2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

A Scene Graph Based Shared 3D World Model for Robotic Applications

Sebastian Blumenthal, Herman Bruyninckx, Walter Nowak and Erwin Prassler

Abstract— **This paper presents a novel approach for representing and maintaining a shared 3D world model for robotic applications. This approach is based on the scene graph concept which has been adapted to the requirements of the robotic domain. A key feature is the temporal and centralized sharing of all available 3D data in the leaves of the graph structure. The approach enables tracking of dynamic objects, incorporates uncertainty and allows for annotations by semantic tags.**

A demonstration is given for a perception application that exploits the temporal sharing of 3D data. A Region of Interest (ROI) is extracted from the stored scene data in order to accelerate processing cycle times.

I. INTRODUCTION

Autonomous robots need internal representations of their environment in order to be able to reason about and to interact with it. With the introduction of mobile manipulation systems the requirements for such representations grew to include full 3D world models that allow grasping objects of interest while avoiding potential collisions with the environment or with humans.

As the real world changes over time it is not sufficient to load environment models from prior knowledge only. Online sensor data must be processed and continuously incorporated into the world model of an application. This scene data is relevant for a variety of "users" like path planners, a task coordinator, motion control modules or perception functionality. Modern robot systems often make use of component-based implementations to realize these kinds of functionalities. Each component encapsulates a particular functionality and only communicates through a set of interfaces. Various robotics frameworks exist that support component-based development such as ROS [1], OROCOS [2] or YARP [3].

Components for planning, coordination, control, and perception tend to have their own internal world representation with custom data types, interfaces or mechanisms to retrieve the latest data. As a consequence parts of the world model are scattered over a variety of components and might remain unconnected or potentially hidden in the code. Such a situation can hinder the reuse of implemented functionality in another component. Also debugging such applications can be tedious as every representation might need to be monitored with a different mechanism. Furthermore each component might produce intermediate results during the data processing that are interesting for other components as well, but which has to be replicated in the other components without a coherent mechanism for sharing the data.

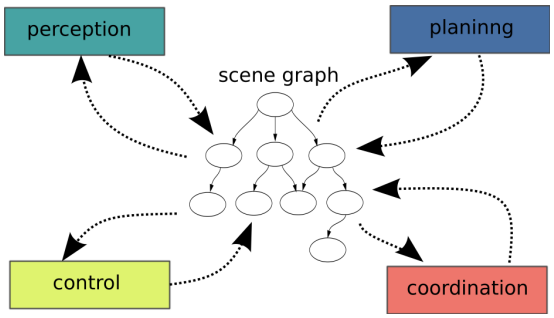

Fig. 1. Overview of a shared world model based on a scene graph. Multiple components can read and write data to the world model.

A better approach is a common and shared world model where each component has access to relevant parts of a scene as indicated by Fig. 1. In this context "relevant" means a certain geometrical subset, e.g. only the robot model and the object to be grasped or certain data at different time stamps. For instance a registration component might need some scene data from a previous processing cycle, or a tracking algorithm needs to know the history of poses of an object.

A unified approach to be able to share all available world model data with other components and for a certain period of time has not emerged in the robotics domain yet.

The problem is to define a shared approach for a complete 3D world model for robotic applications. The goal is to integrate planning, perception, control, and coordination with this world model. This paper describes our initial results to find such an approach. After revisiting related work in Section II, this paper will gather requirements in Section III-A and then present in Section III-B a novel approach for a scene graph based shared 3D world model. It will be illustrated in Section IV how the temporal sharing of data is beneficial to accelerate perception applications. The paper is closed with a conclusion in the last section.

II. STATE OF THE ART

The different needs of specific robotic applications have led to various world model representations. Often these

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics).

Sebastian Blumenthal is with GPS GmbH, Nobelstr. 15, 70569 Stuttgart, Germany, blumenthal@gps-stuttgart.de

Herman Bruyninckx is with the Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium. herman.bruyninckx@mech.kuleuven.be

Walter Nowak is with Locomotec UG, Werner-von-Siemens-Str. 6, 86159 Augsburg, Germany nowak@locomotec.com

Erwin Prassler is with the University of Applied Sciences Bonn-Rhein-Sieg, Grantham-Allee 20, 53757 Sankt Augustin, Germany, erwin.prassler@h-brs.de

representations are driven by the capabilities of the used sensors, ranging from rather sparse to dense geometric representations or from 2D to full 3D models. This paper distinguishes mainly between *grid-based* and *object-oriented* representations.

