International Journal of Humanoid Robotics
© World Scientific Publishing Company

CLASSIFYING OBSTACLES AND EXPLOITING CLASS INFORMATION FOR HUMANOID NAVIGATION THROUGH CLUTTERED ENVIRONMENTS

PETER REGIER[1]
ANDRES MILIOTO[2]
CYRILL STACHNISS[2]
MAREN BENNEWITZ[1]
University of Bonn,
[1] *Endenicher Allee 19a, 53115 Bonn, Germany*
[2] *Nussallee 15, 53115 Bonn, Germany*
{*pregier,maren*}*@cs.uni-bonn.de, amilioto@uni-bonn.de, cyrill.stachniss@igg.uni-bonn.de*

Received 31 March 2019
Revised Day Month Year
Accepted Day Month Year

Humanoid robots are often supposed to share their workspace with humans and thus have to deal with object used by humans in their everyday life. In this article, we present our novel approach to humanoid navigation through cluttered environments, which exploits knowledge about different obstacle classes to decide how to deal with obstacles and selects appropriate robot actions. To classify objects from RGB images and decide whether an obstacle can be overcome by the robot with a corresponding action, e.g., by pushing or carrying it aside or stepping over or onto it, we train and exploit a convolutional neural network (CNN). Based on associated action costs, we compute a cost grid containing newly observed objects in addition to static obstacles on which a 2D path can be efficiently planned. This path encodes the necessary actions that need to be carried out by the robot to reach the goal. We implemented our framework in ROS and tested it in various scenarios with a Nao robot as well as in simulation with the REEM-C robot. As the experiments demonstrate, using our CNN the robot can robustly classify the observed obstacles into the different classes and decide on suitable actions to find efficient solution paths. Our system finds paths also through regions where traditional motion planning methods are not able to calculate a solution or require substantially more time.

1. Introduction

As humanoid robots are designed to work in human environments, one of the tasks that need to be solved consists of navigating through a cluttered environment where stepping over or onto obstacles or moving an object out of the way might be necessary. Thus, they must be able to avoid or to deal with different types of objects located at random places in the environment. Finding suitable robot motions in environments cluttered with objects imposes a high level of complexity to the motion planning problem and is difficult to solve in a computationally efficient manner.

Common approaches to humanoid navigation in complex environments involve whole-body motion planning and multi-contact planning[1,2]. These approaches usually take several

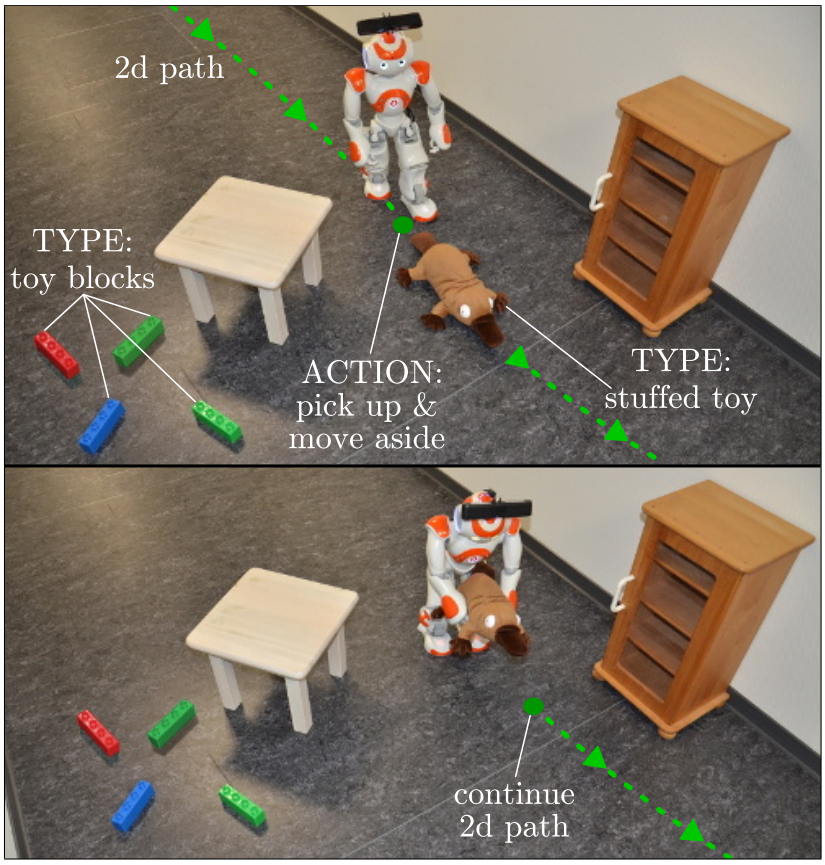


Fig. 1. Application example, in which the path of the robot is blocked. Based on classified non-static obstacles (in this case a stuffed toy and toy blocks) and their associated actions and costs, our system computes a cost grid on which a 2D path can be efficiently computed. The path also encodes the actions that need to be executed by the robot during navigation to reach the goal. As can be seen, the robot moves the stuffed toy aside to clear its path and can then continue walking along the 2D path.

seconds up to minutes to compute a solution. This may prevent the use on a mobile platform or limits it's ability to operate in a reactive way. To limit the computational load, other frameworks employ footstep planning[3,4,5,6] to avoid computing whole-body motion plans. However, if a blocking object needs to be moved aside to reach the goal location, they often do not yield a solution and thus result in a navigation failure.

In this article, we combine the advantages of the existing planning approaches by exploiting semantic information about objects. We present a novel approach to humanoid navigation that combines fast 2D path planning with 3D footstep planning and object manipulation actions in obstructed regions of the path. We consider an indoor environment from which the robot creates a 2D grid map with static obstacles in the absence of clutter using a standard mapping approach[7].

During navigation, the robot adds information about objects of different classes per-

ceived with its camera to the map. We hereby use a convolutional neural network to classify the objects and to decide whether an obstacle can be *stepped over* or *stepped onto*, or moved out of the way by *pushing* or *carrying* it aside. For the classification task, we use our recently developed real-time CNNs[8] that are capable of segmenting RGB images to detect given object classes. This framework provides reliable information about specific object types and their pixel-wise masks (e.g., books, boxes, toys etc.) that we map to appropriate action types, which allow the robot to navigate across the corresponding area. Our approach adds associated stepping and manipulation costs to a 2D map that is used for path planning. In this way, we greatly simplify the full planning problem, as we split the whole plan into several parts, while each part is solved individually. Fig. 1 illustrates a motivating example, where the robot can only reach its goal by manipulating an object. According to the resulting cost map after classifying the obstacles, our planner chooses a path where the robot needs to move the stuffed toy aside.

We implemented our framework in ROS and tested it in various experiments with a Nao humanoid. As the experimental results illustrate, the robot can robustly classify the observed obstacles into different classes and use this information to efficiently find solution paths through passages where objects are blocking the path. The underlying architectures used to extract the semantics from the environment were carefully tailored to this task in order to achieve efficient inference, which allows our approach to run in our small Nao humanoid.

As extension to our previous work[9], we detail how to learn the costs of actions, extend our framework to allow several actions per object class, use point cloud data for deciding which action to take, and present additional experiments.

2. Related Work

Stilman and Kuffner were the first who considered navigation amongst movable objects where the robot can move objects aside if necessary to reach the goal location[10]. The idea of their approach is to decompose the environment representation into disjunct regions and search for obstacle motions that connect two disjoint regions allowing the robot to transition between them. Stilman *et al.* employed the framework on a real humanoid to navigate through an environment with movable chairs and tables[11] where the world state was observed by an external motion capture system. In contrast to this approach, we perform fast planning on a cost grid based on a 2D map, which can be easily obtained by using standard mapping algorithms[7]. Furthermore, we classify objects perceived by the robot and encode different actions associated to the object classes directly in the navigation costs contained in the cost grid. In this way, we combine the planning on a 2D cost grid with local 3D footstep planning and object manipulation.

A variant of the humanoid locomotion problem, in which the robot can utilize objects to get to locations that are otherwise out of reach for the robot, was approached by Levihn *et al.*[12]. The authors introduced the concept of relaxed-constrained planning where the planner is allowed to violate certain constraints. The violation is then locally resolved by using suitable objects, e.g., to overcome a high step height.

The task of collecting objects and delivering them to designated places while clearing cluttered obstacles out of the robot's way, was addressed by Hornung *et al.*[13]. The authors proposed to apply a high-level planner with integrated perception, world modeling, action planning, navigation, and mobile manipulation. Our navigation framework is orthogonal to that approach and could be integrated into such a higher level task planning framework.

