SLAM++: Simultaneous Localisation and Mapping at the Level of Objects

Renato F. Salas-Moreno[1] Richard A. Newcombe[2] Hauke Strasdat[1] Paul H. J. Kelly[1]
Andrew J. Davison[1]

[1]Imperial College London [2]University of Washington

Abstract

We present the major advantages of a new 'object oriented' 3D SLAM paradigm, which takes full advantage **in the loop** *of prior knowledge that many scenes consist of repeated, domain-specific objects and structures. As a hand-held depth camera browses a cluttered scene, real-time 3D object recognition and tracking provides 6DoF camera-object constraints which feed into an explicit graph of objects, continually refined by efficient pose-graph optimisation. This offers the descriptive and predictive power of SLAM systems which perform dense surface reconstruction, but with a huge representation compression. The object graph enables predictions for accurate ICP-based camera to model tracking at each live frame, and efficient active search for new objects in currently undescribed image regions. We demonstrate real-time incremental SLAM in large, cluttered environments, including loop closure, relocalisation and the detection of moved objects, and of course the generation of an object level scene description with the potential to enable interaction.*

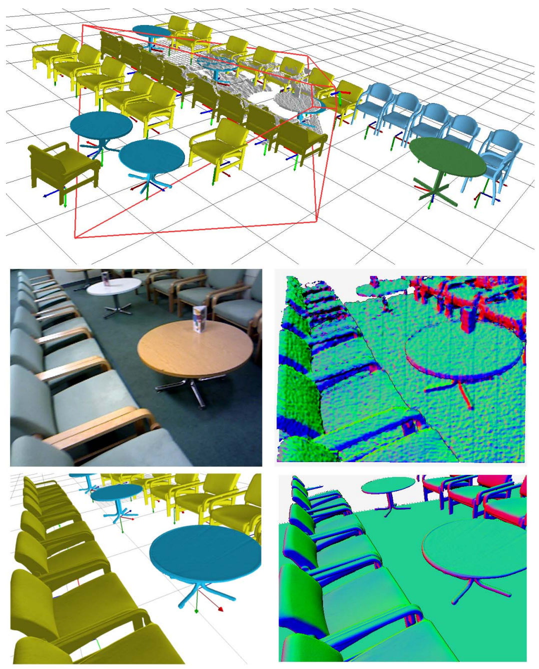


Figure 1. **(top)** In SLAM++, a cluttered 3D scene is efficiently tracked and mapped in real-time directly at the object level. **(left)** A live view at the current camera pose and the synthetic rendered objects. **(right)** We contrast a raw depth camera normal map with the corresponding high quality prediction from our object graph, used both for camera tracking and for masking object search.

1. Introduction

Most current real-time SLAM systems operate at the level of low-level primitives (points, lines, patches or non-parametric surface representations such as depth maps), which must be robustly matched, geometrically transformed and optimised over in order that maps represent the intricacies of the real world. Modern processing hardware permits ever-improving levels of detail and scale, and much interest is now turning to semantic labelling of this geometry in terms of the objects and regions that are known to exist in the scene. However, some thought about this process reveals a huge amount of wasted computational effort; and the potential for a much better way of taking account of domain knowledge *in the loop of SLAM operation itself*.

We propose a paradigm for real-time localisation and mapping which harnesses 3D object recognition to jump over low level geometry processing and produce incrementally built maps directly at the 'object oriented' level. As a hand-held depth camera browses a cluttered scene, prior knowledge of the objects likely to be repetitively present enables real-time 3D recognition and the creation of a simple pose graph map of relative object locations. This graph is continuously optimised as new measurements arrive, and enables always up-to-date, dense and precise prediction of the next camera measurement. These predictions are used for robust camera tracking and the generation of active search regions for further object detection.

Our approach is enabled by efficient GPGPU parallel implementation of recent advances in real-time 3D object detection and 6DoF (Degree of Freedom) ICP-based pose refinement. We show that alongside the obvious benefit of an object-level scene description, this paradigm enables a vast compression of map storage compared to a dense reconstruction system with similar predictive power; and that it easily enables large scale loop closure, relocalisation and great potential for the use of domain-specific priors.

2. Real-Time SLAM with Hand-Held Sensors

In SLAM (Simultaneous Localisation and Mapping), building an internally consistent map in real-time from a moving sensor enables drift-free localisation during arbitrarily long periods of motion. We have still not seen truly 'pick up and play' SLAM systems which can be embedded in low-cost devices and used without concern or understanding by non-expert users, but there has been much recent progress. Until recently, the best systems used either monocular or stereo passive cameras. Sparse feature filtering methods like [5] were improved on by 'keyframe SLAM' systems like PTAM [8] which used bundle adjustment in parallel with tracking to enable high feature counts and more accurate tracking.

