Simultaneous Localization and Mapping with the use of RGB-D Image Processing

Balázs Vecsey*, Dániel Takács*, Zoltán Vámossy*
*Obuda University/John von Neumann Faculty of Informatcs, Budapest, Hungary
vbstudio.vecsey@gmail.com, daniel.takacs2569@gmail.com, vamossy.zoltan@nik.uni-obuda.hu

***Abstract*— For a mobile robot to be able to navigate in an unknown environment it must create a model of that area, but also it has to be able to place itself in that model, and do these things at the same time. This process that consists of these two problems is called Simultaneous Localization and Mapping. Those SLAM systems, that use image processing to gain information for their operations are called Visual Simultaneous and Localization systems. Our aim is to design and create a four wheeled mobile robot, which is able to create a 3D representation of its environment. Then this virtual representation can be freely explore and used for finding a path between two points.**

I. INTRODUCTION

There are a lot of situations in the field of robotics or in the field of image processing, where it is needed to have a 3D representation of the environment and to know the camera poses in a relatively short time. For example a robot to be able to navigate between two points it is important to know its location in the world that it moves in. This is a classic problem in the world of mobile robots, because to localize the robot (camera) in the world, one has to have a model of this world. [8] The same thing is true backwards too, because for the previously mentioned model we need to know the location of the camera.

Basically in an outdoor environment this can be solved relatively simple with the help of GPS. Nowadays with these devices the location of it can be easily estimated with high accuracy in a World size model. However if we move into an indoor environment we have to use a completely different approach, because in these environments one cannot make use of GPS devices or can, but with really low accuracy. Therefore at indoor environments we have to use other tools and solutions to have a SLAM system that is correctly functioning.

The biggest problem that occurs with SLAM system is the uncertainty that comes with the sensor measurements. These are because of measurement errors or because of the technical limits of the chosen device, devices. Probability models are widely used to handle these errors and uncertainties and for efficient estimations [8]. These models generally work with sensor measurements. The robot that is the subject of this paper uses a special camera. Seeing that the focus is on this camera, we cannot speak simply about Simultaneous Localization and Mapping, instead it is Visual Simultaneous Localization and Mapping (VSLAM) [6].

In the last few year lots of new devices showed up in the field of computer entertainment industries that made great effect on the field of digital image processing [14], robotics and mainly mobile robots.

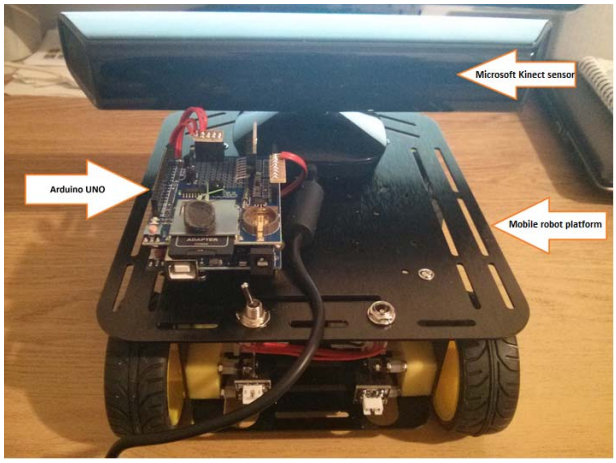

Figure 1. The mobile robot.

There are lots of devices that are more outstanding among these, because of their parameters. For example there is the Microsoft Kinect and the Asus Xtion sensor. These cameras are special, because they provide not only a RGB image, but a Depth (D) image too with the help of a special infrared point cloud [7]. This was a break-through because one had to use stereo camera systems or laser rangefinders to assign three-dimensional data to an RGB image and now it can be done with only one tool. The other advantage of these devices that they are available at a relatively low price, what is more their software support, documentations, etc. are good enough for everyone to start any kind of robot development.

Basically the aim of our project is to design and develop a four wheeled mobile robot, that can process the signals of an RGB-D sensor and from the gained information create a 3D representation of its environment. The finished model can be explored virtually without application and can be used to search for possible paths where the robot can go. The performance is not the highest priority task in this implementation. Considering the measurement errors and uncertainty, we aim to make the model approximate the real world as accurately as possible.