Grey *et al.* recently presented an approach that uses so-called randomized possibility graphs to traverse environments with arbitrary obstacles, in which footstep as well as whole-body motion planning is required[1]. The authors distinguish between passages that are definitely passable by the robot and ones that are definitely impossible to be passed by using approximations of the constraint manifold. So far, their approach has only been tested in simulation with no perception involved. Lin and Berenson considered navigation in uneven terrain using contact planning of palm and foot locations and learned estimates about the traversability of regions[2]. The idea of this approach is to use the learned traversability estimates as a measure how quickly the planner can generate feasible contact sequences. This measure is used in the heuristic function of the contact space planner to guide the search to areas with more contactable regions. Dornbush *et al.* recently developed a planning framework that considers adaptive dimensionality by determining which planning dimensions are relevant in each region of the environment[14]. Their idea was to plan with multiple low-dimensional planning representations simultaneously within a multi-heuristic search. While these approaches also aim at speeding up the search for viable solutions paths, the authors do not consider perception and do not take into account the possibility of actively modifying the environment.

The method proposed by Kaiser *et al.* extracts affordances of geometric primitives to support the planning of whole-body locomotion and manipulation actions[15]. Navigation through cluttered passages has not been considered in their scenario.

Although some of the above mentioned approaches provide a robust way to plan through environments containing cluttered regions, they neglect the semantics of the objects. Semantic information, however, can be effectively exploited for humanoid navigation. Current advances in computer vision using deep CNNs to extract semantics of the environment[16,17,18] have made it possible to infer the semantic classes of objects in cluttered scenes with high accuracy. Such approaches allow us to map each object class to a different action type that we can use for planning in an efficient way. Running our classifier in a fast manner is necessary to avoid adding a large computational overhead to our approach. Thus, we build on top of recent work focusing on *real-time* CNNs[8,19,20,21,22]. Since training deep CNN models is a data intensive task, we present a way alleviate this by generating a large-scale dataset of our interest clutter classes with minimal effort in a semi-autonomous way by crawling images from the Internet.

3. System Overview

Before we describe our approach in detail, we present an overview of the individual components, which are also illustrated in Fig. 2. The first step is the semantic segmentation of the input RGB image. The semantically segmented image is then provided to the object

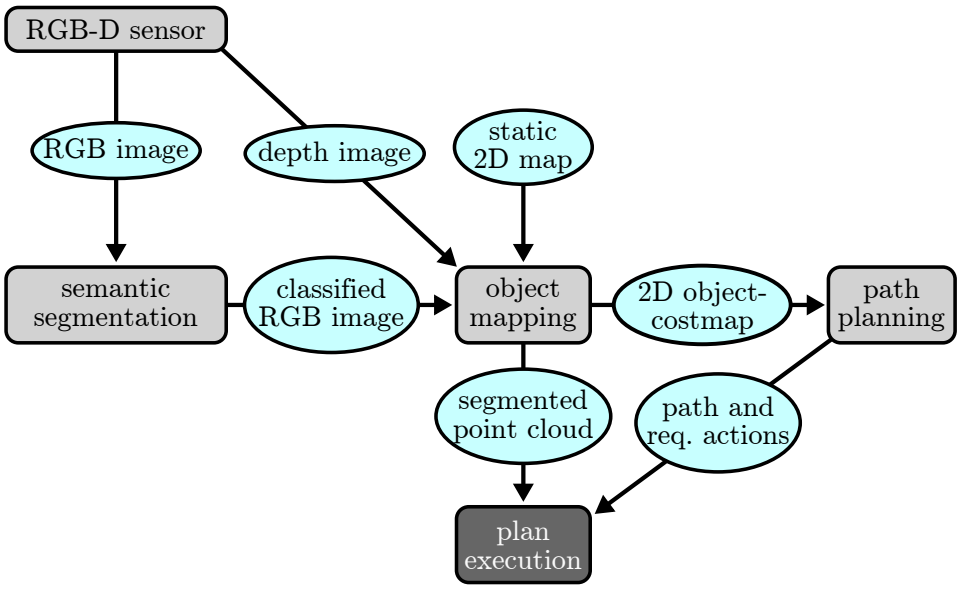


Fig. 2. Overview of our framework. The different components are summarized in Sec. 3.

mapping, which additionally uses 3D point cloud data from the input depth image aligned with the RGB data, and a 2D grid representation of static obstacles in the environment. This static 2D map is constructed using a standard mapping system[7] in the absence of further objects. The output of our object mapping is a 2D cost grid, encoding static obstacles as well as the newly detected and classified objects, on which the path planning takes place. The execution of the computed path, which comprises actions corresponding to the detected objects, is done by the plan execution. The plan execution additionally uses the segmented point cloud when necessary, while invoking the required object actions.

During navigation, our system continuously updates the representation of the environment with information about newly sensed obstacles that might be blocking the way of the robot and replans the robot's path toward the goal if necessary.

4. Semantic Segmentation

The first key step for our approach is the extraction of the semantics of objects in the robot surroundings. Then our approach aims at inferring possible robot actions from the objects in the environment in real-time. This requires a visual classifier that can recognize individual objects accurately from a dictionary of possible classes, while still running fast on a power- and payload-constrained machine such as a small humanoid robot. The state of the art in object detection and semantic segmentation using CNNs makes the accuracy of such algorithms acceptable for this approach to be feasible, but most CNN pipelines are computationally intensive and require large amounts of training data. In order to make the approach applicable on our robot, we rely on a lightweight architecture to achieve a good

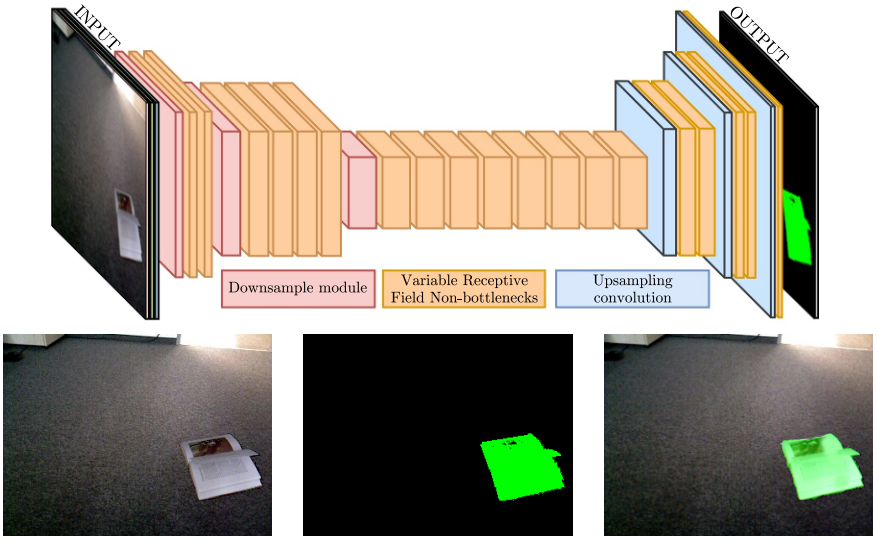


Fig. 3. Top: encoder-decoder semantic segmentation CNN based on the non-bottleneck concept behind ERFNet[21] inferring an image from our dataset. Bottom, left to right: original RGB image, prediction from CNN, and alpha blend for visual qualitative performance assessment. Best viewed in color.

runtime vs. accuracy trade-off. To approach the amount of training data needed, we use pre-trained models from Bonnet's library[8], which already provides useful features in the convolutional layers and we create a large dataset by mixing images recorded by ourselves and a huge amount of scavenged data from the Internet for refining the pre-trained models.

We opt in favor of a semantic segmentation pipeline, which maps each pixel of the robot's camera images into one of the eight classes: *"balls", "books", "boxes", "cars", "dolls", "stuffed toys", "toy blocks", and "background"*, so that each class has at least one navigation action assigned to it. Note that our approach is not limited to these classes and could easily be extended.

4.1. *CNN-Based Semantic Segmentation*

Fig. 3 shows a diagram of the encoder-decoder CNN architecture used in our approach that is build using Bonnet[8]. The chosen architecture is based on ERFNet[21], which proposes to change each computational bottleneck introduced in ResNet[17] and ENet[20] with a "separable non-bottleneck" of a variable receptive field. This module uses a set of separable filters of sizes $[1 \times 3]$ and $[3 \times 1]$ and different dilation rates d, which makes each layer effectively wider without increasing computational cost, allowing the network to be more descriptive without affecting runtime. The choice of using different dilation rates allows the network to have a bigger equivalent receptive field in the image space, capturing long-range dependencies, which is key for big objects. By using a model with these properties, and adjusting the number and width of the layers to fit our data, we can achieve a model that is descrip-

tive enough to provide us with accurate semantic labels while running in real time. In order to achieve even further efficiency in inference to allow the approach to run in a resource constrained platform such as a small humanoid, we exploit TensorRT[23] acceleration of our neural network. In order to achieve this, the used ERFNet backbone needed to be modified to fit all supported operators. At the time of submission, there is no efficient support of dilated convolutions within TensorRT, so in order to efficiently infer full semantic segmentation with our proposed architecture, each $[1 \times 3]$ and $[3 \times 1]$ dilated convolution was replaced by a dense convolution of $[1 \times [3 + 2 \times\ d]]$ and $[[3 + 2 \times\ d] \times 1]$. We observed no drop in performance from this modification, and even though more operations are needed for this, the acceleration obtained by the inference optimization provided in TensorRT was superior.

