Rapid Development of Manifold-Based Graph Optimization Systems for Multi-Sensor Calibration and SLAM

René Wagner Oliver Birbach Udo Frese

***Abstract*— Non-linear optimization on constraint graphs has recently been applied very successfully in a variety of SLAM backends. We combine this technique with a principled way of handling non-Euclidean spaces, 3D orientations in particular, based on manifolds to build a generic and very flexible framework, the Manifold Toolkit for Matlab (MTKM). We show that MTKM makes it particularly easy to solve non-trivial multi-sensor calibration problems while remaining generic enough to handle a very different class of problems, namely SLAM, as well: After an introductory example on single camera calibration we apply MTKM to calibration of stereo vision and IMU w.r.t. the kinematic chain of a service robot, RGB-D and accelerometer calibration of a Microsoft Kinect, stereo calibration on a Nao soccer robot, and several SLAM benchmark data sets illustrating MTKM's versatility. MTKM and all presented examples are available as open source from `http://openslam.org/MTK.html`.**

I. Introduction

Considering the simultaneous localization and mapping (SLAM) problem [30] as a non-linear least squares problem represented as a graph where nodes correspond to robot poses or landmarks and edges to non-linear constraints between them was originally proposed by Lu and Milios [24] and later formalized as GraphSLAM by Thrun et al. [31]. Recently, it was shown that, given this problem formulation, offline SLAM can be solved by finding the maximum likelihood graph through non-linear numerical optimization [7, 27, 12]. Although a generalization to graphs of constraints between arbitrary random variables is possible, use of this method has largely been limited to the SLAM community.

In this paper, we aim to make the technique accessible to a wider audience. We have developed a Matlab framework, the Manifold Toolkit for Matlab (MTKM), that allows for the rapid specification of constraint graph problems which can then be solved using non-linear optimization. In MTKM, random variables and constraint measurements are represented as manifolds which conveniently includes vectors ($\mathbb{R}^n$), orientations ($SO(2), SO(3)$) and arbitrary compounds, i.e. Cartesian products, of these. Using the $\boxplus$-method ("boxplus-method"; [14]), a local, minimal vector view of these more complex topological structures is generated so that a standard non-linear optimization algorithm such as Levenberg-Marquardt can be applied with only minor modification while the underlying global parameterization is singularity free.

We intentionally choose a pure Matlab implementation trading computational performance for ease of use and portability. This focus results in short, easy to read user code that maps to the mathematical formulation as closely as possible. We show how this makes the framework particularly well-suited to multi-sensor calibration. Standard calibration tools only exist for widely-used robot-agnostic sensor setups. Today's robotic systems, however, increasingly rely on a combination of a wide variety of sensors in increasingly complex configurations. Perception and manipulation tasks in particular require precise cross-calibration of all sensors. We apply MTKM to several calibration examples, most notably to multi-sensor calibration on a humanoid service robot.

Fig. 1. Diverse problems solved with MTKM (top left to bottom right): Hand–eye, stereo and IMU calibration on a humanoid service robot, vertical stereo calibration on a humanoid soccer robot, 3D pose adjustment on the parking garage data set [21], and RGB-D and accelerometer calibration.

We also show that the same framework is able to process several established SLAM benchmark data sets with acceptable, although not real-time capable, computation times, producing results equivalent to state-of-the-art SLAM algorithms. However, it is not our goal to present yet another SLAM framework. Our contribution is to integrate the $\boxplus$-method with Matlab's type system and to make least squares optimization on manifolds available to a Matlab user base and to show how MTKM makes it particularly easy to solve multi-sensor calibration problems while retaining a broad genericity so that a different class of problems, SLAM, can also be handled.

Further, we find that MTKM brings about a high degree of flexibility, e.g. to modify the sensor setup to be calibrated or to add additional sensory input to SLAM. In calibration tasks, this is particularly important since calibration is usually not the research focus and rather a matter of getting it done

All authors are with Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Bremen, Germany. U. Frese is also with Universität Bremen, Bremen, Germany. Contact {rene.wagner,oliver.birbach,udo.frese}@dfki.de.

quickly. In SLAM, it facilitates cheap, rapid experimentation.

The remainder of this paper is structured as follows. Section II discusses related work. Section III introduces the general mathematical graph optimization framework and its implementation in MTKM. Sections IV, V and VI apply MTKM to an introductory calibration example, to multi-sensor calibration and to SLAM, respectively.

II. Related Work

A. Calibration

Today's robots are equipped with an increasing number of different types of sensors often including imaging sensors, ranging sensors, kinematic sensors and inertial measurement units (IMU). Calibration of individual sensors is well studied for cameras [32, 35, 11], IMUs [18, 15], and range imaging [9, 8]. Cross-calibrating multiple sensors for sensor data fusion is more difficult. Approaches exist for vision/IMU calibration [23, 25], and calibrating sensors on a robot to its manipulators [28]. In this work, we go one step further and jointly calibrate vision sensors, an IMU and their relation to a humanoid's kinematic chain using the proposed MTKM framework which handles rotations in a non-singular way and exploits the sparsity of the problem automatically. This is contrary to other published calibration routines which often use axis-angle representation for 3D rotations and fill relevant Jacobian blocks manually [1, 5]. The contribution, however, is not this specific kind of calibration but a general framework to implement calibration tasks easily. Also, practice showed that the optimization part is often re-implemented for every new problem. We factor this out enabling maximum code re-use.