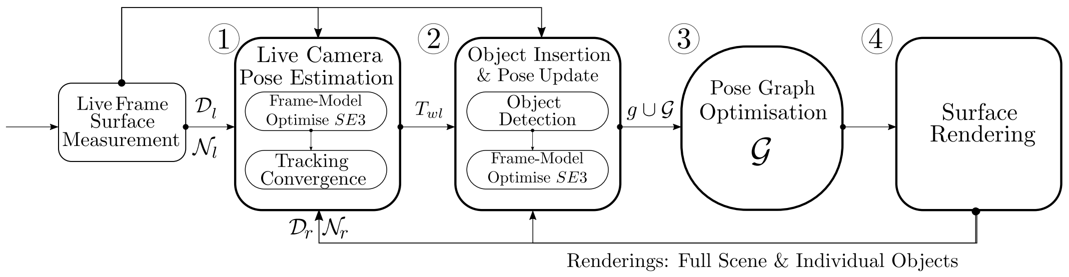
Most recently, a breakthrough has been provided by 'dense SLAM' systems, which take advantage of GPGPU processing hardware to reconstruct and track full surface models, represented non-parametrically as meshes or implicit surfaces. While this approach is possible with an RGB camera [12], commodity depth cameras have now come to the fore in high performance, robust indoor 3D mapping, in particular via the KinectFusion algorithm [11]. New developments such as [18] have tackled scaling the method via a sliding volume, sub-blocking or octrees; but a a truly scalable, multi-resolution, loop closure capable dense non-parametric surface representation remains elusive, and will always be wasteful in environments with symmetry.

From sparse feature-based SLAM, where the world is modelled as an unconnected point cloud, to dense SLAM which assumes that scenes contain continuous surfaces, we have seen an increase in the prior scene knowledge brought to bear. In SLAM++ we step up to the even stronger assumption that the world has intrinsic symmetry in the form of repetitive objects. While we currently pre-define the objects expected in a scene, we intend that the paradigm permits the objects in a scene to be identified and segmented automatically as salient, repeated elements.

Object SLAM has many characteristics of a return to feature-based SLAM methods. Unlike dense non-parametric approaches, the relatively few discrete entities in the map makes it highly feasible to jointly optimise over all object positions to make globally consistent maps. The fact that the map entities are now objects rather than point features, however, puts us in a stronger position than ever before; tracking one object in 6DoF is enough to localise a camera, and reliable relocalisation of a lost camera or loop closure detection can be performed on the basis of just a small number of object measurements due to their high saliency. Further, and crucially, instant recognition of objects provides great efficiency and robustness benefits via the active approaches it permits to tracking and object detection, guided entirely by the dense predictions we can make of the positions of known objects.

SLAM++ relates strongly to the growing interest in semantically labelling scene reconstructions and maps, in both the computer vision and robotics communities, though we stress the big difference between post-hoc labelling of geometry and the closed loop, real-time algorithm we present. Some of the most sophisticated recent work was by Kim *et al*. [7]. A depth camera is first used to scan a scene, similar in scale and object content to the results we demonstrate later, and all data is fused into a single large point cloud. Off-line, learned object models, with a degree of variation to cope with a range of real object types, are then matched into the joint scan, optimising both similarity and object configuration constraints. The results are impressive, though the emphasis is on labelling rather than aiding mapping and we can see problems with missing data which cannot be fixed with non-interactive capture. Other good work on labelling using RDB-D data was by Silberman [16] as well as Ren *et al*. [14] who used kernel descriptors for appearance and shape to label single depth camera images with object and region identity.

Several published approaches use object detection, like we do, with the aim of not just labelling but actually improving reconstruction and tracking; but none of these have taken the idea nearly as far as we have with the combination of real-time processing, full 3D operation, dense prediction and a modern graph optimisation back-end. To give a flavour of this work, Castle *et al*. [4] incorporated planar object detection into sparse feature-based monocular SLAM [5]. These objects, once recognized via SIFT descriptors, improved the quality of SLAM due to their known size and shape, though the objects were simple highly textured posters and the scene scale small. Also in an EKF SLAM paradigm, Ramos *et al*. [13] demonstrated a 2D laser/camera system which used object recognition to generate discrete entities to map (tree trunks) rather than using raw measurements. Finally, the same idea that object recognition aids reconstruction has been used in off-line structure from motion. Bao *et al*. [3] represented a scene as a set of points, objects and regions in two-view SfM, solving expensively and jointly in a graph optimisation for a labelling and reconstruction solution taking account of interactions between all scene entities.