We start from a pre-trained encoder using this modified architecture, which was trained with the COCO dataset[24] and therefore provides rich features even before the semantic segmentation task-specific training. We attach a small decoder that is trained from scratch using random weights and fine-tune the model training end-to-end using back-propagation and a pixel-wise cross-entropy loss of the form

$$L_{\text{semantic}} = -\sum_{c=1}^{C} w_c\, y_c\, \log\left(\hat{y}_c\right) \tag{1}$$

$$w_c = \frac{1}{\log\left(f_c + \epsilon\right)} \quad \text{with} \quad f_c = \frac{1}{P}\sum_{p=1}^{P} \begin{cases} 1 & \text{if } p = c \\ 0 & \text{if } p \neq c \end{cases}, \tag{2}$$

where w_c penalizes class c according to the inverse of its frequency in the ground truth, bounded by a parameter ϵ which is selected by cross-validation and is set to 1.02 in all our experiments.

The training was performed with a dataset of 5,000 images containing roughly 20,000 toy instances with their respective dense masks, which we explain in the following section. The retraining was performed by minimizing Eq. (1) through stochastic gradient descent using the Adam optimizer, with a batch size of 36, a batch normalization momentum of 0.99, and an initial learning rate of 10^{-4}. The learning rate is then halved every 50 epochs, training for 200 epochs over the whole data. We used a channel dropout rate of 10% in the layer before the linear classifier, and a weight decay of 10^{-5} for regularization during training, and use the model with the best validation error measured after each epoch, a practice commonly called early-stopping.

4.2. *Data Collection*

As previously stated, training deep CNNs requires a large amount of labeled training data to obtain the accuracy required to run other approaches on top of the obtained semantics. This effect is particularly magnified when using a semantic segmentation pipeline, because a holistic knowledge of what each image contains is not enough, and labels are required at a pixel level, increasing the effort required to label each image considerably. Even though using pre-trained model helps to reduce the amount of required labeled data, a particular



Fig. 4. Examples of pairs of classes with low inter-class distance, challenging for the CNN, but important for our approach due to different associated object interactions.



Fig. 5. Examples of backgrounds used to generate the synthetic training images. Backgrounds were also crawled from the internet, with queries designed to match viewpoints of places where the robot is likely to operate, such as home and school environments.

case which makes the training more data-hungry is having low inter-class distances, like in our case (see Fig. 4). To circumvent this problem, we collected a dataset with 1,000 objects focusing on the hard examples of inter-class distance, i.e., focusing mostly on labeling objects whose appearance is similar, but which have a different semantic label. This is done to train CNN features that are sensitive enough to allow the classifier to fit the classification hyper plane effectively. To generate a dataset for semantic segmentation, we need pixel-wise masks for each individual object. Since such a labeling is expensive, we collected the data with an RGB-D camera and segmented the objects in the depth channel to obtain the ground-truth mask before feeding them to the CNN as a 3-channel RGB-only image.

To scale up the dataset and make it an order of magnitude bigger, we wrote a script to automatically download images from the Internet with properly formatted queries returning images fulfilling the following conditions: (i) the image contains only one of the desired classes in the dictionary, and (ii) the image contains either an alpha channel making the background transparent or has a blank background. Under these restrictions, the script returns roughly 25,000 images using Google. We further reduce it to roughly 20,000 images after six hours of supervised cleaning by a human. The supervision consists of navigating

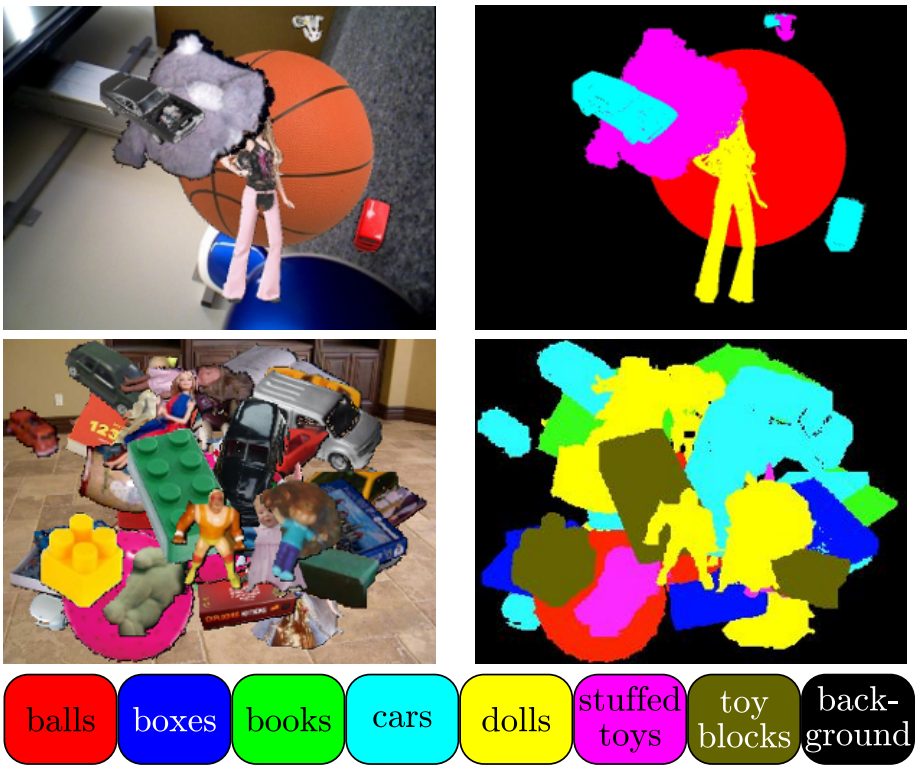


Fig. 6. Examples of generated clutter images with added background. Left: RGB image, right: ground truth. Best viewed in color.

Table 1. Example assignment of possible actions to object classes. The actions are chosen by an expert user according to the robotic hardware, in this case a Nao robot.

Object class	Possible actions
balls	push, step over, pick up
cars, toy blocks	step over, pick up
stuffed toys, dolls	pick up
boxes, books	step onto

quickly through the crawled images and identifying objects that either do not belong to the specific class we are interested in, or whose alpha channel does not fit the object boundary. For our CNN to be usable in realistic environments, the last step in this dataset generation method is to generate 5,000 clutter images from our raw data, containing one of 300 different backgrounds (Fig. 5) and any number of random objects from the database from 0 to 20 objects per image. This can be considered as "synthetic" data, but it is a step closer to the real world, because the images are of real-world objects (Fig. 6).

Table 2. Classification metrics on validation set for different input image resolutions.

Resolution	mAP	Per-class AP							mIoU
		Ball	Books	Boxes	Cars	Dolls	Stuffed toys	Toy blocks	
320 × 240	0.792	0.908	0.747	0.733	0.768	0.801	0.828	0.759	0.585
640 × 480	0.875	0.946	0.878	0.876	0.847	0.874	0.874	0.831	0.715

Table 3. Runtime for segmentation at different image resolutions.

Resolution	Runtime	
	GTX1080Ti	Jetson TX2
320 × 240	10 ms (100 FPS)	89 ms (11 FPS)
640 × 480	33 ms (30 FPS)	245 ms (4 FPS)

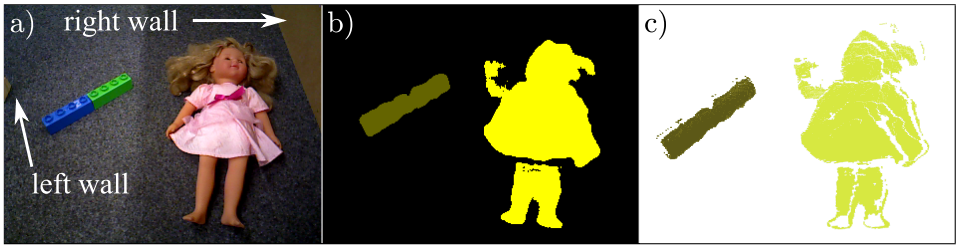


Fig. 7. Data processing for the RGBD data. a) The original RGB image containing a doll and toy blocks between two walls. b) Semantic segmentation results using the Bonnet framework. c) Segmented point cloud of the corresponding objects, using the depth image to get the spatial information of each marked pixel.

