



# UNIVERSITÀ DI PARMA

## ARCHIVIO DELLA RICERCA

University of Parma Research Repository

A KinFu based approach for robot spatial attention and view planning

This is the peer reviewed version of the following article:

*Original*

A KinFu based approach for robot spatial attention and view planning / Monica, Riccardo; Aleotti, Jacopo; Caselli, Stefano. - In: ROBOTICS AND AUTONOMOUS SYSTEMS. - ISSN 0921-8890. - 75:(2016), pp. 627-640. [10.1016/j.robot.2015.09.010]

*Availability:*

This version is available at: 11381/2806071 since: 2021-10-11T10:46:56Z

*Publisher:*

Elsevier

*Published*

DOI:10.1016/j.robot.2015.09.010

*Terms of use:*

Anyone can freely access the full text of works made available as "Open Access". Works made available


*Publisher copyright*

note finali coverpage

(Article begins on next page)

15 January 2025

## AUTHOR QUERY FORM

	<p><b>Journal:</b> Robotics and Autonomous Systems</p> <p><b>Article Number:</b> 2538</p>	<p><b>Please e-mail your responses and any corrections to:</b></p> <p><b>E-mail:</b> <a href="mailto:corrections.esch@elsevier.river-valley.com">corrections.esch@elsevier.river-valley.com</a></p>
---	---	---

Dear Author,

Please check your proof carefully and mark all corrections at the appropriate place in the proof (e.g., by using on-screen annotation in the PDF file) or compile them in a separate list. Note: if you opt to annotate the file with software other than Adobe Reader then please also highlight the appropriate place in the PDF file. To ensure fast publication of your paper please return your corrections within 48 hours.

For correction or revision of any artwork, please consult <http://www.elsevier.com/artworkinstructions>.

Your article is registered as belonging to the Special Issue/Collection entitled “3D Perception with PCL”. If this is NOT correct and your article is a regular item or belongs to a different Special Issue please contact [j.mohideen@elsevier.com](mailto:j.mohideen@elsevier.com) immediately prior to returning your corrections.

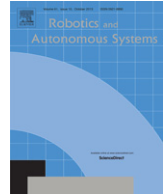
Location in article	Query/Remark <a href="#">click on the Q link to go</a> Please insert your reply or correction at the corresponding line in the proof
<a href="#">Q1</a>	Please confirm that given names and surnames have been identified correctly.
<a href="#">Q2</a>	Color statement has been added to the caption(s) of Figs. 3, 7 and 8. Please check, and correct if necessary.
<a href="#">Q3</a>	The usage ‘Gaussian nr.’ has been changed to ‘Gaussian no.’ in tables 5 and 6. Please check and correct if necessary.
<a href="#">Q4</a>	The repeated vol. 1 is deleted in Ref. [26]. Please check, and correct if necessary.
<a href="#">Q5</a>	Biography is present only in the PDF file. We have inserted it in the TeX file. Please check and correct if necessary.
	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <p style="color: red; margin: 0;">Please check this box or indicate your approval if you have no corrections to make to the PDF file</p> <input style="width: 30px; height: 20px; vertical-align: middle;" type="checkbox"/> </div>

Thank you for your assistance.



Contents lists available at ScienceDirect

## Robotics and Autonomous Systems

journal homepage: [www.elsevier.com/locate/robot](http://www.elsevier.com/locate/robot)

## A KinFu based approach for robot spatial attention and view planning

Q1 Riccardo Monica, Jacopo Aleotti\*, Stefano Caselli

Dipartimento di Ingegneria dell'Informazione, University of Parma, Italy



## HIGHLIGHTS

- A spatial attention approach is presented for a robot manipulator.
- Salient human actions are detected by hand motion tracking and GMM.
- Approximate locations of salient user actions draw robot's attention.
- Next-best views are computed on the GPU using KinFu Large Scale.

## ARTICLE INFO

## Article history:

Available online xxxx

## Keywords:

KinectFusion  
Point Cloud Library  
Robot spatial attention

## ABSTRACT

When a user and a robot share the same physical workspace the robot may need to keep an updated 3D representation of the environment. Indeed, robot systems often need to reconstruct relevant parts of the environment where the user executes manipulation tasks. This paper proposes a spatial attention approach for a robot manipulator with an eye-in-hand Kinect range sensor. Salient regions of the environment, where user manipulation actions are more likely to have occurred, are detected by applying a clustering algorithm based on Gaussian Mixture Models applied to the user hand trajectory. A motion capture sensor is used for hand tracking. The robot attentional behavior is driven by a next-best view algorithm that computes the most promising range sensor viewpoints to observe the detected salient regions, where potential changes in the environment have occurred. The environment representation is built upon the PCL KinFu Large Scale project [1], an open source implementation of KinectFusion. KinFu has been modified to support the execution of the next-best view algorithm directly on the GPU and to properly manage voxel data. Experiments are reported to illustrate the proposed attention based approach and to show the effectiveness of GPU-based next-best view planning compared to the same algorithm executed on the CPU.

© 2015 Published by Elsevier B.V.

## 1. Introduction

In this work a saliency-driven spatial attention approach is presented where a robot manipulator actively updates the 3D representation of the scene only where a change is detected in the environment. Relevant changes in the environment, such as the ones caused by user manipulation activities, draw the robot's attention. The proposed robot system closely links perception and motion planning. Indeed, the robot cannot observe at once the whole environment because of occlusions and kinematic constraints. Hence, the robot is equipped with a sensor on its end-effector, which is used to maintain an updated representation of

the environment by active exploration. The robot focuses the viewpoint of the eye-in-hand sensor towards the regions where user actions are more likely to have produced change.

In particular, the paper proposes an attentional approach for a 6-DoF robot arm in a tabletop scenario. The robot is equipped with a Kinect range sensor (Fig. 1). Relevant changes in the environment are caused by manipulation actions (also named activities) of a person interacting with objects, which may include moving or removing an object in the scene or placing new objects. Potential changes are detected by analyzing the motion of the user hand by motion capture. The 3D environment representation is updated by the robot system only when one or more relevant changes are detected.

To merge information acquired by the range sensor from different views a modified version of KinFu was developed and made available for download. The KinFu Large Scale (KinFu LS) project [1] is an open source implementation of KinectFusion [2] based on the PCL library [3]. The environment is modeled as a volumetric 3D

\* Corresponding author.

E-mail addresses: [rmonica@ce.unipr.it](mailto:rmonica@ce.unipr.it) (R. Monica), [aleotti@ce.unipr.it](mailto:aleotti@ce.unipr.it) (J. Aleotti), [caselli@ce.unipr.it](mailto:caselli@ce.unipr.it) (S. Caselli).<http://dx.doi.org/10.1016/j.robot.2015.09.010>

0921-8890/© 2015 Published by Elsevier B.V.

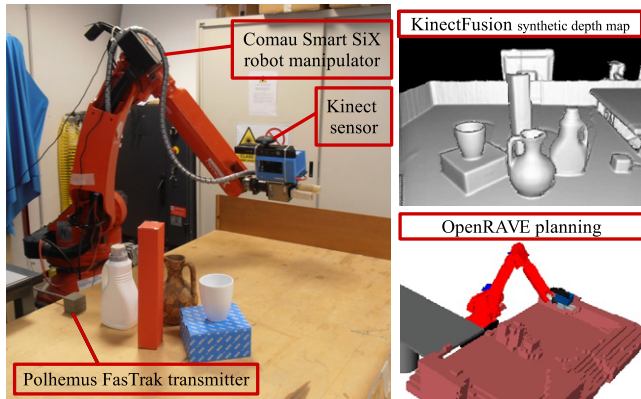


Fig. 1. Experimental setup (left). Example of KinectFusion output (top right). Motion planning environment (bottom right).

voxel grid using a truncated signed distance function. Each voxel is labeled as unknown, empty or occupied. When salient user activities are detected, points of interests are computed from the estimated regions of the environment where something has possibly changed. Then, a Next Best View (NBV) algorithm is executed iteratively to plan the best pose where the sensor must be placed to observe the regions of interest. Multiple views of the robot may be required in order to achieve a complete update of each region of interest. The robot exploration phase of each region of interest continues until information gain is negligible.

The contributions of this work are illustrated next. A method is proposed for saliency-driven reconstruction and update of the environment based on the KinFu algorithm and NBV planning. KinectFusion has been investigated in previous works [4,5] for robotics applications, but the use of KinectFusion with NBV planning on a 3D voxel grid has not been considered before. Moreover, no previous work has investigated novelty detection from user activities through NBV planning. In this work, the original KinFu algorithm has been modified to run the NBV algorithm directly on the GPU and to properly manage voxel data. Experiments are reported to compare the GPU-based NVB algorithm to the same algorithm executed on the CPU. KinFu has also been modified to improve accuracy by exploiting the accurate kinematics of the robot, as in [4], instead of performing egomotion estimation. Two strategies for NBV have been evaluated: keeping the KinFu active during the robot motion between consecutive views and turning it on only at the planned viewpoints. Finally, an algorithm is proposed which segments the human hand trajectory using a Gaussian Mixture Model (GMM) to detect where salient user actions occur. The proposed algorithm has been evaluated on a dataset of user activities and compared to a zero-velocity crossings (ZVC) approach.

The paper is organized as follows. Section 2 presents an overview of the state of the art about novelty detection and attention in robotics. Previous works on NBV planning and automatic segmentation of human tasks are also discussed. Section 3 provides an overview of the proposed approach. KinectFusion is introduced in Section 4. Section 4 also describes the NBV algorithm developed on the GPU, how KinFu was adapted, and the algorithm for determination of the regions of interest. In Section 5 the software architecture based on the PCL library is discussed as well as the integration with the ROS middleware. Section 6 illustrates the experimental results. Section 7 concludes the paper discussing the results and providing suggestions for possible extensions.

## 2. Related work

Most attention-based approaches are strictly focused on locating changes in the environment [6–11] and do not consider

active exploration using NBV sensor planning. Petsch et al. [6] proposed a framework for sensor-based detection of unexpected (surprising) events. Manipulation events are detected from human observation. Alimi et al. [7] presented an attention system for object understanding using 3D data which is able to segment object regions from the background. In [8] a weakly supervised method is described identifying daily actions from video sequences based on the change in the state of the environment. Aksoy et al. [9] proposed a method for learning the semantics of object–action relations by observation using a dynamic graph representation. Herbst et al. [10] presented an algorithm for object discovery from multi-scene Markov random field analysis based on RGB–D data.

Other systems have been proposed for object attention for humanoid robots or stereo-heads. Bottom-up saliency maps are usually defined from blobs of uniform color [12] or local features of the environment [13]. Several spatial attention approaches have been developed in the context of mobile robots to construct and maintain a consistent representation of the environment as the robot moves [14–16]. In particular, Drews et al. [16] introduced a method for novelty detection based on Gaussian mixture models from laser scan data. Attention approaches based on other non-verbal cues were presented in [17,18]. Schauerte et al. [17] investigated a method which combines object visual saliency with directional information from pointing gestures. In [18] the focus of attention of a person in human–robot interaction tasks is estimated by gaze direction estimation.