B. Representing 3D Orientations

When working with 3D orientations a conflict arises: On the one hand, a singularity-free representation (e.g. unit quaternions or orthonormal $\mathbb{R}^{3\times3}$ matrices) is desirable. On the other hand, sensor fusion algorithms typically operate on Euclidean vector spaces $\mathbb{R}^n$. However, no globally singularity-free representation of $SO(3)$ with just three parameters exists.

It was proposed by Ude [33, 34] for least squares optimization and by Kraft [20] for Kalman filtering to use the quaternion exponential map to get a local, minimal (3 parameters) representation and accumulate global results in the original unit quaternion form. Both authors note the relation to more general properties of manifolds but use algorithms specifically modified to operate on the particular quaternion parameterization. In earlier work [14], we proposed two operators $\boxplus$ and $\boxminus$ to decouple the sensor fusion algorithm from the concrete random variable or state representation. In condensed form, this $\boxplus$-method is introduced in III-A.

C. Graph-Based SLAM

Lu and Milios [24] originally proposed the graph formulation of the SLAM problem in 2D. Efficient means of solving it, however, became available much later. Olson et al. [27] propose a method resembling stochastic gradient descent. TORO [10] extends this by using relative poses in a tree for improved performance and by supporting 2D and 3D poses.

Meanwhile, $\sqrt{\text{SAM}}$ [7] further improved performance by exploiting the sparsity of the 2D SLAM problem through incremental, sparse QR decomposition. This was generalized with the iSAM framework [16] processing arbitrary constraints given an implementation of a common C++ interface.

Allowing for the latter while also ensuring a mathematically sound representation of states and constraint measurements was first achieved by SLoM [12] by means of an early version of the $\boxplus$-method [14]. SLoM extracts the sparsity pattern and relies on CSparse [6] for efficient, sparse Cholesky factorization delivering computational performance equivalent to or better than some more problem-specific approaches [13]. MTKM differs from SLoM in that it is implemented in pure, object-oriented Matlab while SLoM makes heavy use of C++ macros and in that the problem specification (constraint graph) is cleanly separated from the optimization algorithm in MTKM, i.e. the user can easily replace the default solver.

More recently, work on graph-based SLAM has focused on improving the computational performance of problem specific SLAM algorithms [19] or general frameworks [21, 17].

III. Graph Optimization with the Manifold Toolkit for Matlab (MTKM)

This section introduces the mathematical graph optimization framework and its implementation in MTKM. We generalize from robot poses and landmarks or camera poses and parameters to arbitrary random variables to be estimated.

A. Manifolds and the $\boxplus$-Method

In earlier work [14], we proposed a principled solution to handling non-Euclidean spaces (e.g. 3D orientations) in sensor fusion algorithms, the $\boxplus$-method, which we discuss here in a condensed, largely informal form to keep the present paper self-contained. For details and proofs see [14].

The key idea behind the $\boxplus$-method is to represent random variables and measurements as manifolds and to exploit the fact that "manifolds are locally homeomorphic to $\mathbb{R}^n$, i.e. [...] we can establish a bijective mapping from a local neighborhood in an n-manifold $\mathcal{S}$ to $\mathbb{R}^n$ [via] two encapsulation operators $\boxplus$ ("boxplus") and $\boxminus$ ("boxminus")"[14]:

$$\boxplus : \mathcal{S} \times \mathbb{R}^n \to \mathcal{S}, \tag{1}$$

$$\boxminus : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^n. \tag{2}$$

These two operators create a local, vectorized view of the globally more complex structure of the manifold. "The operation $y = x \boxplus \delta$ adds a small perturbation vector $\delta \in \mathbb{R}^n$ to $x \in \mathcal{S}$. The inverse operation $\delta = y \boxminus x$ determines the perturbation δ which yields y when $\boxplus$-added to x"[14]. Since the reference point x can be chosen arbitrarily, the entire manifold can be covered even though at each point in time only a local, mapped neighborhood of it is visible via $\boxplus$ and $\boxminus$.

More formally, for a $\boxplus$ that is smooth in its second operand and a $\boxminus$ that is smooth in its first operand, we call

$(\mathcal{S}, \boxplus, \boxminus, V \subset \mathbb{R}^n)$ a $\boxplus$-*manifold* "if the following axioms hold for every $x \in \mathcal{S}$" [14]:

$$x \boxplus 0 = x \tag{3a}$$

$$\forall y \in \mathcal{S}: \quad x \boxplus (y \boxminus x) = y \tag{3b}$$

$$\forall \delta \in V: \quad (x \boxplus \delta) \boxminus x = \delta \tag{3c}$$

$$\forall \delta_1, \delta_2 \in \mathbb{R}^n : \|(x \boxplus \delta_1) \boxminus (x \boxplus \delta_2)\| \leq \|\delta_1 - \delta_2\| . \tag{3d}$$

Thus, 0 is the neutral element w.r.t. $\boxplus$ (3a), $\delta \mapsto x \boxplus \delta$ is surjective (3b), and within the local neighborhood defined by V the parameterization used by $\boxplus$ is unique, i.e. here $\delta \mapsto x \boxplus \delta$ is injective (3c). E.g. for orientations, V is chosen such that $\|\delta\| < \pi$ since otherwise (3c) might be violated due to wraparounds. Finally, (3d) induces a metric [14].