5. Path Planning Utilizing Obstacle Information

In this section, we describe our method for exploiting the semantic information about detected and segmented objects during path and action planning.

5.1. *Actions for Object Classes*

We assume that an expert user assigns for the individual object classes possible actions defining how the robot can overcome such obstacles when necessary. The possible actions for the object types inherently depend on the specific robot that is being deployed. As an example, Tab. 1 shows the actions associated with the object classes for a Nao robot. Note that for some object classes several actions are possible, in which case our system selects the least cost action to deal with the objects on its path. The robot furthermore analyzes the point cloud of the object to decide if an action may not be executable, for example because the object is to high to be stepped over. Those aspects are described in more detail in Sec. 5.3.

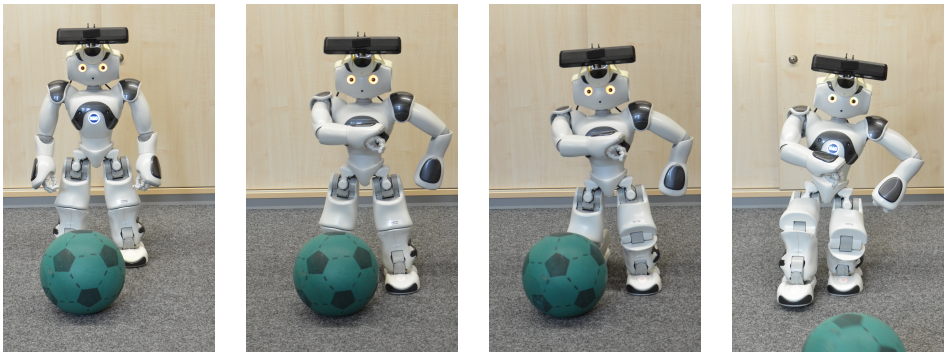


Fig. 8. Pushing: In our implementation, the push action includes tracking the ball locally, positioning relative to the object, and kicking it in the forward direction.

For our scenario, we currently use the following actions:

Standard walking in case of no objects: If the path does not contain any objects, the robot's walking controller follows a 2D path.

Push: If an objects needs to be pushed away, the robot follows the 2D path to the last free grid cell on the path and starts the pushing action. Thus, the robot senses the object locally using the segmented point cloud, moves to a target position relative to the object, and pushes the object out of its way in forward direction. The push action is illustrated in Fig. 8).

Step over and step onto: If the robot needs to traverse an area with objects that need to be stepped over or onto, the robot applies footstep planning in the corresponding region on a height map computed from the point cloud using our previous work[25]. Fig. 9 shows the maximum height of a box the Nao can overcome.

Pick up: If the robot needs to pick up an object and move it out of the way, the robot again follows the 2D path to the last free cell on the path in front of the object and then identifies the object in the segmented point cloud, finds the target position relative to the object, grabs it, rotates $180°$, and puts the object onto the ground, and rotates back $180°$. The pickup action is depicted in Fig. 10.

5.2. *Action Costs*

We define the costs of an action according to the time the robot needs to perform the action. To determine the execution time, we designed an experiment in which the Nao robot had to reach a goal one meter in front of it. The experimental setup of the actions is illustrated in Fig. 11. We measured the time it took the robot to reach the goal by just walking to the target location and by additionally pushing an object out of the way, stepping onto or over objects, and picking up an object. For each case, we performed ten experiments and selected the average execution time as the cost used for the planner. The averaged times

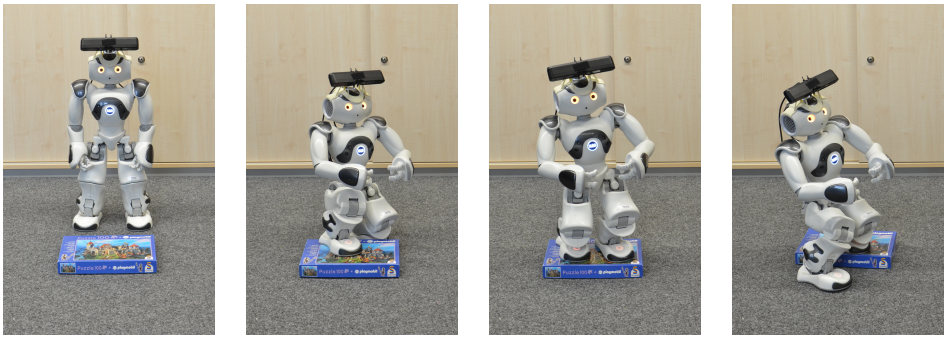


Fig. 9. Stepping onto: The step onto action includes stepping onto the object and also stepping down according to a footstep plan.

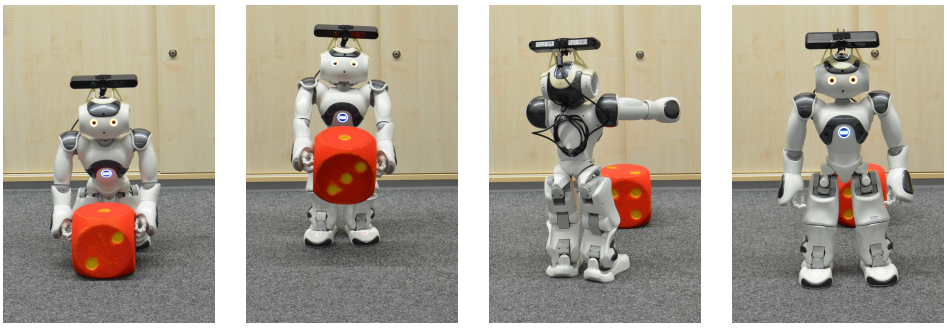


Fig. 10. Picking up: The pick up action includes the tracking of the object locally, positioning relative to the object, executing the grabbing, performing a 180° turn, dropping the object, and rotating back 180°.

of the experiments are provided in Tab. 4. Since the walking is subject to errors induced by the action itself, e.g., shaky walking start after the push action or the difference of the orientation to the goal after the pick up action, it was important to let the Nao robot walk before and after the corresponding action execution to measure the influence of small uncertainties on the time performance.

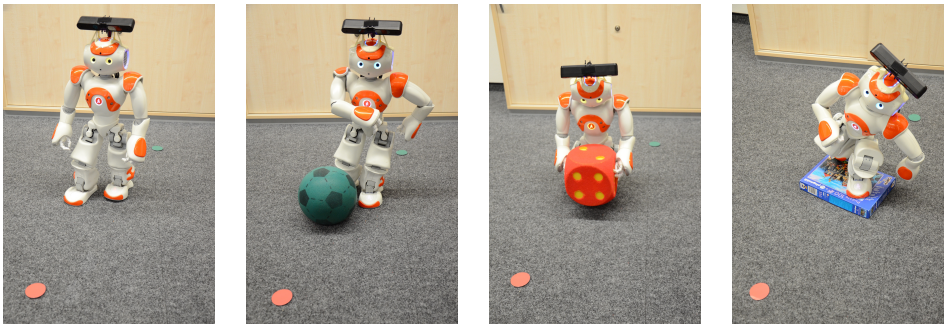
Other factors, e.g., energy consumption or risk of failure, can be considered as well for the cost computation and added to the determined cost.

5.3. *Object Mapping*

In this subsection, we describe how to map objects detected in the environment using RGB-D data onto the cost grid used for planning. We use the semantic segmentation described in Sec. 4 and combine it with the depth information of the RGB-D image to get a segmented point cloud of the corresponding objects (see Fig. 7). Afterwards, we project the segmented point cloud onto a 2D grid map representation of the environment containing

Table 4. Execution times of actions.

Action type	Mean execution time [s]
walk	12
push	25
pick up	55
step over	40
step onto	61



walk push pick up step onto

Fig. 11. Experimental setup to determine the costs of actions according to the completion time. See Sec. 5.2 for a detailed explanation.

inflated static obstacles as illustrated in Fig. 12a. Inflating all objects with the robot radius is a general concept to prevent the robot from colliding with obstacles in case of slight localization errors. We maintain an object database that contains the information about objects in form of object ID, object class, and the set of corresponding 2D grid cells. Note that obstacles that cannot be classified by the CNN and, thus, are considered as background are mapped as static and are not stored in the database of objects that can be manipulated.

Our approach uses a 2D cost grid map for path planning that encodes, in addition to the static obstacles, the costs of the actions corresponding to the observed objects, which are estimated as described in the previous subsection (Sec. 5.2).

