Unified Treatment of Sparse and Dense Data in Graph-based Least Squares

René Wagner Udo Frese Berthold Bäuml

***Abstract*— In this paper, we present a novel method of incorporating dense (e.g., depth, RGB-D) data in a general purpose least-squares graph optimization framework. Rather than employing a loosely coupled, layered design where dense data is first used to estimate a compact SE(3) transform which then forms a link in the optimization graph as in previous approaches [28, 10, 26], we use a tightly coupled approach that jointly optimizes over each individual (i.e. per-pixel) dense measurement (on the GPU) and all other traditional sparse measurements (on the CPU). Concretely, we use Kinect depth data and KinectFusion-style point-to-plane ICP measurements. In particular, this allows our approach to handle cases where neither dense, nor sparse measurements separately define all degrees of freedom (DoF) while taken together they complement each other and yield the overall maximum likelihood solution.**

Nowadays it is common practice to flexibly model various sensors, measurements and to be estimated variables in least-squares frameworks. Our intention is to extend this flexibility to applications with dense data. Computationally, the key is to combine the many dense measurements on the GPU efficiently and communicate only the results to the sparse framework on the CPU in a way that is mathematically equivalent to the full least-squares system. This results in $<20\,\mathrm{ms}$ for a full optimization run.

We evaluate our approach on a humanoid robot, where in a first experiment we fuse Kinect data and odometry in a laboratory setting, and in a second experiment we fuse with an unusual "sensor": using the embodiedness of the robot we estimate elasticities in the kinematic chain modeled as unknown, time-varying joint offsets while it moves its arms in front of a tabletop manipulation workspace. In both experiments only tightly coupled optimization will localize the robot correctly.

I. Introduction

Recently, dense data, such as the depth images used in KinectFusion [21], has gained attention relative to sparse features described by a small number of parameters, such as image corners, extracted lines or planes or relative poses. However, while the latter have widely been used in complex estimation scenarios, ranging from SLAM with aerial vehicles [7] to full-body humanoid robot calibration [4], most investigations on dense data assume a free floating sensor [21, 27, 10]. When working with humanoid robots, however, even in straightforward mobile manipulation scenarios such as ours where Agile Justin transports parts from a storage site to a tabletop workspace and assembles them to larger structures there are very common situations where localization proves tricky unless one takes advantage of the fact that the sensor is mounted on a robot equipped with complementary sensors from odometry or inertial sensing to specific kinematic chains. It is promising to model this situation, the involved sensors and their relationships and fuse the provided information with the dense data from the main sensor. Fig. 1 shows two typical situations where this may be worthwhile: When moving back and forth between different sites in the laboratory the robot's depth sensor often observes just a wall leaving 3 out of 6 degrees of freedom (DoF) unconstrained. When fused with odometry,

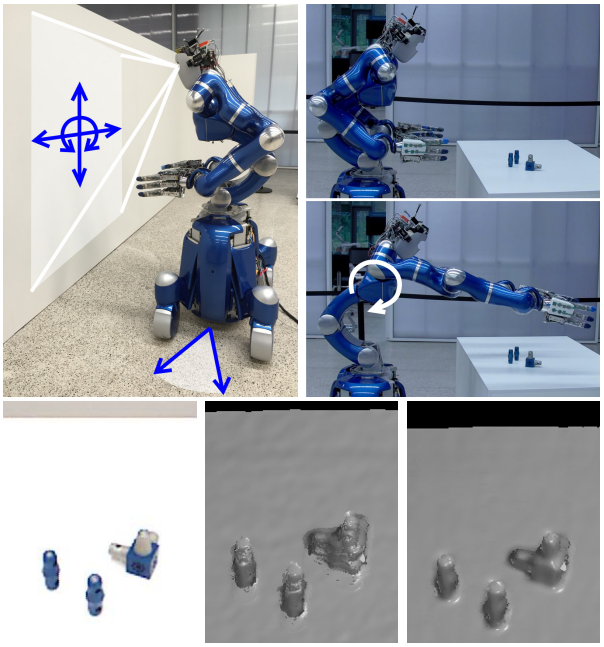

Fig. 1. DLR's humanoid Agile Justin [2] in a mobile manipulation setting. The head mounted Kinect sensor frequently encounters planar structures that only constrain some degrees of freedom (DoF) of the robot pose causing localization via least squares optimization on Kinect data alone to be inaccurate (local minimum due to noise) or to fail completely (nearly singular information matrix). *Top Left:* With odometry alone, even small orientation errors quickly accumulate to large position errors. However, both sensors deliver complementary information: Planar structures in the depth images constrain the sensor orientation while odometry measures the distance traveled fairly accurately. With our unified approach to handling dense and sparse sensor data in LS optimization one sensor can augment undefined or underdefined DoFs in the data of the other sensor resulting in the overall maximum likelihood solution. *Top Right:* In a second experiment, Justin stretches its arms in front of a tabletop manipulation workspace. Elasticities in the torso joints cause the kinematics model to become inaccurate. In our new probabilistic kinematics model these joint offsets are estimated as additional LS variables. *Bottom Left to Right:* RGB image (for reference only), generated 3D model without offset estimation, and with offset estimation for the *stretch-objects* dataset. The structure of the kinematic chain combined with the Kinect sensing the tabletop plane allows our approach to estimate the offsets and prevent objects from washing out in the 3D model. Please also see the video attachment.

R. Wagner and B. Bäuml are with DLR Institute of Robotics and Mechatronics, 82234 Wessling, Germany. U. Frese is with Faculty 3 – Mathematics and Computer Science, University of Bremen, 28359 Bremen, Germany. R. Wagner is also with University of Bremen. Contact {rene.wagner,berthold.baeuml}@dlr.de, ufrese@informatik.uni-bremen.de.

all DoF are constrained. Even more, the result is much better than odometry alone, because orientation and lateral distance comes from the depth sensor and the remaining longitudinal distance is measured by odometry rather precisely. Similarly, when manipulating in front of a tabletop workspace elasticities in the upper body (which are not measured by any sensor) cause a deviation of the kinematic model and the actual robot motion. The Kinect, however only sees the tabletop workspace the geometry of which varies significantly during manipulation and cannot be relied upon to fully constrain the sensor pose. In particular, KinectFusion alone fails when the workspace is empty leaving only a single plane. Similar situations can arise with different sensors or involving other quantities, e.g. calibration parameters, to be estimated alongside. How could such a desirable, flexible fusion be realized in a systematic way? For sparse features the established approach is to model measurement equations for all sensors including the features in a least-squares system and compute the system's maximum-likelihood solution [17, 13, 1]. For dense data the most common approach is to extract sparse features and feed these into the least-squares system, e.g. apply ICP [3] to match depth images and use the resulting relative pose. However, this would not work in a situation with unconstrained DoF as in Fig. 1. It is also theoretically unsatisfying, because ICP by itself performs least-squares estimation, so why not set up one huge unified least-squares system with one measurement for every depth pixel instead of composing several least-squares systems on a software component level? Our paper investigates this approach of building a unified least-squares system to fuse measurements for every dense data item with measurements from complementary sensors. It contributes