A. Grid-based representations

Grid-based representations essentially discretize the workspace in order to encode geometric entities. The most popular variant is the *occupancy grid* [4]. It is a 2D voxelized space with cells that can indicate whether they are free or occupied. The information to describe a state of a cell is a random variable allowing for a stochastic representation that can account for sensor noise. Measurements collected over time can increase or decrease the belief whether space is occupied or not. The approach assumes a ground plane which is typically only valid in indoor environments. For such indoor navigation applications it has been successfully applied [5]. The strengths of an occupancy grid are handling of uncertainty in the sensor readings as well as the ability to represent environments without prior knowledge about the occurring shapes. The grid representation allows for constant time access of the cells. On the other hand it is only an approximation of the workspace and can suffer from quantization errors. It is possible to also represent 3D environments in a 3D occupancy grid. This method can have limitations as 3D grids may require a huge amount of memory.

Octrees [6] address the memory issue. They are full 3D voxel representations based on a recursive workspace subdivision into eight cubes. Empty cubes are neglected and the model results in a tree-like structure of cubes, representing the volume of occupied regions. Compared to a simple 3D voxel grid memory requirements are much lower as the tree adaptively evolves over time.

The ideas of occupancy grid and the octree have been combined into the *Octomap* [7]. The advantage is the multi-resolution and compact representation that can handle dynamic changes as the cells are probabilistically updated. However, explicit encoding of dynamic objects including a pose and its uncertainty is not covered in this approach. A step towards explicit encoding of objects is presented in [8] where the authors propose to use multiple *Octomaps* referenced in a tree-like structure to form a hierarchical scene representation.

The *Collision Map* or *Dynamic Obstacle Map* [9] is a grid-based representation for mobile manipulation applications. It enables planning algorithms to check if a certain manipulator configuration collides with its environment. Each cell represents the occupied space within a sliding time window. If no sensor measurement is received for an occupied cell within a certain duration, the cell is considered to be free. The approach has been successfully applied to the PR2 robot [10] allowing it to recognize objects on a tabletop and to grasp them collision-free. The Collision Map can handle some level of dynamics in the environments. Perceived data can be improved by making use of the robot model as occlusion caused by a manipulator can be identified and removed from the data. However, uncertainty is not addressed explicitly. Robots in the experiments do not move and thus the robots' workspace is limited to the reachability of the manipulators. Being based on a simple 3D grid, scalability of this approach to larger environments might be an issue. The combination of the Collision Map with the Octomap approach seems to be a promising future research direction [10].

In general most grid-based approaches can only deal in a limited way with non-static environments. Real-world objects are solely represented as a set of occupied cells. Without an additional representation it is not clear which cell belongs to which object. This makes interpretation of a scene more difficult. Object tracking including pose prediction is not feasible with purely grid-based world models. The cells of occupancy grids can be seen as an representation accumulated over time, but they do not allow to explicitly access data from previous processing cycles.

B. Object-oriented representations

In contrast to grid-based approaches object-oriented representations do not capture free, occupied or unknown cells. Instead, detected objects or the environment surface are explicitly represented. Typically such representations encode the pose and the shape of objects.

Line maps have been developed for 2D indoor navigation applications. A map consists of multiple segments of lines to group laser scanner values belonging to even surfaces. The benefit is a reduced memory consumption in comparison to 2D occupancy grids. Also the resolution is not limited by an underlying grid. Though this approach imposes new challenges. Perception algorithms need to identify which point belongs to which line, thus an assignment problem must be solved. This problem also holds true for *Landmark-based maps*. This representation consists of salient landmarks which are typically composed of a pose, a feature descriptor and possibly information about the uncertainty of a landmark. Landmark-based maps are often used in vision-based navigation systems [11].

An even more compact representation used for navigation are *Topological maps*. They consist of nodes and edges. Nodes encode certain distinguishable places. Edges indicate the connectivity between the nodes. The *TF* library [12] stores relations between coordinate frames. It organizes the frames in a tree stucture and provides a temporal cache for the transform data. The tree structure imposes limitations on representing multiple hypotheses for a specific frame, for example for an object that is perceived simultaneously by multiple sensors. Landmark-based maps, Topological maps as well as the TF library do not contain full 3D geometric information of the environment. On the contrary, surfaces of geometric objects can be approximated by *point clouds*. They have the advantage that the original density distribution captured by 3D sensors can be preserved. Any geometric shape can be represented. This representation does not suffer from discretization errors like the 3D grid-bases approaches but the arbitrary resolution of a scene can result in an

unlimited memory consumption. Point clouds have been successfully applied to navigation applications [13] and are popular for various 3D perception algorithms [14].