The paper presents a possible approach to solve the Visual SLAM problem. First we process the RGB-D images from the chosen sensor. This processing is multi-staged. First of all we determine the key points of the current frame. Then match and make pairs of these feature points on the previous and the current frames. After we made these pairs we estimate a relative transformation between the two images. The estimation

of this relative transformation is pretty straightforward, because we have depth information beside the RGB data. We construct our 3D model with the use of the RGB-D images and the previously estimated relative transformations. At the income of every new frame based on the existing information we scan for closed loops. Finally the produced data can be passed to a graph optimization framework (such as TORO or g^2o) and the measurement uncertainties can be handled and the program can update the 3D representation.

II. Related Systems

Nowadays several systems exist which deal with the problem of simultaneous localization and mapping. There are examples for these in [1]. It is noticeable, that each of these systems are following a similar development curve that is why we want to give a general view of these in this chapter.

In the referred systems the authors estimate the information of the robot movement from RGB-D images mainly. There are some situations where they use other devices or sensors too to support the RGB-D sensor for higher accuracy. For example, in those cases where the mobile robot rolls on wheels on the ground odometry data can be gained from the turn of the wheels to support. To gain the movement information first of all they extract image features from the incoming frames. When choosing an image feature extraction method we have to take some parameters into account. The method should be invariant to scaling, rotation, translation and lighting, because these can change while the robot is moving. The SIFT [10], SURF [9] and ORB [11] methods are suit these requirements good enough. Of course the decision is highly affected by the needs of the system performance.

After the extraction and matching of image features the next step is to estimate a relative transformation between the two frames. This estimation highly depends on the chosen sensor. In systems where they use stereo cameras depth information can be attached to the image feature points with epipolar geometry. Those systems that use RGB-D sensors are more effective than the previously mentioned ones, because here depth data is available right away. In this case the estimation of the relative transformation consists of transforming the points from one point cloud to another. It is independent of this estimation that the model is 2D or 3D, because the determined relative transformation can be used in both cases.

One of the main problems of SLAM is the closed loop detection [8]. There is a closed loop (loop closure) when the robot after some movement gets back to a place where it was before. For this detection the previously extracted image feature points can be used. An obvious approach can be a naïve implementation which is as follows. One can match the current frame's feature points with all the previously stored feature points every time there is a new frame. If the number of matched points exceeds a given threshold, then there is probably a closed loop. The drawback of this approach is the bigger the area the robot explored, the bigger dataset it has to go through to match the current frame. This method can be developed to only match with images which are the oldest, because the probability of a closed loop is higher at an older area than a newer one.

Last, but not least these systems handle the uncertainties of the measurements in different ways. There are a lot of different approaches to this problem. Filtering or online SLAM is one of the main SLAM paradigms, where these systems widely make use of the Kalman filter or its modified method, the EKF [8]. There is another paradigm that belongs to the main SLAM paradigms too which is smoothing or offline SLAM. In these cases pose graphs are widely used to handle the uncertainties. A pose graph consists of nodes and edges where the nodes are the robot's orientation and location at a given point in time and edges are the relative transformations between these nodes. Up against the online implementations, offline SLAM systems use all the information that is available at once. On the other hand online SLAM systems have an incremental nature, so they use only the information that are currently available.

III. The Applied Approach and its Results

In this section we take a close look at our own implementation and the current results. Fig. 2 shows a detailed flowchart of the system.

First we process the input RGB and D images. Features are extracted from the images using the Speeded Up Robust Features (SURF) algorithm [9]. This one proved to be the most efficient, as opposed to the SIFT and ORB algorithms, which supports the test results stated in other papers [11]. These extracted features are frequently stored as the robot moves on (more on that later). Features from consequent images are matched for pairs, then these features are paired up with their corresponding depth values, if that does not exists, the feature is not used. Also a pair is only used when both points have depth values.