- an analysis of the unified least-squares system that reveals a structure which combines the dense measurements into compact matrix blocks but retains the full system of equations and thus allows us to jointly optimize over data from all sensors at once such that complementary information can supplement each other;
- an algorithm that exploits this structure by processing the dense data on a GPU, thereby solving the unified least-squares system efficiently;
- a case study fusing Kinect depth data with odometry on a humanoid robot (Fig. 1, top left) in a difficult situation with unconstrained DoF;
- a second case study fusing Kinect depth data with a probabilistic kinematics model of a humanoid robot's upper body (Fig. 1, top right, bottom) also with unconstrained DoF in a tabletop manipulation setting.

The remainder of this paper is structured as follows. We first discuss related work (II), then introduce our approach in general terms (III), present our concrete implementation for point-to-plane ICP on Kinect data (IV), show how this can be combined with odometry and a probabilistic kinematics model for SLAM on Justin (V), discuss experimental results (VI), and close with conclusions and future work (VII).

II. Related Work

Least squares (LS) optimization in robotics has been studied extensively within the context of simultaneous localization and mapping (SLAM) culminating in several frameworks for graph-based LS [14, 17, 13, 1]. Applications have traditionally relied on what we call sparse measurement data, i.e. that in itself is compact such as a 2D image feature or can be approximated as a compact representation such as a 3D LIDAR as a $SE(3)$ pose relation [17]. The Kinect sensor spurred interest in dense depth data, or combined with RGB images refered to as RGB-D data. Initial SLAM approaches on Kinect data [9, 5] relied on sparse RGB-D keypoint features and RANSAC as the front-end and pose relation optimization by the graph-based backend. KinectFusion [21] was first to perform fully dense GPU-based real-time environment modeling and a point-to-plane variant of ICP [3] with a special-purpose Gauss-Newton LS optimizer. Fusion of dense Kinect depth data with other sensor modalities was first done [27] by running the same custom optimizer on the weighted sum of the two normal systems of the geometric [21] and photometric [24] (RGB) errors. Interest then shifted to generic graph-based LS backends for global consistency via loop closure constraints [28, 10]. These approaches are layered in that the ICP result is passed to the backend as an $SE(3)$ pose relation. As discussed above this has the great disadvantage that sensor modalities only present on the graph SLAM side cannot stabilize ICP convergence and vice versa. The focus of this line of research has been to treat the Kinect as a self-contained free-floating sensor so that this was not noticed as a fundamental theoretical design problem. Also, fusion of dense Kinect data is usually limited to the built-in depth and RGB sensors with structurally very similar models, leaving only the problem of weighting which is typically solved by a tuning factor [27, 29]. With RGB-centric approaches one can also use keyframes to represent the map [20, 6] and apply semi-dense methods that evaluate near large image gradients only [6]. An alternative to the more common LS backends is to marginalize sensor poses away and store/optimize the equivalent information in a map deformation graph [29]. As for fusion with other sensors, it is very common to combine IMU data with sparse image keypoint features [12, 18, 7] but there does not appear to be equivalent work for dense RGB-D data. It was, however, very recently proposed [16] to fuse joint angle and Kinect depth data to estimate offsets and elasticities of a robotic arm along with a 3D map. They view this as new SLAM problem class (ARM-SLAM). Our approach (developed independently) is more general in that we view it as a general LS problem with a robot-specific model. Our key contribution is the generic integration of dense data in a general LS framework (SLoM): everything but the robot model (V) can be re-used directly and very flexibly applied to other situations/robots. Thus, their approach can be seen as a special-purpose, robot arm-specific implementation of our more general approach, i.e. they lack the split of generic vs. robot specific parts and use a custom gradient descent-based solver. They also use a

TSDF map rather than the higher-quality surfel map and a single global weighting factor rather than a scene-dependent per-pixel noise model. Conceptually closest to our unified LS approach is the work by Ruhnke et al. who also estimate the full LS problem over all 2D LIDAR [22] or 3D Kinect endpoints (surfels) [23]. In contrast to our work they do not analyze or transform (Fig. 2) the structure of the problem and do not use GPU acceleration but consider each surfel as a separate LS variable on the CPU. Presently, we believe the latter to be hard to do in real time such that our approach attempts a balance between handling the complete estimation problem and computational feasibility by grouping surfels into sub maps attached to reference poses [26].

III. Unified Least Squares on Dense Data

Least squares (LS) optimization aims to find the $\hat{X}$ minimizing the squared norm of the residual $F(X) = r$:

$$\hat{X} = \underset{X}{\operatorname{argmin}} \tfrac{1}{2} \left\| F(X) \right\|^2, \tag{1}$$

where in the following we will assume that X is a stacked vector of all variables X_k to be estimated and $F(X)$ consists of all stacked measurement functions $f_i(X) = r_i$ normalized to have unit-covariance:

$$F(X) := \begin{bmatrix} L_1^{-1} f_1(X) \\ \vdots \\ L_m^{-1} f_m(X) \end{bmatrix}, f_i(X) \sim \mathcal{N}\big(0,\, \Sigma_i = L_i L_i^\mathsf{T}\big). \tag{2}$$

A. Why Optimize Jointly?

There are many ways to instantiate (2) for robotics applications. A very large and successful body of research [17] uses pose relations for each of the f_i to perform SLAM. The established approach to incorporating dense data such as Kinect RGB-D data first runs KinectFusion's [21] ICP code which performs a fixed number of Gauss-Newton iterations on the per-pixel measurement equations of a single depth image alone to estimate the Kinect sensor pose. The resulting pose relation estimate is then passed to a generic graph-based LS optimization backend either as a frame-to-frame link between consecutive sensor poses [28] or as a frame-to-map link between the sensor pose and the map [10, 26].