NBV approaches usually assume that a target object is given and do not cope with the problem of detecting regions of interest from user actions. To the best of our knowledge, this is the first work investigating the use of KinectFusion in conjunction with NBV planning on a 3D voxel grid. Furthermore, computation of NBV on the GPU was not considered before, except in [19] where a custom solution was investigated for an autonomous UAV. The NBV algorithm adopted in this work follows the standard approaches proposed by Banta et al. [20] and Connolly [21]. To determine the placement of the sensor in eye-in-hand configuration a viewing sphere is adopted around the region of interest and an objective function is chosen which maximizes the unknown volume. Improvements have been proposed to ensure an overlap among different discrete views in [22]. However, in [22] a turntable is used for moving the object. Approaches for full 6D planning around the object usually consider a single object in the environment [23–28] or assume simple objects to be scanned [29]. Whaitte et al. [30] adopted simple parametric models to represent the objects based on superquadrics. Kriegel et al. [31] presented an exploration system, combining different sensors, for tabletop scenes that supports NBV planning and object recognition. More complete overviews of advanced NBV approaches can be found in [32,33]. In [32] a probabilistic framework for NBV planning in cluttered environments is presented, which estimates the visibility of occluded space using a PR2 robot.

Regarding previous works on segmentation of human hand tasks and automatic extraction of regions of interest, little focus has been placed on the use of Gaussian mixture models [34]. In [34] a GMM method is proposed for automatic segmentation of full-body motion trajectories into basis skills by the intersection of consecutive Gaussians. However, the approach is not intended to identify meaningful and salient manipulation tasks. In [35] a template-based technique was presented based on the computation of velocity features and hidden Markov models. Faria et al. [36,37] proposed an approach to segment and classify the action phases of a task using information about hand orientation and trajectory curvature. In [38] manipulation tasks were segmented by identifying motion breakpoints from the fingertip polygon area. Yeasin et al. [39] used a binocular vision system and feature tracking for learning and segmenting skills from multiple human demonstrations.

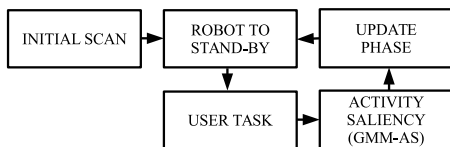


Fig. 2. The general flowchart of the system.

### 3. Overview of the method

The proposed attention-based approach enables a robot manipulator to update the 3D representation of the environment each time a change is detected. The update procedure consists of taking new observations using the eye-in-hand sensor in the regions where a user performed a task. The general behavior of the system is outlined in Fig. 2. First, an initialization phase occurs, where the robot acquires data along a predefined scan trajectory (which is assumed not to collide with any object) using KinectFusion. 3D data acquired after the initial scan only serves as a starting representation of the environment and may include large unseen areas due to object occlusions. Then, the robot moves to a “stand-by” configuration where it waits for a user to perform a task.

The user can interact with the environment by performing object manipulation tasks. In the context of this paper each user task may involve one or more sequential actions (also named activities), each of which can be performed in different regions of the environment. For example, a pick-and-place task requires to direct the attention of the robot to both the initial and final configurations of the object. The trajectory of the user hand during a task is acquired by a motion capture sensor (described in Section 6) and it is analyzed by the GMM-AS algorithm, detailed in Section 4.4. GMM-AS generates a set of Points of Interest (POI). Each POI represents the approximate location of a region of interest where a change in the environment might have occurred.

Then, all the regions of interest are cleared with unknown voxels and the robot starts an update phase (Section 4) to explore all the now unknown regions of interest. After the update phase, the robot returns back to the “stand-by” configuration. The 3D representation of the environment is computed by the KinectFusion algorithm. In this paper the PCL KinFu LS implementation of KinectFusion was adopted and extended. An overview of the algorithm is reported in Section 4.1, while the implementation in Section 5.

The robot update phase is driven by a NBV algorithm [20,21], detailed in Section 4.3. Given a ternary volumetric representation (occupied, empty, unknown) of the environment, the NBV algorithm simulates the Kinect range sensor and produces a sequence of viewpoints ranked by their highest theoretical information gain. Collision free robot movements are planned using the OpenRAVE engine [40]. Occupied and unknown voxels are considered as obstacles in the motion planner. To speedup the planning phase, the representation is conservatively subsampled to voxels of size 4 cm.

### 4. Robot attention approach

The update phase of the environment 3D representation, detailed in Algorithm 1, starts when a set of Points Of Interest is generated (to be explained in Section 4.4). Each POI is characterized by a center point and a surrounding spherical shape  $S(POI)$ , describing the region of interest that it generates. First, all the regions of interest are filled with unknown voxels (line 2) which will be considered as obstacles while planning robot movements. Then, the first POI of the sequence is selected and viewpoints are computed to explore its region of interest  $S(POI)$  by applying the NBV algorithm (line 5). The NBV algorithm scores viewpoints

#### Algorithm 1: Environment representation update phase

---

**Input:**  $WS$ : 3D volumetric environment representation;  
**Input:**  $POIs$ : Point of Interest array;  
**Output:**  $WS$ : The updated representation;

```

1: for each POI in POIs do
2:   ForgetAround( $WS, POI$ );
3: end for
4: for each POI in POIs do
5:   ( $Viewpoints, Gain$ )  $\leftarrow$  NextBestView( $WS, POI$ );
6:   for  $l$  from 1 to size( $Viewpoints$ ) do
7:     if  $Gain[l] < IncTh$  then
8:       break;
9:     end if
10:    ( $Ok, Traj$ )  $\leftarrow$  PlanRobotTo( $Viewpoints[l]$ );
11:    if  $Ok$  then
12:      MoveRobotAlong( $Traj$ );
13:      PerformRobotOscillation();
14:      ( $Viewpoints, Gain$ )  $\leftarrow$  NextBestView( $WS, POI$ );
15:    end if
16:  end for
17: end for
  
```

---

maximizing the estimated information gain. Information gain of each viewpoint  $Gain[l]$  is computed as the number of unknown visible voxels. NBV evaluation is limited to the shape  $S(POI)$  of the current POI, since other POIs will be better served by subsequent observations, centered on them.

All the viewpoints are then attempted one after the other, starting from the most promising in decreasing order. If one viewpoint predicts an information gain less than a given threshold  $S(POI)$  is considered completely reconstructed and the next POI, if any, is selected (line 7). Viewpoints are re-computed after each successful observation (line 14). Once a valid viewpoint is found, a trajectory for the robot is planned (line 10). A viewpoint is skipped either when unreachable, due to robot kinematics, or when already attempted for the current POI. All information acquired by KinectFusion is used to update the environment representation, thus it may happen that a region of interest  $S(POI_i)$  is partially observed during the exploration of another  $POI_j$ . This case is implicitly handled by the proposed algorithm.

When a viewpoint configuration is reached, the robot moves so that the Kinect sensor tilts slightly ( $\pm 5^\circ$ ) around the sensor horizontal axis (line 13), allowing KinFu to work properly on a continuous input stream of depth images from that viewpoint. The tilting motion is modeled in the NBV algorithm by configuring the simulated sensor with a wider vertical field of view than the real Kinect sensor. Two variations of the NBV exploration strategy have been evaluated. In the first, called “KinFu On ViewPoint” (KFOVP), KinFu is allowed to acquire new data only from the observation viewpoints as described above. In the second, called “KinFu On Motion” (KFOM), KinFu is also kept active during the motion of the robot between two consecutive viewpoints. We hypothesize that KFOM should require a lower number of views to complete the environment update phase. Both KFOVP and KFOM use KinFu output as the 3D volumetric reconstruction of the scene. KinFu is not reset after each viewpoint observation.

#### 4.1. The KinectFusion algorithm

KinectFusion is an iterative algorithm for real-time tracking of a moving depth camera and 3D fusion of the environment observations in a volumetric data structure. Software implementations exploit highly parallel GPU techniques (CUDA). The 3D environment representation is accessible at any time, provided a fast access to the GPU memory is available. KinectFusion represents the environment as an implicit surface model using a truncated signed distance function (TSDF). TSDF is a clamped function  $R^3 \rightarrow R$  which

maps the 3D coordinates  $(x, y, z)$  to the distance from the nearest surface, negative inside objects and positive outside. KinectFusion samples a TSDF volume in a regular grid of voxels with fixed size. Each voxel contains two values: a TSDF sampled value  $v$  and a weight  $w$  that counts the number of times the voxel has been observed.

Each KinectFusion iteration consists of four steps. *Ray casting*: A synthetic depth map (point cloud) is generated from the current TSDF volume, as seen by a virtual sensor placed in the last known sensor position. *Depth map conversion*: The latest measured depth image is converted into an organized point cloud. *Camera tracking*: The point cloud is aligned with the synthetic depth map using a modified point-to-plane ICP (Iterative Closest Point) algorithm and then the motion of the sensor is estimated from the ICP transformation. *Volumetric integration*: The point cloud is converted to global coordinates and merged with the TSDF volume. *Ray casting* and *volumetric integration* steps are described next to explain how KinectFusion was adapted to work with the NBV algorithm.

In the *ray casting* step, each ray of the sensor is sampled with constant step and traversed until it exits the TSDF volume or a zero crossing of the TSDF is found. When a zero crossing of the TSDF is found a new point is saved in the point cloud using a trilinear interpolation of the TSDF values of neighboring voxels. Since KinectFusion is executed on the GPU, each ray is evaluated in parallel.

During the *volumetric integration* phase, the latest sensor pose and the corresponding observed points are converted in global coordinates. Then, the voxels intersected by the view ray are updated up to the observed point as follows. The weight value is used to average the new value with past values and it is increased at each new voxel observation. The weight value is capped to a maximum value  $M_w$ : thus, after some observations, the update procedure becomes a running average. The update equations of the TSDF and weight values of each intersected voxel can be written as follows:

$$v_{SDF} = \|p - o\| - \|c - o\| \quad (1)$$

$$v' = \frac{v \cdot w + \text{clamp}(v_{SDF}/M_v, -1, 1)}{w + 1} \quad (2)$$

$$w' = \min(w + 1, M_w) \quad (3)$$

where  $p$  is the point observed by the sensor from position  $o$  and  $c$  is the position of the voxel in the TSDF volume, along the view ray  $\overline{op}$ . The temporary value  $v_{SDF}$  represents the Signed Distance Function, not yet truncated. The  $v_{SDF}$  value is normalized by the maximum value  $M_v$  and clamped between  $-1$  and  $1$  to obtain the updated TSDF  $v'$ . Empty voxels far outside the surface have a TSDF value equal to  $1$ , empty voxels near the surface have a TSDF value between  $1$  and  $0$ , while occupied voxels inside the surface have a TSDF value below  $0$  (Fig. 3). Voxels never observed have a  $0$  weight and their TSDF value remains at its default  $0$  value. Since the view ray stops at the observed point, voxels inside objects are never updated, even though from update Eq. (2) the TSDF value should be set to  $-1$ .

#### 4.2. KinFu Large Scale

A limitation of the KinectFusion algorithm is that it can neither acquire data outside the TSDF volume, nor use data outside the TSDF volume for sensor tracking. Besides, the TSDF volume cannot be expanded indefinitely due to limited GPU memory. Improved versions of KinectFusion have been developed based on the concept of downloading part of the 3D representation on the CPU memory, thus freeing space on the GPU for data processing [41]. This operation is called *shifting*.

