

# Spatial scene representation and navigation in a mobile robot using RGB-D camera

Marek Jaszuk, University of Information Technology and Management, Rzeszów, Poland

Wojciech Pałka, Academic High School, Rzeszów, Poland

Michał Furgał, University of Information Technology and Management, Rzeszów, Poland

Dawid Darlak, Academic High School, Rzeszów, Poland

Janusz A. Starzyk, School of Electrical Engineering and Computer Science Ohio University, Athens, USA, and University of Information Technology and Management, Rzeszów, Poland

## Abstract

The paper presents an algorithm for creating 3D scene representation to be used in a mobile robot. It uses data collected with an RGB-D camera. The RGB scene view is further processed with semantic segmentation algorithm based on convolutional neural networks to obtain binary masks of the recognized objects. The masks are matched to the depth view of the scene, which gives point clouds reflecting 3D shape of objects. Then the point clouds are averaged to reduce all objects into single points. In this way we obtain minimalistic representation of a scene, which is easy for storing and processing even in case of very large and complex scenes. This scene representation is aimed to support a mobile robot in recognizing scenes, and finding locations. The presented solution is able to efficiently deal with incompleteness of information coming from the real world.

## 1 Introduction

The problem of 3D scene mapping has been addressed by many researchers. This task is of particular importance for autonomous robots that have to track in real-time their location and create an internal representation of the scene, the task which is also known as simultaneous location and mapping (SLAM)[1]. To create 3D map of a scene, we need to combine the RGB image data, and the spatial data. SLAM task become much easier after the launch of cheap RGB-D cameras. The most notable device from this category is Microsoft Kinect. When combined with the KinectFusion software package it allows to reconstruct surface model of the scene even in real-time [2]. The device can also be mounted on a mobile robot to support the scene perception [3]. The robotic systems are also frequently equipped with laser scanners, known as LIDAR-s.

There are two basic types of LIDAR: 2D and 3D. 2D devices are relatively cheap, and allow for acquiring the data from 360°. Unfortunately, the acquisition of the scene data by such devices is from a single plane only. The 3D scanners allow for acquiring the real 3D scene data, but they are much more expensive. In addition, most of the scanners currently available on the market are based on a mechanically rotated elements, which limits the speed of data acquiring. This speed is much slower, than the speed offered by the RGB-D camera like Kinect.

The Kinect is able to acquire the depth data from a few meters, while the laser scanners register data from much larger distances like hundreds of meters. Laser scanners became more popular after introduction of cheap solid-state scanners. A number of companies announces such products, but

very few are yet available. An example is the Livox family sensors. The speed of data acquiring (when compared to depth cameras) is still a limitation of such devices.

For autonomous robots, the geometrical reconstruction of the scene is usually not sufficient, because the robot needs deeper understanding of the scene, after semantic recognition of surrounding objects. In recent years we observed a significant progress in the field of computer vision thanks to the success of deep convolutional neural networks [4]. They allow to perform various tasks related to visual scene analysis. In order to recognize the semantic scene, in addition to recognizing the category of the object, the important task is to find the object location, which is implemented by identifying the rectangle surrounding the recognized object like in the Faster R-CNN [5]. Identifying the rectangle, however, is not always sufficient, because it does not reflect the shape of the objects. To obtain the precise information about the shape, it is necessary to create the binary masks matching the pixels belonging to each object, which is possible using e.g. Mask R-CNN [6]. For real-time systems, besides precision of results, we also have to take into account the speed of the computations. Using GPU acceleration is currently a standard approach. This however, is still not always sufficiently fast. The mentioned Faster R-CNN algorithm allows for processing about 7 frames per second (FPS) on a GPU. This is quite fast, but for real-time systems, larger frame rate is expected. The situation has been greatly improved by introduction of the fast algorithms of object detection, like YOLO [7], which allow for processing at the speed rate exceeding 40 FPS. In addition, the pixel-level object masking speed has been significantly improved [8].

The last thing that we have to discuss in effective robot navigation is the scene representation. There are two widely applied methods of 3D world representation: surface models and voxel models. Both of them are applied in robotics [9][10] for building scene representation. Both kinds of scene representation have their advantages and disadvantages. The main disadvantage of both of them is that they collect massive amounts of data, which are hard to handle. If the memorized environment is large and contains a rich variety of scenes, it is hard to recognize the scene of interest, and locate the robot within the world, unless an external location system, like GPS, is available. The speed of memory processing is also critical, if the robot needs to compare the perceived scene with the memory during its normal operation, i.e. in real time. The scene representation presented in this work is addressed to solve such problems.

In our approach the scene complexity is reduced maximally by representing objects as points. This is quite simple, but allows for storing sufficient information about the scene. The other, less important details might be used only temporarily when the robot perceives the scene, e.g. for collision avoidance, but they are not stored permanently in the memory.