1
Live Camera Pose Estimation
Frame-Model Optimise $SE3$
Tracking Convergence
2
Object Insertion & Pose Update
Object Detection
Frame-Model Optimise $SE3$
3
Pose Graph Optimisation
$\mathcal{G}$
4
Surface Rendering
Live Frame Surface Measurement
$\mathcal{D}_l$
$\mathcal{N}_l$
T_{wl}
$g \cup \mathcal{G}$
$\mathcal{D}_r$ $\mathcal{N}_r$
Renderings: Full Scene & Individual Objects

Figure 2. Outline of the SLAM++ pipeline. Given a live depth map $\mathcal{D}_l$, we first compute a surface measurement in the form of a vertex and normal map $\mathcal{N}_l$ providing input to the sequentially computed camera tracking and object detection pipelines. **(1)** We track the live camera pose T_{wl} with an iterative closest point approach using the dense multi-object scene prediction captured in the current SLAM graph $\mathcal{G}$. **(2)** Next we attempt to detect objects, previously stored in a database, that are present in the live frame, generating detection candidates with an estimated pose in the scene. Candidates are first refined or rejected using a second ICP estimation initialised with the detected pose. **(3)** We add successfully detected objects g into the SLAM graph in the form of a object-pose vertex connected to the live estimated camera-pose vertex via a measurement edge. **(4)** Rendering objects from the SLAM graph produce a predicted depth $\mathcal{D}_r$ and normal map $\mathcal{N}_r$ into the live estimated frame, enabling us to actively search only those pixels not described by current objects in the graph. We run an individual ICP between each object and the live image resulting in the addition of a new camera-object constraint into the SLAM graph.

3. Method

SLAM++ is overviewed in Figure 2, and detailed below.

3.1. Creating an Object Database

Before live operation in a certain scene, we rapidly make high quality 3D models of repeatedly occurring objects via interactive scanning using KinectFusion [11] in a controlled setting where the object can easily be circled without occlusion. A mesh for the object is extracted from the truncated signed distance volume obtained from KinectFusion using marching cubes [10]. A small amount of manual editing in a mesh tool is performed to separate the object from the ground plane, and mark it up with a coordinate frame such that domain-specific object constraints can be applied. The reconstructed objects are then stored in a simple database.

3.2. SLAM Map Representation

Our representation of the world is a graph, where each node stores either the estimated $\mathbf{SE}(3)$ pose (rotation and translation relative to a fixed world frame) T_{wo_j} of object j, or T_{wi} of the historical pose of the camera at timestep i (see Figure 5). Each object node is annotated with a type from the object database. Each $\mathbf{SE}(3)$ measurement of the pose of an object Z_{i,o_j} from the camera is stored in the graph as a factor (constraint) which links one camera pose and one object pose. Additional factors can optionally be added to the graph; between camera poses to represent camera-camera motion estimates (e.g. from ICP), or domain-specific *structural* priors, e.g. that certain types of objects must be grounded on the same plane. Details on graph optimisation are given in Section 3.5.

3.3. Real-Time Object Recognition

We follow the approach of Drost *et al.* [6] for recognising the 6DoF pose of 3D objects, represented by meshes, in a depth image. We give details of our parallel implementation, which achieves the real-time detection of multiple instances of multiple objects we need by fully exploiting the fine-grained parallelism of GPUs.

As in the Generalised Hough Transform [2], in Drost *et al.*'s method an object is detected and simultaneously localised via the accumulation of votes in a parameter space. The basis of voting is the correspondence between Point-Pair Features (PPFs); four-dimensional descriptors of the relative position and normals of pairs of oriented points on the surface of an object. Points, with normal estimates, are randomly sampled on a bilateral-filtered image from the depth camera. These samples are paired up in all possible combinations to generate PPFs which vote for 6DoF model configurations containing a similar PPF.

A global description for each object mesh is quickly generated on the GPU by discretising PPFs with similar values and storing them in search data structures built using parallel primitive operations such as reduction, scan and sort (see Algorithm 1), provided by modern GPU template libraries such as Bolt [1]. Similar structures are also built from each live frame. This process typically takes <5ms for 160K PPFs and could also be used in the future to describe new object classes on the fly as they are automatically segmented.

Matching similar features of the scene against the model can be efficiently performed in parallel via a vectorised binary search, producing a vote for each match. Accumulation of votes into a shared buffer can be prohibitively expensive on the GPU, where many threads would require atomic operations when incrementing a common memory location

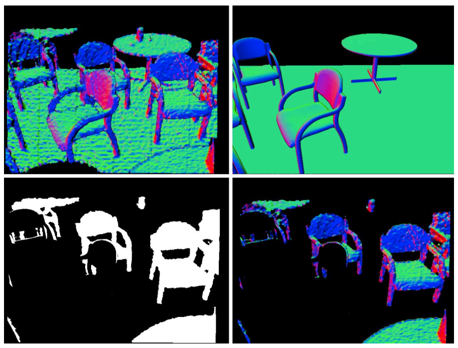


Figure 3. Object recognition with active search. **(top-left)** Measured normal map from sensor. **(top-right)** Predicted view from graph. **(bottom-left)** Undescribed regions to be searched are in white. **(bottom-right)** Masked normal map used for recognition.