To choose the action for an object, we consider the segmented point cloud to exclude some possible actions listed in Tab. 1 that are predicted to be not executable by the robot depending on the size of the object. We then assign the cheapest action of the remaining actions to each object and, thus, to the cost map. For example, a small ball can either be pushed away (cheapest action), stepped over, or the ball can be picked up in order to free the path while for a big ball, e.g., a basket ball, neither action would be expected to be executable so that the object would be marked as static in the map. The applied heuristics to suggest a viable action for the Nao humanoid are listed in Tab. 5. In general more advance operations for analyzing the geometry of the segmented object could be

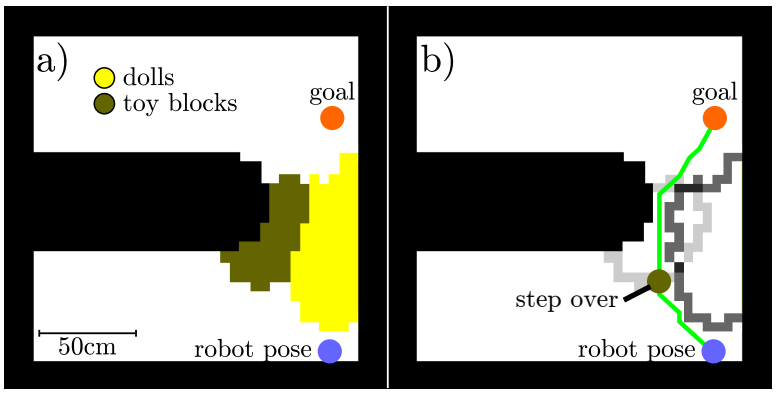


Fig. 12. a) Visualization of the projection of the objects onto a 2D grid. Inflated static obstacles are represented as black cells and cells with detected objects from the segmented point cloud (see Fig. 7c) are color coded. Yellow cells correspond to the doll, brown cells to the toy blocks. b) Resulting 2D cost map for planning with the costs of the object actions encoded in the border cells of the objects. The border cells of the toy blocks (light gray), which can be stepped over, have lower cost than the cells corresponding to the doll (dark gray), which needs to be moved away so that the computed path contains the action to step over the blocks. The overlapping border cells of the inflated objects are the sum of their action costs.

Table 5. Heuristics for the viable actions of the Nao robot.

Action type	**Geometric object features**
push	maximum height $< 20\ cm$
pick up	longitudinal axis $< 30\ cm$
step onto	maximum height $< 7\ cm$
step over	maximum height $< 6\ cm$ and transverse axis $< 5\ cm$

performed. However, this leads to a trade-off between the accuracy of the suggested action and the time performance of the planning framework.

In Fig. 12b, the cost value is represented by the gray level of the object border cells. Hereby, the costs of the overlapping border areas between the inflated segmented objects are the sum of the corresponding action costs of both/multiple objects, which means that in these regions several object actions will be necessary. In this example, the path leads across the toy block. From Tab. 1, our planner can chose between the step over and the pick up action and decides to step over the block since this is the cheaper action.

Note that our system pauses the mapping while the robot executes a push, step, or pick up action to free as much resources as possible to perform the action. Thus, no object tracking takes place during that time.

5.4. *Path Planning*

Since the cost map contains the information about all obstacles, i.e, static and not static, and encodes the potential object actions and their associated costs, we can efficiently use

A* search [26] with the Euclidean distance as the heuristic function to find a path for the robot on the cost grid. If the path leads through any object area, the corresponding class and, thus, the possible actions can be derived from the object database. In the example shown in Fig. 7 and Fig. 12, our approach computes the green path with the action to step over the toy blocks.

5.5. *Updating the Action*

Note that the path planner generally provides only suggestions regarding the action types. Should the execution module not find a solution for a proposed actions, e.g., due to object-environment configuration or classification errors, the action would be changed for the corresponding object such that our planner would seek a different solution on the updated cost grid. This can also happen in case the footstep planner does not find a sequence of valid footholds for a region containing objects to be stepped over or onto, see Sec. 6.3 for an example.

6. Experimental Evaluation

In this section, we present experiments to evaluate the performance of the CNN-based classification framework for the navigation task at hand (Sec. 6.1) and to demonstrate the capabilities of our system with respect to efficient planning and navigation in various real-world experiments with a Nao humanoid (Sec. 6.2). Additionally, we performed experiments with the REEM-C [27] humanoid (Sec. 6.3) in simulation to show a generalization of our approach to other robot platforms that are able to perform more advanced actions. For step over heuristic of the REEM-C, we set the maximum height and transverse axis to 15 cm and 10 cm, respectively. For the pick up action, we designed a one-handed grab procedure and determined the 10 cm as maximum value for both minor axes of the object. Throughout this section, the illustrated grid maps are similar to Fig. 12a, i.e., they show the inflated object classes (rather than the actual costs) for better visualization. The resolution of the grid maps was set to 5 cm for all experiments. The inflation radius was chosen as half the shoulder width of the considered humanoid (15 cm for the Nao and 25 cm for the REEM-C).

6.1. *Classification Results*

The first set of experiments is designed to show that our semantic segmentation approach is applicable on mobile robot platforms in terms of accuracy, runtime, and computational resources. As detailed in Sec. 4.2, we train a semantic segmentation CNN with 5,000 images, generated from a database of 20,000 different object instances, and over 300 backgrounds. The network is then evaluated on a test set containing 1,000 real-world images collected in our lab and pixel-wise annotated. Semantic segmentation approaches which assign a label to each pixel in an image are typically evaluated using the mean Jaccard index, also called

mean intersection over union (mIoU), defined as

$$mIoU = \frac{1}{C} \sum_{i=1}^{C} \frac{tp_i}{tp_i + fp_i + fn_i}, \tag{3}$$

where C is the number of classes, and tp, fp, and fn are the pixel-wise number of true positives, false positives, and false negatives per-class, respectively. For approaches that in the end work on objects, a commonly used measure is the mean average precision (mAP):

$$mAP = \frac{1}{C} \sum_{i=1}^{C} \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_i(r), \tag{4}$$

where r corresponds to a value of recall in the precision-recall curve for each class, and $p_i(r)$ is the value of precision corresponding to recall r for class i. Predicted instances are defined as a positive detection when they have more than 50% IoU overlap with the ground truth mask.

In our experiments, we found that the quality of the classification depends mostly on the size of the input images, and therefore report in Tab. 2 the results for two resolutions of the used sensor (ASUS Xtion PRO). For a resolution of 320×240, we achieve a mAP over all classes of 0.79 and for 640×480 the mAP increases to 0.88. These results are encouraging since they show that starting from pre-trained weights, a network can be trained solely on images crawled from the Internet and few hours of human supervision to clean the dataset from improper objects present in the query results and wrong masks in the alpha channel. The limitations of off-line batch training in terms of labeled data collection and pre-definition of the classes are hard to circumvent and currently an open research area in computer vision.

As with the accuracy of the model, the runtime of the CNN is also highly dependent on the input resolution. In Tab. 3, we show the runtime of the model in different hardware and using different resolutions. The results show that the approach is usable also in resource-constrained hardware, such as the NVIDIA Jetson TX2, where we achieve a framerate of 4 Hz-11 Hz depending on the image resolution.

6.2. *Real-World Experiments with a Nao Humanoid*

The second set of experiments is designed to show the behavior of our planning framework in different real-world navigation scenarios. The experiments demonstrate the advantages of our method in comparison to pure footstep or whole-body-motion planning systems, as these approaches are not capable of finding a solution to the goal when manipulation actions are required.

We equipped the Nao robot with a ASUS Xtion PRO to show the full capability of our navigation framework. For 6D localization we apply Monte Carlo localization as developed by Hornung *et al.*[28] and extended by Maier *et al.*[29] for depth camera data.

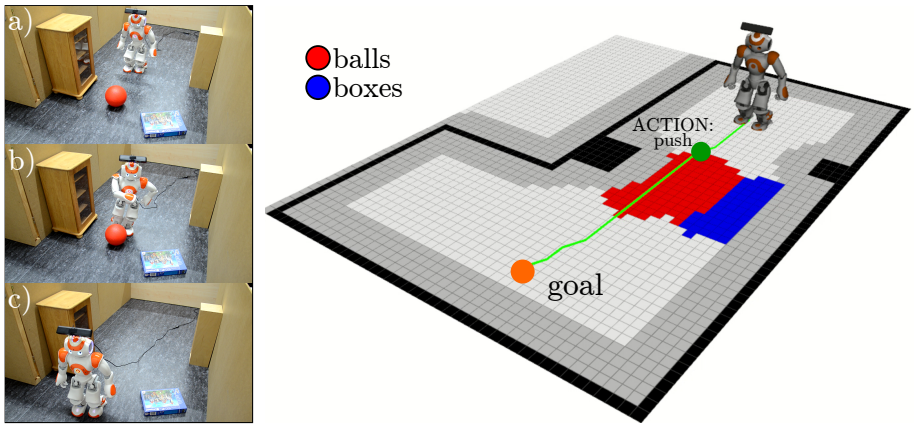


Fig. 13. Path planning with manipulation actions: Experiment with a Nao robot (left) and segmented objects with inflation radius projected onto the grid map (right). *Left*: a) The robot detects the box and the ball on its way to the goal and decides to push the ball in order to clear the path, since this is the cheapest path on the corresponding cost map. b) After the robot has followed the path close to the object, it performs the pushing action. c) The robot continues walking along the path, which does not contain any further objects.