The paper is organized as follows. In Section 2 we describe building local scene representation. In particular, we describe the experimental setup and the method of matching visually recognized objects to the depth data, and identifying point location of the objects in the local view of the robot. In Section 3 we describe the algorithm of transforming the local coordinates of objects to the global scene model and present test results. Section 4 presents the conclusions.

## 2 Building local scene representation

In this section we analyze the mechanism of building internal scene representation by a mobile robot exploring a 3D scene.

### 2.1 Previous work

We already presented the basic mechanism for creating the discussed kind of scene representation, but that work was related to an agent exploring a virtual world created using a game engine [11]. This simplified the problem, because the mobile agent could use the data delivered by the game engine used to create a digital world. When dealing with the real world data registered during mobile robot operation, such simplification is not possible, and we have to overcome all the difficulties coming from imperfect object recognition, incompleteness of information, and different kinds of noises. We also developed a method for comparing scenes, which allows for deciding if the scene was already memorized by the robot, or not [11]. The method is based on the distance matrix created for the scene, which is then compared to the memorized scene, and gives the similarity factor between both the scenes.

### 2.2 The experimental set

The robot that we use in our experiments is the Parallax Arlo mobile platform, which allows for easy mounting of additional equipment (**Figure 1**). On the top of the platform we mounted the RGB-D camera, which in our case is the Microsoft Kinect v2. This is our only sensor delivering information about the scene. The depth camera registers the data from a limited distance, while the RGB camera is not limited by the distance. Single shot taken by the robot from a particular location gives only information about the positions of scene elements which are in the range of the depth camera measurements. In most of the devices available on the market, this range does not exceed several meters.



**Figure 1** The Arlo mobile robot platform with the Kinect sensor used in experiments

To export the most computationally intensive parts of data processing, we extended our experimental set by a computational server equipped with GPU cards. Thus, the main robot data processing system is located on the server. A laptop placed on the platform receives the depth and RGB scene view from the sensor, and sends them to the server through local WiFi connection. After the data have been processed on the server and the robot steering system makes the decision about further robot operation, the appropriate signals are sent back to the laptop. Then the laptop sends the commands to the robot microcontroller, which transforms them into rotation of the robot wheels. The speed of data transmission in contemporary computer networks is not a significant limitation, thus the robot can operate without notable delays. The experimental advantage of moving computations to the server, is that we can preview in real time the internal states of the robot during its operation.

### 2.3 Locating objects in 3D space

On the server the RGB scene view undergoes processing using tools based on convolutional neural networks. This allows for recognition and location of a set of objects within the robot's field of view (**Figure 2**). The recognition and location is, however, not sufficient, because our approach assumes, that the shape of the object must be identified so that it will be easier to manipulate. There is a number of tools that can be used for this purpose, and we analyzed the available choices. Our initial choice was to use the Mask R-CNN [6]. To train the model we created our own dataset representing the objects that can be found inside our university building, like different kinds of furniture.



**Figure 2** The robot facing the scene

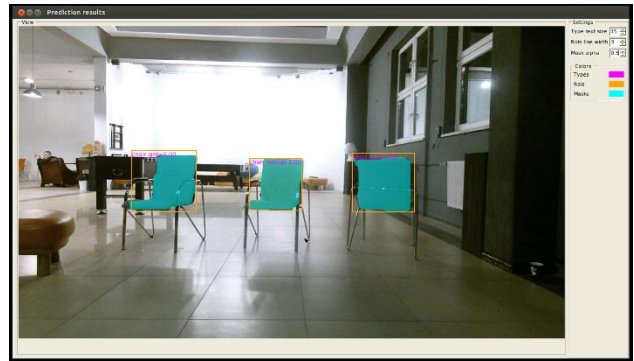
The Mask R-CNN itself allows for processing speed of about 5 FPS. But we have to take into account, that image masking, although the most computationally intensive element of the processing sequence, is not the only operation performed by the main robot steering system. Thus the real speed of the system operation was a bit lower - approximately 3 FPS.

Assuming that the robot is not going to move very quickly, this speed of operation could be accepted. The experiments showed however, that the results generated by the Mask R-CNN are not stable. The instability is revealed in the fact that in a sequence of images received from the robot, at least some objects are not always masked, despite that they are visible very well. For the robot perception, this means, that some objects appear and disappear. It is also possible, that the object changes its type between subsequent frames. This happens in case of some similarity of two categories. This kind of effects makes it quite hard to maintain a stable scene view. We were not able to eliminate this effect by longer training or delivering more training data.