Algorithm 1: Build global description from PPFs

```
input  : A set of $N$ point-pair features (PPF)
output: Set of search data structures
// Encode the PPF index and hash key
$codes \leftarrow$ new $array$;
foreach $i \leftarrow 0$ to $N$ - 1 in parallel do
    Compute the hash key $hk$ from PPF $i$;
    $codes[i] = hk \ll 32 + i$;
$codes \leftarrow$ Sort($codes$);
// Decode PPF index and hash key
$key2ppfMap \leftarrow$ new $array$;
$hashKeys \leftarrow$ new $array$;
foreach $i \leftarrow 0$ to $N$ - 1 in parallel do
    $key2ppfMap[i] = \sim(1 \ll 32)$ & $code[i]$;
    $hashKeys[i] = code[i] \gg 32$;
// Count PPF per hash key and remove
   duplicate keys
$hashKeys, ppfCount \leftarrow$ Reduce($hashKeys,$
$sumOp$);
// Find the first PPF index of a block
   of PPFs in key2ppfMap having equal
   hash key
$firstPPFIndex \leftarrow$ ExclScan($ppfCount$);
return $hashKeys, ppfCount, firstPPFIndex,$
$key2ppfMap$;
```

to avoid race conditions. To overcome this, each vote is represented as a 64-bit integer code (Figure 4), which can then be efficiently sorted and reduced in parallel.

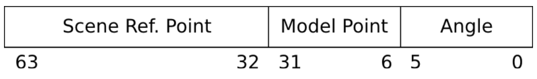


Figure 4. Packed 64-bit integer vote code. The first 6 bits encode the alignment angle, followed by 26 bits for the model point and 32 bits for the scene reference point.

Sorting puts corresponding model points, scene reference points and alignment angles contiguously in memory. This is followed by a parallel reduction with a *sum* operation to accumulate equal vote codes (Algorithm 2). After peak finding, pose estimates for each scene reference point are clustered on the CPU according to translation and rotation thresholds as in [6].

Algorithm 2: Accumulate Votes

```
input  : A set of $M$ vote codes
output: Vote count per unique vote code
// Sort voteCodes in parallel
$voteCodes \leftarrow$ Sort($voteCodes$);
// Count votes with equal voteCode
$voteCodes, voteCount \leftarrow$ Reduce($voteCodes,$
$sumOp$);
return $voteCodes, voteCount$;
```

3.3.1 Active Object Search

Standard application of Drost *et al.*'s recognition algorithm [6] in room scenes is highly successful when objects occupy most of the camera's field of view, but poor for objects which are distant or partly occluded by other objects, due to poor sample point coverage. It is here that we realise one of the major benefits of our in-the-loop SLAM++ approach. The view prediction capabilities of the system mean that we can generate a mask in image space for depth pixels which are not already described by projected known objects. The measured depth images from the camera are cropped using these masks and samples are only spread in the undescribed regions (see Figure 3).

The result of object detection is often multiple cluster peaks, and quantised localisation results. These must be passed to ICP refinement as explained in the next section.

3.4. Camera Tracking and Accurate Object Pose Estimation using ICP

Camera-Model Tracking: In KinectFusion [11], the up-to-date volumetric map is leveraged to compute a view prediction into the previously estimated camera pose, enabling estimation of the live camera using a fast dense iterative closest point (ICP) algorithm [15]. In SLAM++, in contrast to the relatively incomplete models available at early stages of mapping with KinectFusion, we track against a complete high quality multi-object model prediction. Following [11], we compute a reference view prediction of the current scene geometry consisting of a depth map $\mathcal{D}_r$ and normal map $\mathcal{N}_r$ rendered into the previously estimated frames pose T_{wr} representing the 6DoF rigid body transform defined as a member of the special Euclidean group

$\mathbf{SE}(3)$. We update the live camera to world transform T_{wl} by estimating a sequence of m incremental updates $\{\tilde{T}_{rl}^{n}\}_{n=1}^{m}$ parametrised with a vector $\mathbf{x} \in \mathbb{R}^6$ defining a twist in $\mathbf{SE}(3)$, with $\tilde{T}_{rl}^{n=0}$ as the identity. We iteratively minimise the whole depth image point-plane metric over all available valid pixels $u \in \Omega$ in the live depth map:

$$E_c(\mathbf{x}) = \sum_{u\in\Omega} \psi\left(e(u,\mathbf{x})\right) \ , \tag{1}$$

$$e(u,\mathbf{x}) = \mathcal{N}_r(u')^\top \left(\exp(\mathbf{x})\hat{v}_l(u) - v_r(u')\right) . \tag{2}$$