6.2.1. *Path Planning with Manipulation Actions*

In the situation depicted in Fig. 13a, the robot detected a box and a ball that obstructed the way. To reach the goal location, the robot could either push the ball aside or step onto the box before continuing walking. Based on the computed cost map our planner found a path across the ball. The robot followed the path to the vicinity of the ball and then performed a push action (see Fig. 13b). After pushing the ball aside, the path to the goal was free and the robot continued walking (see Fig. 13c.)

6.2.2. *Replanning the Path During Execution*

The next two experiments are designed to show the replanning capabilities of our framework when the current 2D path does not appear to be optimal anymore according to an updated cost map.

In the first experiment shown in Fig. 14, our robot first detected only a single row of blocks and the stuffed toy blocking the way to the goal. Accordingly, stepping over the two detected blocks was the cheaper solution compared to picking up the stuffed toy and putting it aside. While following the path and updating the cost map, the robot subsequently encountered more blocks, so that cheapest path was now lead through the stuffed toy and the plan was change to include the action to pick up the stuffed toy. Thus, the robot followed the path to the last free cell before the object and then executed the pick up action, before it followed the remaining path to reach the goal. A video of this experiment is available online[a].

The next experiment in Fig. 15 shows a scenario with two different routes to the goal.

[a] `https://youtu.be/WO94iXT3V1I`

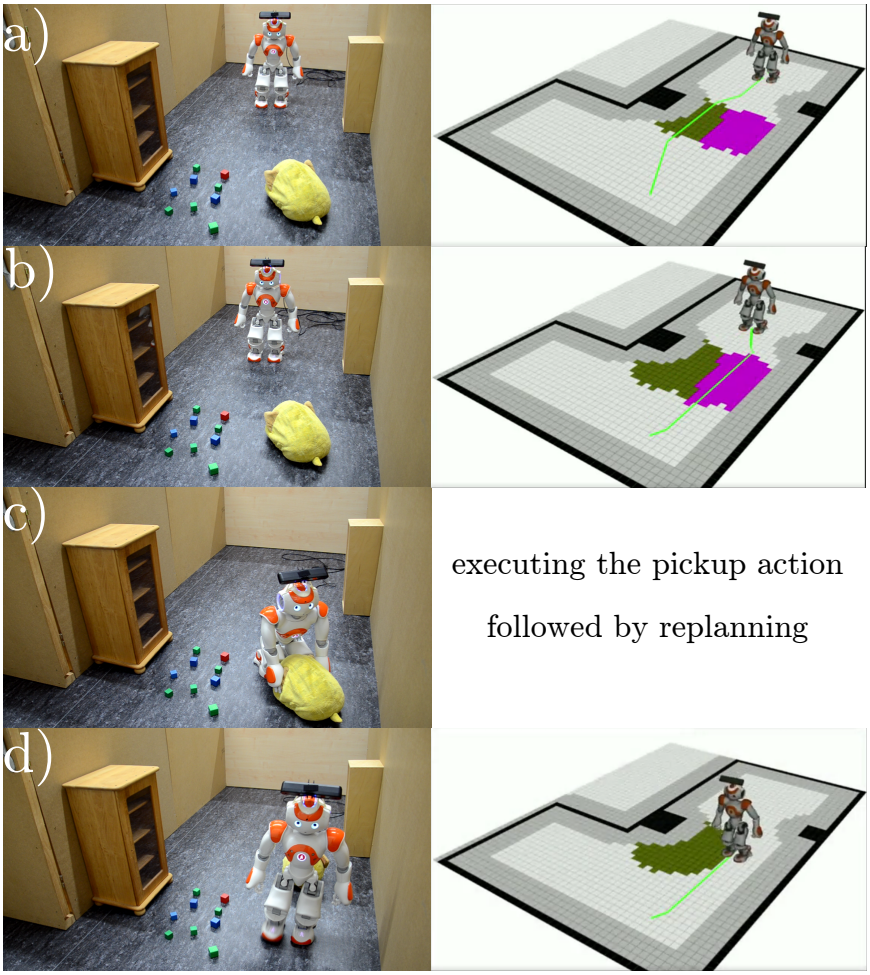


Fig. 14. Replanning the path on an updated cost grid during execution. a) In this case, the robot initially observes only two toy blocks at the foremost row and the stuffed animal in its field of view. The cheapest solution computed by our planner on the corresponding cost grid leads through the toy blocks, which require footstep planning. b) While moving forward, the robot detects further toy blocks and computes a new path on the updated cost grid. Now the path leads through the stuffed animal, suggesting a pick up action. This solution is cheaper since it contains one manipulation action rather than several step over actions. c) The robot executes the pickup action and puts the object aside. d) Afterwards our planner updates the cost map and replans the path. The robot can now simply follow the path to its goal.

Since the balls were too large for the Nao to step over them (Fig. 15a), the robot could either push the balls out of the way, pick them up, or take a detour around the large central obstacle. In this case, our planner found a path that included the detour, since interacting with multiple objects had higher associated costs. In Fig. 15b another obstructing object, in this case a stuffed toy, was detected, where our planner decided based on the updated cost map to perform a pick up action to clear the path.

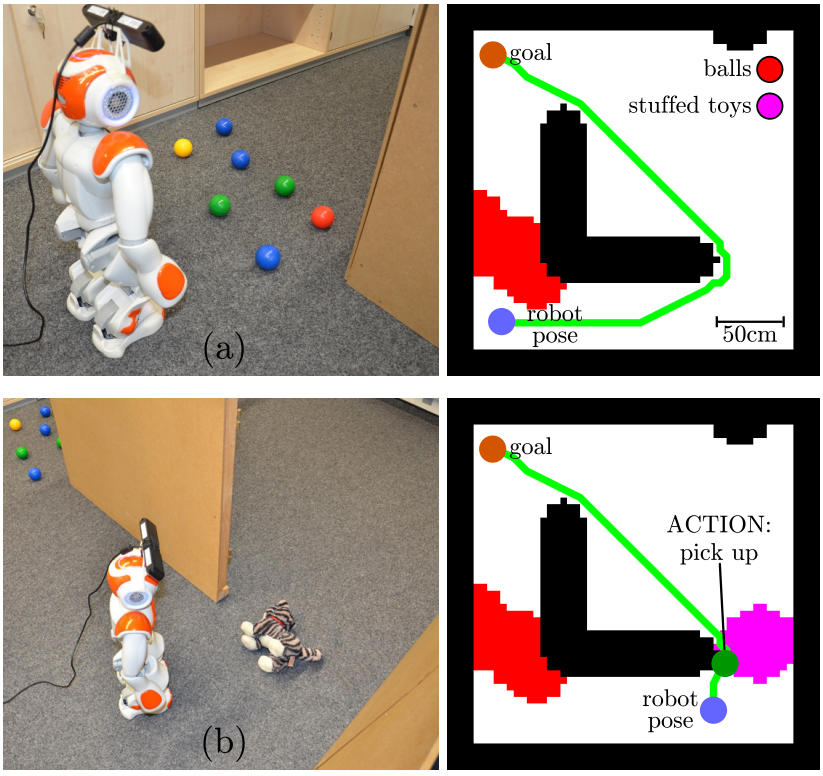


Fig. 15. Replanning the path during execution: Nao humanoid and segmented objects with inflation radius projected onto the grid map. a) The robot perceives seven objects classified as balls in its way and decides to take the detour around the L-shaped static obstacle since this appears to be the cheapest solution according to the cost map. b) On its way to the goal, the robot detects a further object (in this case a stuffed toy). Our framework now decides to pick up the object to reach the goal location.