We analyzed also alternative tools, and all of them demonstrated this kind of instability. The only solution to the problem, that we could find, was to cumulate the results from a sequence of images, which would make the scene

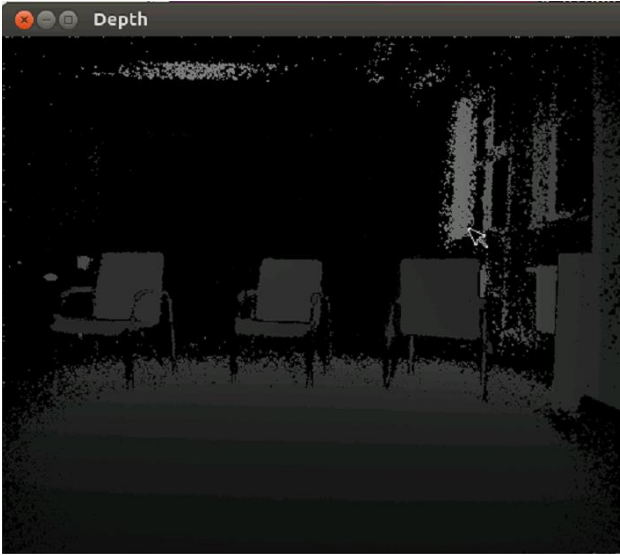
perception more stable. Such cumulation will cause, however, further slowing down of the sensory data processing. The improvement could be brought by, a faster masking algorithm, which still maintains stability of results, and quality of masks at least comparable to Mask R-CNN. It is not an easy task, because most of the available algorithms operate with speed not larger than a few FPS.

Our expectations were met by YOLACT [8]. The implementation of this algorithm allows for processing 30FPS with high quality of masks. In this way, we obtain the binary masks of the objects (**Figure 3**). Within the illustrated scene three objects were recognized: one orange chair and 2 gray chair (indicated by the text inside the surrounding rectangles). The algorithm still demonstrates the kind of instability that we discussed with Mask R-CNN. However, higher processing speed of YOLACT allowed for processing more frames in real time. Thus we accumulate the results from several subsequent frames to characterize the scene. If in a given location, an object is identified in more than 50% of the frames, it is assumed, that the object is there. Cumulating the results from 3 subsequent frames significantly improves stability of the recognized objects, but we used 5 subsequent frames to further reduce any instabilities.

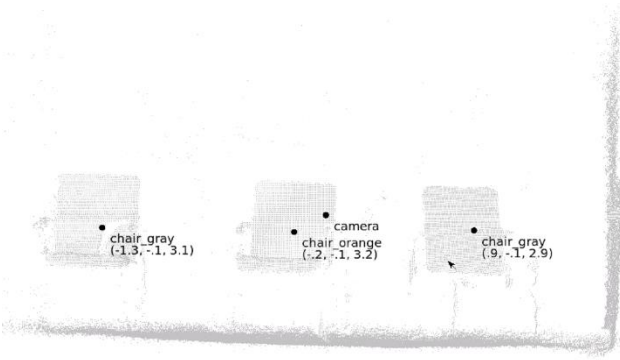


**Figure 3** The binary masks of objects obtained from YOLACT

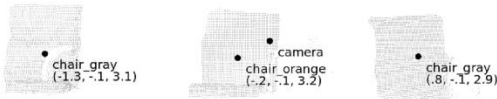
Besides the contours of the recognized objects, we need to determine their location in 3D space using information from the depth Kinect camera. The raw depth view obtained from Kinect is shown in **Figure 4**. Each pixel in the image represents the distance from the camera to respective element of the scene. Such a spatial data representation is not convenient for further computations. Thus we transform the depth data into a point cloud. The point cloud is equivalent of the depth image, but it represents not the distance from the camera, but the points in 3D coordinates with the origin in the camera location. The code that we implemented to do that is written using CUDA to accelerate the computation on a GPU. The point cloud obtained for the discussed scene view is presented in **Figure 5**. **Figure 5** indicates also the point locations of particular objects. To get them we match the binary masks from **Figure 3** to the point cloud. In this way we obtain a filtered point cloud (**Figure 6**). Then we average all the filtered points to obtain single point locations of individual objects within the local camera coordinates.



**Figure 4** The depth view of the scene



**Figure 5** The frontal point cloud view



**Figure 6** The filtered frontal point cloud view

We use the coordinate system typical for 3D graphics, where the z axis is pointing towards the scene, the y axis is pointing up, and the x axis is pointing to the right. The camera location is (0, 0, 0). The locations of objects in the camera coordinates are the basis for building the global scene memory.

As one can see in **Figure 3**, the masks generated for the chairs do not cover their legs. This is not due to limited

accuracy of the masking algorithm, but deliberate action. We prepared the training data for YOLACT in this way, to avoid including the chair legs. The reason for that is the depth data (**Figure 5**), where we can see that the legs of the chairs are mainly black. The black color means, that the location is undetermined, and in consequence, these elements are mostly invisible in the point cloud (**Figure 5**). Trying to include them would only increase fluctuations in the object locations. The reason is the metallic reflective material, which reveals the camera limitation. Also very thin elements are problematic because of the depth camera resolution. We believe, that the new Azure Kinect will improve the situation due to higher depth resolution, and we are going to try it soon.