The problem with this layered approach is that the KinectFusion Gauss-Newton solver is not guaranteed to converge. This happens particularly when the data fails to constrain some DoF of the sensor pose (a very common situation in indoor environments dominated by large planar surfaces). Heuristics [21] to detect such cases are not perfectly reliable. Thus, in the worst case a problematic estimate goes unnoticed and is passed as a valid pose relation link, in the best case unconstrained DoF lead to a pose relation being discarded altogether. Passing the information matrix last used by the KinectFusion Gauss-Newton solver to the generic backend in addition to the mean does not fully solve this as there is always some spurious information along unconstrained DoF which makes the least squares result ill-defined and may then spoil data association within the ICP. By contrast, when jointly optimizing, other sensors can complement the unconstrained DoF leading to a well-defined least squares problem including the ICP problem which taken by itself would be ill-defined. Our approach discussed below will make this possible.

B. Exploiting Structure Beyond Sparsity: Unified LS

At the core of any quasi-Newton method for non-linear least squares optimization essentially the following normal system needs to be solved at each iteration:

$$J^\mathsf{T} J \delta = -J^\mathsf{T} r, \tag{3}$$

where $J = \mathrm{d}_X F(X) := \frac{\mathrm{d}}{\mathrm{d}X} F(X) \in \mathbb{R}^{M \times N}$ is the measurement Jacobian, $r = F(X) \in \mathbb{R}^M$ the measurement residual, $\delta \in \mathbb{R}^N$ a small incremental change that is applied to improve the estimate $\hat{X}$, and M and N the sum of all measurement and variable degrees of freedom respectively.

$J^\mathsf{T} J$ is naturally sparse since not every measurement depends on all variable DoFs to be estimated. State-of-the-art graph-based least squares frameworks [17, 13] encode these dependencies in a hypergraph from which the sparsity pattern is computed such that only relevant dense blocks of $J^\mathsf{T} J$ need to be considered when solving (3).

Unfortunately, the software interfaces require each measurement function to compute the residual r_i from which J_i is typically computed numerically (although analytical derivatives can be provided). When dense measurements treat, e.g., every pixel of a 640×480 image as a separate measurement these become very large, i.e. for a single image $r_i \in \mathbb{R}^{640 \cdot 480}$, and, assuming a dependency on a single 6DoF pose only, $J_i \in \mathbb{R}^{640 \cdot 480 \times 6}$. Thus, even if we parallelize the evaluation of the measurement function on the GPU, the overhead of copying results over to the CPU alone would be prohibitive. Instead, our initial motivation for this work was that there is no need for access to r and J individually when solving (3) and that for the example above $(J_i^\mathsf{T} J_i) \in \mathbb{R}^{6 \times 6}$ and $(J_i^\mathsf{T} r_i) \in \mathbb{R}^{6 \times 1}$ are very compact.

For the following, more rigorous, analysis let us return to equations (1) and (2) and consider the case where every measurement function $f_{k,i}(X)$ computes a single residual, e.g. for every pixel i in every image k. Computation of the information matrix

$$\Omega = \sum_k \sum_i J_{k,i}^\mathsf{T} J_{k,i} \tag{4}$$

using a naive approach requires double summation over all i and k. However, as illustrated in the diagram in Fig. 2, each $f_{k,i}$ can be seen as a composition of three functions

$$f_{k,i} = h_{k,i} \circ g_k \circ \mathrm{select}_k \,. \tag{5}$$

The selection step encodes the sparsity pattern, i.e. it selects only those individual variables from the stacked variable vector X that g and h depend on. This is the part commonly handled by graph-based LS frameworks as discussed above and usually returns a set of sensor poses, calibration parameters, etc. Further, g, which we call the *composition function*, then translates these into the most compact form possible, typically into a single relative sensor pose T, which is then passed to h, the core sensor model. We now note that neither select, nor g depend on the pixel index i. The chain

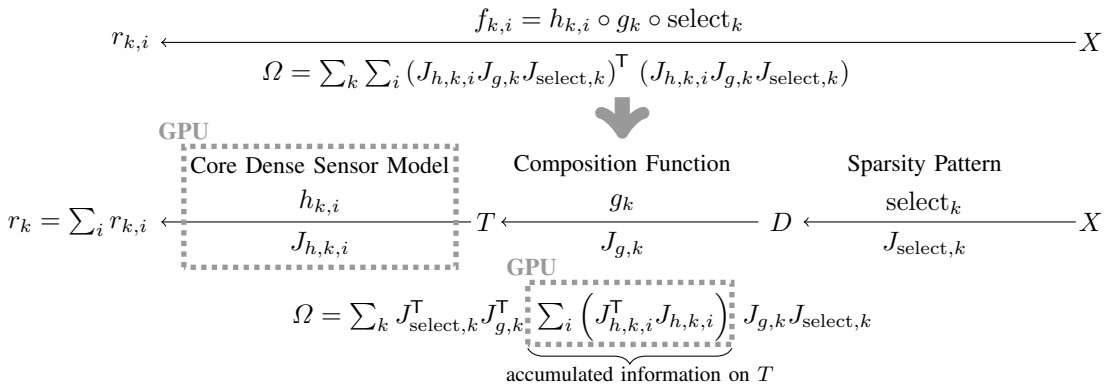

Fig. 2. The unified least squares (LS) problem on dense data treats each per-image k and per-pixel i measurement individually in one huge LS problem together with all other sparse measurements. Structural analysis (III-B) reveals that by splitting each measurement function into a composition $f_{k,i} = h_{k,i} \circ g_k \circ \text{select}_k$, where select handles the sparsity, i.e. picks only the set of dependent variables D (many poses, calibration parameters, etc.) from all variables X, g translates this into a minimal form T (in IV just one relative pose) and h is the core sensor model, we can compute the solution of the full, huge LS problem but still efficiently evaluate the dense data in a small parallel inner loop on the GPU. Note that, here, we show the information matrix only but the same applies to the gradient.

rule allows us to split the evaluation of the measurement Jacobian and re-arrange the summation in the information matrix computation as illustrated in Fig. 2. Thus, all pixel-dependent information can be accumulated in a parallel inner loop on the GPU. The role of the *composition function* g is two-fold: It ensures that the inner loop only needs to compute the minimal-sized matrix block, but it also ensures that the highly-optimized GPU-code is entirely application independent. All application specifics are handled by g and select. Our implementation is an extension of the SLoM framework [13], where select determines the relevant blocks of SLoM's large information matrix and gradient which are then updated with the results of $h \circ g$ in every iteration of SLoM's Gauss-Newton or Levenberg-Marquardt optimizer.