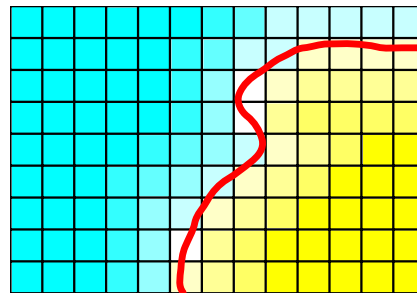


Fig. 3. An example of 2D TSDF volume. The red line represents the object surface. The yellow area represents the negative values inside the object, while the cyan area the positive ones. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)<sup>^</sup>

PCL KinFu LS uses a cyclical buffer to obtain a faster shifting procedure, without the need of memory deallocation and reallocation. The TSDF volume is allocated as a 3D matrix of fixed size. Whenever shifting occurs, the origin of the TSDF volume is translated, so it can represent a different area of the workspace. Slices of the TSDF volume are downloaded from the GPU memory to the CPU RAM. The now free slices on the GPU are then refilled with existing data (if any) from the CPU, or reset to unknown state ( $v = 0, w = 0$ ). The new TSDF volume is partially overlapped with the old one, hence some data is kept on the GPU memory. Since the TSDF volume is organized as a 3D cyclical buffer only the origin is shifted and no data needs to be moved on the GPU. Downloaded slices would need a large amount of CPU memory, if they were saved as a voxel grid. The solution proposed by KinFu LS is to convert slices to a point cloud. Each point is defined by the 3D global coordinates of the voxel and the TSDF value. The weight value is lost in the process. Moreover, to reduce the size of the point cloud, all points with TSDF value  $v = 1$  or never observed ( $w = 0$ ) are removed from the representation.

KinFu LS introduces a number of translated and scaled reference frames. Reference frame  $\{W\}$  is the world frame in real metric coordinates. The expanded frame  $\{E\}$  is the origin in expanded coordinates, scaled with respect to world coordinates so that each TSDF voxel has edge  $1$ , i.e. a point  ${}^E p$  in frame  $\{E\}$  can be expressed as:

$${}^E p = \frac{w p}{e} \quad (4)$$

where  $e$  is the voxel size. The TSDF shift frame  $\{H\}$  is centered at the voxel of index  $(0, 0, 0)$  in the TSDF volume in expanded coordinates. The TSDF frame  $\{O\}$  is centered at the voxel of the TSDF volume that determines the shifting operation, in expanded coordinates. The TSDF frame  $\{M\}$  has the same origin of  $\{O\}$  but it is defined in real world metric coordinates. An overview of the reference frames is shown in Fig. 4. Reference frames defined above are updated at each shifting operation. In the following, the notation  ${}^b_a t$  indicates the translation vector of reference frame  $a$  with respect to frame  $b$ . Given that the TSDF volume is cyclical, a point  ${}^H p$  in TSDF shift coordinates can be expressed in extended coordinates  ${}^E p$  as:

$${}^E p = [({}^H p + {}^O_H t + S_V) \bmod S_V] + {}^E t \quad (5)$$

where  $S_V$  is an array which contains the size of the TSDF volume in voxels and  $\bmod$  is the component-wise modulo operator.  $S_V$  is added to  $({}^H p + {}^O_H t)$  so that the result is always positive.

In the attention-based system a spherical region of interest  $S(POI)$  of the TSDF volume has to be cleared if the following inequality holds:

$$\|{}^W S_c - {}^W p\| < {}^W S_r \quad (6)$$

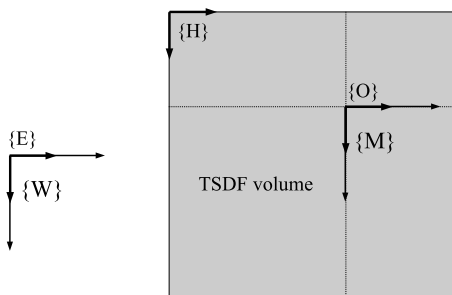


Fig. 4. A 2D representation of the KinFu LS reference frames. Unit vectors for  $\{O\}$ ,  $\{E\}$  and  $\{H\}$  are shorter, since they use expanded (scaled) coordinates.

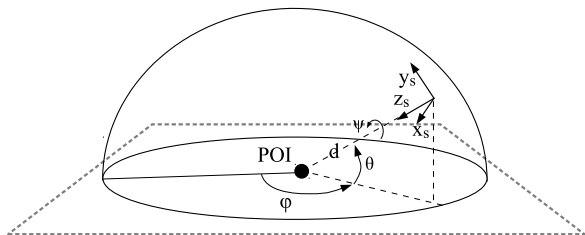


Fig. 5. The sphere sampled by NBV algorithm around the POI with radius  $d$ .

where  ${}^W S_c$  is the sphere center and  ${}^W S_r$  is the radius. TSDF voxels can be processed in parallel using a CUDA kernel. Each thread evaluates a different voxel, with index in TSDF shift coordinates  ${}^H p$ . To verify inequality (6),  ${}^H p$  has to be converted to  ${}^W p$  for each thread using Eq. (5) and then Eq. (4). However, it can be noticed that Eq. (4) can be applied to inequality (6) in advance, once for all on CPU. The CUDA kernel can then operate directly in the expanded coordinates  $\{E\}$ , and considering that

$${}^O S_c = {}^E S_c + {}^O E t \quad (7)$$

inequality (6) can be rewritten as:

$$\|{}^O S_c - {}^O p\| < {}^O S_r. \quad (8)$$

Hence, a slightly simpler equation than (5) may be used to obtain the point to be evaluated for inclusion in  $S(POI)$ :

$${}^O p = ({}^H p + {}^O H t + S_V) \bmod S_V. \quad (9)$$

If the voxel satisfies the inequality it is set to unknown.

### 4.3. Next-best view approach

This section describes the GPU-based NBV algorithm. The algorithm follows the two-step approach proposed in [20,21]. In the first step, candidate viewpoints are sampled on a sphere with fixed radius centered on the current  $POI$ . Each sensor position is uniquely defined by assigning the radius ( $d$ ), the  $POI$  and three attributes: longitude ( $\phi$ ), latitude ( $\theta$ ) and rotation around the sensor axis ( $\psi$ ) as shown in Fig. 5. Rotation  $\psi$  is mainly important to ease the search for a reachable pose of the robot manipulator. The three attributes are uniformly sampled in their ranges:  $\phi \in [0^\circ, 360^\circ]$ ,  $\theta \in [0^\circ, 90^\circ]$ ,  $\psi \in [0^\circ, 360^\circ]$ , to cover all the upper hemisphere. The sampling intervals are respectively  $30^\circ$ ,  $10^\circ$  and  $45^\circ$ , resulting in a total of 960 sampled viewing poses. The radius  $d$  is fixed at 0.8 m having the Kinect sensor a minimum sensing distance of 0.5 m.

In the second step, viewpoints are evaluated and sorted in decreasing order by a scoring function which counts the number of unknown visible voxels. The NBV algorithm must find where each viewing ray, passing through a pixel of the virtual sensor, intersects the first non-empty voxel in the environment representation. Thus,

### Algorithm 2: NextBestView( $KinFu, POI$ )

**Input:**  $KinFu$ : KinFu world representation;  
**Input:**  $POI$ : Current Point of Interest;  
**Output:**  $Viewpoints$ : Array of viewpoints;  
**Output:**  $Gains$ : Scores assigned to each viewpoint;  
1:  $Viewpoints \leftarrow \text{GenerateViewpointsAround}(POI)$ ;  
2:  $Gains \leftarrow \text{array}(\text{size}(Viewpoints))$ ;  
3:  $KinFu.ForceShiftToPoint(POI)$ ;  
4: **for**  $I$  **from** 1 **to**  $\text{size}(Viewpoints)$  **do**  
5:    $Gains[I] \leftarrow 0$ ;  
6:    $Voxels \leftarrow KinFu.RayCast(Viewpoints[I])$ ;  
7:   **for each**  $Voxel$  **in**  $Voxels$  **do**  
8:     **if**  $S(POI).Contains(Voxel)$  **and**  $Voxel.IsUnknown$  **then**  
9:        $Gains[I] \leftarrow Gains[I] + 1$ ;  
10:     **end if**  
11:   **end for**  
12: **end for**  
13:  $\text{OrderViewpointsAndGains}(Viewpoints, Gains)$ ;

the second phase is the most time consuming. Unknown voxels are counted only if inside the current shape  $S(POI)$ . Therefore, a voxel  $v$  viewed by the simulated sensor of the NBV algorithm will contribute to the expected gain  $Gain[I]$  of a viewpoint with index  $I$  as:

$$Gain[I] \leftarrow \begin{cases} Gain[I] + 1 & \text{if } v = \text{unkn.} \wedge v \in S(POI) \\ Gain[I] & \text{otherwise.} \end{cases} \quad (10)$$

The proposed algorithm, named NBV-GPU, exploits the TSDF volume itself. Indeed, it may be observed that the ray casting phase of KinFu can be adapted to obtain the observed TSDF voxels, provided the point of view of the virtual sensor can be arbitrarily set. The main advantage of this novel solution is that the GPU allows fast computation of the NBV at higher resolution than the CPU can achieve. Moreover, it does not require data to be extracted from the TSDF volume and converted on the CPU, and it uses exactly the same sensor model as KinFu.

In particular, the ray casting phase of KinFu has been modified to be executed on-demand by the NBV algorithm, from an arbitrary pose. When traversed by the NBV algorithm rays must stop at zero crossings (default behavior), but also when an unknown voxel (with 0 weight) is intersected. An overview of the NBV procedure, called *NextBestView* in Algorithm 1, is shown in Algorithm 2. Initially, a shifting is forced to set the origin  $\{M\}$  of the TSDF volume to the  $POI$  (line 3). This shift loads the region of interest on the TSDF volume. Unfortunately, as stated in Section 4.2, all shifting events, including those that happen when KinFu is active, cause the loss of both the weight values and the empty voxels with TSDF value  $v = 1$ . Hence, empty voxels far from the surface and the unknown voxels become impossible to distinguish. This information must be restored in the TSDF volume for the NBV to work properly. Keeping track of both unknown and empty voxels is achieved by using a custom octree data structure on CPU memory. This data structure was designed to efficiently store points observing that the number of unknown and empty voxels is much higher than the number of occupied ones. Details are provided in Section 5.2.

The NBV-GPU algorithm was compared to a standard algorithm where NBV is computed on the CPU (NBV-CPU). As explained above NBV-CPU requires volumetric data to be extracted from the GPU TSDF volume to the CPU. In addition, extracted data needs to be converted to a standard ternary representation (occupied, empty and unknown voxels). A conversion rule can be defined as follows:

$$\begin{cases} w = 0 & \rightarrow \text{unknown voxel} \\ w = 0 & \begin{cases} v \leq 0 & \rightarrow \text{occupied voxel} \\ v > 0 & \rightarrow \text{empty voxel.} \end{cases} \\ w > 0 & \end{cases} \quad (11)$$

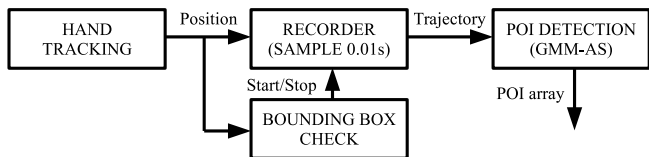


Fig. 6. The trajectory recorder and POI detection as functional blocks.

To speed-up ray casting NBV-CPU uses a standard PCL octree data structure, which is queried recursively to find the first intersection. In particular, NBV-CPU uses the *getIntersectedVoxelCenters* method of the *pcl::octree::OctreePointCloudSearch* class, configured to stop at the first non-empty octree voxel. This algorithm is optimized with respect to a raycasting algorithm in which the ray is followed through a voxel grid with a fixed step (named NBV-CPU-step in Section 6.4), since the ray skips empty volumes thanks to the octree data structure.