Typically line, landmark, or topological maps as well as 3D point sets or 3D meshes form a world model consisting of only that type. In addition, some approaches exist that contain mixed or more complex representations. Here we will call such complex objects *scene objects*. For instance such a model could contain point clouds as well as task-relevant objects with a basic shape or a surface mesh.

One example for such more complex scene objects is the *Local Dynamic Map (LDM)* [15] which is a world model for autonomous vehicle applications. Environments can contain static elements like roads or traffic signs and highly dynamic objects like the vehicles. The LDM is essentially a database with explicitly modeled objects with a rather deep inheritance hierarchy to represent objects of different type like static, dynamic or conceptual (e.g. traffic jam). Each object has attributes, uncertainties and relevant relationships to other objects. This world model is able to explicitly represent dynamic objects. It is sensor independent, it is compact and it can focus on certain regions for instance with a moving horizon. Raw data cannot be stored in this approach.

The database-driven world model [16] can store spatial and temporal relations between objects including uncertainty. In contrast to the LDM this world model does not have an inheritance hierarchy to represent objects. Objects (or features) are represented by a type that could be either a point set or a set of lines or a poly-line, a name, the origin frame, a tag, a set of attributes, timestamps for creation and observation time and an error model. Coordinate transformations are captured in an oriented graph of transformation matrices including a 6-DoF error model. New information can be added by custom algorithms, so called *processes*. This approach seems promising, however it does not explicitly address when to delete data and how to maintain the objects in a hierarchical manner. An open source implementation is not available.

A 3D world model representation successfully used in the computer animation and computer games domain is the *scene graph* [17]. The scene graph uses a directed acyclic graph (DAG) to represent a scene by topological and spatial relations of objects. The DAG structure can be used to capture hierarchies by (recursive) grouping of the elements in a scene. A couple of standards emerged to describe scene graphs like PHIGS, VRML or COLLADA. There are concepts available for distributed scene graphs that can synchronize local copies between multiple clients [18]. Another notable feature is the ability to store geometric data in multiple resolutions. However scene graphs have not been established in robotic applications so far. Their focus is on rendering pre-modeled scenes, so they lack support of online sensor data processing as needed in robotics. Temporal caching of data and uncertainty representation are missing as well.

In contrast to grid-based world models the object-based representations excel at handling dedicated objects and thus enable tracking of them. Though, object-based world models require rather strong heuristics of possible types of objects and might have difficulties in handling unknown elements of a scene. A consequent storing and sharing of all available data including sensor raw data is not foreseen in most approaches.

III. APPROACH

We will first define a set of requirements in order to find a suitable approach for a shared world model for robotic applications.

A. Requirements for a world model

A world model that can be used in conjunction with computational components for perception, planning, control and coordination needs to satisfy a number of requirements. These requirements are listed as follows without implying any particular order:

1) 3D model: The world representation needs to fully reflect the fact that robots, in particular mobile manipulation systems, operate in 3D environments. Depending on the type of "user" different data representations are required. For example, a component that controls the motion of a manipulator is in particular interested in kinematic chains, a planning component will typically work on (triangle) meshes, while perception components may process point cloud data captured from 3D sensors. All these types need to be expressed within the same world model.

2) Sharing of data: The world model needs to provide a mechanism that stores and provides data across multiple components, across multiple robots and across a (limited) time span. Having the data accessible in a central place allows to exploit and combine every available piece of information. The world model should support access to sub-parts of a scene and thus needs some means to address these parts. Data consistency among all potential "users" of the world model is a major challenge.

3) Efficiency: Mobile manipulation systems need to react to changes in the environment within a reasonable amount of time to prevent dangerous situations such as collisions with humans. Efficiency in the sense of (near) real-time capabilities is an important requirement for a world model. This issue has multiple aspects.

One aspect relates to efficient access to data in the world model. For example, a path planning algorithms does not need to evaluate the geometries in every detail especially in case of objects far away from the actual planned path. Therefore multi-resolution data access should be possible. While the robot explores its environment the performance should not significantly degrade. A world model needs to be able to scale dynamically in its dimensions.

Another aspect of efficiency is the ability to put computational assets near to the data. Allowing to execute algorithms *within* the world model will reduce potential communication overhead and shorten cycle times for data processing.

4) Tracking ability: As real world scenes change over time, the scene objects need to be expressed in such a way that a trajectory of an object can be deduced from its history of poses. Such a history will help to make estimates for future states of the scene.

5) Uncertainty handling: Sensor data typically contains noisy measurements, perception algorithms might make simplifying assumptions that reduce the accuracy of the results, or some parts of a scene change unexpectedly. A world model should be able to express how certain it can be about its data and thus allowing mechanisms to handle the above situations more robustly.