Further, we lift the notion of a Gaussian distribution onto $\boxplus$-manifolds by defining [14]

$$\mathcal{N}(\mu, \Sigma) := \mu \boxplus \mathcal{N}(0, \Sigma), \tag{4}$$

"where $\mu \in S$ is a an element of the $\boxplus$-manifold but $\Sigma \in \mathbb{R}^{n \times n}$ a matrix as for regular Gaussians"[14]. This also means that the parameterization of $\boxplus$/$\boxminus$ defines the interpretation of covariance matrices.

The other important property we exploit is the fact that the Cartesian product of manifolds yields another manifold. For two (and by induction arbitrary numbers of) $\boxplus$-manifolds, we get the *compound* $\boxplus$*-manifold* $S = S_1 \times S_2$ with componentwise $\boxplus$/$\boxminus$-operators [14]:

$$(x_1, x_2) \boxplus \left[\begin{smallmatrix} \delta_1 \\ \delta_2 \end{smallmatrix}\right] := (x_1 \boxplus_{\mathcal{S}_1} \delta_1, x_2 \boxplus_{\mathcal{S}_2} \delta_2) \tag{5}$$

$$(y_1, y_2) \boxminus (x_1, x_2) := \begin{bmatrix} y_1 \boxminus_{\mathcal{S}_1} x_1 \\ y_2 \boxminus_{\mathcal{S}_2} x_2 \end{bmatrix} . \tag{6}$$

We use this to construct compound $\boxplus$-manifolds from a set of $\boxplus$-manifold primitives and to create stacked vectors of random variables as discussed below.

For the purposes of this paper, the most relevant $\boxplus$-manifold primitives are $\mathbb{R}^n$ with trivial operators $\boxplus := +$ and $\boxminus := -$, 2D orientations ($SO(2)$) represented as an angle and $\boxplus$/$\boxminus$-operators which simply respect the wraparound at $\pm\pi$, and 3D orientations ($SO(3)$) either as unit quaternions, or, as will be assumed in the following, as orthonormal matrices $\mathbb{R}^{3\times3}$ with $\boxplus$/$\boxminus$-operators implemented by means of the matrix exponential [14]

$$x \boxplus \delta = x \exp \delta \qquad y \boxminus x = \log\left(x^{-1} y\right), \tag{7}$$

i.e. δ defines the axis and the angle $\|\delta\|$ of a relative rotation (Rodriguez formula). Other Lie groups can be handled analogously using their respective exponential maps. Rigid body transforms in 2D and 3D are treated as compounds $SE(2) := \mathbb{R}^2 \times SO(2)$ and $SE(3) := \mathbb{R}^3 \times SO(3)$, respectively.

In the following, we assume both random variables and measurements to be $\boxplus$-manifolds so that a generic algorithm can modify and compare them using $\boxplus$ and $\boxminus$, respectively, as if they were flat vectors without any knowledge of the global, possibly non-Euclidean structure of the $\boxplus$-manifold which is automatically enforced by $\boxplus$/$\boxminus$.

B. Constraint Graphs and Least Squares on $\boxplus$-Manifolds

Given a set of random variables $x_{1,\ldots,n}$ to be estimated, the constraint graph is defined by the set of constraints $m_{1,\ldots,m}$ between them where each constraint

$$m_i = (z_i, \Sigma_i, f_i, Y_i = \{x_j | \operatorname{dep}(z_i, x_j)\}) \tag{8}$$

consists of a measurement z_i with covariance Σ_i and a measurement function f_i, which given a set of dependent random variables Y_i computes the expected measurement $\hat{z}_i$.

In both SLAM and calibration problems it is common to assume all measurement errors to be normally distributed, i.e.

$$f_i(Y_i) \boxminus z_i \sim \mathcal{N}(0,\, \Sigma_i) . \tag{9}$$

We first normalize each constraint given the Cholesky decomposition of its covariance $L_i L_i^T = \Sigma_i$ since [12]

$$L_i^{-1}(f_i(Y_i) \boxminus z_i) \sim \mathcal{N}(0,\, I) , \tag{10}$$

i.e. multiplied by L_i^{-1} all measurement errors can be treated as identically distributed with identity covariance. We further define the stacked vector X of random variables

$$X = (x_1, \ldots, x_n)^T \tag{11}$$

and the stacked error function F

$$F(X) = \begin{pmatrix} L_1^{-1}(f_1(Y_1) \boxminus z_1) \\ \vdots \\ L_m^{-1}(f_m(Y_m) \boxminus z_m) \end{pmatrix} . \tag{12}$$

Note that each Y_i in (12) relates to a subset of all x_j only. If this subset is small for most constraints, as is the case for SLAM in particular, the problem is naturally sparse.

It is straight forward to show [14] from (3) that the maximum likelihood solution can be determined by solving the least squares problem

$$\hat{X} = \underset{X}{\operatorname{argmin}} \tfrac{1}{2} \|F(X)\|^2 \tag{13}$$

using any adequate non-linear least squares optimization algorithm [3] such as Gauss-Newton or, for rank deficient problems, Levenberg-Marquardt with only one modification: vector addition and subtraction must be replaced by $\boxplus$ and $\boxminus$, respectively, when working with $\boxplus$-manifold variables [14].