IV. Point-to-Plane ICP on Kinect Data

We will now derive the core sensor model h for point-to-plane ICP on Kinect data. The basic idea, popularized by KinectFusion [21], is to match a Kinect depth image to a 3D surface model by generating a synthetic depth and normal image via projection into the camera image plane and minimizing, for each image k and pixel i, the point-to-plane distance between each point and the surface [21]

$$h_{k,i}(T_k) = (T_k v_{k,i} - \hat{v}_{k,i})^\mathsf{T} \hat{n}_{k,i}, \tag{6}$$

where $\hat{v}_{k,i}, \hat{n}_{k,i} \in \mathbb{R}^3$ are the surface point and normal respectively, $v_{k,i} \in \mathbb{R}^3$ the point from the current depth image associated to $\hat{v}_{k,i}$ via projective data association [21], $T_k \in SE(3)$ the transform from camera to surface coordinates, and we use Tv to denote transformation of a point:

$$Tv := T_r v + T_t, \tag{7}$$

where $T_r \in SO(3)$ and $T_t \in \mathbb{R}^3$ are the rotational and translational parts of T. $h_{k,i}$ is assumed to be subject to distance-dependent noise using the same model as [26]. SLoM lifts the notion of quasi-Newton numerical optimization from the familiar $\mathbb{R}^n$ into the tangent space of non-Euclidean manifolds $(SO(2), SO(3), \mathcal{S}^2)$ via its $\boxplus$-operator [13] (and the inverse $\boxminus$). The same formalism was later adopted by g2o [17] and we want to keep it here, too, which requires analytical $\boxplus$-derivatives. The $\boxplus$-operator works by perturbing manifold variables with small, minimally parameterized changes $\delta \in \mathbb{R}^3$ which gives rise to the definition of derivatives on $\boxplus$-manifolds based on the usual derivative on $\mathbb{R}^n$, i.e. within the perturbation space spanned by $\boxplus$-adding a small δ to the manifold variable [11]:

$$\mathrm{d}_x f(x) := \mathrm{d}_\delta (f(x \boxplus \delta) \boxminus f(x))|_{\delta=0} \tag{8}$$

For Lie groups $\boxplus$ is implemented by means of the familiar Lie group exponential map. Note, however, that this formalism is more general since any Lie group is also a manifold but not vice versa (e.g. $\mathcal{S}^2$). In the following we will work with derivatives of functions of $SE(3)$ transforms. Intuitively, one can think of these in the usual sense, i.e., for every input and output DoF (component of the Jacobian), answering the question of how much the output changes given small changes in the input.

Also, the chain rule applies as usual, such that we can ground most derivatives in the following elementary derivatives on $SO(3)$ [11]:

$$\mathrm{d}_R R^\mathsf{T} = -R^\mathsf{T} \tag{9a}$$
$$\mathrm{d}_R Rv = -[Rv]_\times \tag{9b}$$
$$\mathrm{d}_{R_1}(R_1 R_2) = I_3 \tag{9c}$$
$$\mathrm{d}_{R_2}(R_1 R_2) = R_1, \tag{9d}$$

where $R, R_1, R_2 \in SO(3)$, $v \in \mathbb{R}^3$, and $[\cdot]_\times$ is the matrix having the same effect as the cross product.

We can now determine the derivative of (6) as follows. First, it is easy to see that

$$\mathrm{d}_a (a^\mathsf{T} b) = \mathrm{d}_a \left(\sum_i a_i b_i \right) = b^\mathsf{T} \tag{10}$$

where $a, b \in \mathbb{R}^n$, and we also lift (9b) to $SE(3)$ such that

$$\mathrm{d}_T Tv = \mathrm{d}_{T_r, T_t}(T_r v + T_t) = \left[-[T_r v]_\times \;\; I_3 \right], \tag{11}$$

where $T \in SE(3)$ and $v \in \mathbb{R}^3$. Application of the chain rule then yields the desired result:

$$\begin{aligned} J_{h,k,i}(T_k) &= \mathrm{d}_{T_k}(T_k v_{k,i} - \hat{v}_{k,i})^\mathsf{T} \hat{n}_{k,i} \\ &= \hat{n}_{k,i}^\mathsf{T} \mathrm{d}_{T_k}(T_k v_{k,i} - \hat{v}_{k,i}) \end{aligned}$$

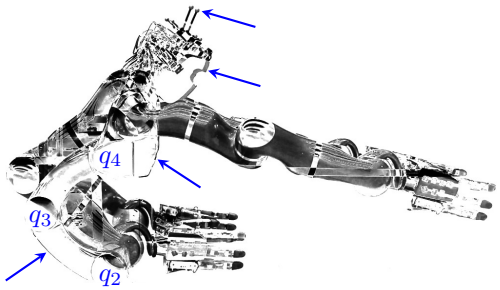

Fig. 3. The difference image of the start and end configurations of Justin stretching its arms (cf. Fig. 1, top right) shows that the added torque causes a small joint offset in q_2, virtually no offset in q_3 and a significant offset in q_4. Note the duplicate Vicon marker and other spots marked with arrows.

$$\begin{aligned} &= \hat{n}_{k,i}^{\mathsf{T}} \mathrm{d}_{T_k}(T_k v_{k,i}) \\ &= \hat{n}_{k,i}^{\mathsf{T}} \begin{bmatrix} -[T_{r,k} v_{k,i}]_\times & I_3 \end{bmatrix} \\ &= \begin{bmatrix} (T_{r,k} v_{k,i} \times \hat{n}_{k,i})^{\mathsf{T}} & \hat{n}_{k,i}^{\mathsf{T}} \end{bmatrix}. \end{aligned} \tag{12}$$

Thus, for each pixel i, (6) yields a 1×1 matrix as the residual $r_{k,i}$ and (12) a 1×6 vector as the Jacobian $J_{h,k,i}$. We compute these with one CUDA call per image k in parallel over all i on the GPU. We do not, however, copy them to the CPU directly. Instead, as part of the same parallel CUDA kernel, we compute the squared residual $(r^{\mathsf{T}} r)_{k,i}$, the gradient $(J^{\mathsf{T}} r)_{h,k,i}$, and (exploiting symmetry) only the lower triangular part of $(J^{\mathsf{T}} J)_{h,k,i}$ and then perform a two-stage parallel reduce operation [19] to compute the sum of each of these components over all pixels in an image. For each Kinect image, only these resulting 28 `float` values are copied over to the CPU, typically after 125 µs per frame.

V. Modelling Agile Justin

We apply our method for SLAM on the humanoid robot Agile Justin with a dense 3D map and fusion of Kinect depth images with odometry and a probabilistic kinematics model.