#### 4.4. Determination of regions of interest

Task trajectories are automatically recorded when the user hand is inside the bounding box of the workspace. The user hand is tracked by a motion capture sensor, described in Section 6. When the hand exits the bounding box, the trajectory is analyzed. An overview of the procedure for trajectory acquisition, segmentation and POIs extraction is shown in Fig. 6.

The proposed algorithm for POIs extraction, named Gaussian Mixture Models Activity Saliency (GMM-AS), is reported in Algorithm 3. The trajectory is defined as  $T = \{q_i\} = \{(x_i, y_i, z_i)\}$ , a sequence of points  $q_i$  with  $i \in 1 \dots N$ , where  $N$  is the length of the trajectory. Motion analysis for the determination of the POIs is executed in the augmented 4D space  $S = \{(x_i, y_i, z_i, t_i)\}$  which includes time stamp information (lines 1–4). Including the time stamp helps the separation of points close in space but far in time. Indeed, self-intersections should not be encoded by the same Gaussian in the model.

GMM-AS fits a Gaussian Mixture Model on the trajectory  $p(S) = \sum_{j=1}^K \pi_j \mathcal{N}(S | \mu_j, \Sigma_j)$  through Expectation Maximization (EM), EM is computed in function *ExpectationMaximization* (line 8), which takes as its arguments  $S$  and the number of Gaussians  $GC$  to be fitted, and it returns the *GMM*. The EM algorithm is initialized as in [34] by splitting the time span of the trajectory in equal segments and by assigning a Gaussian to each segment. The means and covariances of each Gaussian component are initialized using the sample mean and the sample covariances of the points which fall in the corresponding segment.

Since to apply the EM algorithm the number of Gaussians must be known, EM is executed multiple times with an increasing number of Gaussian components, starting from 1. For each *GMM* a *BIC* index [34] is computed by *ComputeBIC* (line 9). The lower the value of the *BIC* index the better the *GMM* fits the trajectory. It is assumed that the best fitting *GMM* model is in the first local minimum of the *BIC* index. The algorithm terminates when the current *BIC* exceeds the best found *BIC* by a small threshold *BICTh* (lines 10–14). The Gaussians are ordered by increasing timestamp mean value in function *SortByTimestamp* (line 15).

It can be observed that, in the GMM, intermediate hand movements between manipulation actions (e.g. object transportation phases) are likely to be represented by long and narrow Gaussians, that exhibit a single dominant direction with high variance (Figs. 7 and 8). On the opposite, Gaussians with more balanced variances and a higher weight appear where the user performs a manipulation action. This can be explained by observing that user actions are usually performed at slow speed and involve changes of hand

#### Algorithm 3: GMM-AS POI detection

**Input:**  $T$ : the trajectory, sequence of points;  
**Output:**  $POIs$ : the POI array;

- 1:  $S \leftarrow$  empty set;
- 2: **for**  $i$  **from** 1 **to**  $\text{size}(T)$  **do**
- 3:  $S \leftarrow S \cup \{[x_i \ y_i \ z_i \ t_i]\}$ ;
- 4: **end for**
- 5:  $MinBIC \leftarrow +\infty$ ;
- 6:  $GC \leftarrow 1$ ;
- 7: **repeat**
- 8:  $GMM \leftarrow \text{ExpectationMaximization}(S, GC)$ ;
- 9:  $BIC \leftarrow \text{ComputeBIC}(GMM, S)$ ;
- 10: **if**  $BIC < MinBIC$  **then**
- 11:  $(MinBIC, BestGMM, c^*) \leftarrow (BIC, GMM, GC)$ ;
- 12: **end if**
- 13:  $GC \leftarrow GC + 1$ ;
- 14: **until**  $BIC > MinBIC + BICTh$ ;
- 15:  $\text{SortByTimestamp}(BestGMM)$ ;
- 16: **for**  $g$  **from** 1 **to**  $\text{size}(BestGMM)$  **do**
- 17:  $s_g = \text{ComputeSaliency}(G_g, c^*)$ ;
- 18: **end for**
- 19:  $POIs \leftarrow$  empty array;
- 20: **for**  $g$  **from** 2 **to**  $\text{size}(BestGMM) - 1$  **do**
- 21:  $th_g \leftarrow \frac{1}{2 \cdot A} \sum_{i=1}^A (s_{g+i} + s_{g-i})$ ;
- 22: **if**  $s_g > POITh \cdot th_g$  **then**
- 23:  $POIs \leftarrow POIs + \{[G_g \cdot \mu_x \ G_g \cdot \mu_y \ G_g \cdot \mu_z]\}$ ;
- 24: **end if**
- 25: **end for**

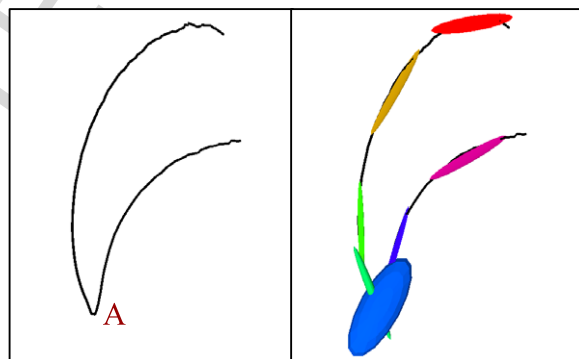


Fig. 7. An example trajectory (left image) obtained from the placing task of an object in point A. The superimposed GMM (right image). Only the blue “thick” Gaussian correctly determines a POI. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

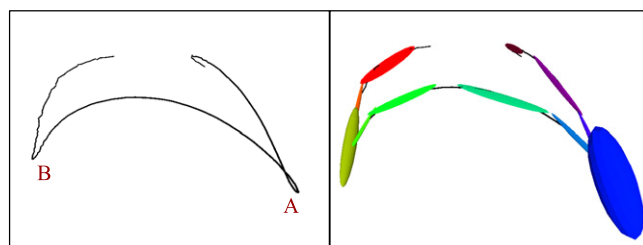


Fig. 8. An example trajectory (left image) obtained from the task of picking and placing an object from A to B. The superimposed GMM (right image). Only the blue and the yellow “thick” Gaussians correctly determine two POIs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

direction, hence more points are sampled without a dominant direction. This facts can be exploited by evaluating each Gaussian  $G_g$  using a *ComputeSaliency* function (line 17) defined as follows:

$$\sigma_i = \sqrt{|\lambda_i|} \quad i \in \{1, 2, 3\} \quad (12)$$



$$s = \frac{\sigma_1 \cdot \sigma_2 \cdot \sigma_3}{\sigma_1 + \sigma_2 + \sigma_3} \cdot \pi \cdot c^* \quad (13)$$

where  $\lambda_1$ ,  $\lambda_2$  and  $\lambda_3$  are the eigenvalues of the Gaussian G covariance matrix, marginalized over Cartesian coordinates,  $s$  is the saliency,  $c^*$  is the Gaussian count and  $\pi$  is the weight (prior) of the Gaussian in the model. The saliency  $s_g$  of each Gaussian is compared to the average saliency of the neighbor Gaussians (lines 20–25). The first and last Gaussians are ignored, since they often suffer from boundary effects (line 20). If saliency is higher than the average saliency multiplied by *POIth* (line 22), the spatial coordinates ( $G_g \cdot \mu_x$ ,  $G_g \cdot \mu_y$ ,  $G_g \cdot \mu_z$ ) of the mean of the current Gaussian are added as a new *POI*. It must be noted that there is no reliable correlation between the shape (or size) of a Gaussian and the shape of the potential region of interest. Indeed, the shape of the Gaussian is mostly determined by the user's (local) trajectory and not by the shape of the object. Thus, the spherical region of interest  $S(POI)$  is set with a fixed radius ( $W_{S_r} = 0.2$  m), larger than most common objects in manipulation scenarios. The user can, in principle, perform an arbitrary number of tasks. Each task can include an arbitrary number of actions. However, after each task the user must wait for the robot to complete the environment update phase before performing the following task. GMM-AS may generate multiple close *POIs* which, however, do not affect the performance of the system since these close *POIs* are simultaneously explored.

The GMM-AS algorithm was compared to an adaptive zero-velocity crossings (ZVC) algorithm, which is based on the idea of looking for points of interest where slower hand movements appears. First, a mean filter with width  $(2B + 1)$  is applied to the trajectory to reduce noise. The average speed at index  $i$  is then computed as:

$$\bar{v}_i(r) = \frac{1}{\|R_i(r)\|} \sum_{j \in R_i(r)} \frac{\|q_j - q_{j-1}\|}{t_j - t_{j-1}} \quad (14)$$

where  $R_i$  is the set of the indices of the neighboring samples of  $q_i$  within a radius  $r$ , defined as:

$$R_i(r) = \{j \mid \forall k \in ([j, i] \cup [i, j]), \|q_k - q_i\| < r\}. \quad (15)$$

A slow hand movement is detected whenever the local speed is significantly lower than the speed averaged on a wider range. In particular, the average speed is computed for two values of  $r$ :  $r_1 = 10$  cm and  $r_2 = 25$  cm. Segments of trajectory that satisfy  $\bar{v}_i(r_1) \cdot ZVCTh < \bar{v}_i(r_2)$ , where *ZVCTh* is a constant multiplier, generate a *POI* at the average time index of the segment.

## 5. Software architecture

### 5.1. Extension of KinFu Large Scale for robot attention

KinFu LS defines *KinfuTracker* as the main class. *KinfuTracker* can be instantiated to a function object where the *operator()* method accepts a new depth map and executes one iteration of the algorithm. Two storage classes, *TsdfVolume* and *CyclicalBuffer* manage the world model. *TsdfVolume* acts as a proxy for the TSDF volume loaded on the GPU. *CyclicalBuffer* manages TSDF shifting and updates the reference frames. A nested class *WorldModel* manages the point cloud with intensity (*pcl::PointXYZI*) that contains the downloaded slices. The intensity value here represents the TSDF value  $v$ . Overall nesting relations are shown in Fig. 9.

To use KinFu as a persistent environment representation for the robot attention algorithm the following features were added. First, KinFu has to be interrupted and restarted on demand, for example when the robot reaches the stand-by configuration. Second, part of the environment has to be cleared, i.e. set to unknown to generate the regions of uncertainty of the *POIs*.

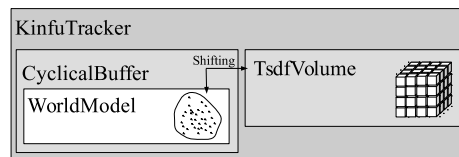


Fig. 9. Relations among the most important classes of KinFu LS.

Interrupting KinFu can easily be achieved by not calling the *operator()* method as there is no active thread in the implementation. However, as the ICP algorithm works under the assumption of small sensor movements between consecutive readings, if a movement is too large egomotion estimation by ICP does not converge. This case happens if KinFu is suspended and resumed from another viewpoint. Therefore, to solve this issue, KinFu must be provided with the actual pose of the sensor. To this purpose a parameter, named *hint*, was added to *operator()*. The *hint* parameter accepts a struct of type *Hint*, defined as in Listing 1.

```
struct Hint
{
    enum Type
    {
        HINT_TYPE_NONE,
        HINT_TYPE_HINT,
        HINT_TYPE_FORCED
    };
    Type type;
    Eigen::Affine3f pose;
};
```

Listing 1: The *Hint* struct.