C. The Manifold Toolkit for Matlab: The $\boxplus$-Method in Matlab

A key challenge in developing MTKM was the integration of manifold primitives and compounds with the Matlab type system through its object oriented programming extensions.

To keep code using manifolds very concise manifold objects are designed to look like a vector as much as possible from the outside. The operators $+$ and $-$ are overloaded to perform $\boxplus$ and $\boxminus$ after checking for semantic validity as per the signatures in (1) and (2). **`size`** and **`length`** are overloaded to return the degrees of freedom (DOF) of the $\boxplus$-parameterization.

Manifold primitives are the basic building block and implement the respective $\boxplus$/$\boxminus$-operations as specified in III-A. $\mathbb{R}^n$ is represented as the class `Rn` which takes its

dimensionality as a constructor parameter and holds the contained vector as the member `.vec`. $SO(2)$ is handled by `SO2` encapsulating an angle `.phi`; $SO(3)$ by `SO3` with a rotation matrix `.Q`. Although $SE(2)$ and $SE(3)$ could be represented as compounds we also provide specialized implementations for added performance (due to less indirection) and convenience. Compared to `SO2` and `SO3`, `SE2` and `SE3` add a 2D or 3D position vector `.pos` and the methods `transform()` to get the 3×3 or 4×4 homogeneous transformation matrix mapping a point from local to parent coordinates, and `tolocal(v)` and `fromlocal(v)` to convert a vector from the parent coordinate system to local coordinates and vice versa.

Compound manifolds are created by the function `make_compound` based on a list specifying the member name and type of each sub-manifold plus, optionally, their dimensionality, e.g. for a 3D pose:

```
pose_t = mtk.make_compound('position', @mtk.Rn, 3,
                           'orientation', @mtk.SO3);
```

The returned handle can then be used in place of a class constructor to instantiate objects. Compounds come with an auto-generated implementation of $\boxplus$ and $\boxminus$ and a method `idxOf('submanifoldname')` which determines the start of a sub-manifold component in the stacked δ. This is required to, e.g. populate covariance matrices programmatically.

D. The Manifold Toolkit for Matlab: Least Squares

The primary goal behind MTKM was to get all the tedious book-keeping details normally involved with least squares optimization out of the user's way. So, what are these?

MTKM provides the manifold primitives and a mechanism to construct compound manifolds as discussed above. In particular, this means that singularities in the representation of orientations are not an issue and that data structures and common operations on these are bundled together.

The `OptimizationProblem` class handles the problem description. Unknown random variables and constraints between them are added using the `add_random_var()` and `add_measurement()` methods as discussed in the examples below. From these, a stacked vector X of random variables with appropriate $\boxplus$ and $\boxminus$ implementation, the stacked error function F and the numerical approximation[1] of its Jacobian J is computed automatically. The complex index management machinery involved in this is hidden entirely from the user. The sparsity of the problem is exploited in that only blocks of J are considered where measurements (columns) are dependent on the corresponding random variable (row). All other blocks are known to be zero.

Any non-linear optimization algorithm that uses the error function and the Jacobian to inspect the local behavior of the problem and compares and adjusts intermediate solutions can be applied as the solver. Existing solvers can be re-used as long as they only invoke the supported manifold operations

[1]MTKM does not currently compute the Jacobian analytically as the user would need to specify analytical derivatives of each measurement function.

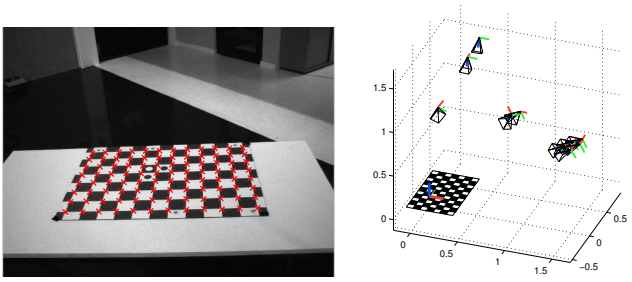

Fig. 2. Results of the single camera calibration: World points projected into the image using the resulting parameters (left). The camera's relation to the checkerboard as estimated in the extrinsic parameters (right). For better presentation, only a subset of the total of 18 camera poses is plotted.

(III-C). Checks such as `isnumeric` or other attempts to access the parameters as a vector need to be adapted. In a case study with `marquardt` from the `immoptibox` [26], we have found that this mainly affects input validation and sanity checks but not the core algorithm (see bundled patch).

MTKM itself comes with text-book-style Levenberg-Marquardt [29] and Gauss-Newton implementations and with examples of common measurement functions (e.g. pinhole projection, 2D and 3D pose relations, 2D landmarks).

IV. Introductory Calibration Example

In this section, we illustrate the use of MTKM in an introductory example by performing single camera calibration, i.e. finding the maximum likelihood estimate of the camera's pin-hole model (intrinsic) parameters, often including radial distortion. Therefore, a calibration target whose 3D geometry is known a priori is observed by the camera from different perspectives and the image error of observed vs. projected (via the pin-hole model) target points is minimized.

