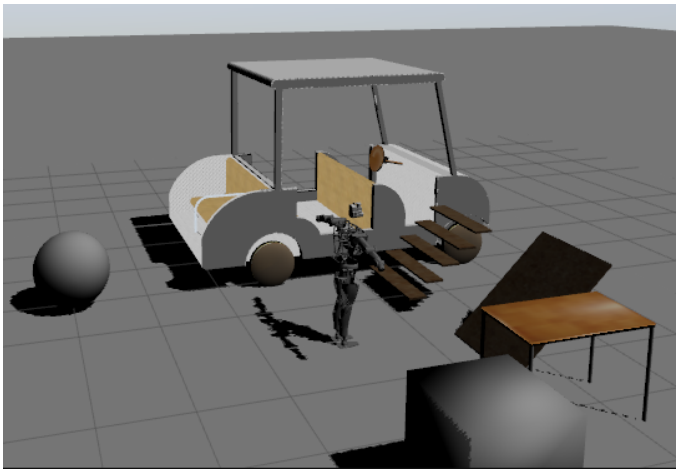

Figure 5. The Gazebo environment that was used to test the algorithm with the points and laser scanners.

$(\ x=0 \quad y=0 \quad z=1\ )^T$). Comparison is calculated with the following equation:

$$\frac{2-\sqrt{(x_{in}-x_{compare})^2+(y_{in}-y_{compare})^2+(z_{in}-z_{compare})^2}}{2}$$

which is always used on unit vectors, assuring the result is a scalar between 0 and 1, 0 meaning completely different, and 1 meaning completely alike.

4) Viewer: The last thing we implemented is a 'viewer', which listens to the ROS topics published by the feature calculation program, and visualizes everything. This viewer represents the found plane segments in colors between red and purple. This color is created by a rgb-value based on the outcome of the comparison. A value of 125 for the red color is always given, so that every point can be seen. The value for blue differs from 0 to 255. For this the measure of equality is used as follows: $equality^5 * 255 = color$. We take the fifth power of the equality, in order to exaggerate the measure in which the ground should be horizontal: a 45° slope is fairly hard to stand on, especially for a robot.

This viewer enables the user to easily see what works and what works not and is a useful combination between listening to ROS topics and the PCL Visualizer class.

IV. Results

A. Footstep Planning

In this section we will show the results of our footstep oriented environment segmentation on various types of point cloud.

1) Gazebo's Points2 Sensor: The most logical sensor to try, in this case, is Gazebo's built in Points2 sensor. This sensor is mounted on the MultiSense-SL head. This sensor was used in a world with some objects, some of which one should want to stand on, others of which one should not. This world can be seen in figure 5.

The resulting image can be seen in figure 6. As can be seen the side of the golf cart is considered bad to stand on, just like the slope. The table and floor are good. Unfortunately planes parallel to the slope are also found in the stairs. For that reason the stairs are also calculated to a red color value. This problem is mainly caused by two reasons:

1) The PCL plane segmentation function has no way of setting a threshold of cloud density. In other words: cloud segments that exist of only five parallel lines of points can be fitted by a plane model, no matter how far these lines are apart from each other. This way, instead of only finding horizontal planes in the stairs, it is also possible to find planes that are fitted through several stairs, and are diagonal on the ground plane. However unwanted this result is, no solution to this problem could be found.
2) Another problem is that the point cloud finds very distorted points. As can be seen, only points in a grid form are found on the ground and on the slope. This is most probably a result of the manner in which the point cloud sensor was implemented in Gazebo.
   As a result of this noise in the point cloud, the processing of especially the stairs could have been worsened. Also finding good segments using any other algorithm than plane segmentation would be harder in the slope, because of the vertical stripes with point clouds, instead a desired dense cloud, like can be seen on the table top.

2) Gazebo's Laser Range Scanner: Because of the gaps in the point cloud, the laser range scanner was put to use. This scanner returns points over one axis, and is capable of being rotated. This results in the capability of collecting point data of the surroundings. When collecting data for 10 seconds, the point cloud of figure 7 is retrieved. The first thing that can be seen is that this point cloud is more accurate than the point cloud retrieved by the Points2 sensor; all planes are straight and the points are divided more evenly. Another thing that can be noticed when looking at this data points, is that the laser range scanner always returns the maximum value, when no points are found. This results in a dome of invalid points around the scanned environment. This will later be referred to as the 'noise dome'.