The obvious limitation of determining point locations of objects in a scene is that we don't know the whole shape of the object when we see it only from a single robot position. The position of the same object, can be different, when it is seen from another perspective. The robot while exploring the scene registers new views all the time, and can discover new, previously invisible, parts of the already identified objects. Thus the position of the object is not something fixed, but it must be updated along with the scene exploration. This would lead to instability of the robot memory, which is undesirable. Thus we developed an efficient way of stabilizing locations of objects in the memory, which is discussed in the next section.

### 3 Global scene representation

The scene view obtained from a single robot position is not suitable for memorizing the scene, because the robot changes its location all the time, and the scene representation should be independent of the temporary point of observation. Thus we have to transform the positions of the objects to some global space.

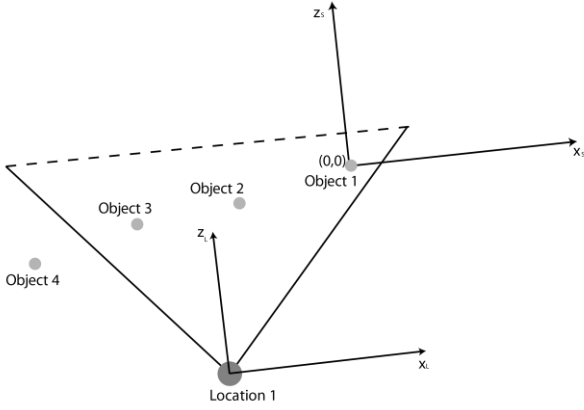
#### 3.1 Transforming local into global coordinates

In the real world there is no such thing, as global coordinates. Everything that we see, is seen from relative first person perspective. But the memorized mental model of the world is a combination of numerous local scenes (episodes), which are combined into global memory, forming a hierarchical structure. The approach to structure the robot memory should be similar. In this paper we will focus on constructing global memory within a single scene, and this approach can be extended to more scenes and complex worlds later.

If there are no global coordinates, that means, that the coordinate system is arbitrary. The only requirement, is that it should be stable. But everything can move, so there are no fully reliable reference points. Our approach assumes, that every part of a scene can be a reference to the other parts of the same scene, or even other scenes. The location of an object within a scene can be determined, if the robot perceives other, previously located reference objects within the same scene.



As demonstrated in Section 2, a single robot position allows for recording local object coordinates. Now it is time, to make the coordinates independent of the temporary position of the robot. Let us assume, the scene looks like in **Figure 7**. The triangle shows the depth camera view and range (dashed line). The image is only schematic, because there is no precisely defined range of the depth camera, as it depends on the surfaces visible in the scene. Within the camera view we have three visible objects: O1, O2, and O3. Thus **Figure 7** is the equivalent to the situation presented in **Figure 3**, where the robot can see three objects.



**Figure 7** The sample scene seen from Location 1 (top view) – the first step of scene memory building

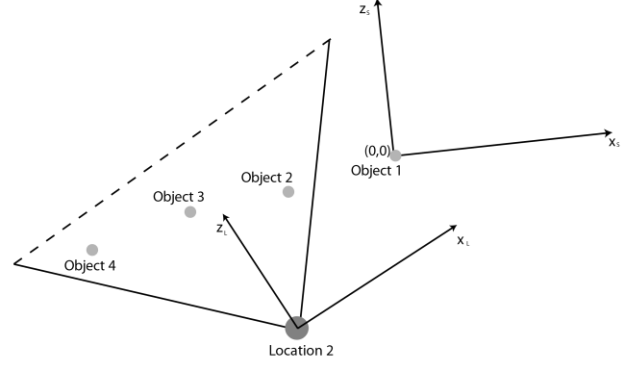
In the first step of its operation, the robot starts building its global scene memory, which is based on the global coordinate system. The global scene coordinates have the same axes as local, but the difference is that they are associated to the scene in a permanent way. The origin of the global scene coordinates can be located in arbitrary point, like Object 1 in **Figure 7**. The scene coordinate axes ( $x_s, y_s, z_s$ ) are parallel to the local camera view axes ( $x_L, y_L, z_L$ ). Computing the scene coordinates of all visible objects is straightforward, given their local coordinates. It is enough to subtract from the local coordinates of an object the local coordinates of the origin of the global scene coordinates

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{O_S} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{O_{L1}} - \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{(0,0,0)_{L1}}. \quad (1)$$

Also the camera location in scene coordinates can be easily found by putting:  $\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{O_{L1}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

To simplify further considerations, and easily explain the idea of memory building, we have to take into account the technical limitations of the experiment. The main assumption is that the robot travels on a flat surface, and the camera is fixed to the robot, so the robot cannot change the angle of observation independently of its movement. This assumption leads to the situation, where the point of observation moves only in 2 dimensions, although the world it sees is 3D. We also assume that the y coordinate is constant, and movement of the robot can be expressed using only 2 cartesian coordinates ( $x_s, z_s$ ).

Let us now assume, that the robot moves from the original position L1 to a new location L2 (**Figure 8**). What has changed is that a new object appeared in robot's field of vision (Object 4). We know its local position obtained directly from the camera measurement. However, we do not know, how to determine its position with respect to the scene coordinates. Moreover, the scene origin in **Figure 8** is no longer visible. But this is not a problem as long, as the robot sees at least 2 objects, with known scene positions. In our case these are Object 2 and Object 3.