A. Probabilistic Forward Kinematics Model

Justin's torso includes a mechanism to drive the chest up and down using three joints q_2, q_3 (both actuated) and q_4 coupled to q_2 and q_3 through cables (Fig. 3). Allowed commanded motion is strictly up and down but joint elasticities cause rotational and forward motion particularly of the head so we would like to estimate these based on a probabilistic model. Adding one joint offset parameter per joint would create redundant DoFs and likely cause instabilities in the optimizer. It also neglects the physical behavior of the robot (Fig. 3). There is virtually no offset in q_3 and the signs of the offsets in q_2 and q_4 due to load are always the same. Also, we have found that the ratio of the offsets in q_2 and q_4 remains similar across different runs. Thus, we use the following model: For each time step k there is one offset parameter $\hat{\psi} \in SO(2)$ and one ratio parameter $\hat{\phi}_k \in \mathbb{R}$ both subject to colored noise, i.e. priors say that $\hat{\psi}_k$ is approximately zero (with $\sigma_{\psi,\text{abs.}} = 10°$) and $\hat{\phi}_k$ approximately 0.2 (with $\sigma_{\phi,\text{abs.}} = 0.05$) and that neither change over time (with $\sigma_{\psi,\text{rel.}} = 0.1°$ and $\sigma_{\phi,\text{rel.}} = 0.005$ respectively). The estimated joint angles are then

$$\hat{q}_2 = q_2 + \hat{\phi} \cdot \hat{\psi}, \tag{13a}$$

$$\hat{q}_4 = q_4 + (1 - \hat{\phi}) \cdot \hat{\psi}. \tag{13b}$$

B. Composition Function

The formulation of the point-to-plane distance measurement function in (6) is intentionally idealized. In an actual application, it depends on more than just a single transform T as a variable. On Agile Justin it depends on the robot pose $\hat{R}_k$, the map pose $\hat{S}$, the Kinect pose $\tilde{K}_k := K(\hat{\phi}_k, \hat{\psi}_k)$ and the pose C_k from which the synthetic depth/normal image was generated (relative to the map):

$$f_{k,i} = (\hat{R}_k \tilde{K}_k v_{k,i} - \hat{S} C_k \hat{v}_{k,i})^{\mathsf{T}} ((\hat{S} C_k)_r \hat{n}_{k,i}), \tag{14}$$

where $(\cdot)_r$ denotes the rotational part of a transform, $\hat{R}_k$, $\hat{S}$, $\hat{\phi}_k$ and $\hat{\psi}_k$ are SLoM variables (picked by select), and C_k is treated as a constant which could be an arbitrary transform but is usually chosen to be the previous best estimate of the Kinect pose. $K(\hat{\phi}_k, \hat{\psi}_k)$ is computed from the kinematics model, the extrinsic Kinect calibration parameters, and the joint offset parameters $\hat{\phi}_k$ and $\hat{\psi}_k$. It is the job of the *composition function* g to translate (14) into the form of (6). For the function itself this is straightforward; re-arranging the transforms yields:

$$\begin{aligned} f_{k,i} &= (C_k^{-1} \hat{S}^{-1} \hat{R}_k \tilde{K}_k v_{k,i} - \hat{v}_{k,i})^{\mathsf{T}} \hat{n}_{k,i} \\ &= h_{k,i}(\underbrace{C_k^{-1} \hat{S}^{-1} \hat{R}_k \tilde{K}_k}_{:= T = g_k(\text{select}_k(X))}). \end{aligned} \tag{15}$$

Similarly, the chain rule allows us to split the application specific derivative from the generic one in (12). We need derivatives w.r.t. all SLoM variables, i.e.

$$\begin{aligned} \mathrm{d}_{\hat{R}_k} f_{k,i} &= \mathrm{d}_{\hat{R}_k} (C_k^{-1} \hat{S}^{-1} \hat{R}_k \tilde{K}_k v_{k,i} - \hat{v}_{k,i})^{\mathsf{T}} \hat{n}_{k,i} \\ &= \mathrm{d}_{T_k} (T_k v_{k,i} - \hat{v}_{k,i})^{\mathsf{T}} \hat{n}_{k,i} \cdot \mathrm{d}_{\hat{R}_k} C_k^{-1} \hat{S}^{-1} \hat{R}_k \tilde{K}_k, \end{aligned} \tag{16}$$

$$\begin{aligned} \mathrm{d}_{\hat{\psi}} f_{k,i} &= \mathrm{d}_{\hat{\psi}_k} (C_k^{-1} \hat{S}^{-1} \hat{R}_k \tilde{K}_k v_{k,i} - \hat{v}_{k,i})^{\mathsf{T}} \hat{n}_{k,i} \\ &= \underbrace{\mathrm{d}_{T_k} (T_k v_{k,i} - \hat{v}_{k,i})^{\mathsf{T}} \hat{n}_{k,i}}_{\text{generic}} \cdot \underbrace{\mathrm{d}_{\hat{\psi}_k} C_k^{-1} \hat{S}^{-1} \hat{R}_k \tilde{K}_k}_{\text{application specific}} \end{aligned} \tag{17}$$

and analogously for $\hat{S}$ and $\hat{\phi}_k$. Further application of the chain rule yields the derivative w.r.t. $\hat{R}_k$ (and analogously w.r.t. $\hat{S}$, $\hat{\phi}_k$ and $\hat{\psi}_k$):

$$\begin{aligned} \mathrm{d}_{\hat{R}_k} T_k &= \mathrm{d}_{\hat{R}_k} C_k^{-1} \underbrace{\left((\hat{S}^{-1} \hat{R}_k) \tilde{K}_k \right)}_{:= B} \\ &= \mathrm{d}_B C_k^{-1} B \cdot \mathrm{d}_{\hat{R}_k} \underbrace{(\hat{S}^{-1} \hat{R}_k)}_{:= A} \tilde{K}_k \\ &= \mathrm{d}_B C_k^{-1} B \cdot \mathrm{d}_A A \tilde{K}_k \cdot \mathrm{d}_{\hat{R}_k} \hat{S}^{-1} \hat{R}_k, \end{aligned} \tag{18}$$

where the remaining elementary derivatives are given in Appendix A, and $\mathrm{d}_{\hat{\phi}_k} \tilde{K}_k$ and $\mathrm{d}_{\hat{\phi}_k} \tilde{K}_k$ are computed through numerical differentiation of the kinematics model. We can now apply $\mathrm{d}_{\hat{R}_k} T_k$, $\mathrm{d}_{\hat{S}_k} T_k$, $\mathrm{d}_{\hat{\phi}_k} T_k$ and $\mathrm{d}_{\hat{\psi}_k} T_k$ to turn the very compact form of the core sensor model measurement information from IV into its larger application-specific form (14×14 for $(J^{\mathsf{T}} J)_k$ and 1×14 for $(J^{\mathsf{T}} r)_k$) which is then added on top of the corresponding blocks of SLoM's large normal system (3) at every solver iteration.