Three modes of operations are allowed: *HINT\_TYPE\_NONE* causes KinFu to use the default ICP egomotion tracking; *HINT\_TYPE\_FORCED* disables ICP and forces the use of the *pose* parameter; *HINT\_TYPE\_HINT* where ICP is enabled but it initialized with the *pose* parameter and not with the last sensor pose. In the proposed system, the pose of the sensor is computed from the robot kinematics. The third mode was added to allow an additional hybrid approach.

Clearing parts of the environment affects both *TsdfVolume* and *WorldModel* classes. The point cloud contained by the *WorldModel* class is in expanded coordinates  $\{E\}$ . Points not existing in this model are considered unknown when loaded into the TSDF volume. All the points in the *WorldModel* which verify inequality (6) (Section 4.2) in expanded coordinates are set to unknown. The sphere must also be cleared in the TSDF volume, accessed through the *TsdfVolume* class in reference frame  $\{O\}$ . This procedure is executed in parallel, on the GPU, for each voxel of the TSDF volume, using Eq. (9) to clear voxels that satisfy inequality (8). If the voxel is inside the sphere, the voxel is set to unknown (TSDF value  $v = 0$  and weight  $w = 0$ ). A wrapper method was added to the main *KinfuTracker* class to convert the sphere to  $\{E\}$  and  $\{O\}$  coordinates and consistently clear both the *WorldModel* and the *TsdfVolume*.

### 5.2. Extension of KinFu Large Scale for next-best view

KinFu LS defines a *RayCaster* class that can be used to execute raycasting from an arbitrary pose in the TSDF volume. The *RayCaster* class can be configured with arbitrary intrinsic parameters: focal length and image size. The default configuration is changed, by setting a wider vertical field of view, when simulating the tilting motion of the real sensor described in Section 4. *RayCaster* uses a CUDA kernel to perform ray casting and to fill a vertex map with the voxels that are viewed by the virtual sensor. A Boolean template parameter was added to enable

the kernel to be used for the next-best view algorithm without affecting the original behavior.

For the next-best-view implementation rays need to be stopped at the first non-empty voxel. A knowledge map was added on the GPU, filled with values indicating if the ray intercepted an unknown voxel, an occupied voxel or exited the volume intercepting only empty voxels. Unknown voxels in the knowledge map are counted, to estimate the number of unknown voxels seen by the virtual sensor, only if the corresponding point in the vertex map is inside the shape  $S(POI)$  (Eq. (10)).

As stated in Section 4.2, shifting loses distinction between unknown ( $w = 0$ ) and empty ( $v = 1$ ) voxels when downloading data from the GPU, as  $w$  is discarded in the *WorldModel*. For this reason, KinFu LS was modified in such a way that the distinction between unknown and empty voxels is properly saved and restored for the shifted slices. Adding additional information to the KinFu *WorldModel* would be extremely expensive in terms of memory occupation. Thus, a data structure was introduced to only distinguish empty and unknown voxels. This structure, named *BitmaskOctree*, is based on the `pcl::octree::OctreeBase` class. *BitmaskOctree* operates at lower (1/8) resolution with respect to the TSDF volume. To match with the TSDF resolution, each octree leaf contains a 3D matrix `boost::bitmask` of  $8 \cdot 8 \cdot 8 = 512$  bits. The compression is lossless in terms of resolution, however the actual value is binarized to 0 (unknown) if  $w = 0$  and 1 (known, i.e. empty or occupied) if  $w > 0$ . Octree leaves that are completely unknown are not stored in memory. The minimum amount of memory for a bitmask with a single point is 64 bytes. However, this event is very unlikely, since known voxels tend to form clusters and most of the bitmasks will be completely set. Indeed, leaves in the *BitmaskOctree* that are completely set are on average 69% of the total number of leaves.

For the sake of generality *BitmaskOctree* was wrapped in a listener class. *CyclicalBuffer* was modified to accept a listener class which implements the *WeightCubeListener* interface, defined in listing 2. The `onReset` and `onClearSphere` methods are called when clearing the whole workspace or a spherical region of interest. When a shifting occurs, the method `onNewCube` is called to save TSDF weights before the shifting procedure. In addition, the `retrieveOldCube` method is polled during shifting and at each KinFu iteration until binarized weight information for the new TSDF volume origin is returned, extracted from the *BitmaskOctree*. The weight information is then merged with the TSDF volume in the GPU as follows. If the *BitmaskOctree* leaf is 1, the weight value in the TSDF volume is incremented by 1. Otherwise, the value on the TSDF volume is left unchanged. If incrementing the weight value results in a previously unknown TSDF voxel changing to known, the TSDF value is initialized to 1 (empty). The load operation may be executed at any time, amid KinFu iterations, as soon as the `retrieveOldCube` returns the weight information. Indeed, voxels changing from unknown to empty in the TSDF do not affect the correctness of the KinFu LS algorithm, because during standard execution the two types of voxel are treated in the same manner.

```
class WeightCubeListener
{
public:
    virtual void onReset() = 0;
    virtual void onClearSphere(...) = 0;
    virtual void onNewCube(...) = 0;
    virtual bool retrieveOldCube(...) = 0;
};
```

Listing 2: The *WeightCubeListener* interface. Parameters are omitted.

A multithread solution was implemented to execute asynchronously the setting and retrieval of values in the *BitmaskOctree*,

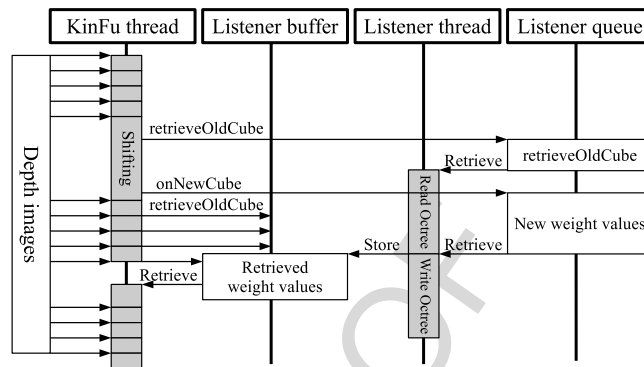


Fig. 10. The multithread shifting procedure.

without disrupting the KinFu execution, as shown in the sequence diagram in Fig. 10. The listener is an active object, with its own processing thread. At the first `retrieveOldCube` call the request is scheduled in a queue and, when the result is ready, it is returned during one of the subsequent polling calls to the method. Also, the `onNewCube` calls return immediately, while data is placed in the same queue for later processing.

### 5.3. System integration in ROS

KinFu LS was integrated into the ROS (Robot Operating System) framework and it is available for download ([https://github.com/RMonica/ros\\_kinfu](https://github.com/RMonica/ros_kinfu)). Integration is based on the KinFuLS ROS wrapper [42], which publishes the synthetic depth map to a ROS topic and the current tracked sensor pose. The PCL KinFu LS implementation is primarily designed to acquire data from the Kinect sensor, visualize it and save data to a file before termination. A post-processing phase is required to obtain a polygon mesh, by calling an executable which runs the Marching Cubes algorithm. In contrast to this standard behavior, as explained in Section 5.1 the attention-based system requires the use of KinFu as a persistent tool for environment representation.

In this work, the main *kinfu* node from [42] was extended to handle the modifications of KinFu LS. The extended *kinfu* node interacts with the ROS environment by accepting two types of messages: commands and requests. Commands affect the behavior of the system and do not require any further processing. Commands include `SUSPEND`, `RESUME`, `CLEAR_SPHERE`. Each command may also supply a hint or a forced hint, which act as defined in Section 5.1. Also, a `SET_FORCED_TF_FRAME` command binds the KinFu tracking to a reference frame using `ros-tf`. This feature is used in this work to feed KinFu tracking with the robot odometry. Commands change asynchronously the state of a command subscriber, which applies the new configuration at the next iteration of the KinFu.

Request messages ask the *kinfu* node to publish a part of the environment. The part of the environment may be extracted in various formats, such as a point cloud, a polygon mesh or a voxel grid. Also, the projection of the environment on a virtual sensor (used for NBV) is executed through a request. Each request is served by a dedicated thread, different from the main KinFu thread. Requests copy data from the *WorldModel* and the *BitmaskOctree* after waiting for any KinFu pending operation in the *WeightCubeListener* queue or any `retrieveOldCube` unfinished calls. Each request thread must acquire an exclusive lock on KinFu to copy data. However, additional processing can be performed in parallel, such as execution of marching cubes to obtain the polygon mesh, or subsampling to reduce the size of the point cloud in the response.



Fig. 11. The two environments used for testing KinFu tracking accuracy.

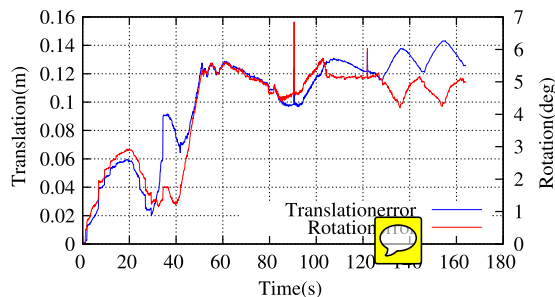


Fig. 12. KinFu egomotion tracking error in the first scenario of Fig. 11.

Table 1  
3D object reconstruction errors.

Scenario	Tracking method	Mean abs. (m)	St. dev. (m)
1	Egomotion	0.031	0.040
1	Robot kinematics	0.005	0.006
2	Egomotion	0.027	0.041
2	Robot kinematics	0.006	0.008

## 6. Experimental evaluation

The experimental setup includes a small robot arm (Comau SMART SiX) with six degrees of freedom and a Kinect sensor mounted on the end-effector. The Kinect is calibrated with respect to the robot wrist. A Polhemus FasTrak is used for 3D hand tracking, calibrated with respect to the robot base frame. FasTrak is an electro-magnetic device which tracks in real-time the position and orientation of a small receiver located on the wrist of the user. The advantage of using a magnetic motion capture device is that it does not suffer from occlusion problems. The overall configuration of the system is shown in Fig. 1. The software runs under the ROS framework on an Intel Core i7 4770 at 3.40 GHz, Nvidia GeForce GTX 670.

### 6.1. Evaluation of Kinect sensor tracking solutions

Initial tests were performed to evaluate to what extent the internal drift of KinFu egomotion tracking may affect the robot attention system. In this section, KinFu egomotion tracking is compared with an alternative solution where KinFu is fed with the robot forward kinematics. Tests were performed by moving the robot arm along the trajectory used for the initial observation of the environment. Experiments were conducted in two scenarios with different objects as shown in Fig. 11. Tracking errors are computed as the difference between the robot forward kinematics (ground truth) and the KinFu egomotion estimation. Fig. 12 shows the translation (Euclidean distance) and rotation (angle-axis representation) errors in the first scenario. The errors increase with time during robot motion. Egomotion tracking accumulates a large drift, as the translation error at the end is about 12 cm.