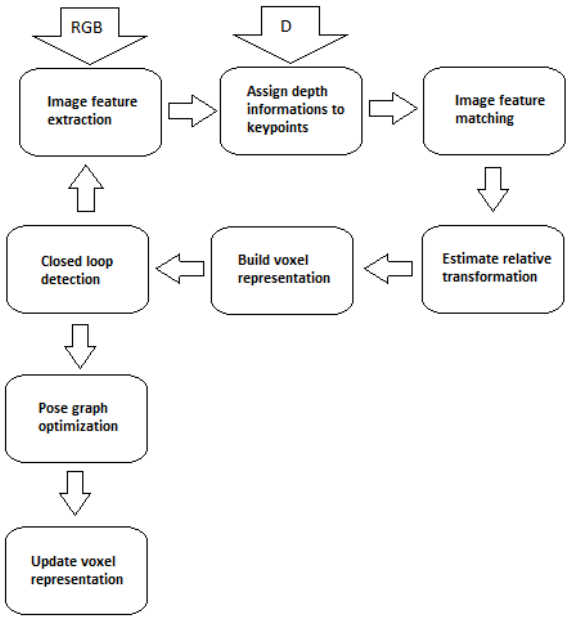

Figure 2. Image processing and SLAM flowchart.

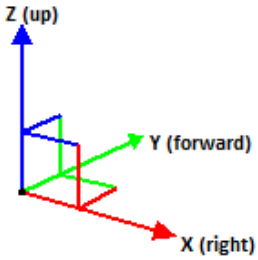

Figure 3. Direction conventions in our coordinate system.

Fig. 4 shows an example of this, where the red marks show the features without a depth value.

A. *Relative transformation*

To calculate the relative transformation, the previously extracted and matched feature pairs are used. Our 3D coordinate system uses the right hand-rule, with the direction conventions as seen in Fig. 3. The image's X, Y and depth coordinates have to be projected into this coordinate system. This is done by projecting the X axis as is the Y axis to the Z axis but mirrored, and the depth to the Y axis in positive direction [13].

The Kinect provides a horizontally mirrored image, so to save some processing time, instead of mirroring the actual bitmap image in the start, we only mirror that one axis during the projection by multiplying it by –1.

The following formula shows the exact method our coordinates are projected:

$$P_x = -2 \cdot \left(\frac{k_x}{w-1} - 0.5 \right) \cdot d \cdot FOV_x, \quad (1)$$

$$P_y = d, \quad (2)$$

$$P_z = -2 \cdot \left(\frac{k_y}{h-1} - 0.5 \right) \cdot d \cdot FOV_y. \quad (3)$$

where w and h are the dimensions of the input image, k_x and k_y are the current coordinates inside the image, and d is the current depth. With the P_x and P_z values, it first converts the coordinate to a minus one – plus one range. Because of the mirroring it is multiplied by –1, then multiplied with the depth value, and then by the corresponding field of view value of that axis. The P_y coordinates are simply the depth values without any further transformation. The fields of view (FOV) are two constant values. In our case – because we work with Kinect for Xbox360 sensors – these values are

$$FOV_x = \tan\left(\frac{57°}{2}\right); \; FOV_y = \tan\left(\frac{43°}{2}\right). \quad (4)$$

To determine the relative transformation between the two point clouds, we need at least three pairs. Of course with only three points the results are more affected by noise and false values. On the other hand, too many point pairs are also brings lots of noise because more points mean more extreme outliers. The golden mean is if we use a fair amount of pairs with inlier filtering (more on that later) [15].

The centroids of the point clouds are calculated with the following formula:

$$centroid_A = \frac{1}{N} \sum_{i=1}^{N} P_A^i. \quad (5)$$

$$centroid_B = \frac{1}{N} \sum_{i=1}^{N} P_B^i \quad (6)$$

When the centroids are subtracted from every point in their corresponding point clouds (A and B respectively), the results are two point clouds with their centroids in the same origin, only differing by their orientation. The easiest way to determine this orientation is with Singular Value Decomposition. SVD of an $m \times n$ real or complex matrix E is a factorization of the form $E = U \cdot S \cdot V^T$, where the m columns of U and the n columns of V are called the left-singular vectors and right-singular vectors of E, respectively, and the diagonal entries in S are known as the singular values of E. Because our input matrix is an $m \times m$ matrix, all three output matrices will be $m \times m$ as well.