However, the camera poses relative to the target points are not known in advance. Thus, the 3D transformation between the target coordinate system and the camera coordinate system at each observing pose, the extrinsic parameters, must also be estimated in addition to the intrinsic ones. The projection itself establishes the constraints between these random variables. Using MTKM, this is captured as a measurement function

```
function z= project_point(intrinsic,x2world,x2cam,p_world)

world2cam = x2cam.transform() * inv(x2world.transform());
p_cam = world2cam * [p_world;1]; % Transform to cam coords

% Project onto the image plane
z = pinhole(intrinsic.focal_length.vec, intrinsic.offset.
    vec, intrinsic.distortion.vec, p_cam(1:3));
```

taking as arguments: the intrinsic camera parameters as a compound, two `SE3` transformations, and the point in world coordinates to be projected. The two transformations are successive steps of the 3D transformation `world2cam` from world coordinates (defined by the calibration target) to camera coordinates enabling use for problems which also estimate an intermediate transformation, e.g. stereo-calibration. For single camera calibration `x2cam` is the identity. The body of the function computes `world2cam` to

Listing 1. Single camera calibration script

```
load_data;

% define custom manifold compound type
intrinsic_t = mtk.make_compound('focal_length', @mtk.Rn,1,
                                'offset', @mtk.Rn,2,
                                'distortion', @mtk.Rn,1);

% compute initial intrinsic and extrinsic parameters
compute_initial_parameters;

% create optimization problem
o = mtk.OptimizationProblem();

% add intrinsic parameters
intrinsic_id = o.add_random_var(intrinsic);

cam2world_id_for_img = cell(num_images);
for i=1:num_images  % for each calibration image
    p_img = Z{i}; p_world = M{i};  % get measurements

    % add extrinsic parameter (camera pose)
    cam2world_id_for_img{i} = o.add_random_var(
         cam2world_for_img{i,1});

    % add measurements (checkerboard corner points)
    for j=1:size(p_img,2)
        o.add_measurement(p_img(:, j), @project_point, {
            intrinsic_id, cam2world_id_for_img{i}}, {mtk.
            SE3(eye(4)), p_world(:, j)}, Sigma);
    end;
end

[X] = nrlm(@o.fun, o.X0);  % solve problem
```

transform `p_world` into camera coordinates, projects it into the image using the function `pinhole` (implementing the well-known pin-hole model) and finally returns the function's result in `z`.

The main script for single camera calibration using MTKM is given in Listing 1. First, data is loaded and a custom manifold type `intrinsic_t` is created (`SE3` is pre-defined). The problem specific `compute_initial_parameters` script instantiates all random variables and assigns initial parameters to them in this case via Zhang's method [35]. The parameters to be estimated are added to an `OptimzationProblem` instance using `.add_random_var()`, which returns an ID. As all measurements depend on the intrinsic parameter its random variable is added right away. While iterating over `num_images` checkerboard images, the corresponding extrinsic random variables are added to the optimization problem. Single measurements of the corners on the pattern are added in a nested loop using `.add_measurement()` taking the following arguments (cf. (8)): The actual measurement `p_img`, a handle to the measurement function `project_point`, a cell array containing the IDs of all dependent random variables and another cell array containing arbitrary user data (in our case the identity transformation and the checkerboard point in the world `p_world`), and the associated uncertainty of the measurement `Sigma`. The cell array entries are passed to the measurement function each time it is evaluated. The IDs are automatically mapped to the current value of the corresponding random variable by the framework. User data is passed unmodified as constants.

Once all measurements have been added, the automatically generated stacked error function and initial stacked random variable vector are available as `@o.fun` and `o.X0`, respectively, which are then passed to `nrlm`, the Levenberg-Marquardt solver bundled with MTKM.

The data set used for validating this example contains 18 images of a camera (1616×1220 px) observing a checkerboard pattern from different poses. Calibration results are presented in Fig. 2. The rms residual is about $(0.26, 0.29)$ px. Convergence was achieved in 10 iterations and, due to passing each observed target point individually, $542.83s$. A modified measurement function operating on all target points of a checkerboard image at once (see `project_points` below) gets this down to $6.34s$ (same number of iterations). For comparison, the special-purpose camera calibration routine by [5] needs $3.34s$ on the same data set (all timings on a Xeon W3520 2.67GHz/32bit Linux/Matlab R2008b).

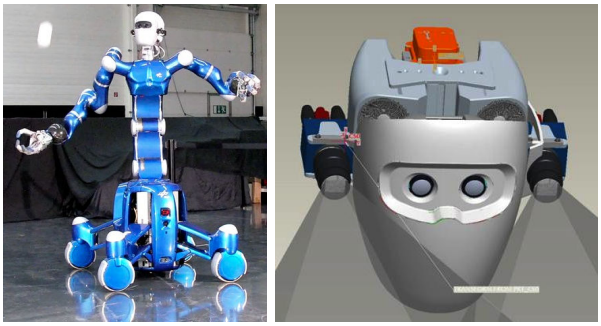

Fig. 3. DLR's humanoid *Rollin' Justin* catches a ball (left). CAD drawing of DLR's *Rollin' Justin's* head showing the stereo camera setup and the orange-boxed IMU (right). Images courtesy of DLR Institute of Robotics and Mechatronics. Used with permission.