Here $v_r(u')$ and $\mathcal{N}_r(u')$ are the projectively data associated predicted vertex and normal estimated at a pixel correspondence $u' = \pi(K\hat{v}_l(u))$, computed by projecting the vertex $v_l(u)$ at pixel u from the live depth map into the reference frame with camera intrinsic matrix K and standard pin-hole projection function π. The current live vertex is transformed into the reference frame using the current incremental transform $\tilde{T}_{rl}^{n}$:

$$\hat{v}_l(u) = \tilde{T}_{rl}^{n} v_l(u) \ , \tag{3}$$

$$v_l(u) = K^{-1}\dot{u}\mathcal{D}_l(u) \ , \tag{4}$$

$$v_r(u') = K^{-1}\dot{u}'\mathcal{D}_r(u') \ . \tag{5}$$

We chose ψ as a robust Huber penalty function in place of the explicit point compatibility check used in [11] which enables a soft outlier down weighting. A Gauss-Newton based gradient descent on Equation (1) results in solution of the normal equations:

$$\sum_{u\in\Omega} J(u)^\top J(u)\mathbf{x} = \sum_{u\in\Omega} \psi^{'}(e)J(u) \ , \tag{6}$$

where $J(u) = \frac{\partial e(\mathbf{x},u)}{\partial \mathbf{x}}$ is the Jacobian, and ψ' computes the robust penalty function derivative given the currently estimated error. We solve the 6×6 linear system using Cholesky decomposition. Taking the solution vector $\mathbf{x}$ to an element in $\mathbf{SE}(3)$ via the exponential map, we compose the computed incremental transform at iteration $n+1$ onto the previous estimated transform $\tilde{T}_{rl}^{n}$:

$$\tilde{T}_{rl}^{n+1} \leftarrow \exp(\mathbf{x})\tilde{T}_{rl}^{n} \ . \tag{7}$$

The estimated live camera pose T_{wl} therefore results by composing the final incremental transform $\tilde{T}_{rl}^{m}$ onto the previous frame pose:

$$T_{wl} \leftarrow T_{wr}\tilde{T}_{rl}^{m} \ . \tag{8}$$

Tracking Convergence: We use a maximum of $m = 10$ iterations and check for poor convergence of the optimisation process using two simple criteria. First, we do not attempt to track against the predicted model if its pixel coverage is less than $\frac{1}{8}$ of a full image. Second, after an optimisation iteration has completed we compute the ratio of pixels in the live image which have been correctly matched with the predicted model ascertained by discounting pixels which induce a point-plane error greater than a specified magnitude ϵ_{pp}.

Tracking for Model Initialisation: We utilise the dense ICP pose estimation and convergence check for two further key components in SLAM++. First, we note that the real-world objects we use as features are often ambiguous, and not well discriminated given a single view. Therefore, given a candidate object and detected pose, we run camera-model ICP estimation on the *detected* object pose, and check for convergence using the previously described criteria. We find that for correctly detected objects, the pose estimates from the detector are erroneous within $\pm 30^\circ$ rotation, and ± 50cm translation. This allows a more conservatively set threshold ϵ_{oi} and early rejection of incorrect objects.

Camera-Object Pose Constraints: Given the active set of objects that have been detected in SLAM++, we further estimate relative camera-object pose parameters which are used to induce constraints in the scene pose graph. To that end, we run the dense ICP estimate between the live frame and *each* model object currently visible in the frame. The ability to compute an individual relative pose estimate introduces the possibility to prune poorly initialised or incorrectly tracked objects from the pose graph at a later date. By analysing the statistics of the camera-object pose estimate's convergence we can keep an inlier-outlier style count on the inserted objects, and cull poorly performing ones.

3.5. Graph Optimisation

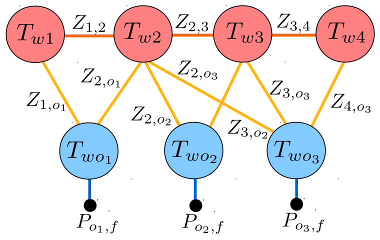


Figure 5. Example graph illustrating the pose of the moving camera over four time steps T_{wi} (red) as well as the poses of three static objects in the world T_{wo_j} (blue). Observations of the object o_j at time i are shown as binary camera-object constraints Z_{i,o_j} (yellow) while the relative ICP constraints between two cameras are shown as $Z_{i,i+1}$ (orange). We also apply unary structural constraints $P_{o2,f}$, $P_{o3,f}$ encoding prior information.