For the orientation we also need a covariance matrix H, which can be determined the following way:

$$H = \sum_{i=1}^{N} \left(P_A^i - centroid_A \right) \left(P_B^i - centroid_B \right)^T \quad (7)$$

$$[U, S, V] = SVD(H) \quad (8)$$

$$R = VU^T \quad (9)$$

Each point is stored in a 3×1 vector (x, y, z). With this, we get a 3×3 rotation matrix which is the average

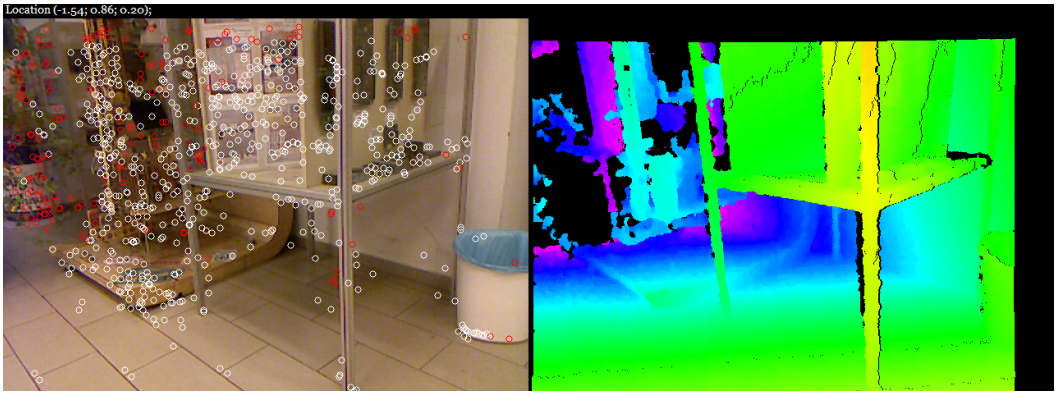

Figure 4. Extracted image features (left), depth image (right). Features without depth value are colored red.

Figure 5. A voxel scene constructed from 30 nodes (right) using 1cm^3 voxels. RGB image from one of the nodes for comparison (left).

rotation. There is a special case where sometimes the SVD produce a reflection matrix, which is numerically correct, but does not make any sense in reality. If the determinant of the rotation matrix R is smaller than zero, multiply the third column in V with -1, then recalculate R with these new values.

Now, determining the translation vector using the previous calculations is done this way:

$$t = -R \cdot centroid_A + centroid_B \tag{10}$$

With rotation matrix R and translation vector t determined, a transformation matrix can be made by simply putting these into a 4×4 matrix the following way:

$$tr = \begin{bmatrix} R_{00} & R_{01} & R_{02} & t_0 \\ R_{10} & R_{11} & R_{12} & t_1 \\ R_{20} & R_{21} & R_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{11}$$

Those feature points which are on the edge of an object might provide a false depth value. This and the general noise can degrade the quality of the transformation, and because of this we felt a need for an inlier filtering. Assuming we have two point clouds named A and B, both have the same number of points, and points with the same indices are pairs. We determined the relative transformation which projects the points from A to B with a certain amount of error. This projection is done by multiplying A with the rotation matrix and then adding the translation vector. The projected point is called B'. To filter out those pairs which differ too much from the average, check how far each B is from B', if it exceeds a certain threshold, the pair is considered an outlier and thrown away.

B. Closed loop detection

Closed-loop detection uses the same methods as stated above. When the robot determines relative transformation between two images, it does not go from the previous frame to the current frame, but from the last node to the current frame. A node is a previously saved frame, and a new node is only saved when the robot has moved or rotated by a specific amount. By doing so, it not just conserves a great deal of memory and storage space, but and also filters out any meaningless nodes where there is no transformation at all, so when the robot is stopped, no new nodes are added. Also the speed of the robot does not determine the node frequency. This significantly speeds up any future calculations, like the closed-loop detection.

In our case the conditions for creating a new node are either a relative distance from the last node by at least 30 cm or a relative rotation of at least 12 degrees in any direction. These thresholds were determined by trial and error. The initial values were 25 degrees and 40 cm, in this case, the overlap between the two images were too little, more likely producing an insufficient amount of features or no feature at all. To fix this, we started decrementing these values until finding the optimal 12 degrees and 30 centimeters, as stated above. It is important to add, that the actual rotation or translation (at least one of them) is always a little greater than the actual threshold, so the camera should not be moved too fast.