To further evaluate the accuracy of the two solutions three parameters were measured for each object and compared to the ground truth values: the object height and the lengths of the major

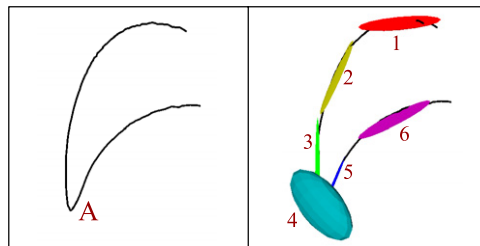


Fig. 13. A trajectory representing an object placement task to location A (left) and the superimposed GMM (right).

and minor axis of the object projection on the horizontal table. Average errors for the parameters are reported in Table 1. The 3D reconstruction using the robot kinematics exhibits much higher accuracy than the one obtained using KinFu egomotion tracking. For this reason, during all the experiments of the attention-based system presented in Sections 6.3 and 6.4, the KinFu was fed with robot kinematics.

### 6.2. GMM-AS algorithm evaluation

The robustness of the GMM-AS algorithm was evaluated on a dataset of 330 tasks performed by four subjects. The dataset is available at <https://github.com/RMonica/hand-polhemus-dataset>. The dataset contains 110 placement tasks of new objects in the environment, 110 object removal tasks from the environment and 110 pick-and-place tasks. Each recorded task trajectory was inspected manually and ground truth POIs were placed where actions actually occurred, i.e. at the initial location of the object for removal tasks, at the final location of the object for insertion tasks and at both the initial and final object locations for pick-and-place tasks. Object placement and removal tasks were evaluated together as they are characterized by a single action and they exhibit quite similar trajectories. Instead, pick-and-place tasks are characterized by two actions. Actions were counted as correctly detected if the distance to the true POI was within 20 cm (POI radius). False positives (FP) were counted when a POI was detected far from any ground truth POIs. False negatives (FN) were counted when the algorithm did not detect any POI near a ground truth POI. Tests were performed at different speeds of the task by undersampling ( $\frac{1}{2} \times$ ,  $\frac{1}{4} \times$ ) and oversampling ( $2 \times$ ,  $4 \times$ ) the trajectories. Additional tests were performed by adding to the trajectories white Gaussian noise.

The GMM-AS and the ZVC-like algorithm show comparable results on the dataset (Table 2). Also, ZVC-like shows a slightly better result than GMM-AS when the two algorithms are tested at different speeds (Table 3). However, to obtain this result the noise-reduction window parameter  $B$  of ZVC must be at most 4. Such higher bound for  $B$  reduces the maximum noise that the ZVC-like algorithm is able to compensate. As seen in Table 4, GMM-AS is able to detect POIs even with 2.5 cm of standard deviation, while the ZVC-like algorithm fails.

The ZVC-like algorithm takes some milliseconds of computational time, while the GMM-AS takes a few (2–8) seconds. Nonetheless, this higher execution time does not affect the performance of the robot attention system which is mostly affected by planning and by the slow velocity of the robot.

Fig. 13 shows an example trajectory of an object placement task. The saliency assigned to each of the Gaussians is reported in Table 5. Only one Gaussian (the fourth) is classified as a salient action (placement) and generates a single POI. Another example, for a pick-and-place task, is shown in Fig. 14 and Table 6. In this case, only the second and the fifth Gaussians are detected as salient actions generating two POIs, which correspond to object picking and placing respectively.

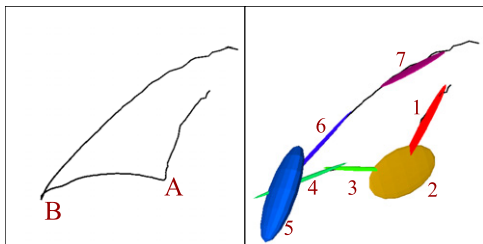


Fig. 14. A trajectory representing a pick-and-place task from location A to B (left) and the superimposed GMM (right).

### 6.3. Evaluation of the robot attention system

This section presents two complete experiments of the attention based system (Fig. 15) using NBV-GPU. Each experiment was performed two times using the same recorded user hand trajectory, the first one with KFOVP and the second with KFOM. In the first experiment, the user placed a plastic bottle in the environment, among a group of boxes. In the second experiment, the user picked up a jug from a pallet and then placed the jug on a wooden disc after passing over a briefcase. The tasks generated POIs and corresponding regions of interest as shown in Fig. 16.

Table 2

Evaluation of GMM-AS and ZVC-like ( $\lambda = 2$ ,  $POI_{th} = 2.5$ ,  $B = 4$ ,  $ZVC_{th} = 1.72$ ).

Tasks	Algorithm	Precision	Recall
Placements and removals	ZVC-like	96.4%	97.3%
Placements and removals	GMM-AS	95.4%	95.9%
Pick-and-place	ZVC-like	94.5%	98.2%
Pick-and-place	GMM-AS	97.7%	97.3%

GMM-AS generated a single correct POI in the first experiment. In the second experiment GMM-AS generated a total of three POIs, two of them almost superimposed where the jug is moved.

Fig. 17 shows images of OpenRAVE planning, robot NBV configurations, KinFu synthetic depth maps and environment representations of KFOVP exploring the single POI in experiment 1. A video of KFOM in experiment 1 is available at <http://rimlab.ce.unipr.it/documents/KinFuNBV2015.wmv>. Fig. 18 shows all the five views performed by KFOVP in experiment 2. The first view (first column) observes the empty space above the pallet where the jug is picked up (first POI). The other four views are needed to reconstruct the second and the third (almost superimposed) POIs where the jug is moved on the wooden disc.

Fig. 19 illustrates the final result for the KFOVP experiments. The average accuracy of the reconstructed objects involved in the

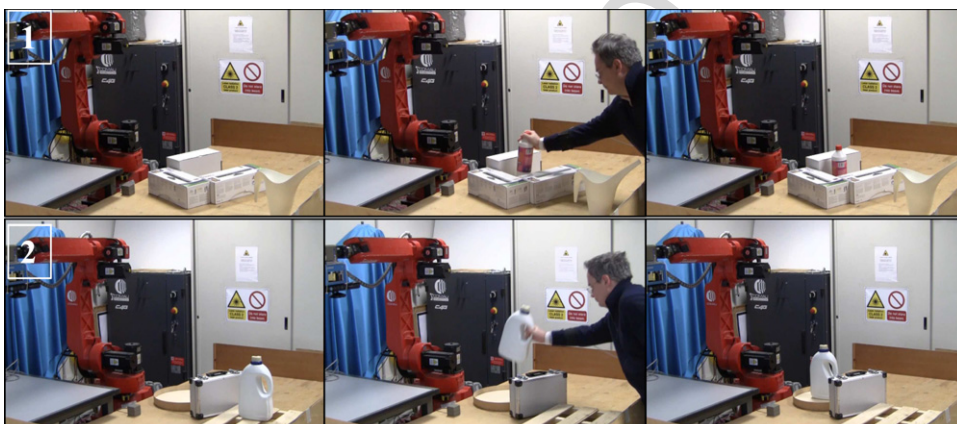


Fig. 15. The environment before (left), during (center) and after (right) the user task. First experiment (top row) and second experiment (bottom row).

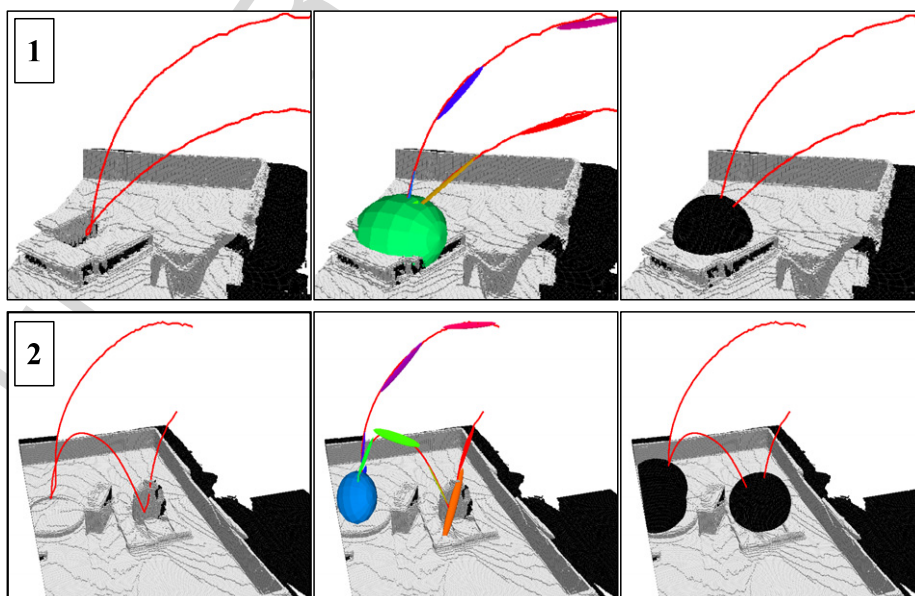
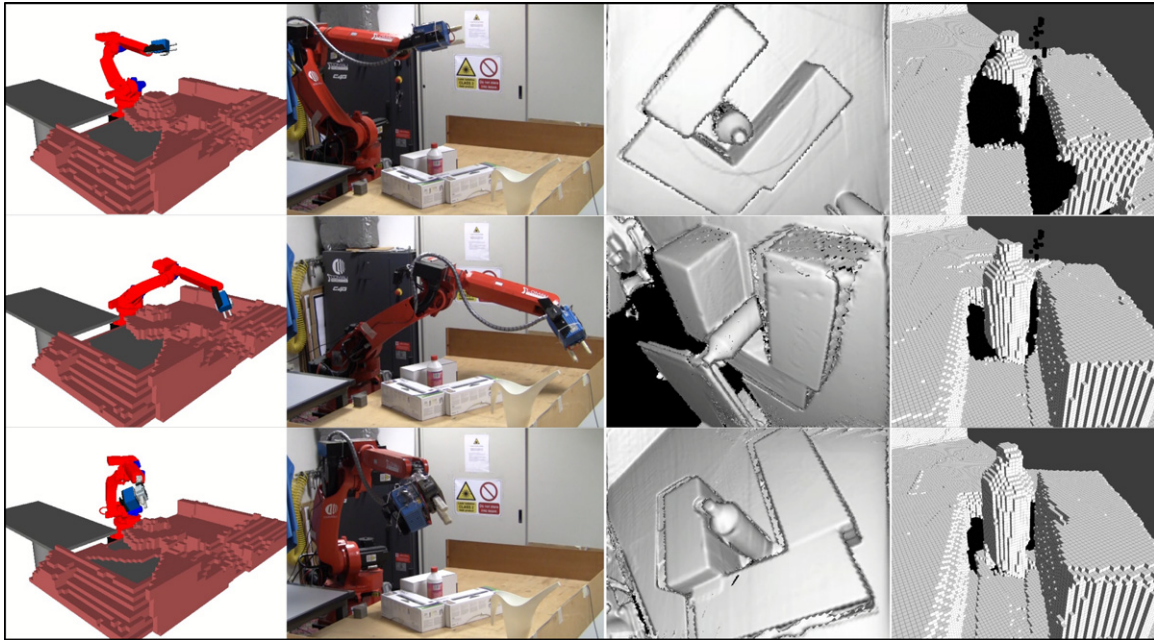
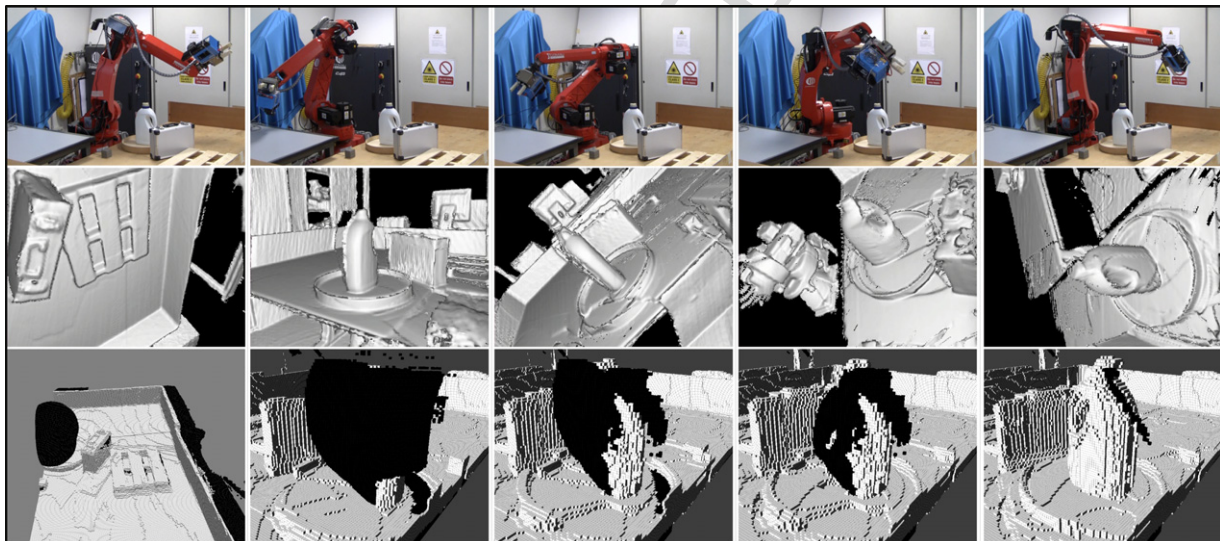