V. Multi-Sensor Cross Calibration

A. Rollin' Justin

The proposed toolkit is also used in a more sophisticated setup to cross-calibrate vision and inertial (IMU) sensors, and their relation to a robot's kinematic chain. This calibration routine is part of DLR's humanoid *Rollin' Justin* [4] effort in catching two pitched balls each with one arm. Fig. 3 shows the robot catching a ball and illustrates its sensor setup. For further details on the setup please refer to [2].

Measurements for the calibration procedure are acquired in two steps. First, both cameras observe the corners of a horizontally aligned checkerboard pattern while the inertial sensor measures gravity in its coordinate system. This allows us to calibrate both camera's intrinsic parameters, their transformation (stereo) and the rotation between inertial sensor and cameras (translation is measured manually). Two measurement functions map this relation between parameters and measurements. The first, `project_points`, is simply an extension of `project_point` to project multiple world points. The second measurement function

```
function z = g_world2g_imu(cam2world, cam2imu)

z = cam2imu.Q * inv(cam2world.Q) * [.0 .0 -9.81]';
```

Listing 2. *Rollin' Justin* ball catching calibration script

```
% load p_img_l, p_world_l. p_img_r, p_world_r, g_imu,
% p_lhand_img_[l,r], p_rhand_img_[l,r]
load_data;

intrinsic_t = mtk.make_compound('focal_length', @mtk.Rn,1,
                                'offset', @mtk.Rn,2,
                                'distortion', @mtk.Rn,1);

% create and init i_left, i_right, left2right, left2imu,
% left2head, p_hand_l, p_hand_r, left2world_for_img
compute_initial_parameters;

o = mtk.OptimizationProblem();

% add intrinsic parameters
i_left_id = o.add_random_var(i_left);
i_right_id = o.add_random_var(i_right);

% add transformations
left2right_id = o.add_random_var(left2right);
left2imu_id = o.add_random_var(left2imu);
left2head_id = o.add_random_var(left2head);

% add marker positions on hand
p_hand_l_id = o.add_random_var(p_hand_l);
p_hand_r_id = o.add_random_var(p_hand_r);

left2world_id_for_img = cell(num_images);
for i=1:num_images
    % add extrinsic parameter for every image to problem
    left2world_id_for_img{i} = o.add_random_var(
        left2world_for_img{i});
    o.add_measurement(p_img_l{i}(:), @project_points, {
        i_left_id, left2world_id_for_img{i}}, {mtk.SE3(
        eye(4)), p_world_l{i}(:)}, Sigma_chk);
    o.add_measurement(p_img_r{i}(:), @project_points, {
        i_right_id, left2world_id_for_img{i},
        left2right_id}, {p_world_r{i}(:)}, Sigma_chk);

    if (g_imu{i}(1) ~= -1) % test for valid IMU
        measurement
        o.add_measurement(g_imu{i}(:), @g_world2g_imu, {
            left2world_id_for_img{i}, left2imu_id}, {},
            Sigma_g);
    end;
end

for i=1:size(p_lhand_img_l, 1) % add left arm measurements
    o.add_measurement(p_lhand_img_l(i,:)', @project_marker
        , {i_left_id, left2head_id, p_hand_l_id}, {mtk.
        SE3(eye(4)), lhand2head{i,1}}, Sigma_hand);
    o.add_measurement(p_lhand_img_r(i,:)', @project_marker
        , {i_right_id, left2head_id, p_hand_l_id,
        left2right_id}, {lhand2head{i,1}}, Sigma_hand);
end;

for i=1:size(p_rhand_img_l, 1) % add right arm
    measurements
    o.add_measurement(p_rhand_img_l(i,:)', @project_marker
        , {i_left_id, left2head_id, p_hand_r_id}, {mtk.
        SE3(eye(4)), rhand2head{i,1}}, Sigma_hand);
    o.add_measurement(p_rhand_img_r(i,:)', @project_marker
        , {i_right_id, left2head_id, p_hand_r_id,
        left2right_id}, {rhand2head{i,1}}, Sigma_hand);
end;

[X] = nrlm(@o.fun, o.X0); % solve problem
```

rotates the gravity $(0,\ 0,\ -9.81)^T$ from the world coordinate system (perpendicular to the leveled checkerboard) into the camera coordinate system using the inverse of the extrinsic rotation `cam2world.Q` and from there into the inertial sensor coordinate system using the sought parameter `cam2inertial.Q`.

In the second step, both cameras observe a marker attached to the robot's left and right hand in different kinematic configurations of arm and head (see Fig. 4), enabling calibration of the transformation between the left camera coordinate system and the last head link of the kinematic chain. Therefore, the third measurement function

```
function z = project_marker(intrinsic, left2head, p_hand,
    left2cam, hand2head)

left2hand = mtk.SE3(inv(hand2head.transform()) * left2head
    .transform());

z = project_point(intrinsic, left2hand, left2cam, p_hand);
```

performs a projection of a single point `p_hand` on the hand into the image. For this, the intrinsic parameters are simply passed to the function, while the extrinsic parameters are combined from `hand2head`, which is precomputed from forward kinematics, and the to be estimated `left2head` transformation. To use a single measurement function for both cameras, `left2cam` allows to define a transformation between the considered camera and the left camera, that will be combined with the extrinsic transformation in `project_point`.

Listing 2 shows the MTKM implementation of estimating all parameters jointly using the introduced measurement functions. The overall skeleton is the same as above; there are just more measurements. Since the actual position of the visual marker relative to the hand is not known a priori, two 3D points (one for each hand) are added to the problem.