6.3. *Replanning Actions During Execution*

The final experiment demonstrates the capabilities of our planning system to replan the action for an object during execution. In this scenario (Fig. 16), the robot had to cross a narrow passage with several blocks. The initial plan contained step over actions over two blocks in the middle. However, after approaching the blocks and initiating footstep planning for the step over action, our footstep planner[25] could not find a sequence of valid footholds in order reach the subgoal and failed. According to the failed action, the cost map was updated with the new action pick up for the first block on the path. Replanning on the updated map lead to a solution containing the same objects but with another action for the first object. The robot had to pickup the first block on the path and step over the second one. After performing the pick up action and replacing the block, the path planner found a valid sequence of steps to cross the region.

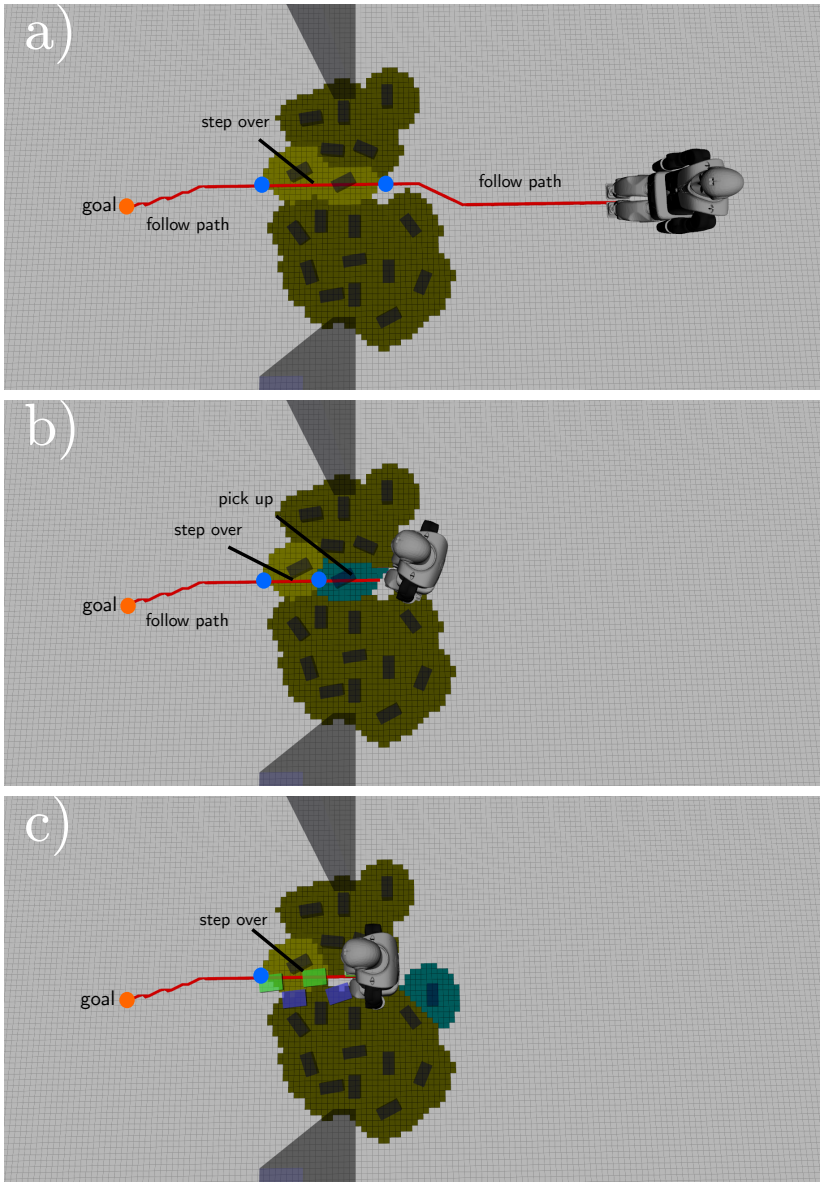


Fig. 16. Replanning actions during execution: Experiment with a REEM-C[27] robot in simulation. a) Initially, our planner computes a path that suggests to perform the *step over* actions over the two blocks to be computed with a footstep planning algorithm (between the two blue circles). b) However, due to the configuration of the blocks, the footstep planner could not find a valid footstep plan. Therefore, our planner selects an alternative action for the first object on the 2D path. Based on the action table, our framework decides to *pick up* the first object and recomputes the plan. c) After the robot has picked up and moved the object out of the way, the footstep planner finds a path over the remaining region.

6.4. *Summary of the Experiments*

In sum, our experimental evaluation shows that semantic information about objects can help humanoid navigation to efficiently plan and effectively execute navigation actions that include handling objects. We demonstrate with our experiments that state-of-the-art semantic segmentation CNNs such as Bonnet[8] are well-suited to solve the associated perception problem of classifying the individual objects in the vicinity of the robot. Based on this information, the robot can combine 2D planning, footstep planning, and an effective object-dependent action selection approach. As a result of that, complex to compute complete whole-body plans or higher-level action plans provided by a symbolic action planner can be avoided. With our framework, the robot can perform planning of complex navigation tasks online, in simulation as well as in the real world. Our experiments demonstrate this for the computation of new plans as well as for online replanning. The combined maximum runtime of the mapping and planning step is 0.2 s for a reasonable number of objects (two to five) on an *Intel Core i7-4710MQ*, without the object classification and segmentation, which runs in another thread with 5 Hz. Thus, the claims made in this article have been backed up with our experimental evaluation.

6.5. *Implementation Details*

For our experimental setup, we use the Robot Operating System [30] (ROS) as a communication backbone between the different framework components. For the experiments with the Nao robot, we used the ROS packages [31] that contain the drivers for communication with the NaoQI API [32], the robot model, and the walking module. For localization, we also used ROS packages for Monte Carlo localization using Octomap as environment representation [33,34,35].

The experiments with the REEM-C were simulated in Gazebo [36] and visualized with RViz [37]. We integrated our footstep execution with the PAL Robotics repositories [38,39] available on GitHub that provide the robot model and the communication interface.

7. Future Work and Limitations

In future work, we plan to determine the actions for objects by using a neural network that can predict the object affordance directly. In the last years, progress has been made that shows the potential of such approaches [40,41,42,43]. The detection of affordances could reduce the amount of engineering required to identify the actions of objects and also reduce false action suggestion of our planner that are only noticed during execution, since affordances consider not only the object properties but also its surrounding.

We designed the actions in Sec. 5.1 to overcome objects that are impeding the way to the goal. The advantage of our planner is that we can easily exchange the action executing modules with more advance approaches. Similar to Hornung *et al.*[13], one could use an module that differs between one handed and two handed grabbing depending on the object for the pick up action. For the push action one could differ between pushing the object in the forward direction or to the side depending on the configuration of the environment.

Furthermore, one could apply the ideas of this research to different scenarios, where the robot has not only to reach an obstructed goal, but also improve the configuration of the environment, e.g., by cleaning it up. Here the robot needs to reason about the object placement when a manipulation action is suggested. To achieve this, further investigation of environment models, motion execution models, high level planning, database integration is required. For further reading on this subject, we refer to [10,11,13,44,45,46].

Wrong object classification of the CNN or non-executable actions caused by wrong heuristics for action execution can of course happen. In these cases, the map and the path will be updated when new information becomes available. As a result, the robot's resulting path might not be the optimal one. We experienced the following classification errors in our experiments:

- A single object was perceived as two objects of different classes, specially when detected at the border of the field of view of the sensor, because the object was only partially perceived and/or pixels further away from the robot cover a bigger surface of the environment and thus provide less information for the CNN.
- False positive object detection of the background.

In most cases, the error was compensated by the next sensor measurement and the map as well as the path were updated accordingly.

8. Conclusion

In this article, we presented a novel framework for humanoid motion planning that exploits the knowledge about obstacle classes during the planning process for fast planning and efficient humanoid navigation. As one central contribution we propose to train a convolutional neural network to distinguish between different object classes and use this information to construct a cost grid during navigation. The cost grid represents the static obstacles in the environment as well as the costs of actions that need to be carried out by the robot to cross cluttered regions. These cluttered regions typically contain obstacles that can be overcome by the robot, i.e., by actions such as stepping over or onto, pushing, or picking up. The robot then uses the cost grid for efficiently planning its path to the goal location and the 2D path implicitly contains all necessary actions to be executed by the robot. Should replanning be necessary during the execution due to failed actions, the cost grid is updated with the costs of alternative actions and the path is replanned.

As we showed in various experiments, the trained neural network is able to robustly distinguish between the different obstacle classes and thus provides relevant information to the robot's planner. We demonstrated that a humanoid robot can exploit the knowledge about classified obstacles during navigation and efficiently find solution paths that contain appropriate actions to deal with the obstructing objects. Furthermore, we showed that in case the current solution cannot successfully be executed, the robot invokes replanning and our system provides an alternative path.