**Figure 8** The view of the scene after robot moves to a new location

To determine the scene coordinates of a new object, we have to know, how the position of the robot has changed. We find that information by determining the relative robot motion with respect to the scene objects. To do that, it is convenient to describe the motion using polar coordinates, by combining robot motion from rotation, and translation. The scene coordinates of a new object are calculated by transforming the Location 2 coordinates into scene coordinates, as follows:

$$\begin{bmatrix} x \\ z \end{bmatrix}_S = R \begin{bmatrix} x \\ z \end{bmatrix}_{L2} + T - \begin{bmatrix} x \\ z \end{bmatrix}_{(0,0)_{L1}}, \quad (2)$$

where  $R$ , and  $T$  are the rotation translation matrices of the robot moving between L1 and L2. Explicitly this can be written as:

$$\begin{bmatrix} x \\ z \end{bmatrix}_S = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}_{L2} + \begin{bmatrix} t_x \\ t_z \end{bmatrix} - \begin{bmatrix} x \\ z \end{bmatrix}_{(0,0)_{L1}} \quad (3)$$

What we have to find, is the rotation, and translation parameters. To solve the problem, we move to the homogeneous coordinates, which leads to a single transformation matrix:

$$\begin{bmatrix} x \\ z \\ 1 \end{bmatrix}_S = T_R \begin{bmatrix} x \\ z \\ 1 \end{bmatrix}_{L2}, \quad (4)$$

or explicitly:

$$\begin{bmatrix} x \\ z \\ 1 \end{bmatrix}_S = \begin{bmatrix} \cos\theta & -\sin\theta & t_x - x_{(0,0)_{L1}} \\ \sin\theta & \cos\theta & t_z - z_{(0,0)_{L1}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \\ 1 \end{bmatrix}_{L2}, \quad (5)$$

where  $\theta$  is the rotation angle of the local robot coordinates with respect to the scene coordinates. This is a single

matrix equation, with 4 parameters to be determined. A single data point is insufficient to solve eq. (4), but when we combine equations for 2 points (objects  $O2$  and  $O3$  in **Figure 8**), the equation system becomes solvable:

$$\begin{cases} \begin{bmatrix} x \\ z \\ 1 \end{bmatrix}_{O2_S} = T_R \begin{bmatrix} x \\ z \\ 1 \end{bmatrix}_{O2_{L2}}, \\ \begin{bmatrix} x \\ z \\ 1 \end{bmatrix}_{O3_S} = T_R \begin{bmatrix} x \\ z \\ 1 \end{bmatrix}_{O3_{L2}} \end{cases}, \quad (6)$$

To solve the system of equations let's express them in the following form (4 equations with 4 unknowns):

$$\begin{bmatrix} x_{O2_S} \\ z_{O2_S} \\ x_{O3_S} \\ z_{O3_S} \end{bmatrix} = \begin{bmatrix} \cos\theta x_{O2_{L2}} - \sin\theta z_{O2_{L2}} + t_x - x_{(0,0)_{L1}} \\ \cos\theta z_{O2_{L2}} + \sin\theta x_{O2_{L2}} + t_z - z_{(0,0)_{L1}} \\ \cos\theta x_{O3_{L2}} - \sin\theta z_{O3_{L2}} + t_x - x_{(0,0)_{L1}} \\ \cos\theta z_{O3_{L2}} + \sin\theta x_{O3_{L2}} + t_z - z_{(0,0)_{L1}} \end{bmatrix}, \quad (7)$$

or as a product:

$$\begin{bmatrix} x_{O2_S} \\ z_{O2_S} \\ x_{O3_S} \\ z_{O3_S} \end{bmatrix} = \begin{bmatrix} x_{O2_{L2}} & -z_{O2_{L2}} & 1 & 0 \\ z_{O2_{L2}} & x_{O2_{L2}} & 0 & 1 \\ x_{O3_{L2}} & -z_{O3_{L2}} & 1 & 0 \\ z_{O3_{L2}} & x_{O3_{L2}} & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta \\ \sin\theta \\ t_x - x_{(0,0)_{L1}} \\ t_z - z_{(0,0)_{L1}} \end{bmatrix}, \quad (8)$$

where  $\begin{bmatrix} x_{O2_S} \\ z_{O2_S} \\ x_{O3_S} \\ z_{O3_S} \end{bmatrix}$  are known coordinates, because we

determined them in the previous iteration, and  $\begin{bmatrix} x_{O2_{L2}} \\ z_{O2_{L2}} \\ x_{O3_{L2}} \\ z_{O3_{L2}} \end{bmatrix}$  are

also known because they are determined locally. To get the

desired parameters  $\begin{bmatrix} \cos\theta \\ \sin\theta \\ t_x - x_{(0,0)_{L1}} \\ t_z - z_{(0,0)_{L1}} \end{bmatrix}$ , it is enough to find the