C. Odometry, Additional Priors and Graph Optimization

Odometry is modeled as pose relations combined with prior information that the z-position is roughly but not exactly known as in our previous work [26, §VI-A,B]. In contrast to [26], we now use a sliding window, i.e. the SLoM graph consists of robot poses for the last 10 Kinect and odometry measurements; earlier poses are fixed (no longer optimized) when they leave the sliding window. Optimization is performed on odometry, prior and dense Kinect point-to-plane ICP measurements with the same resolution pyramid approach as [21] for the latter. Every time a new Kinect frame comes in we perform three Levenberg-Marquardt SLoM iterations on the full unified LS system at each level and after this full optimizer run update the surfel map.

D. Surface Map Representation

Previously [26], we used truncated signed distance function (TSDF) [21] to represent the map as an implicit surface. We now use surfel maps [15]: Each surfel is a disc represented as a center, radius, normal and confidence weight, stored in unordered, flat arrays on the GPU. Extraction of depth and normal images [8] and map updates [15] work by projection of the surfel list into the image and the same weighted averaging (equivalently to a simple Kalman filter) as [21], yet surface normal quality is noticably better as normals are encoded explicitely and many Kinect pixels need to agree on the normal for a surfel hypothesis to be kept.

VI. Experiments

To tackle the tricky situations discussed in the introduction, we devised two experimental setups. In the first, we had Agile Justin drive back and forth parallel to a wall in our lab (Fig. 1) while the Kinect is directed at the wall. The commanded trajectory is a straight line of $4\,\mathrm{m}$ length (one way, Fig. 4) and 8 round trips are executed for a total distance traveled of $64\,\mathrm{m}$ per dataset. We repeat this at different distances to the wall to generate the *multi-wall-0.9m*, *multi-wall-1.6m* and *multi-wall-2m* datasets. The wall leaves some DoF unconstrained in the Kinect data, odometry measures motion on the ground plane only and small orientation errors accumulate to position errors that would quickly make e.g. gripping of objects impossible without re-detection, and finally the prior on the z-position constrains only a single DoF. Note that the choice of a straight line as the trajectory is deliberate as motion along an arc combined with the measurement of the orientation relative to the wall would constrain the position. Due to the small scale of the environment we use a single surfel map although the composition function (15) treats the map pose as a variable which would allow for multiple map segments as we did previously with TSDF maps [26]. For the driving datasets, the SLoM optimizer was invoked with a new Kinect depth image every $10\,\mathrm{cm}$ since we need to ensure that LS measurements are stochastically independent, especially because we are filtering over the surfel map. Otherwise, systematic errors in the Kinect depth image would "burn" into the map. The resulting trajectories are plotted in Fig. 6.

Fig. 4. For the experiment fusing Kinect depth data with odometry, Justin drives $4\,\mathrm{m}$ from the start pose (left) to the end pose (right) and back for 8 consecutive round trips per *multi-wall-** dataset.

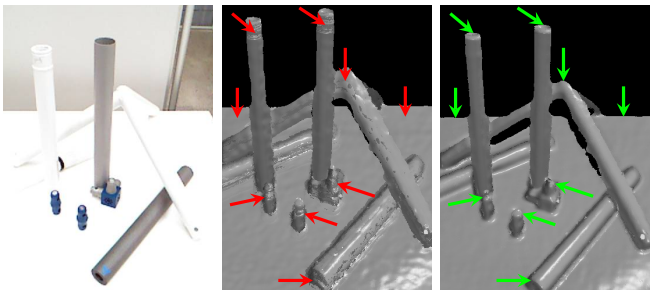

Fig. 5. *stretch-clutter* dataset: RGB (left, for reference), the washed out 3D model generated with original kinematics model (center) and the consistent 3D model generated with our new probabilistic kinematics model fused with Kinect depth through our unified dense/sparse LS approach (right). The perspective of the renderings is identical – arrows pointing to the same pixels highlight inconsistencies in the original model (e.g. too tall tubes).

In the second experiment, Justin is positioned in front of a tabletop workspace. It does not drive around, but this is not known a-priori and only sensed through actual odometry measurements. It then extends its arms forward as if to manipulate objects, which excites the mentioned elasticities in the torso. The arms are partially visible in the Kinect depth images and masked out with a robot model consisting of spheres [25]. Background pixels are also masked out as they are unrealiable. The arm stretching motion is repeated four times increasingly fast and the whole sequence is repeated with three different scenes: with an empty tabletop where ICP alone would fail (*stretch*), with few, small objects that do not yet form reliable geometric constraints (*stretch-objects*) but allow judging model consistency visually (Fig. 1), and with many objects on the table (*stretch-clutter*) to show that additional geometry does not hurt either (Fig. 5). With these datasets, we enable joint offset estimation after the map has stabilized (after 30 frames while the robot is not moving yet) and then run SLoM every time a new Kinect image comes in. The estimated head trajectory and joint offset parameters for the *stretch* dataset is plotted in Fig. 7. The plots for the other datasets show very similar performance so that we have summarized these by means of average estimation errors in Tab. I. Renderings of the generated 3D maps without and with joint offset estimation are shown in Figs. 1 and 5.

Ground truth is provided by a Vicon system (6 ceiling-mounted 16Mpx cameras covering a volume of $6\text{x}6\text{x}2\mathrm{m}^3$) tracking retro-reflective markers attached to Justin's head. The specified precision is $<1\,\mathrm{mm}$ for the position and $<0.5°$ for the orientation under ideal conditions. In practice, marker occlusions can degrade this significantly especially when Justin's head is tilted forward.