Using the laser scanner, and an 'assembler' to send the points of the last 10 seconds in one point cloud, the algorithm

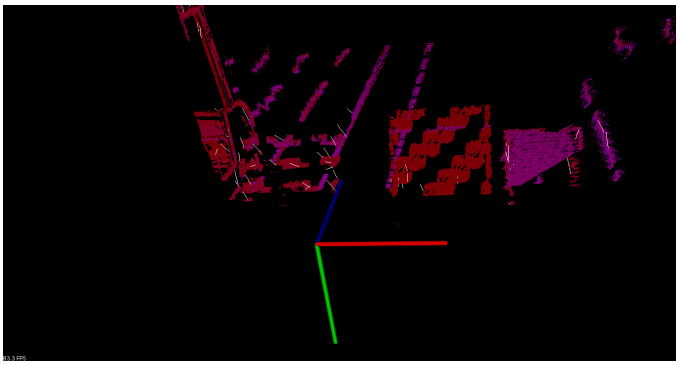

Figure 6. The output image of our algorithm when used with the Points2 sensor in the world of figure 5. A purple tint indicates that that part of the point cloud consists of a surface that is good to stand on. A red tint indicates the opposite. The white arrows are the calculated mean surface normals. From left to right part of the golf cart, stairs, slope and table can be seen. Table is seen as a good solution.

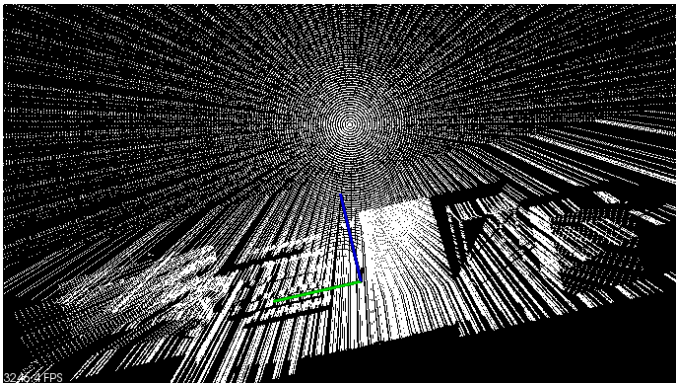

Figure 7. An image of the point cloud retrieved when collecting points from the rotary laser range scanner for 10 seconds, using the Gazebo world of figure 5.

provides the image as seen in figure 8. As can be seen, the table top, each of the stairs and part of the golf cart are correctly detected and calculated to be viable stepping planes. A problem that occurs, however, is that the slope is also detected as a plane with a mean normal that is pointing straight up.

This has the following explanation: The plane that is found, consists of not only the slope, but also numerous points in the 'noise dome' that surrounds the environment. This results in a surface normal that is calculated based on not only the slope normals, but also on the 'dome normal', which is calculated by only one small ring of points and thus faulty.

3) A Point Cloud from the Real World: To prove that the algorithm works, and that is should also work in real-life, in stead of a simulator, we also downloaded a point cloud from the internet, which was created from a real life sensor. The point cloud that was used was the one from a Point Cloud Library tutorial[7]. This resulted in the image that can be seen in figure 9. In this image it can be seen that planes for the ground and walls are found and that the right value is computed for them. The couch forms a more difficult challenge for the algorithm, because the seat and backrest are more curved. Still most of the planes are evaluated correctly.

[7]http://www.pointclouds.org/documentation/tutorials/using_kinfu_large_scale.php

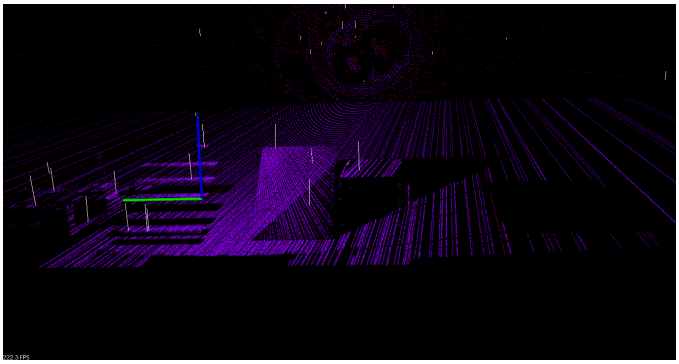

Figure 8. The output of the algorithm when using a rotary laser range scanner in the Gazebo world from figure 5. The same color codes as before count.

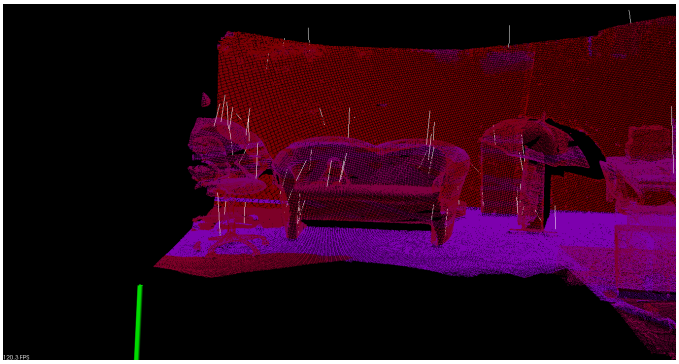

Figure 9. The output of the algorithm when using a point cloud that was retrieved from a real environment.