Fig. 16. The environment 3D representation with user hand trajectory (left), GMM (center) and generated spherical regions of uncertainty (right) for the first (top row) and second experiment (bottom row) in black color.



**Fig. 17.** KFOVP snapshots in experiment 1. From left to right: OpenRAVE planning, robot NBV configuration, KinFu synthetic depth map and reconstruction in the ternary representation (unknown voxels are displayed in black), for each view pose (top to bottom).



**Fig. 18.** KFOVP snapshots in experiment 2. Robot NBV configuration (top row), KinFu synthetic depth map (middle row) and the reconstruction in the ternary representation (bottom row, unknown voxels are displayed in black) for each view pose (left to right).

**Table 3**  
Evaluation of GMM-AS and ZVC-like at different speed multipliers, on the whole dataset.

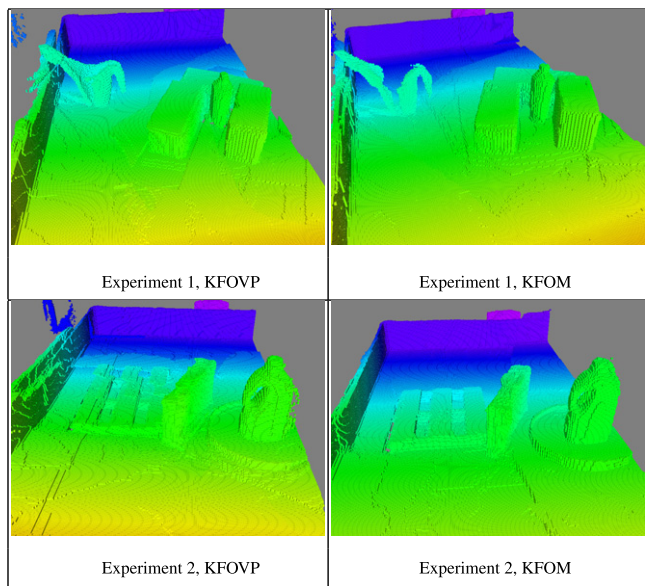
Speed multiplier	Algorithm	Precision	Recall
0.25	ZVC-like	96.6%	93.0%
0.25	GMM-AS	98.0%	86.1%
0.50	ZVC-like	95.7%	97.7%
0.50	GMM-AS	97.0%	95.9%
1.00	ZVC-like	95.5%	97.7%
1.00	GMM-AS	96.6%	96.6%
2.00	ZVC-like	95.7%	97.7%
2.00	GMM-AS	96.8%	92.5%
4.00	ZVC-like	96.4%	97.7%
4.00	GMM-AS	97.0%	80.7%

**Table 4**  
Evaluation of GMM-AS and ZVC-like at various noise standard deviation values.

Std. dev. (cm)	Algorithm	Precision	Recall
0.1	ZVC-like	95.4%	97.7%
0.1	GMM-AS	97.5%	98.4%
2.5	ZVC-like	95.1%	12.5%
2.5	GMM-AS	97.0%	80.2%
5.0	ZVC-like	97.5%	0.5%
5.0	GMM-AS	98.6%	56.8%

accuracy as shown in Table 7. A qualitative score from 1 to 10 was given to evaluate the completeness of the reconstruction of each object. In the first experiment KFOM is not significantly faster than KFOVP, even if KFOM requires one less view. This can be explained by the slower robot movement during acquisition required by KFOM. The difference in total completion time is, however, higher

tasks was measured by using the same parameters introduced in Section 6.1. KFOM and KFOVP achieve the same reconstruction



**Fig. 19.** Final environment representation for KFOM and KFOVP in the two experiments.

**Table 5**

The saliency computed for each Gaussian in Fig. 13.

Gaussian no.	Saliency	Threshold	POI
1	0.005880	–	NO
2	0.001646	0.040424	NO
3	0.001352	0.031084	NO
4	0.073616	0.002673	YES
5	0.001749	0.038675	NO
6	0.002383	–	NO

**Table 6**

Saliency computed for each Gaussian in Fig. 14.

Gaussian no.	Saliency	Threshold	POI
1	0.002341	–	NO
2	0.025118	0.002693	YES
3	0.001191	0.036059	NO
4	0.001855	0.035187	NO
5	0.066844	0.002574	YES
6	0.000679	0.035919	NO
7	0.003140	–	NO

**Table 7**

Reconstruction error, completeness, number of views and execution time for KFOM and KFOVP.

	Abs. error (m)	Compl.	Views	Time (s)
Experiment 1				
KFOVP	0.005	8	3	194
KFOM	0.008	9	2	191
Experiment 2				
KFOVP	0.005	10	5	402
KFOM	0.007	7	3	304

**Table 8**

Time (seconds) of the most relevant algorithm phases, averaged for each experiment.

Phase	Experiment 1		Experiment 2	
	KFOVP	KFOM	KFOVP	KFOM
Clear $S(POI)$	0.07	0.07	0.08	0.06
NBV-GPU	10.43	10.91	12.21	12.83
Robot movement	6.86	27.76	7.58	20.71

**Table 9**

Execution times for one phase of NBV-GPU, NBV-CPU and NBV-CPU-step (times in seconds).

	NBV-GPU	NBV-CPU	NBV-CPU-step
Execution time	10.6	211.5	619.6

**Table 10**

Comparison of overhead in standard KinFu LS and NBV-GPU (times in ms).

Phase	Std KinFu LS	NBV-GPU
Shifting	830	836
Weights download	0	283
Read octree (parallel)	0	3396
Weights upload	0	45
Write octree (parallel)	0	1638
Total time	830	6232

in the second experiment where KFOVP requires two views more than KFOM. Table 8 shows the execution times of specific phases of the algorithm. As expected, the robot movement is slower in KFOM. In addition, for both KFOVP and KFOM the NBV-GPU algorithm is about 20% slower in the second experiment than in the first. This indicates that the computational time depends on the shape of the environment. In general, it can be concluded that there are no large differences between KFOM and KFOVP in terms of reconstruction accuracy and completeness. KFOM is slightly faster than KFOVP when a large number of views is required.

Fig. 20 shows the number of unknown voxels in the 3D representation, for each POI. The number of voxels decreases as the robot explores the environment and it never reaches 0 due to unknown voxels inside objects. In experiment 2, the number of unknown voxels in the first POI, above the pallet, decreases almost to 0 at the first view. Also, in experiment 2 it can be noticed that when using KFOM the robot observes voxels in the second and third POI simultaneously as it travels to the first view pose to observe the first POI, hence the number of unknown voxels of the second and third POI decreases even before NBV planning starts for these two POIs.

#### 6.4. Evaluation of the developed KinFu LS extensions

A test scenario, shown in Fig. 21, was set up to compare the performance of NBV-GPU and NBV-CPU. A single POI is detected when a plastic bottle is placed by the user on top of a box. The NBV-GPU algorithm takes about 10.6 s to compute the next best view, thus confirming the results found in Section 6.3. In contrast, when running the NBV-CPU algorithm at the same resolution the experiment was aborted due to excessive execution time. In Table 9 the execution times for the first NBV phase of the experiment are reported, for each NBV algorithm. The step used for the NBV-CPU-step algorithm was 0.3 cm, about half a voxel size. These results confirm the advantage in terms of efficiency of using NBV-GPU, i.e. executing the ray casting phase on the GPU. In Table 10 the computational overhead introduced by NBV-GPU when downloading and uploading weight data between the *BitmaskOctree* and the TSDF volume is analyzed. A shifting operation takes about 6 s, compared to 800 ms of the standard KinFu LS implementation. However, thanks to the developed multithreading architecture, read and write operations on the octree run in parallel. Indeed, the actual overhead with respect to the standard KinFu LS implementation is only about 283 ms, which is due to the weight download phase, when KinFu must be locked exclusively. An additional exclusive lock must be acquired when uploading on the GPU the stored weight values from the CPU, however this phase requires only 45 ms.

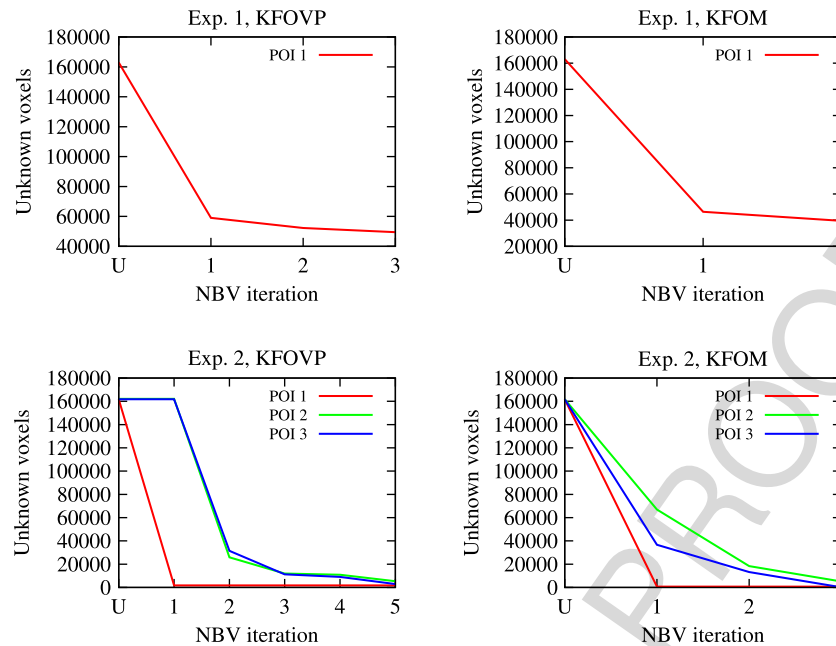


Fig. 20. The number of unknown voxels inside each uncertainty region, after user task (U) and at each robot NBV iteration.