6) Semantic annotations: The annotation of elements in the scene with semantic attributes or "tags" should be possible. A complex scene can contain a wide range of different types of recognized objects or elements. This information, if available, needs to be preserved such that "users" of a world model are able to identify their relevant parts of the scene.

7) Reusability: The implementation of a world model should be reusable and thus easily integrable into new robotic components and applications. It should not depend on any specific framework including communication middlewares or application specific object representations.

B. A scene graph based world model

The approach presented in this paper is inspired by the scene graph concept. Existing implementations for scene graphs vary as they are tailored towards the needs of specific applications. In this work we will adapt the concept of a scene graph to meet the requirements of robotic applications, as listed in Section III-A. Thus we will call the proposed approach *Robot Scene Graph (RSG)*. A prototype implementation is available as open source C++ library[1] though it does not yet cover all functionality as explained here. The remainder of this section defines the properties of the *RSG* and explains how the requirements can be met.

1) Spatial organization: The core idea is to represent a full 3D scene as a directed acyclic graph (DAG) of geometric entities. Topological and spatial relations between objects are modeled similar as in existing scene graphs for computer animation. The scene graph consists of two different types of nodes: *leave nodes* and *inner nodes*. Only inner nodes can have child nodes. The design of the graph elements complies to the *composite* software pattern [19] as it helps to represent part-whole hierarchies formed by the scene graph. The exact interpretation of the graph relations is dependent on the *semantic domain* as defined in section III-B.2. Every leave or inner node in the *RSG* has at least the following properties.

- Each node can have multiple parent nodes. The root node is a node that has no parents.
- Each node has a globally unique ID. These IDs are a mean to abstract from pointers and allows identification of nodes beyond the system boundaries of a computer.
- Each node has attached attributes. These attributes are stored as a list of key value pairs. This mechanism allows to tag graph entities with e.g. *semantic annotations* or debug information. However the exact usage and meaning depends on the actual application and requires some kind of convention on the used vocabulary. The scene graph offers the flexibility and infrastructure to be able to realize such tagging, but it does not define any semantics - this is left to the actual application.
- Each node has a bounding collision geometry. A sphere is used to approximate the boundary. The origin of the sphere is expressed relative to the frame of the of the node.

Leave nodes contain the actual 3D data like point clouds or triangle meshes. When running robotic 3D perception applications some intermediate data will be created and further processed by perception algorithms. The raw data, the intermediate processing data as well as the final resulting data will be stored in leaves within the scene graph. This is in contrast to the traditional scene graph approach in the computer animation domain where typically only pre-modeled 3D data is loaded. The **GeometricNode** is a leave that stores 3D data. It is a container that contains a single reference to any type of 3D data. Possible data types range from primitive shapes like boxes or cylinders to unconstrained geometries like point clouds or meshes. Any geometric data is assumed to be *immutable*. Once new data is added read-only access will be granted only.

The inner nodes that can be used to create the scene structure are essentially *Groups* and *Transforms*.

- **Group**: The Group is a node that allows for the actual graph relations, as it extends a node to have further child nodes. The group nodes are responsible for management of the children: A node "owns" its children - it will trigger removal of its parent-child relations when it is deleted. A child node is deleted automatically when it does not have parents any more (except for the root node). A deletion of a single node can trigger a cascaded deletion for a subtree. As stated earlier an ID can identify any node, so it can be also used as an ID to specify a sub-part of a scene, as a Group and all its successive child nodes (if any) form a subgraph.
- **Transform**: A transform node is a special group node that expresses a geometric transform between its parents and children. In robotic applications these transforms tend to change over time. For example the robot is moving in a world frame, dynamic objects are moving in the environment or the links of a manipulator move while grasping an object of interest. To be able to determine the current velocity of dynamic objects or to predict their future positions we need to store (a limited) history of the transforms. Each transform node in the scene graph stores the data in a cache with associated time stamps[2]. Data that exceeds a certain duration is

[1] `http://www.best-of-robotics.org/brics_3d/`

[2] This is similar to the implementation of the TF library [12]. However here it is used within a graph instead of a tree structure.

deleted. This forms a short-term temporal cache to allow for *tracking* and pose *prediction* of objects.

- **UncertainTransform**: The UncertainTransform is a more specialized Transform node to add uncertainty information to the transform data of the world model. It extends an entry of the temporal cache to contain beside the time stamp and transform also a covariance matrix. This matrix represents a six degree of freedom error model for a transform as defined in [20]. There might be other choices for an error model as well and we regard this as rather arbitrary choice. The point is to foresee a place where to hook in such an error model in the scene graph in general. The temporal caches and storage of uncertainty information extends the concept of existing scene graphs.