We formulate the problem of estimating the historical poses of the depth camera T_{wi} at time i and the poses of the static objects T_{wo_j} as graph optimisation. Z_{i,o_j} denotes the 6DoF measurement of object j in frame i and Σ_{i,o_j}^{-1} its inverse measurement covariance which can be estimated using the approximated Hessian $\Sigma_{i,o_j}^{-1} = J^\top J$ (with J being the Jacobian from the final iteration of the object ICP). $Z_{i,i+1}$ is the relative ICP constraint between camera i and $i+1$, with $\Sigma_{i,i+1}^{-1}$ the corresponding inverse covariance. The absolute poses T_{wi} and T_{wo_j} are *variables* which are modified during the optimisation, while Z_{i,o_j} and $Z_{i,i+1}$ are measurements and therefore *constants*. All variables and measurements have 6DoF and are represented as members of $\mathbf{SE}(3)$. An example graph is shown in Figure 5.

We minimize the sum over all measurement constraints:

$$\begin{aligned} E_m &= \sum_{Z_{i,o_j}} || \log(Z_{i,o_j}^{-1} \cdot T_{wi}^{-1} \cdot T_{wo_j}) ||_{\Sigma_{i,o_j}} \\ &+ \sum_{Z_{i,i+1}} || \log(Z_{i,i+1}^{-1} \cdot T_{wi}^{-1} \cdot T_{wi+1}) ||_{\Sigma_{i,i+1}} \quad (9) \end{aligned}$$

with $||\mathbf{x}||_\Sigma := \mathbf{x}^\top \Sigma^{-1} \mathbf{x}$ the Mahalanobis distance and $\log(\cdot)$ the logarithmic map of $\mathbf{SE}(3)$. This generalised least squares problem is solved using Levenberg-Marquardt; the underlying normal equation's sparsity is exploited using a sparse Cholesky solver [9]. The pose Jacobians are of the form $\pm \frac{\partial}{\partial \epsilon_i} \log(A \cdot \exp(\epsilon) \cdot B)|_{\epsilon=0}$, approximated using higher order Baker-Campell-Haussdorf expansions [17].

3.5.1 Including Structural Priors

Additional information can be incorporated in the graph in order to improve the robustness and accuracy of the optimisation problem. In our current implementation we apply a structural planar prior that the objects are located on a common ground plane. The world reference frame w is defined in such that the x and z-axes lie within the ground plane with the y-axis perpendicular into it. The ground plane is implicitly detected from the initial observation Z_{1,o_1} of the first object; its pose T_{wf} remains fixed during optimisation. To penalize divergence of the objects from the ground plane, we augment the energy,

$$E_{m\&p} = E_m + \sum_{P_{o_j,f}} || \log(P_{o_j,f}^{-1} \cdot T_{wo_j}^{-1} \cdot T_{wf}) ||_{\Sigma_{o_j,f}} \ , \quad (10)$$

using a set of unary constraints $P_{o_j,f} = \exp((\upsilon^\top, \theta^\top)^\top)$ with $\upsilon, \theta \in \mathbb{R}^3$ (see Figure 5). In particular, we set the translation along the y-axis, $\upsilon_2 = 0$, as well as the rotational components about the x and z-axis to zero, $\theta_1 = 0$ and $\theta_3 = 0$. We use the following inverse prior covariances:

$$\Sigma_{o_j,f}^{-1} = \text{diag}(0, w_{\text{trans}}, 0, w_{\text{rot}}, 0, w_{\text{rot}}) \ , \quad (11)$$

with positive weights w_{trans} and w_{rot}. Since the objects can be located anywhere within the $x-z$ plane and are not constrained in their rotation about the y-axis, the corresponding weights are set to zero; the prior components $\upsilon_1, \upsilon_3, \theta_2$ do not affect the energy and can be set to arbitrary values.

3.6. Other Priors

The ground plane constraint can have value beyond the pose graph. We could guide point-pair feature sampling and make votes for poses only in the unconstrained degrees of freedom of objects. In the ICP method, again a prior could be used as part of the energy minimisation procedure to pull it always towards scene-consistent poses. While this is not yet implemented we at least cull hypotheses of object positions far from the ground plane.

3.7. Relocalisation

When camera tracking is lost the system enters a relocalisation mode. Here a new local graph is created and tracked from, and when it contains at least 3 objects it is matched against the previously tracked long-term graph (see Figure 6). Graph matching is achieved by considering both the local and long-term graphs as sets of oriented points in a mesh that are fed into the same recognition procedure described in Section 3.3. We use the position of the objects as vertices and their x-axes as normals. The matched vertex with highest vote in the long-term graph is used instead of the currently observed vertex in the local graph and camera tracking is resumed from it, discarding the local map.