inverse of the transformation matrix, and multiply it by the vector of known scene coordinates:

$$\begin{bmatrix} \cos\theta \\ \sin\theta \\ t_x - x_{(0,0)_{L1}} \\ t_z - z_{(0,0)_{L1}} \end{bmatrix} = \begin{bmatrix} x_{O2_{L2}} & -z_{O2_{L2}} & 1 & 0 \\ z_{O2_{L2}} & x_{O2_{L2}} & 0 & 1 \\ x_{O3_{L2}} & -z_{O3_{L2}} & 1 & 0 \\ z_{O3_{L2}} & x_{O3_{L2}} & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_{O2_S} \\ z_{O2_S} \\ x_{O3_S} \\ z_{O3_S} \end{bmatrix}, \quad (9)$$

The inverse of the transformation matrix can be found algebraically or numerically. In our experiments we use the numerical solution. Given the transformation parameters, we can compute scene coordinates of all of the newly seen objects using eq. (5), which transforms the local coordinates of objects to their scene coordinates.

The two objects in the global coordinates are the minimal number of objects that must be used to determine the robot transformation parameters. In case of more objects, previously located in the scene coordinates and present in the field of view, all of them can be used to increase reliability of the computation and reduce the solution error that may result from imperfect measurements. We can take pairs of such objects, determine the transformation parameters from each of the pairs, and average the results or combine all previously located object coordinates to obtain one

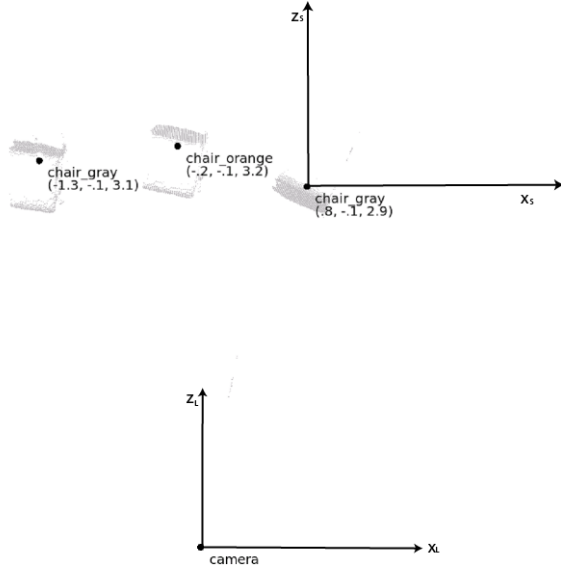
overdetermined system of equations. Solution of such system yields the smallest least square error for given set of measurements.

The limitation of the method, to some degree, is the requirement, that the robot needs to have at least two objects within the field of view, to be able to determine its own transition within the scene, as well as locations of new objects. To satisfy this requirement, the masking algorithm should be trained to recognize as many object types as possible. But still there is a chance that the robot, can stand in front of a wall, and lose any recognizable objects from its view. In this situation, the robot should start exploring the scene in a random way, to find any recognizable object. Then there are two possible cases. The first one is that the scene the robot sees, is already memorized one. In this case the robot is able to recognize its location within the scene, and continue the exploration of the known environment. The other case is the situation, where the robot does not recognize the visible combination of objects. In this case, it should build the new scene memory. It is possible, that it will find the relation between the two scenes later. Similar situation is when the robot is switched off to break the experiment, moved to some other location, and then switched on again to start the experiment. The robot's memory is maintained, because it is kept on the server, and the robot only connects to the memory when it is running. In consequence of such approach, the robot spatial memory consists of a number of smaller scenes. The relation between different parts of memory can be found at later time, when the robot finds object combination on the boundary of two scenes. Actually, in such cases, the coordinates of objects in one scene could be recalculated to make all objects match the common coordinate system. This makes sense if there are many points shared between the scenes, like objects in the same room. In case of very few contact points (like in case of separate rooms), it is more convenient to leave both scenes separate and only memorize the relationship between scenes. The detailed discussion of this issue goes beyond the scope of this article, and will be discussed separately.

## 3.2 Experimental results

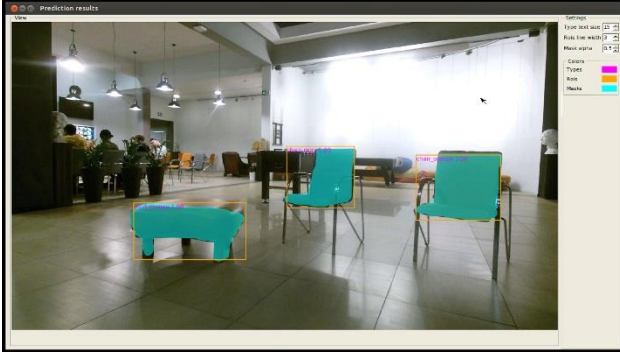
Now we will illustrate the presented considerations with the results of an experiment. In **Figure 3** we presented the initial scene view with masked objects. The respective top view of the point cloud, filtered using the masks, is presented in **Figure 9**. In this iteration the robot locates the scene coordinate origin in arbitrary object. In our case this is a gray chair visible to the right ( $O1$ ). The axes of the scene coordinate system are parallel to the local camera coordinates.