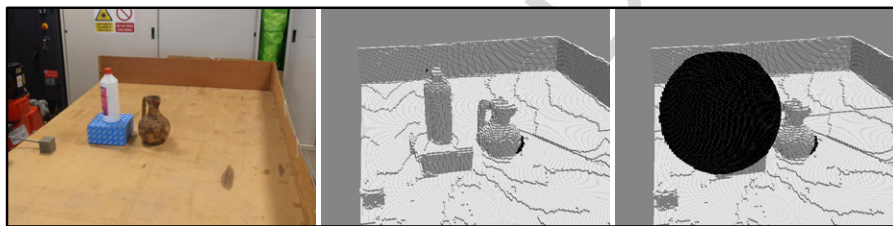


Fig. 21. Environment (left), ternary representation (center) and POI (right), for the experiment described in Section 6.4.

## 7. Conclusions

This paper presented a novel robot spatial attention approach for a robot manipulator equipped with a range sensor (Kinect) in eye-in-hand configuration. Relevant manipulation actions performed by the user are detected by tracking the motion of the human hand and by applying a GMM-based algorithm for saliency estimation. Approximate locations of the salient user actions are also computed and used to focus the robot's attention. The 3D environment representation is updated by exploiting a NBV planner and a modified version of the PCL KinFu algorithm. In contrast to previous works, next-best view planning is performed directly on the GPU to enable fast computation of NBV at high resolution. KinFu was also modified to exploit the robot kinematics to improve the accuracy of 3D reconstruction.

Future work will investigate strategies to further improve the quality of the reconstructed model, like planning the scan trajectories of the robot instead of planning discrete next best views. Human motion analysis could also be improved by exploiting Kinect range data and skeleton tracking. Moreover, a possible extension of the spatial attention system is to recognize both the objects and the type of actions performed by the user from the changes in the environment.

## References

- [1] KinectFusion extensions to large scale environments. <http://www.pointclouds.org/blog/srcs/theredia/index.php>.
- [2] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohi, J. Shotton, S. Hodges, A. Fitzgibbon, Kinectfusion: Real-time dense surface mapping and tracking, in: IEEE Intl Symposium on Mixed and Augmented Reality, ISMAR, 2011, pp. 127–136.
- [3] R. Rusu, S. Cousins, 3D is here: Point Cloud Library (PCL), in: IEEE Intl Conference on Robotics and Automation, ICRA, Shanghai, China, 2011.
- [4] R. Wagner, U. Frese, B. Bauml, Real-time dense multi-scale workspace modeling on a humanoid robot, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2013, pp. 5164–5171.
- [5] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, J. McDonald, Robust real-time visual odometry for dense RGB-D mapping, in: IEEE Intl Conference on Robotics and Automation, ICRA, 2013, pp. 5724–5731.
- [6] S. Petsch, D. Burschka, Representation of manipulation-relevant object properties and actions for surprise-driven exploration, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2011, pp. 1221–1227.
- [7] P. Alimi, D. Meger, J. Little, Object persistence in 3D for home robots, in: The Semantic Perception, Mapping, and Exploration, SPME Workshop, 2012.
- [8] A. Fathi, J. Rehg, Modeling actions through state changes, in: IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2013, pp. 2579–2586.
- [9] E.E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, F. Wörgötter, Learning the semantics of object-action relations by observation, Int. J. Robot. Res. 30 (10) (2011) 1229–1249.
- [10] E. Herbst, X. Ren, D. Fox, RGB-D object discovery via multi-scene analysis, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2011, pp. 4850–4856.
- [11] E. Herbst, P. Henry, D. Fox, Toward online 3-D object segmentation and mapping, in: IEEE Intl Conference on Robotics and Automation, ICRA, 2014, pp. 3193–3200.
- [12] F. Orabona, G. Metta, G. Sandini, Object-based visual attention: a model for a behaving robot, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR, 2005, pp. 89–89.
- [13] L. Goncalves, A. Oliveira, R. Grupen, A framework for attention and object categorization using a stereo head robot, in: XII Brazilian Symposium on Computer Graphics and Image Processing, 1999, 1999, pp. 143–152.
- [14] J. Clark, Spatial attention and the maintenance of representations of a robot's environment, in: Workshop on Perception for Mobile Agents, 1998.
- [15] R. Finman, T. Whelan, M. Kaess, J. Leonard, Toward lifelong object segmentation from change detection in dense RGB-D maps, in: European Conference on Mobile Robots, ECMR, 2013, pp. 178–185.
- [16] P. Drews, P. Núñez, R. Rocha, M. Campos, J. Dias, Novelty detection and segmentation based on gaussian mixture models: A case study in 3D robotic laser mapping, Robot. Auton. Syst. 61 (12) (2013) 1696–1709.

- [17] B. Schauerte, J. Richarz, G. Fink, Saliency-based identification and recognition of pointed-at objects, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2010, pp. 4638–4643.
- [18] Z. Yücel, A. Salah, C. Meriçli, T. Meriçli, R. Valenti, T. Gevers, Joint attention by gaze interpolation and saliency, IEEE Trans. Cybern. 43 (3) (2013) 829–842.
- [19] B. Adler, J. Xiao, J. Zhang, Finding next best views for autonomous UAV mapping through GPU-accelerated particle simulation, in: IEEE/RSJ Intl Conference on Intelligent Robots and Systems, IROS, 2013, pp. 1056–1061.
- [20] J. Banta, L.R. Wong, C. Dumont, M. Abidi, A next-best-view system for autonomous 3-D object reconstruction, IEEE Trans. Syst. Man Cybern. A 30 (5) (2000) 589–598.
- [21] C. Connolly, The determination of next best views, in: IEEE Intl Conference on Robotics and Automation, ICRA, Vol. 2, 1985, pp. 432–435.
- [22] R. Pito, A solution to the next best view problem for automated surface acquisition, IEEE Trans. Pattern Anal. Mach. Intell. 21 (10) (1999) 1016–1030.
- [23] L. Torabi, K. Gupta, Integrated view and path planning for an autonomous six-DOF eye-in-hand object modeling system, in: IEEE/RSJ Intl Conference on Intelligent Robots and Systems, IROS, 2010, pp. 4516–4521.
- [24] S. Kriegel, C. Rink, T. Bodenmuller, A. Narr, M. Suppa, G. Hirzinger, Next-best-scan planning for autonomous 3D modeling, in: IEEE/RSJ Intl Conference on Intelligent Robots and Systems, IROS, 2012, pp. 2850–2856.
- [25] S. Foix, G. Alenyà, J. Andrade-Cetto, C. Torras, Object modeling using a ToF camera under an uncertainty reduction approach, in: IEEE Intl Conference on Robotics and Automation, ICRA, 2010, pp. 1306–1312.
- [26] K. Morooka, H. Zha, T. Hasegawa, Next Best Viewpoint (NBV) planning for active object modeling based on a learning-by-showing approach, in: Fourteenth Intl Conference on Pattern Recognition, Vol. 1, 1998, pp. 677–681.
- [27] Y. Li, Z. Liu, Information entropy-based viewpoint planning for 3-D object reconstruction, IEEE Trans. Robot. 21 (3) (2005) 324–337.
- [28] G. Walck, M. Drouin, Automatic observation for 3D reconstruction of unknown objects using visual servoing, in: IEEE/RSJ Intl Conference on Intelligent Robots and Systems, IROS, 2010, pp. 2727–2732.
- [29] S. Chen, Y. Li, Vision sensor planning for 3-D model acquisition, IEEE Trans. Syst. Man Cybern. B 35 (5) (2005) 894–904.
- [30] P. Whaite, F. Ferrie, Autonomous exploration: Driven by uncertainty, IEEE Trans. Pattern Anal. Mach. Intell. 19 (3) (1997) 193–205.
- [31] S. Kriegel, M. Brucker, Z.C. Marton, T. Bodenmuller, M. Suppa, Combining object modeling and recognition for active scene exploration, in: IEEE/RSJ Intl Conference on Intelligent Robots and Systems, IROS, 2013, pp. 2384–2391.
- [32] C. Potthast, G.S. Sukhatme, A probabilistic framework for next best view estimation in a cluttered environment, J. Vis. Commun. Image Represent. 25 (1) (2014) 148–164.
- [33] J. Aleotti, D. Lodi Rizzini, R. Monica, S. Caselli, Global registration of mid-range 3D observations and short range next best views, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2014.
- [34] S.H. Lee, I.H. Suh, S. Calinon, R. Johansson, Learning basis skills by autonomous segmentation of humanoid motion trajectories, in: IEEE-RAS Intl Conference on Humanoid Robots, 2012, pp. 112–119.
- [35] J.-S. Lin, D. Kulic, Online segmentation of human motion for automated rehabilitation exercise analysis, IEEE Trans. Neural Syst. Rehabil. Eng. 22 (1) (2014) 168–180.
- [36] D. Faria, J. Dias, 3D hand trajectory segmentation by curvatures and hand orientation for classification through a probabilistic approach, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2009, pp. 1284–1289.
- [37] D.R. Faria, R. Martins, J. Lobo, J. Dias, Extracting data from human manipulation of objects towards improving autonomous robotic grasping, Robot. Auton. Syst. 60 (3) (2012) 396–410.
- [38] S.B. Kang, K. Ikeuchi, Toward automatic robot instruction from perception-temporal segmentation of tasks from human hand motion, IEEE Trans. Robot. Autom. 11 (5) (1995) 670–681.
- [39] M. Yeasin, S. Chaudhuri, Toward automatic robot programming: learning human skill from visual data, IEEE Trans. Syst. Man Cybern. B 30 (1) (2000) 180–185.
- [40] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, J. Kuffner, Grasp planning in complex scenes, in: 7th IEEE-RAS Int'l Conference on Humanoid Robots, 2007, pp. 42–48.
- [41] H. Roth, M. Vona, Moving volume Kinectfusion, in: Proceedings of the British Machine Vision Conference, BMVA Press, 2012, pp. 112.1–112.11.
- [42] M. Korn, KinFu ROS wrapper <http://fsstud.is.uni-due.de/svn/ros/jis/kinfu>.



**Riccardo Monica** is a Ph.D. student at the Department of Information Engineering of the University of Parma. In 2014 he received the Master degree in **Computer Engineering**. His main research interests lie in robot perception of 3D environments.



**Jacopo Aleotti** received his Laurea degree in Electronic Engineering and Ph.D. degree in Computer Engineering, both from the University of Parma, Italy, in 2002 and 2006, respectively. Since 2014 he is fixed-time assistant professor at the University of Parma. From 2006 to 2013, he conducted postdoctoral research with the Department of Information Engineering, University of Parma. In 2003, he was Marie Curie Fellow at the Learning System Laboratory, Orebro University, Sweden. His current research interests include robot learning from demonstration, robot manipulation and range sensing, human-robot interaction and virtual reality.



**Stefano Caselli** is Professor of Computer Engineering and Director of the Laboratory of Robotics and Intelligent Machines (RIMLab) at the Department of Information Engineering of the University of Parma, Italy. He received his Laurea degree in Electronic Engineering in 1982 and his Ph.D. degree in Computer and Electronic Engineering in 1987, both from the University of Bologna, Italy. His current research interests include the development of autonomous and remotely operated robot systems, service robotics, and real-time systems.