2) Semantics of the graph relations: The meaning of the parent-child relations in the graph needs to be further defined in order to interpret a scene graph correctly. Depending on the context there are different ways of understanding these relations, here we will call them *semantic domains*:

- **Spatial domain**: A node inherits all accumulated rigid transforms from its parents. The transforms along a path from one node to another node of interest are applied in a sequential manner. Each node along such a path that is of type *transform node* will apply one of its transforms from its cache (which one will be defined by a time stamp which is part of a query). If a node is not a transform node then it is treated as an identity transform. Nodes can have multiple paths by its preceding parents as the structure is a graph. This case expresses that multiple information to the same entity is available. For example, an object could be detected by two sensors at the same time. A configurable policy for resolving such situations needs to be set up for a specific application. Possible policies comprise selecting the most promising path like the "latest" path denoted by the latest time stamps associated with the transforms or choosing a path with help of the semantic tags. Note that this ability to represent multiple information to a single object differs to the definition in existing scene graphs. There, multiple paths define multiple instances of the same object.
- **Uncertain transform domain**: Similar to the transform nodes in the spacial domain UncertainTransforms are treated in sequential manner along a path. Additionally, the covariance with respect to the transform between two nodes accumulates along a path via the *compounding* operation as described in [20]. If there are multiple paths between two nodes they are merged according to the *composition* operation. If a Transform node appears along a path it is assumed to have a pre-configured default covariance matrix. All other node types will be ignored.
- **Semantic attributes domain**: The attributes assigned to a node are only valid for the single node, they are not inherited to children or parents.
- **Collision geometry domain**: A node inherits the accumulated collision geometry from its children: all boundaries of the collision geometries are contained in the collision geometry of the node. This allows for efficient collision checks for objects in the scene graph as each node subsumes a collision boundary for its subgraph. If the geometry of a collision query does not intersect with the boundary of a node then it does not intersect with any of its children. The individual checks for the children can be skipped. The collision geometries of the nodes need to be updated every time a new geometry is inserted.
- **Temporal domain**: A time stamp is valid for only that node. It is not inherited. Time stamps are assigned to the actual 3D data in the graph: the geometries as well as the transforms in the temporal cache of a transform node.

The above semantics can be realized with the help of dedicated graph traversals by applying the *visitor* software pattern [19]. It is possible to extend the semantic domains with new ones by defining them and implementing custom graph visitors accordingly.

3) Temporal organization: Beside the spatial organization with the help of the graph structure and the semantic domains that define its interpretation the world model needs to express and handle *time*. Thus the *RSG* has time stamps attached to geometric and transform data. With this it is possible to define strategies when and how to delete data. Two possible approaches are:

- **Sliding time window**: A graph traverser is executed periodically and deletes data that is older than a fixed maximum duration. The traverser can be also configured to take only a subgraph into account. So it is possible to have multiple traversers with different settings for the sliding windows for different subgraphs. A node that is part of multiple such traversed subgraphs will have multiple parents and will be only deleted when its last parent reference is removed - in this case it would remain longest in the subgraph of the traverser with the longest sliding window.
- **Manual deletion**: A node and its subgraph is deleted manually. This can be useful in applications where the duration of the data storage shall be determined by the duration of a processing cycle. Given that all such data is part of a group node it will be enough to delete this group node at the end of the cycle (or at the beginning of the next one).

Both strategies will be demonstrated in an example application in Section IV-B.

4) Computational organization: All elements in the scene graph are uniquely addressable by their IDs so we will use this property to define a generic interface for computational assets for scene data processing. Such a computational unit will be called *function block*. Function blocks can be loaded to an instance of an *RSG* - they can be seen as some form of plugins. This allows to move the computation near to the

data. The interface of a function block essentially consists of four functions:

- *setData*: Sets the IDs defining the relevant nodes for the processing.
- *execute*: Triggers the execution. The implemented functionally will make use of the IDs defined with *setData* to query the scene graph and retrieve relevant data for processing. Possibly new data is inserted to the scene graph.
- *getData*: Returns the IDs of the new inserted nodes (if any).
- *configure*: Configures parameters for the internal algorithms. They are passed as a key value list.

Defining input and output data by IDs has the clear advantage to find a single interface to all algorithms executed within the *RSG*. Though the loose coupling of data types will put some extra burden on the programmer of such a function block to deduce and check if the IDs correctly map to the required data types. Correctly assigning inputs and outputs of a set of function blocks can be difficult as faults will be visible only at application runtime. It is the opinion of the authors that model driven tools can fill this gap. A more elaborate discussion is beyond the scope of this paper.