Detecting a closed-loop works the following way. Every saved node is in chronological order, each following the next one with a relative transformation. If we can find a match between two nodes outside of this order, like for example the last one, and one 20 nodes before, then we can assume the robot had circled back to a point it had visited before. Let's call these two nodes A and B, where A is the older node and B is the newer. Calculating a transformation matrix between A and B will result in a slightly different transformation than the one that accumulated through the relative transformations of all the nodes between them. This is due to the inaccuracy of the sensors and noise. Let us call this direct transformation m, and the newly suggested position of B based on this transformation B'.

We assume that this direct transformation is more precise that the accumulated one, so B must be in B'-s position.

$$B' = A \cdot m. \tag{12}$$

B can be easily moved to *B*', but all the nodes in between must slide along with it, like a branch of a tree. To do this, first determine the transformation matrix from *B* to *B*':

$$diff = B^{-1} \cdot B'. \tag{13}$$

Node *A* must stay where it is, node *B* must move by *diff*, and all the nodes in between must move proportionately from not moving (an identity matrix) to moving by *diff*, based on their relative location between *A* and *B*. Luckily for us, the SlimDX library which used for the 3D rendering has a built in function for correctly interpolating transformation matrices, we only need to calculate that relative location *d*, which is a number between 0 and 1.

$$d = \frac{i - A.index}{B.index - A.index} \tag{14}$$

$$node_i = node_i \cdot Lerp(I, diff, d). \tag{15}$$

Where $d \in [0,1]$, *i* is the node index, *I* is an identity matrix and *Lerp*() is the function that calculates the linear interpolation. After this, we must not forget about the nodes consecutive to *B*. They all must be transformed by *diff*, same way as *B*.

$$node_i = node_i \cdot diff. \tag{16}$$

C. *Route-finding in volumes*

Because our robot moves on the floor, with walls and drops equally counting as obstacles, if we can separate the floor from the other voxels, the route finding process can be reduced to a 2D problem. For finding the shortest route we use the Dijkstra's algorithm [12].

Separating the floor is easy: starting from the camera position and going downwards the first solid voxel we hit is a floor element. From there with wave front propagation the whole surface can be travelled. Since the voxel floor is never quite flat because of the noise and the imprecise joins, a threshold must be defined as to where the wave can jump or drop one voxel, and where consider it an obstacle. This gives a flat map of the floor, where the coordinates are in sync with the horizontal coordinates in the 3D space.

Dijkstra's shortest path algorithm uses a weighted graph, where the weight is the physical distance between two nodes, so it can find the shortest path on a graph of any shape (like a road map). In our case with the voxels only two kind of weight exists: horizontal or vertical where the weight is 1, and diagonal where the weight is $\sqrt{2}$. In practice these are used with integers instead, because of performance considerations, using 1000 and 1414.

The threshold for jumps and falls is one voxel which is perfect for 40mm^3 voxels.

Even with this low threshold, slopes with up to 45 degrees are still being considered as floor, so we added a new feature to our wave front propagation. When a wave front jumps or drops one voxel, it remembers this for the next 2 iterations, and if encounters any kind of jump or drop during this, it immediately considers it an obstacle. This is not affected by noise, but perfectly filters those slopes.

IV. CONCLUSION

To sum it up in this paper we described the problems of Simultaneous Localization and Mapping and what solutions are used in related systems in general. We presented our SLAM system, which works with our four wheeled mobile robot. The system we implemented is a Visual SLAM system, which estimates the information of the robot movement from RGB-D images and from the image feature points extracted from these frames. The representation that is made by our robot is a three-dimensional model, which is based on voxels. The system can detect closed loops and accordingly to these detections it corrects the 3D voxel model. The uncertainties which are caused by the sensor measurements are handled by a graph optimization