All computations run in real time at Kinect frame

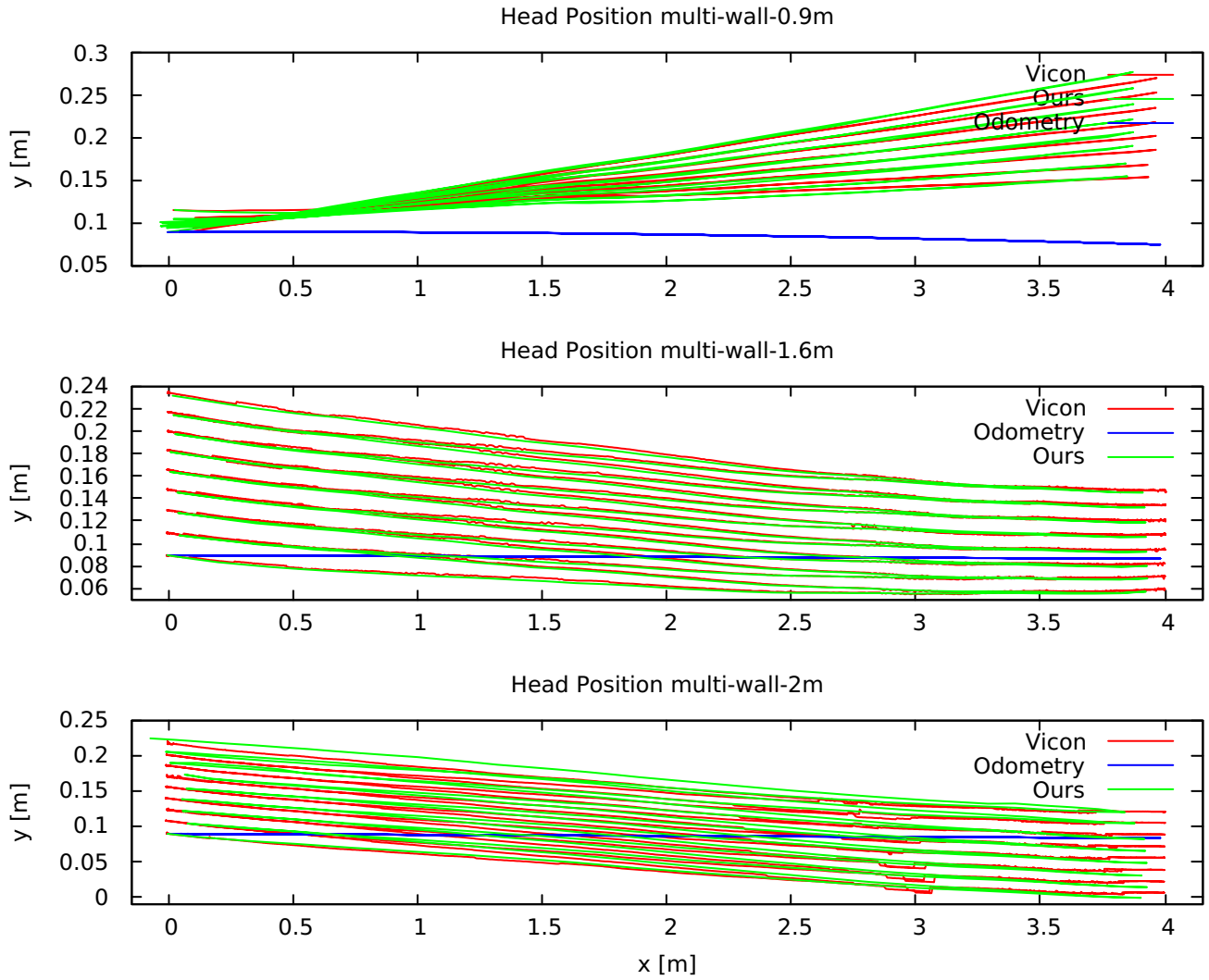

Fig. 6. Head position (relative to the initial odometry pose) for the driving datasets. The wall is located to the top of the trajectory. Odometry does not sense the drift perpendicular to the wall well and basically reports the commanded trajectory. Our unified dense/sparse LS approach recovers the actual motion. Note that our trajectory seems somewhat short on the right because we plot poses with Kinect data only (every 10cm) and w.r.t. the Vicon data that the tabletop datasets contain a small motion only while here we rely on global Vicon precision throughout the tracking volume which is much lower. In some cases there are even easily identifiable Vicon errors (bottom, at x=3m) as the robot cannot jump sideways.

Dataset	Rotation $\cdot 10^{-4}$ [rad]			Position $\cdot 10^{-4}$ [m]		
	x	y	z	x	y	z
stretch	1.23	2.08	4.19	1.26	0.71	0.63
stretch-objects	2.10	3.58	2.41	2.26	4.44	1.36
stretch-clutter	1.71	1.63	2.51	3.18	4.93	1.74

TABLE I

AVERAGE ERRORS OF ESTIMATED HEAD POSE VS. VICON

rate (30Hz). Average computing times using a single CPU core of a Xeon E5-2667v3 and a GTX Titan X GPU were as follows. With the *stretch-*/multi-wall-** datasets, 1.5 ms for pre-processing, 2 ms/4.2 ms to synthetize a depth&normal image from the map, 17.2 ms/18.5 ms for each full optimizer run, 1.6 ms/5.4 ms per map update.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed an approach to handle dense data in least-squares (LS) estimation with the goal of fusing it with complementary sensors. The initial idea is to make every dense data-item an individual per-pixel measurement in a single, huge LS problem, which is theoretically rigorous but sounds computationally challenging. However, careful analysis of the resulting structure shows that all dense measurements of a single image can be combined into a 6×6 information matrix and 1×6 gradient vector on a relative pose which can be done efficiently on a GPU while still, without approximation, estimating the full, huge LS problem.

We have shown in a case study with Agile Justin that this approach facilitates sensor fusion on a humanoid robot because it allows to formulate robot specific models flexibly. This is "embodiment" as it allows the perception system to use the information provided by the humanoid's body.

Future work will include to extend the formalism to a full SLAM system where we do not filter over the map and instead estimate it as a first-class LS variable and to