The computation is decoupled from communication to foster reusability. Input is specified by IDs which have no dependency to any specific communication middleware.

5) Shared Resource: The *RSG* shares all its scene data with all function blocks. A set of function blocks is able to read data and add new data. As geometric data is defined to be immutable it can be accessed concurrently. Transform nodes are not immutable but they provide a temporal cache such that inserting new transform data will not affect retrieving transform data at a specific older time stamp - as long as queries do not reach beyond the limit of the cache.

The *RSG* can be naturally used as a shared world model within a multi-threaded application. However crossing the system boundaries of a process or a computer requires additional communication mechanisms. Many component-based frameworks in robotics including ROS, OROCOS and YARP provide a communication layer for distributed components. These frameworks are mostly message-oriented and do not support a shared data structure well. To overcome this hurdle we will allow the *RSG* to create and maintain local copies of the scene graph similar to [18]. Subsequent graph updates can be encapsulated in any frameworks message type as describes as follows.

All functions to modify nodes in the scene graph will be aggregated in a central API for the *RSG*. This API has to implement the *observer* software pattern [19] to notify any change to the observers. To realize communication based on a middleware a specific observer for this communication framework needs to be implemented and attached to the *RSG*. Furthermore a receiver needs to be implemented to translate again the messages into appropriate function calls of the API. Each framework can have its own observer implementation without affecting an implementation of the *RSG* itself. Please note that a mechanism to ensure consistent concurrent modification of the same node in the local copies is not yet included and remains future work. A promising solution for this problem is discussed in [18].

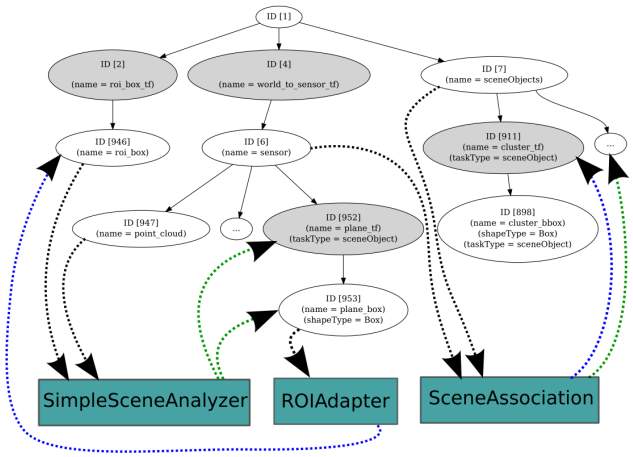

Fig. 2. Example of a *RobotSceneGraph (RSG)*. The function blocks read and add new data to the shared world model as indicated by black and green arrows. Blue arrows denote updates of transform nodes.

The concept of the *RSG* satisfies the requirements for a shared world model approach as described earlier. The ability to share data over a certain time span can be used for example to make perception faster with the help of a dynamic Region of Interest derived from the scene graph. The next section will give an example of such an application.

IV. EVALUATION

This section presents an application based on the *RSG* world model. Here we pick the perception problem of the RoboCup@Work[3] scenario as it is meant as competition to benchmark mobile manipulation systems.

A. Scenario description

The task is to detect and manipulate a set of known objects like screws or aluminum profiles with a KUKA youBot[4]. For evaluation we choose one of the aluminum profiles to be detected. The software runs on the internal PC of the robot. A Kinect 3D camera is attached to the rear of the robot such that it can observe objects near the right side of the robot.

B. Perception solution with shared world model

To solve the task as described above four function blocks have been implemented: a *SetupBlock*, a *SimpleSceneAnalyzer*, a *ROIAdapter* and a *SceneAssociation* module. The *RSG* including the function blocks is embedded into a ROS node that is subscribed to the point cloud data published by the Kinect camera driver. Figure 2 illustrates the interaction of the *RSG* and the involved function blocks.

The *SetupBlock* is executed once at the start of the application. It creates the basic scene graph structure including nodes to represent transforms between the world frame and the robot as well as the transform between robot and sensor

[3] http://www.robocupatwork.org/
[4] http://youbot-store.com/

frame. Furthermore it initially sets up a Region of Interest (ROI) based on a Transform node and a GeometricNode containing a box as shape data.

Whenever new data from the Kinect camera arrives a processing cycle is triggered which involves execution of the *SimpleSceneAnalyzer*, the *ROIAdapter* and the *SceneAssociation* function blocks in a sequential manner. First the point cloud data is added to the *RSG* as a GeometricNode as child of the sensor group node. Then the *SimpleSceneAnalyzer* block gets executed. It contains a set of perception algorithms to detect aluminum profiles in the current scene. As input the ID of the freshly inserted point cloud and the ID for the GeometricNode containing the ROI are passed. All intermediate point clouds are stored as GeometricNodes in the *RSG* as child nodes to the sensor group as well. Before each processing cycle starts all such intermediate data from the previous cycle is manually deleted as new data of the scene is available.