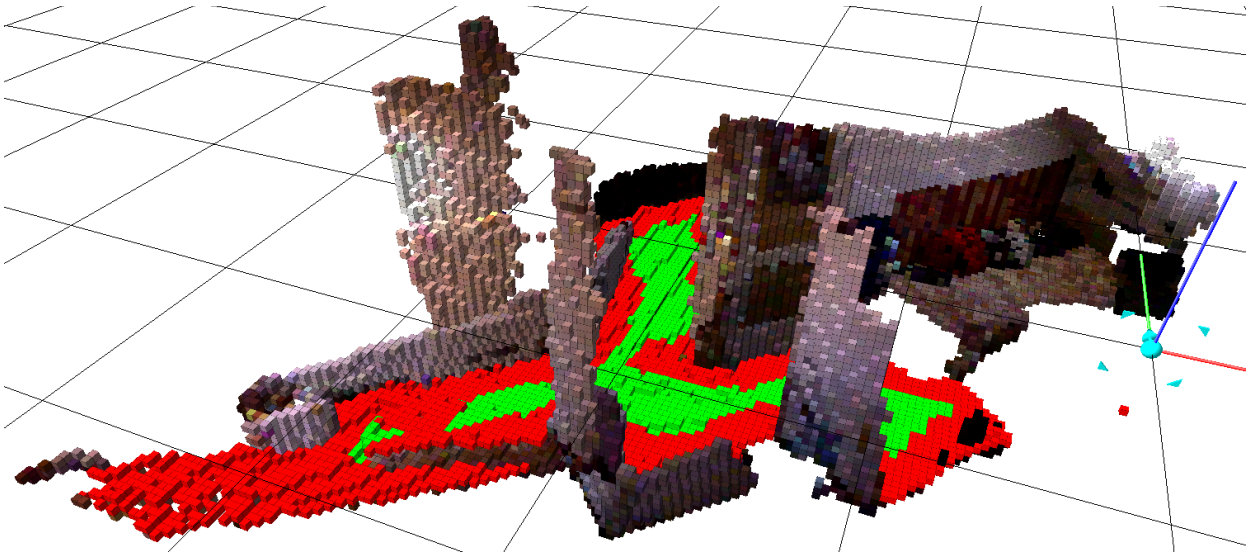

Figure 6. A more complex scene (bedroom and hallway) using 4cm^3 voxels, with the floor painted to green where the robot can safely move and red where it is too close to the wall.