The robot starts traveling across the scene. Its motion is a combination of rotation and translation. The next view that the robot registered is shown in **Figure 10**. What has changed within the scene view is that a new object was identified (orange pouf), and one of the previously seen gray chairs is no longer within the field of view. This kind of scene view change was illustrated in **Figures 7** and **8**.

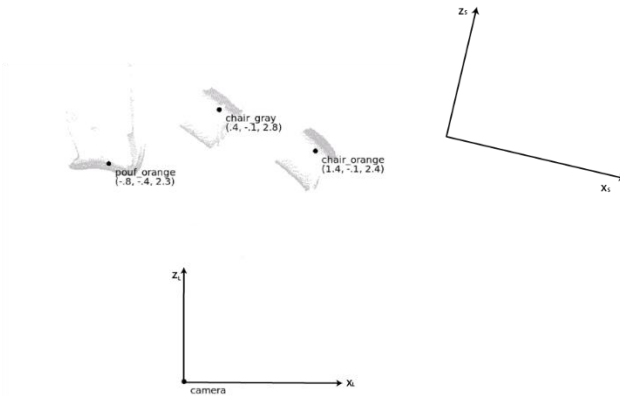


**Figure 9** The top view of the point cloud – initial robot location

Let us see the top view of the point cloud of the second view (**Figure 11**). The goal of the robot is to locate the orange pouf within the scene coordinates. The scene coordinate origin is no longer visible, but it is enough that the scene position of gray chair and orange chair are already known.



**Figure 10** The second scene view with masked objects



**Figure 11** The second scene top view with masked objects

Let us have a closer look at the calculations performed during the presented experiment. In Location 1 (**Figure 7**), the

coordinates of objects of interest are the following:  $O1_{L1} = [0.84, -0.13, 2.96]$ ,  
 $O2_{L1} = [-0.23, -0.17, 3.21]$ ,  
 $O3_{L1} = [-1.31, -0.17, 3.17]$ .

These are the local coordinates expressed in meters. Making them global is easy, because it requires subtracting the coordinates of object that serves as the origin of the scene coordinates. In our case this is the  $O1$  object. In this way we get scene coordinates of objects:

$O1_S = [0.0, 0.0, 0.0]$ ,  
 $O2_S = [-1.07, -0.04, 0.25]$ ,  
 $O3_S = [-2.15, -0.04, 0.21]$ .

Then the robot moves to Location 2, and our goal is to determine the scene coordinates of object 4. For this purpose we need the local coordinates of objects 2 and 3 in L2:  
 $O2_{L1} = [1.48, -0.17, 2.42]$ ,  
 $O3_{L1} = [0.47, -0.17, 2.80]$ .

Next we find the solution of Equation 9, which yields the vector specifying the transformation parameters:

$$\begin{bmatrix} \cos\theta \\ \sin\theta \\ t_x \\ t_z \end{bmatrix} = \begin{bmatrix} 0.92 \\ 0.39 \\ -0.66 \\ 0.40 \end{bmatrix}, \quad (10)$$

From the above we can find the rotation angle of the robot between L1 and L2g, which is  $23^\circ$ . The local coordinates of Object 4 recorded in L2 are:  $O4_{L2} = [-0.81, -0.47, 2.35]$ . After applying equation 5, with parameters 10, we get the following scene coordinates:  $O4_S = [-3.16, -0.30, -0.70]$ .

### 3.3 The system efficiency

The computer that runs all the processing of data delivered by the robot is equipped with two Intel Xeon CPU E5-2697, which allow for running up to 72 processes simultaneously. It is also equipped with 3 GPU cards: NVIDIA RTX 2080ti, TITAN Xp, and TITAN X. The most important factor influencing the speed of processing is the possibility of accelerating the computational tasks on the GPU-s. The element that consumes most of the computational power is the YOLACT masking algorithm. The time of processing a single frame from the camera on the RTX device takes about 33ms. The other elements of the computational chain were also implemented using CUDA to use the GPU acceleration. The most important tasks are: transforming depth data into point cloud, filtering the cloud with the masks, and computing the averages. Actually, we are using 2 GPU-s. While on one GPU the operations related to the point cloud processing are performed, the other GPU computes the masks for the next frame. Sequential processing of these two tasks on a single GPU would cause additional delay. The multiple CPU power is used only in a minor way.

In consequence of the presented approach to computations, we are able to process each scene about 15 times per second, which includes the masking, and the depth data processing. For the purpose of the experiment, this is very fast. It should be remembered, however, that we took the approach of averaging the results of masking to increase the reliability of masks. Thus we process only one depth image

per several RGB images. Assuming that we take 5 images to the average, the robot is able to process the scene information about 5 times per second.