In a first step all points that are within the dimensions of the box defined by the ROI are extracted. Then the point cloud is sub-sampled with an Octree based filter. The dominant plane is estimated by fitting a plane model with the RANSAC algorithm. The inliers and outliers of the model are stored in separated point clouds. The point cloud based on the outliers is clustered based on the Eucledian distance. Then for each cluster an approximate oriented bounding box is computed by extraction of the Principle Components. The volume of the bounding boxes is used as feature for classification. All detected clusters that are classified as aluminum profile trigger insertion of a new Transform node pointing to the center of the object and a new GeometricNode with the calculated bounding box data as its child node. They represent the currently perceived objects of interest in the scene. In addition the attribute with key "taskType" and value "sceneObject" gets assigned to both nodes. All intermediate processing data as well as the result of the algorithm are now accessible for other function blocks. The IDs of the GeometricNodes form the set of output IDs and are passed as input to the *ROIAdapter* function block.

The purpose of the *ROIAdapter* is to adapt the position and dimensions of the ROI box based on the currently perceived objects. It picks the first found object (typically there is only one such object in the scenario) and updates the transform of the stored ROI based on the objects transform but relative to the scene root node. The ROI influences the perception algorithms in the *SimpleSceneAnalyzer* within the next cycle and thus utilize information that is already stored in the scene graph. To some regard this is in contrast to most traditional "pipe line" processing approaches that do not make use of the already available scene information from earlier processing steps.

Finally the *SceneAssociation* block maintains all perceived objects over a longer period of time than the perception cycle. The input IDs specify (a) the subgraph containing the perceived objects and (b) the subgraph of the "scene" that holds the scene objects. Both subgraphs are traversed individually and all nodes with the attribute ("taskType", "sceneObject") are collected. Then they are associated by selecting the pair with the shortest Euclidean distance of their centroids. A threshold defines if the closest object is too far away to be treated as a new object. New scene objects are inserted as follows: a new Transform node relative to the root node of the "scene" is inserted and all children of the scene object are taken over by adding new parent-child relations to the Transform node. Here it means to take over the GeometricNode for the box shape. Any available geometric data or even subgraph could be taken over in general. If a perceived object can be successfully associated to one of the existing scene objects then the recent pose gets inserted as a new entry into the cache of the Transform node of the scene object. As the *SceneAssociation* is responsible for maintaining a scene it triggers the graph traversals that implement the sliding time window. The used implementation deletes any Transform in the "scene" subgraph that has an empty temporal cache. The maximum allowed duration of the entries in cache is $10\,\mathrm{s}$. That means either an object is updated at least once per $10\,\mathrm{s}$ or it gets deleted. With the help of the "sceneObject" tag it is possible to quickly query the *RSG* for the relevant scene objects (and accessing their pose history if necessary).

C. Experimental results

Experiments have been conducted to see how much this example application can benefit from the usage of a dynamic ROI that is maintained in the scene graph. The application was executed multiple times with enabled and disabled dynamic ROI. For each run at least 200 processing cycles have been recorded. The dynamic ROIs did not negatively affect the quality of results as long as the ROI does not get suddenly lost e.g. in cases of quick changes of the camera pose. The used performance metric is the computational time. The average values and standard deviations for each proceeding step are depicted in Fig. 3. The individual bars represent the processing steps as follows: (1) the ROI box based point cloud extraction, (2) the sub-sampling based on an Octree, (3) the RANSAC based plane segmentation, (4) the Euclidean clustering and (5) the cluster evaluation based on the bounding box.

The results show the processing time improves for every step. The ROI extraction itself can profit moderately. Although it always has to evaluate the same number of input points it has to create less points for a smaller ROI for the resulting point cloud. The remaining steps can profit by a speed up factor of up to 15 as the number of input points can be reduced significantly via the ROI. In total the average processing time is approx. 4 times faster with the dynamic ROI. Of course the results are only valid for the used test based in with combination the experimental environment. However they show that the temporal storage of scene data in a central world model can be exploited to improve perception applications.