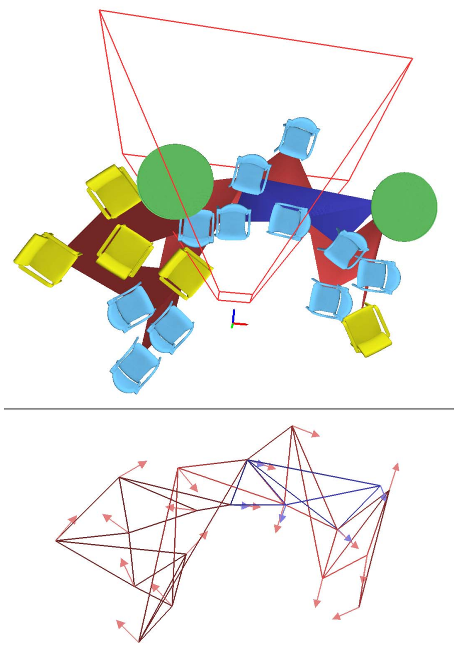


Figure 6. Relocalisation procedure. When tracking is lost a local graph (blue) is created and matched against a long-term graph (red). **(top)** Scene with objects and camera frustum when tracking is resumed a few frames after relocalisation. **(bottom)** Oriented points extracted for matching. The connectivity depicts the detection sequence and is not used by the recognition procedure.

4. Results

The in-the-loop operation of our system is more effectively demonstrated in our *submitted video* than on paper, where the advantages of our method over off-line scene labelling may not be immediately obvious. We present demonstrations of a new level of real-time localisation and mapping performance, surpassing previous SLAM systems in the quality and density of geometry description obtained given the very small footprint of our representation; and rivalling off-line multi-view scene labelling systems in terms of object identification and configuration description.

4.1. Loop Closure

Loop closure in SLAM occurs when a location is revisited after a period of neglect, and the arising drift corrected. In SLAM++, small loops are regularly closed using the standard ICP tracking mechanism. Larger loop closures (see Figure 7), where the drift is too much to enable matching via predictive ICP, are detected using a module on based matching fragments within the main long-term graph in the same manner as in relocalisation (Section 3.7).

4.2. Large Scale Mapping

We have run SLAM++ in environments including the large common room shown in Figure 1 (size 15×10×3m), with two types of chair and two types of round table, all constrained to a common ground plane. The real-time process lasted around 10 minutes, including various loop closures, relocalisations due to lost tracking. 34 objects were mapped across the room. Figure 1 gives a good idea of mapping performance. Note that there are no priors in the system concerning the regular positioning of tables and chairs, only that they sit on the ground plane.

4.3. Moved Object Detection

We demonstrate the ability to detect the movement of objects, which fail ICP gating due to inconsistency (Figure 8).

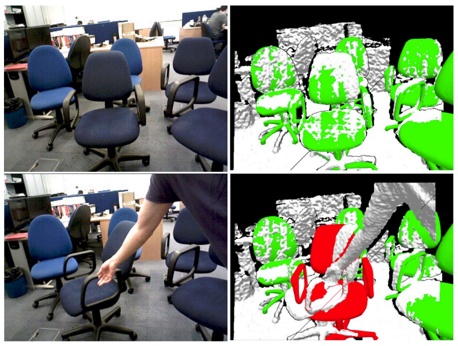


Figure 8. When a mapped object is largely inconsistent with good measurements (green) it is marked as invalid (red) and observations from it are stopped to avoid corrupting the rest of the graph.

4.4. Augmented Reality with Objects

Finally, the ability to semantically predict complete surface geometry from partial views allows novel context-aware AR capabilities such as path finding, to gracefully avoid obstacles while reaching target objects. We apply this to command virtual characters to navigate the scene and find places to sit as soon as the system is started (without the need to scan a whole room). (Figure 9).

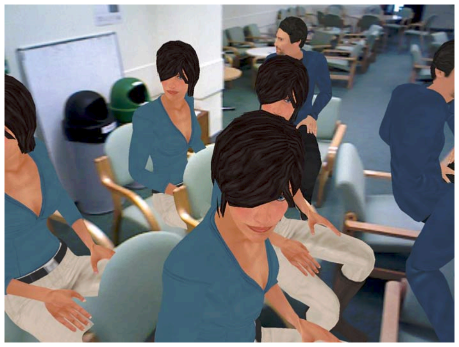


Figure 9. Context-aware augmented reality: virtual characters navigate the mapped scene and automatically find sitting places.

4.5. System statistics

We present system settings for mapping the room in Figure 7 (10×6×3m) using a gaming laptop. We compare the memory footprint of SLAM++ with KinectFusion [11] for the same volume (assuming 4 Bytes/voxel, 128 voxels/m).

Framerate	20 fps
Camera Count	132
Object Count	35
Object Class Count	5
Edge Count	338
Graph Memory	350 KB
Database Memory	20 MB
KinectFusion Memory	1.4 GB
Approx. Compression Ratio	1/70

5. Conclusions and Future Work

In this paper we have shown that using high performance 3D object recognition in the loop permits a new approach to real-time SLAM with large advantages in terms of efficient and semantic scene description. In particular we demonstrate how the tight interaction of recognition, mapping and tracking elements is mutually beneficial to all. Currently our approach is well suited to locations like the interiors of public buildings with many repeated, identical elements, but we believe is the first step on a path to more generic SLAM methods which take advantage of objects with low-dimensional shape variability or in the long term which can segment and define their own object classes.