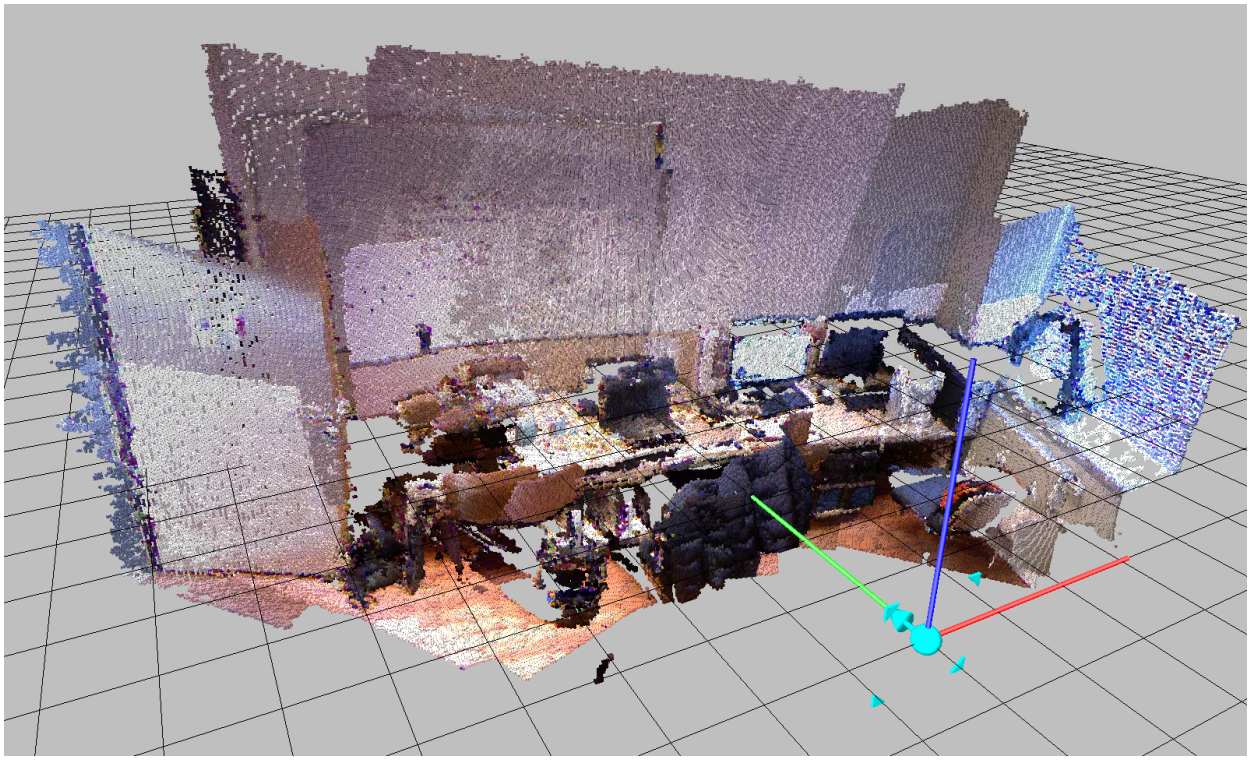

Figure 7. A voxel scene of the university's lab with 1cm^3 voxels. The patches where projection of different images meet is clearly visible.

framework and according to this the system updates and corrects the voxel representation.

The proposed system was tested at the open corridors of the university, where there were enough obstacles and features. Further tests were made in similar, but in more feature rich environments such as a bedroom (Fig. 6) and a small office (Fig. 7).

Of course we have further plans with the system and robot together. One of them is to explore as big areas as possible to estimate the limits of the current implementation. We would like to implement and try out online SLAM too and compare them to our current implementation. And last, but not least we want our robot to create a 3D representation of an unknown environment fully autonomously.

REFERENCES

[1] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, Dieter Fox: „RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments", Experimental Robotics, The 12th International Symposium on Experimental Robotics, Springer Berling Heidelberg, 2014. pp 477-491.

[2] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, Wolfram Burgard: „An Evaluation of the RGB-D SLAM System", Robotics and Automation (ICRA), 2012 IEEE International Conference on 14-18 May 2012, pp. 1691-1696

[3] Dominik Belter, Michal Nowicki, Piotr Skrzypczynski, Krzysztof Walas, Jan Wietrzykowski: „Lightweight RGB-D SLAM System for Search and Rescue Robots", Progress in Automation Robotics and Measuring Techniques, Advances in Intelligent Systems and Computing Volume 351, 2015, pp 11-21

[4] Dominik Belter, Michał Nowicki, Piotr Skrzypczyński: „On the Performance of Pose-based RGB-D Visual Navigation Systems", Computer Vision – ACCV 2014, Lecture Notes in Computer Science Volume 9004, 2015, pp 407-423

[5] Francisco Ferrer Sales: „SLAM and Localization of People with a Mobile Robot using a RGB-D Sensor", A dissertation presented for the degree of Master of Science in Electrical and Computer Engineering, 2014, University of Coimbra, p. 109

[6] Virgile Högman: „Building a 3D map from RGB-D sensors", Master's Thesis in Computer Science, KTH, Stockholm, Sweden, p. 69.

[7] L. Somlyai - Mobil robot localization using RGB-D camera, ICCC 2013, Proceedings of IEEE 9th International Conference on Computational Cybernetics. Tihany: IEEE Hungary Section, 2013, pp. 131-136.

[8] Bruno Siciliano, Oussama Khatib: „Springer Handbook of Robotics", ISBN: 978-3-540-23957

[9] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool: „SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346-359, 2008

[10] David G. Lowe: „Distinctive Image Features from Scale-Invariant Keypoints", *International Journal of Computer Vision,* 60, 2 (2004), pp. 91-110.

[11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski: „ORB: an efficient alternative to SIFT or SURF", ICCV '11 Proceedings of the 2011 International Conference on Computer Vision, Barcelona, pp: 2564-2571

[12] D. Stojcsics, "Autonomous waypoint-based guidance methods for small size unmanned aerial vehicles", *Acta Polytechnica Hungarica,* Vol. 11, No. 10. pp. 215–233, 2014

[13] A. Rövid, "Machine vision-based measurement system for vehicle body inspection," *Acta Polytechnica Hungarica*, Vol. 10, No. 5. pp. 145–158, 2013.

[14] Szabolcs Sergyán, "Parameter Optimization of Dominant Color Histogram Descriptor in Content-Based Image Retrieval Systems", Int. Journal of Circuits, Systems and Signal Processing, 8: pp. 286-291. 2014

[15] *Szénási S. Distributed region growing algorithm for medical image segmentation. INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING 8: (2014) pp. 173-181.*