## 4 Conclusions

We demonstrated a method for building spatial scene memory. The advantage of this method is minimal requirement with respect to the storage, because the objects are represented as single points. This kind of representation allows also for quick real time searching and comparison of different scenes, even if the scenes are large in terms of the number of objects. In this way, the robot is able to find the relation between what it sees, and what is registered within the scene memory. The whole robot memory is built as a collection of different scenes, which can be combined, when the robot finds relation between them.

The presented solution was demonstrated for a limited case of a robot moving on a flat surface, which results in a constant point of observation, and simplifies the computations. The solution however can be extended to arbitrary kind of motion in 3D. The only difference in case of a robot moving freely in 3D is that it has to follow at least 3 objects to be able to assess its own motion (our simplified 2D case requires only 2 objects).

The hardware solution and the code that we developed, heavily relies on using GPU for acceleration. This allowed us to achieve high processing speed, which is sufficient for performing experiments with the current solution, further extending the functionalities of the robot.

The method relies on an external tool, which is the masking algorithm. Despite huge progress in this field, such masking algorithms have their limitations, which were partially discussed in the paper. Considering the effort taken to improve this kind of tools all around the world, the situation should improve in the future.

There is a number of things that have to be done in the future. We need to develop a path finding algorithm. After analyzing the known algorithms we are not satisfied with them because in our case the scene representation requires developing different methods for navigation. The other thing is the motion within the scene. The current solution assumes that everything is static, but in real scenes there are also moving elements. We also want to develop a communication system with the robot using the speech channel. The weak part of the methodology, when we want to consider the robot an autonomous learning system, is that the training data are prepared manually. Ultimately the robot should be able to distinguish the parts of the scene itself, and learn from the object's manipulation. Moreover, we want to use the robot to perform a Motivated Learning scenario, where the robot will be able to perform some actions, and find the best strategies to satisfy its needs [13]. This of course requires extending the robot with a manipulator.

## Acknowledgement

This work was supported by the grant from the National Science Centre DEC-2016/21/B/ST7/02220.

## 5 Literature

- [1] Cadena, C.: Carlone, L.: Carrillo, H.: Latif, Y.: Scaramuzza, D.: Neira, J.: Reid, I.: Leonard, J.J.: Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*. 32 (6): 2016, pp. 1309–1332
- [2] Newcombe, R.A.: Izadi, S.: Hilliges, O.: Molyneaux, D.: Kim, D.: Davison, A.J.: Kohi, P.: Shotton, J.: Hodges, S.: Fitzgibbon, A.: KinectFusion: Real-time dense surface mapping and tracking. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, Basel, Switzerland, 26–29 October 2011, pp. 127–136
- [3] Endres, F.: Hess, J.: Sturm, J.: Cremers, D.: Burgard, W.: 3-D mapping with an RGB-D camera. *Transactions on Robotics* 30, 2014, pp. 177–187.
- [4] Goodfellow, I.: Bengio Y.: Courville A.: *Deep Learning*, MIT Press, 2016 <http://www.deeplearning-book.org>
- [5] Ren, S.: He, K.: Girshick, R.: Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016 <https://arxiv.org/abs/1506.01497>
- [6] He, K.: Gkioxari, G.: Dollár, P.: Girshick, R.: Mask R-CNN, 2017 <https://arxiv.org/abs/1703.06870>
- [7] Redmon, J., Divvala, S.: Girshick, R. Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection, 2015 <https://arxiv.org/abs/1506.02640>
- [8] Bolya, D.: Zhou, C.: Xiao, F.: Lee, Y.J.: YOLACT: Real-time Instance Segmentation, ICCV, 2019
- [9] Trevor, A.J.B.: Rogers, J.G.: Christensen, H. I. Planar Surface SLAM with 3D and 2D Sensors. In *IEEE International Conference On Robotics and Automation*, St. Paul, MN, 2012
- [10] Kim, B.: Kohli, P.: Savarese, S.: 3D Scene Understanding by Voxel-CRF, *The IEEE International Conference on Computer Vision (ICCV)*, 2013, pp. 1425–1432
- [11] Jaszuk, M.: Starzyk, J.A.: Building Internal Scene Representation in Cognitive Agents, *Knowledge, Information and Creativity Support Systems: Recent Trends, Advances and Solutions*, Andrzej M.J. Skulimowski, Janusz Kacprzyk eds, in series: *Advances in Intelligent Systems and Computing*, 364, Springer, 2016, pp. 479–491
- [12] Schwarz, M.: Behnke, S.: Semantic RGB-D Perception for Cognitive Service Robots, In: *RGB-D Image Analysis and Processing*, Springer, 2019, pp. 285–308
- [13] Starzyk, J.A.: Graham, J.: MLECOG - Motivated Learning Embodied Cognitive Architecture, *IEEE Systems Journal*, vol. 11, no. 3, 2017, pp. 1272–1283