Acknowledgments

We thank Philipp Karkowski for providing the footstep planning framework and his contributions to the conference version of this article. We furthermore thank NVIDIA for their GPU donation, which was used for training the semantic segmentation models.

References

1. M. Grey, A. Ames, and C. Liu, "Footstep and motion planning in semi-unstructured environments using randomized possibility graphs," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017, pp. 4747–4753.
2. Y. Lin and D. Berenson, "Humanoid navigation in uneven terrain using learned estimates of traversability," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2017, pp. 9–16.
3. R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2014, pp. 279–286.
4. M. Fallon, P. Marion, R. Deits, T. Whelan, M. Antone, J. McDonald, and R. Tedrake, "Continuous humanoid locomotion over uneven terrain using stereo fusion," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2015, pp. 881–888.
5. A. 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 (RA-L)*, vol. 2, no. 4, pp. 1856–1863, 2017.
6. D. Wahrmann, A.-C. Hildebrandt, T. Bates, R. Wittmann, F. Sygulla, P. Seiwald, and D. Rixen, "Vision-based 3d modeling of unknown dynamic environments for real-time humanoid navigation," *The Int. Journal of Humanoid Robotics (IJHR)*, vol. 16, no. 1, pp. 1—-34, 2019.
7. G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Trans. on Robotics (TRO)*, vol. 23, no. 1, pp. 34–46, 2007.
8. A. Milioto and C. Stachniss, "Bonnet: An Open-Source Training and Deployment Framework for Semantic Segmentation in Robotics using CNNs," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019, pp. 7094–7100. [Online]. Available: https://arxiv.org/abs/1802.08960
9. P. Regier, A. Milioto, P. Karkowski, C. Stachniss, and M. Bennewitz, "Classifying obstacles and exploiting knowledge about classes for efficient humanoid navigation," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2018, pp. 820–826.
10. M. Stilman and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *Intl. Journal of Robotics Research (IJRR)*, vol. 2, no. 4, pp. 479–503, 2005.
11. M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner, "Planning and executing navigation among movable obstacles," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2006, pp. 820–826.
12. M. Levihn, K. Nishiwaki, S. Kagami, and M. Stilman, "Autonomous environment manipulation to assist humanoid locomotion," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014, pp. 4633–4638.
13. A. Hornung, S. Boettcher, C. Dornhege, A. Hertle, J. Schlagenhauf, and M. Bennewitz, "Mobile manipulation in cluttered environments with humanoids: Integrated perception, task planning, and action execution," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2014, pp. 773–778.
14. A. Dornbush, K. Vijayakumar, S. Bardapurkar, F. Islam, M. Ito, and M. Likhachev, "A single-planner approach to multi-modal humanoid mobility," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018, pp. 4334–4341.

15. P. Kaiser, D. Gonzalez-Aguirre, F. Schueltje, J. Borras, N. Vahrenkamp, and T. Asfour, "Extracting whole-body affordances from multimodal exploration," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2014, pp. 1036–1043.
16. L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, vol. 40, no. 4, pp. 834–848, 2018.
17. K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
18. H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 2881–2890, July 2017.
19. A. Milioto, P. Lottes, and C. Stachniss, "Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018, pp. 2229–2235. [Online]. Available: https://arxiv.org/abs/1709.06764
20. A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: Deep neural network architecture for real-time semantic segmentation," *arXiv preprint*, vol. 1606.02147, 2016.
21. E. Romera, J. Alvarez, L. Bergasa, and R. Arroyo, "ERFNet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Trans. on Intelligent Transportation Systems (ITS)*, vol. 19, no. 1, pp. 263–272, 2018.
22. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.
23. "NVIDIA TensorRT. Programmable Inference Accelerator." [Online]. Available: https://developer.nvidia.com/tensorrt
24. T. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. Zitnick, "Microsoft COCO: Common objects in context," *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, vol. abs/1405.0312, pp. 740–755, 2014.
25. P. Karkowski, S. Oßwald, and M. Bennewitz, "Real-time footstep planning in 3D environments," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2016, pp. 69–74.
26. P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. on Systems Science and Cybernetics (SSC)*, vol. 4, no. 2, pp. 100–107, 1968.
27. "REEM-C. A biped humanoid robot." [Online]. Available: https://pal-robotics.com/robots/reem-c
28. A. Hornung, K. M. Wurm, and M. Bennewitz, "Humanoid robot localization in complex indoor environments." in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010, pp. 1690–1695.
29. D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3D environments based on depth camera data," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2012, pp. 692–697.
30. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proc. of the ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.
31. "ROS: Aldebaran Nao." [Online]. Available: https://wiki.ros.org/nao
32. "NAOqi API." [Online]. Available: https://doc.aldebaran.com/2-1/naoqi/index.html
33. "ROS: Humanoid localization." [Online]. Available: https://wiki.ros.org/humanoid_localization
34. "ROS: OpenNI." [Online]. Available: https://wiki.ros.org/openni2_launch
35. "ROS: Octomap." [Online]. Available: https://wiki.ros.org/octomap
36. N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot

simulator," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154.
37. "ROS: RViz." [Online]. Available: https://wiki.ros.org/rviz
38. "PAL Robotics." [Online]. Available: https://pal-robotics.com
39. "GitHub: PAL Robotics." [Online]. Available: https://github.com/pal-robotics
40. J. Sawatzky, A. Srikantha, and J. Gall, "Weakly supervised affordance detection," in *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2795–2804.
41. A. Nguyen, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Detecting object affordances with convolutional neural networks," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2765–2770.
42. A. Roy and S. Todorovic, "A multi-scale cnn for affordance segmentation in rgb images," in *Proc. of the Europ. Conf. on Computer Vision (ECCV)*. Springer, 2016, pp. 186–201.
43. T. Luddecke and F. Worgotter, "Learning to segment affordances," in *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2017, pp. 769–776.
44. M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, "Rosplan: Planning in the robot operating system," in *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2015, pp. 333–341.
45. S. Krivic, M. Cashmore, D. Magazzeni, B. Ridder, S. Szedmak, and J. H. Piater, "Decreasing uncertainty in planning with state prediction." in *Proc. of the Intl. Conf. on Artificial Intelligence (IJCAI)*, 2017, pp. 2032–2038.
46. Z. Saigol, B. Ridder, M. Wang, R. Dearden, M. Fox, N. Hawes, D. M. Lane, and D. Long, "Efficient search for known objects in unknown environments using autonomous indoor robots," in *IROS Workshop on Task Planning for Intelligent Robots in Service and Manufacturing*, 2015.



Peter Regier studied at the University of Duisburg-Essen in Germany and finished his degree in Mechanical Engineering in 2014. Currently, he is a PhD student in the Humanoid Robots Lab at the University of Bonn. His research focuses on autonomous navigation, perception, human-robot-interaction, and educational technology. During his time in Bonn, he was engaged in developed of robotic software for several EU-funded projects.



Andres Milioto is a research assistant and Ph.D. student at the University of Bonn since 2017. He received his Electrical Engineering degree from Universidad Nacional de Rosario, Argentina in 2016, where he was best of his class. During this time, he was involved in several robotics projects for private companies, including the construction of a large-scale iron pellet stacker and software development for robotics arms in welding applications, in Argentina, Mexico, and Italy. Preceding this position in Bonn, he worked for iRobot (USA) on software development and hardware integration, developing behaviors and communication protocols for state-of-the-art, SLAM-enabled, consumer robots.



Cyrill Stachniss is a professor at the University of Bonn and heads the lab for Photogrammetry and Robotics. Before working in Bonn, he was a lecturer at the University of Freiburg in Germany and a senior researcher at the Swiss Federal Institute of Technology. Cyrill Stachniss finished his habilitation in 2009 and received his PhD thesis entitled “Exploration and Mapping with Mobile Robot” supervised by Wolfram Burgard at the University of Freiburg in 2006. From 2008-2013, he was an associate editor of the IEEE Transactions on Robotics, since 2010 a Microsoft Research Faculty Fellow, and received the IEEE RAS Early Career Award in 2013. Since 2015, he is a senior editor for the IEEE Robotics and Automation Letters.



Maren Bennewitz is professor for Computer Science at the University of Bonn, Germany, and head of the Humanoid Robots Lab. She got her Ph.D. in Computer Science from the University of Freiburg in 2004. Before she moved to Bonn in 2014, she was a Postdoc and assistant professor at the University of Freiburg. The focus of her research lies on robots acting in human environments. In the last few years, she has been developing several innovative solutions for robotic systems co-existing and interacting with humans. Among them are techniques for efficient navigation with humanoid and wheeled robots as well as for reliably detecting and tracking humans from sensor data and analyzing their motions.