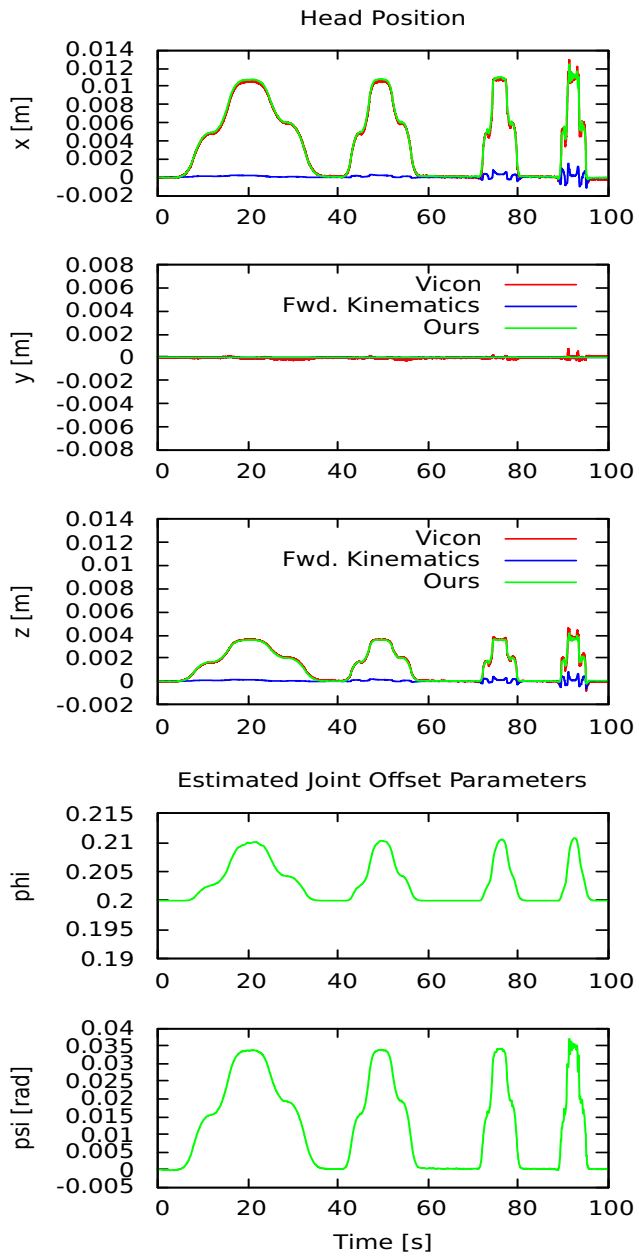

Fig. 7. Head position (relative to the initial head pose) and estimated joint offset parameters for the *stretch* dataset (empty tabletop workspace). Our estimate almost exactly matches Vicon groundtruth.

further refine the robot model, particularly to handle shaking motion of the robot in combination with a head mounted IMU. Imagine the robot standing still in front of a wall (Fig. 1) but with some shaking. The IMU would sense the shaking, odometry would prevent unbounded drift and those DoF observable from the wall would be even more precise. Another extension would be a core dense sensor model for the structurally very similar RGB-D visual odometry [24] analogously to IV.

APPENDIX

A. Elementary $\boxplus$-Derivatives

The derivative (18) of the Agile Justin-specific composition function relies on three new elementary $\boxplus$-derivatives concerning $SE(3)$ transforms A, B which follow from the definition in (8) and the elementary $SO(3)$ derivatives in (9):

$$\mathrm{d}_A AB = \begin{bmatrix} I_3 & 0 \\ -[A_r B_t]_\times & I_3 \end{bmatrix} \tag{19a}$$

$$\mathrm{d}_B AB = \begin{bmatrix} A_r & 0 \\ 0 & A_r \end{bmatrix} \tag{19b}$$

$$\mathrm{d}_A A^{-1} B = \begin{bmatrix} -A_r^\mathsf{T} & 0 \\ \left([A_r^\mathsf{T} B_t]_\times A_r^\mathsf{T} - [A_r^\mathsf{T}]_\times A_r^\mathsf{T}\right) & -A_r^\mathsf{T} \end{bmatrix} \tag{19c}$$

REFERENCES

[1] S. Agarwal, K. Mierle, et al. Ceres solver. http://ceres-solver.org.
[2] B. Bäuml et al. Agile Justin: An upgraded member of DLR's family of lightweight and torque controlled humanoids. In *ICRA*, 2014.
[3] P. J. Besl and N. McKay. A method for registration of 3-D shapes. *TPAMI*, 14(2), 1992.
[4] O. Birbach, U. Frese, and B. Bäuml. Rapid calibration of a multi-sensorial humanoids upper body: An automatic and self-contained approach. *IJRR*, 34(4-5), 2015.
[5] F. Endres, J. Hess, N. Engelhard, et al. An evaluation of the RGB-D SLAM system. In *ICRA*, 2012.
[6] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *ECCV*, 2014.
[7] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *RSS*, 2015.
[8] G. Guennebaud et al. Splat/Mesh Blending, Perspective Rasterization and Transparency for Point-Based Rendering. In *SPBG*, 2006.
[9] P. Henry, M. Krainin, et al. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *ISER*, 2010.
[10] P. Henry, D. Fox, A. Bhowmik, et al. Patch volumes: Segmentation-based consistent mapping with RGB-D cameras. In *3DV*, 2013.
[11] C. Hertzberg. *3D-Sensors – Axiomatization, Modeling, Calibration, Localization and Mapping*. PhD thesis, University of Bremen, 2015.
[12] C. Hertzberg, R. Wagner, et al. Experiences in building a visual SLAM system from open source components. In *ICRA*, 2011.
[13] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1), 2013.
[14] M. Kaess, H. Johannsson, R. Roberts, et al. iSAM2: Incremental smoothing and mapping using the bayes tree. *IJRR*, 31(2), 2012.
[15] M. Keller, D. Lefloch, M. Lambers, et al. Real-time 3D reconstruction in dynamic scenes using point-based fusion. In *3DV*, 2013.
[16] M. Klingensmith, S. Srinivasa, and M. Kaess. Articulated robot motion for simultaneous localization and mapping (ARM-SLAM). *IEEE Robotics and Automation Letters (RA-L)*, 1(2), 2016.
[17] R. Kümmerle, G. Grisetti, et al. g2o: A general framework for graph optimization. In *ICRA*, 2011.
[18] S. Leutenegger, S. Lynen, M. Bosse, et al. Keyframe-based visual-inertial odometry using nonlinear optimization. *IJRR*, 34(3), 2015.
[19] J. Luitjens. Faster parallel reductions on Kepler. http://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler/, 2014.
[20] M. Meilland and A. I. Comport. On unifying key-frame and voxel-based dense visual SLAM at large scales. In *ICRA*, 2013.
[21] R. A. Newcombe, S. Izadi, et al. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*, 2011.
[22] M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Highly accurate maximum likelihood laser mapping by jointly optimizing laser points and robot poses. In *ICRA*, 2011.
[23] M. Ruhnke, R. Kümmerle, G. Grisetti, et al. Highly accurate 3D surface models by sparse surface adjustment. In *ICRA*, 2012.
[24] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense RGB-D images. In *ICCV Workshops*, 2011.
[25] R. Wagner, U. Frese, and B. Bäuml. 3D modeling, distance and gradient computation for motion planning: A direct GPGPU approach. In *ICRA*, 2013.
[26] R. Wagner, U. Frese, and B. Bäuml. Graph SLAM with signed distance function maps on a humanoid robot. In *IROS*, 2014.
[27] T. Whelan, H. Johannsson, et al. Robust real-time visual odometry for dense RGB-D mapping. In *ICRA*, 2013.
[28] T. Whelan, M. Kaess, J. Leonard, et al. Deformation-based loop closure for large scale dense RGB-D SLAM. In *IROS*, 2013.
[29] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, et al. ElasticFusion: Dense SLAM without a pose graph. In *RSS*, 2015.