In the backrest of the couch, some purple planes are found. These are considered 'good to stand on', because they are returned together with the desk, that can be seen on the right side of the image. This is due to the same problem as was treated in the first point in section IV-A1: The points in the couch are considered as being part of the desktop, whereas they really should have been part of a plane in the backrest of the couch.

V. Conclusion

From the results shown in section IV-A it can be concluded that using planar segmentation and normal estimation to find planes in the environment that can be stepped on by the robot should work. This should work in the Gazebo Simulator, if a noiseless point cloud is given and the points that do not have any value are set to NaN or omitted.

Furthermore it is safe to conclude that some improvements can be done in the segmentation method, but that this robust version works in theory.

VI. Future Work

A. Footstep Planning

Improvements can still be made on the footstep planning. These improvements can be put in two categories.

1) Improvements on current footstep planning: At first the current planning algorithm will be treated. The first thing that could be improved is the planar segmentation. One would want to find planes that are uninterrupted. Also planes with a small curve in them should be allowed. Initially this could be solved by (re)implementing region growing, as explained in section III-A1. Another solution would be to implement the segmentation refinement as described in [6].

Ideally other factors for stability of foot placements could be learned: for example when dealing with a rocky area, when hill climbing or walking through a ditch, a stable foot placement might not always be the flattest area, but a V-shaped hole in the ground.

Another improvement in finding good foot placement locations is retrieving the size of the plane and comparing that with the size of the robot's foot. This would for example make the robot take bigger stairs, when available, to minimize the chance of slipping when traversing upwards.

The last improvement that could be made on the current software is ignoring the points that are in the 'noise dome' of the laser range scanner. This can be done by calculating the distance of a point from the robot, and discarding it if it is above a certain threshold. This should improve the precision of the algorithm, because the computed surface normals of the planes that are currently found will then be more accurate.

2) Improvements on the overall footstep planning: To actually be able to use the algorithm, footstep locations should be found in the planes that are currently found suitable for walking on. These locations should be based on where the robot's current position is, where it wants to go and the measure of traversal difficulty for the terrain between these positions. As covered in section II-A, Path planning algorithms like A* are suitable for this goal.

B. Integration with inverse kinematics

The footstep planning should output a trajectory with several Cartesian foot placement coördinates. The path between each foot placement should be covered by the swing of the legs, while the robot stays in balance. This is not trivial for the Atlas robot[8], because it is quite a tall and heavy robot with relatively small feed.

To keep balance the relation between the contact points with the walking surface and the center of mass has to be calculated [9]. ROS contains a humanoid robots kinematics library, but this library was made for the smaller Nao robot. For the Nao robot the pelvis is a good reference point to calculate the center of mass, but for the Atlas robot the hip joints are not fixed which means that there is no guarantee that the pelvis is leveled.

Even when the inverse kinematics problem is solved and the intended swing of the legs can be calculated, a good control of the upper body (including both arms) is needed to balance the momentum during a step. At this moment such high level control of the Atlas robot still has to be developed.

So we can conclude with the observation that the DARPA Robotics Challenge and the robots developed for this challenge will revolutionize the application of robots in rescue situations, but that at the moment there are still enough open issues which require a substantial research effort.

ACKNOWLEDGEMENTS

We would like Norbert Heijne and Sander Nugteren for their effort to solve the inverse kinematics problem of the Atlas robot, to design an animation with resembles a first step and to make the geometrical calculation how the upper body could support the balance of the robot.

REFERENCES

[1] E. Krotkov and J. Manzo, "Virtual robotics challenge technical guide," DISTAR Case 20776, February 2013, Draft Version 5.

[2] J. Bourgeot, N. Cislo, and B. Espiau, "Path-planning and tracking in a 3d complex environment for an anthropomorphic biped robot," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3. IEEE, 2002, pp. 2509–2514.

[3] R. Cupec, I. Aleksi, and G. Schmidt, "Step sequence planning for a biped robot by means of a cylindrical shape model and a high-resolution 2.5 d map," *Robotics and Autonomous Systems*, vol. 59, no. 2, pp. 84–100, 2011.

[4] M. Kalakrishnan, J. Buchli, P. Pastor, and S. Schaal, "Learning locomotion over rough terrain using terrain templates," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 167–172.

[5] T. Rabbani, F. van Den Heuvel, and G. Vosselmann, "Segmentation of point clouds using smoothness constraint," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 36, no. 5, pp. 248–253, 2006.

[6] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-time plane segmentation using rgb-d cameras," in *RoboCup 2011: Robot Soccer World Cup XV*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7416, pp. 306–317.

[7] R. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[8] M. Inja, N. Heijne, S. Nugteren, and M. de Waard, "Project ai - the darpa robotics challange - f.o.o.t.l.o.o.s.e." Project Report, Universiteit van Amsterdam, February 2013.

[9] M. Vukobratović and B. Borovac, "Zero-moment pointthirty five years of its life," *International Journal of Humanoid Robotics*, vol. 1, no. 01, pp. 157–173, 2004.