We then add measurements in three separate loops. First, checkerboard measurements of the left and right camera are added. Note how the two extrinsic arguments of the measurement function are used differently for the left and right image to estimate extrinsic `left2world_id_for_img` and stereo parameters `left2right_id` using only one measurement function `project_points`. Additionally, the corresponding IMU measurement is added for each checkerboard image.

Visual marker measurements are added in two separate loops, one for each arm. Again, only one measurement function, `project_marker`, is used to handle both projections by setting the additional transformation parameter `left2cam` differently for the left camera (`eye(4)`) and for the right cameras (`left2right_id` (stereo)).

To validate the calibration method, 18 different poses were recorded, including 9 with IMU measurements and a horizontally placed checkerboard pattern. The rms residual is about $(0.25, 0.28)$ pixel for the corner measurement, and about $(0.029, 0.015, 0.02) m/s^2$ for the IMU's accelerometer measurement. Additionally, the robot observed 9 left arm poses and 12 right arm poses (see Fig. 4 for two examples), with a joint RMS of $(1.39, 1.2)$ pixel. Convergence was achieved in 8 iterations $(26.5s)$.

B. Kinect and Nao Calibration

We use MTKM to calibrate a Microsoft Kinect (Fig. 1), estimating pin-hole parameters of RGB and infrared (IR) cameras and their relation (stereo) by observing a checkerboard (Fig. 5, left and middle). As the board's depth is implicitly

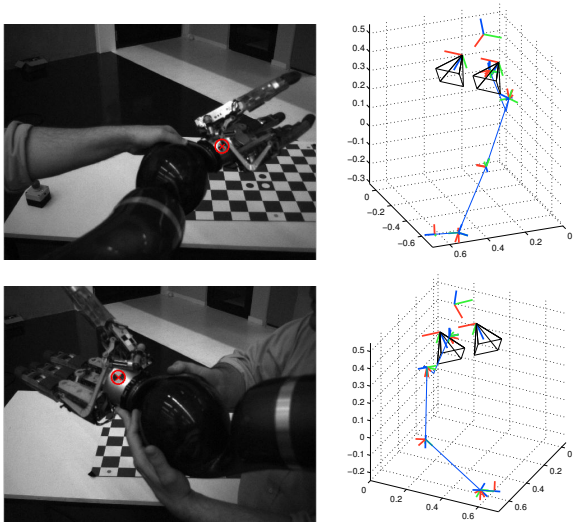

Fig. 4. Two different arm configurations observed by the head cameras using a visual marker (left). Corresponding calibration result for the configurations as computed over all measurements (right). The illustration shows all relevant frames, including both cameras, the IMU (above the cameras), and their relation to the hand marker position through forward kinematics.

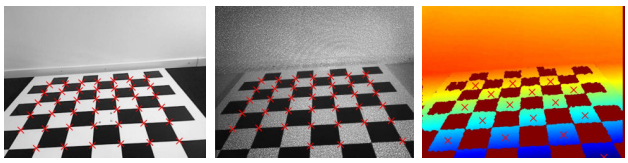

Fig. 5. From left to right: RGB (already converted to grayscale), infrared (IR) and disparity image from a Kinect sensor observing the leveled checkerboard. Used feature points are marked as red crosses.

estimated through the extrinsic parameters, the mapping from depth to disparity is calibrated using well perceived points in the disparity image (Fig. 5, right). Additionally, the Kinect sensor contains an accelerometer, so the rotation from the cameras to the accelerometer and the accelerometer's scale factor can be estimated when the checkerboard is leveled.

In another scenario, MTKM is used to calibrate the stereo vision of the humanoid robot NAO (Fig. 1) for robot soccer. In this robot, two cameras are aligned on a vertical base-line, rotated by $43°$, one facing forward, the other down. Thus, even a large checkerboard appears only partially in each camera image at the same time, making the estimation difficult. To keep the problem well-conditioned (mostly defining each camera's intrinsic parameters) we record separate (mono) images of the full board in addition to the partial stereo images. Compared to regular stereo calibration, the code only needs to be extended by adding the extra `cam2world` variables and checkerboard measurements. This underlines MTKM's flexibility as considerably less effort is spent integrating additional measurements in contrast to [1] or [5].

VI. SLAM

Besides ease of use, the ability to use MTKM for a wide range of problems was a second important goal. We show this flexibility by solving benchmark data sets representing three different classes of SLAM problems: For *2D pose adjustment* we used selected samples from the SPA data set collection ([19]; Fig. 6), for *2D feature-based SLAM* the DLR Spatial Cognition data set ([22]; Fig. 6), and for *3D (6DOF) pose adjustment* the synthetic sphere ([10]; Fig. 7) and the parking garage ([21]; Fig. 1) data sets – each based on the well-known measurement models for pose relations and landmark measurements. The final RSS is nearly identical to that of other frameworks, e.g. for the parking garage data set, it is 1.26840 compared to 1.26837 (SLoM [12]) and 1.23869 (g2o [21]; slightly different parameterization of rotations). Performance-wise, the size of the SLAM problems illustrates that, although the algorithms behind MTKM can be fast, the pure Matlab implementation needs to perform many small operations in loops (as opposed to few but large, vectorized operations which Matlab is optimized for) and is thus orders of magnitude slower than comparable C++ code. E.g. on the parking garage data set, MTKM needs $62s$ per Gauss-Newton iteration, SLoM $135ms$, and g2o $85ms$/$55ms$ (first iteration/all subsequent iterations). A redesign of MTKM to use SLoM wrapped as a MEX Matlab extension behind the scenes has been considered but not investigated further as the performance gain is doubtful if measurement functions remain implemented in plain Matlab code and since it would tie the user to the SLoM-provided optimizers.