V. CONCLUSION

This paper has presented the *Robot Scene Graph (RSG)*, a novel approach to represent and manage 3D scenes for

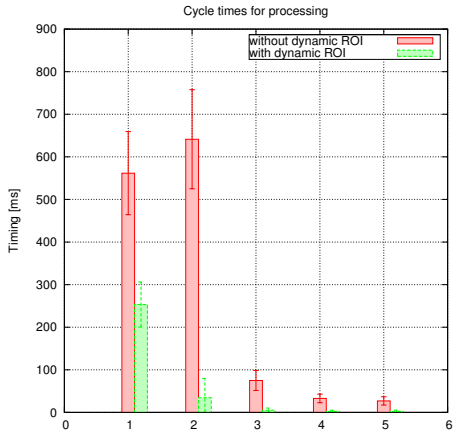

Fig. 3. Timing comparison for the processing steps: (1) the ROI box extraction, (2) sub-sampling, (3) plane segmentation, (4) clustering and (5) cluster evaluation.

robotic systems. The approach uses the scene graph concept and adapts it towards the requirements of full 3D model support, sharing of data, efficiency, tracking ability of dynamic objects, uncertainty handling, semantic annotations and reusability. To evaluate the concept a sample perception application is presented that is able to successfully exploit knowledge from the temporal shared data in the scene graph. The experiments show that the processing cycle times can be accelerated. The *RSG* implementation is available as open source C++ library.

A number of issues remain for future work. A demonstration on how to integrate the *RSG* with planning or motion control components needs to be implemented to further verify the concept. Also the perception algorithms could be much more sophisticated in the sense of feature extraction and classification. A more generic collision representation, alternative temporal caching strategies, exploration of graph structures for uncertainty representation or advanced communication and synchronization mechanisms for distributed scene graphs are promising future research directions.

VI. ACKNOWLEDGMENTS

The authors would like to thank all the partners of the BRICS project for their valuable comments.

REFERENCES

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[2] H. Bruyninckx, "Open robot control software: the orocos project," in *IEEE International Conference on Robotics and Automation*, vol. 3, 2001, pp. 2523 – 2528.

[3] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: Yet another robot platform," *International Journal on Advanced Robotics Systems*, vol. 3, no. 1, pp. 43–48, 2006.

[4] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer Magazine*, vol. 22, no. 6, pp. 46–57, 1989.

[5] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[6] N. Fairfield, G. Kantor, and D. Wettergreen, "Real-time slam with octree evidence grids for exploration in underwater tunnels," *Journal of Field Robotics*, vol. 24, no. 1-2, pp. 03–21, 2007.

[7] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," in *Proceedings of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010, software available at http://octomap.sf.net/.

[8] K. Wurm, D. Hennes, D. Holz, R. Rusu, C. Stachniss, K. Konolige, and W. Burgard, "Hierarchies of octrees for efficient 3d mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 4249–4255.

[9] R. Rusu, I. Sucan, B. Gerkey, S. Chitta, M. Beetz, and L. Kavraki, "Real-time perception-guided motion planning for a personal robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 4245–4252.

[10] M. Ciocarlie, K. Hsiao, E. G. Jones, S. Chitta, R. B. Rusu, and I. A. Sucan, "Towards reliable grasping and manipulation in household environments," in *Proceedings of RSS 2010 Workshop on Strategies and Evaluation for Mobile Manipulation in Household Environments*, New Delhi, India, 2010.

[11] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," in *Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, June 2007.

[12] T. Foote, W. Meeussen, and E. Marder-Eppstein. (2013, February) tf2 - ros wiki. [Online]. Available: http://ros.org/wiki/tf2

[13] A. Nüchter, *3D Robotic Mapping The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*. Springer, 2009.

[14] R. Rusu, Z. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008.

[15] Z. Papp, C. Brown, and C. Bartels, "World modeling for cooperative intelligent vehicles," in *Intelligent Vehicles Symposium*, 2008, pp. 1050–1055.

[16] J. Curn, D. Marinescu, and V. Cahill, "A flexible approach to management and processing of collaborative vehicular perception data," in *Proceedings of the Workshop on Emergent Cooperative Technologies in Intelligent Transportation Systems at the 2010 IEEE Intelligent Transportation Systems Conference 2010*, Sept. 2010.

[17] D. Shuey, D. Bailey, and T. Morrissey, "Phigs: A standard, dynamic, interactive graphics interface," *IEEE Computer Graphics and Applications*, vol. 6, no. 8, pp. 50–57, 1986.

[18] M. Naef, E. Lamboray, O. Staadt, and M. Gross, "The blue-c distributed scene graph," in *Proceedings of the workshop on Virtual environments 2003*. ACM, 2003, pp. 125–133.

[19] G. Erich, H. Richard, J. Ralph, and V. John, *Design patterns: elements of reusable object-oriented software*. Addison Wesley Publishing Company, 1995.

[20] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," *Autonomous robot vehicles*, vol. 1, pp. 167–193, 1990.