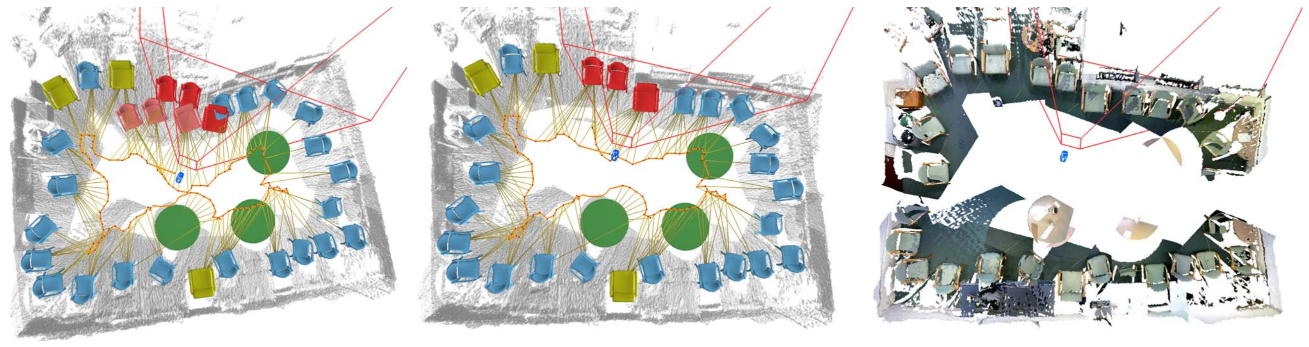


Figure 7. Loop closure. **(left)** Open loop drift during exploration of a room; the two sets of objects shown in red were identified as corresponding. The full SLAM++ graph is displayed with yellow lines for camera-object constraints and orange lines for camera-camera constraints. **(middle)** Imposing the new correspondences and re-optimising the graph closes the loop and yields a more metric map. **(right)** For visualisation purposes only (since raw scans are not normally saved in SLAM++), we show a coloured point cloud after loop closure.

Acknowledgements

Renato Salas-Moreno's research was supported by a PhD scholarship from AMD. We are grateful to Lee Howes for useful discussions. We also acknowledge support from ERC Starting Grant 210346.

References

[1] Advanced Micro Devices Inc. Bolt C++ Template Library. 2012. 3

[2] D. H. Ballard. Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognition*, 12(2):111–122, 1981. 3

[3] S. Y. Bao, M. Bagra, Y.-W. Chao, and S. Savarese. Semantic Structure From Motion with Points, Regions, and Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 2

[4] R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray. Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007. 2

[5] A. J. Davison. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2003. 2

[6] B. Drost, M. Ulrich, N. Navab, and S. Ilic. Model globally, match locally: Efficient and robust 3D object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 3, 4

[7] Y. M. Kim, N. J. Mitra, D.-M. Yan, and L. Guibas. Acquiring 3D Indoor Environments with Variability and Repetition. In *SIGGRAPH Asia*, 2012. 2

[8] G. Klein and D. W. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007. 2

[9] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g^2o: A General Framework for Graph Optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011. 6

[10] W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proceedings of SIGGRAPH*, 1987. 3

[11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. 2, 3, 4, 5, 7

[12] R. A. Newcombe, S. Lovegrove, and A. J. Davison. DTAM: Dense Tracking and Mapping in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011. 2

[13] F. T. Ramos, J. I. Nieto, and H. F. Durrant-Whyte. Combining object recognition and SLAM for extended map representations. In *Experimental Robotics*, volume 39 of *Springer Tracts in Advanced Robotics*, pages 55–64. Springer, 2008. 2

[14] X. Ren, L. Bo, and D. Fox. Indoor Scene Labeling using RGB-D Data. In *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*, 2012. 2

[15] S. Rusinkiewicz and M. Levoy. Efficient Variants of the ICP Algorithm. In *Proceedings of the IEEE International Workshop on 3D Digital Imaging and Modeling (3DIM)*, 2001. 4

[16] N. Silberman and R. Fergus. Indoor Scene Segmentation using a Structured Light Sensor. In *Workshop on 3D Representation and Recognition, in conjunction with International Conference on Computer Vision*, 2011. 2

[17] H. Strasdat. *Local Accuracy and Global Consistency for Efficient Visual SLAM*. PhD thesis, Imperial College London, 2012. 6

[18] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. J. Leonard. Kintinuous: Spatially Extended KinectFusion. In *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*, 2012. 2