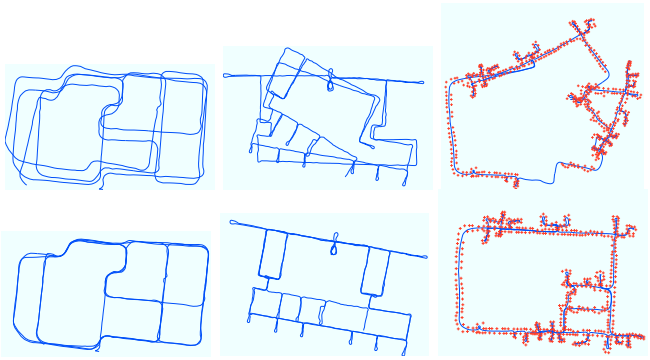

Fig. 6. 2D SLAM data sets before (top row) and after optimization with MTKM (bottom row). From left to right: two selected SPA data sets [19]; the DLR Spatial Cognition data set [22].

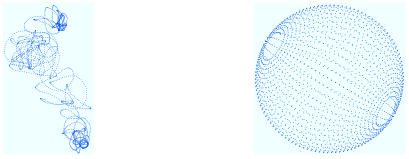

Fig. 7. Synthetic sphere data set [10] before ("mednoise" variant; left) and after optimization with MTKM (right).

We still find MTKM highly useful in a SLAM context because of its flexibility in integrating new sensory data or changing the way measurement functions use the data, and since one can directly use Matlab facilities for analysis and visualization of results. If computation time is an issue we recommend prototyping with MTKM in Matlab and porting this to C++ using SLoM once the system generally works.

VII. Conclusions

We have implemented the $\boxplus$-method in Matlab and built a framework for non-linear least squares optimization on

constraint graphs, the Manifold Toolkit for Matlab (MTKM). We have shown that although this approach has its origins in SLAM it is particularly well-suited to solving non-trivial calibration problems. Using MTKM, even complex calibration tasks such as the calibration of DLR's *Rollin' Justin* can be expressed so concisely that the code was printed in full (apart from the computation of an initial guess). We also showed that MTKM is general enough to handle SLAM problems. We hope MTKM will prove useful to the robotics community in getting calibration tasks done quickly and facilitate experimentation with SLAM problems perhaps also in teaching.

Acknowledgments

This work was partly supported under DFG grants FR2620/1-1 and SFB/TR 8 Spatial Cognition and under BMBF grant 01IS09044B. We thank *Rollin' Justin*'s Team at DLR for providing the robot, G. Grisetti for the parking garage data set and C. Hertzberg for fruitful discussion on and help with SLoM.

References

[1] OpenCV (open source computer vision). `opencv.willowgarage.com`, 2011.

[2] O. Birbach, U. Frese, and B. Bäuml. Realtime perception for catching a flying ball with a mobile humanoid. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2011.

[3] A. Björck. *Numerical methods for least squares problems*. SIAM, Philadelphia, 1996.

[4] C. Borst, T. Wimböck, F. Schmidt, M. Fuchs, B. Brunner, F. Zacharias, P. R. Giordano, R. Konietschke, W. Sepp, S. Fuchs, C. Rink, A. Albu-Schäffer, and G. Hirzinger. Rollin' Justin: Mobile platform with variable base. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2009.

[5] J.-Y. Bouguet. Camera calibration toolbox for Matlab. `vision.caltech.edu/bouguetj/calib_doc`, 2011.

[6] T. A. Davis. *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms. SIAM, Philadelphia, 2006.

[7] F. Dellaert. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. In *Robotics Science and Systems*, 2005.

[8] S. Fuchs and G. Hirzinger. Extrinsic and depth calibration of ToF-cameras. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2008.

[9] K. D. Gremban, C. E. Thorpe, and T. Kanade. Geometric camera calibration using systems of linear equations. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1988.

[10] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics Science and Systems*, 2007. `openslam.org/toro`.

[11] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[12] C. Hertzberg. A framework for sparse, non-linear least squares problems on manifolds. Master's thesis, Universität Bremen, 2008. `openslam.org/slom`.

[13] C. Hertzberg, R. Wagner, O. Birbach, T. Hammer, and U. Frese. Experiences in building a visual SLAM system from open source components. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2011.

[14] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, submitted March 2010. Preprint: `arxiv.org/abs/1107.1119`.

[15] M. Hwangbo and T. Kanade. Factorization-based calibration method for MEMS inertial measurement unit. In *IEEE Int. Conf. on Robotics and Automation*, 2008.

[16] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics*, 24(6): 1365–1378, 2008. `openslam.org/iSAM`.

[17] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2011.

[18] A. Kim and M. Golnaraghi. Initial calibration of an inertial measurement unit using an optical positioning system. In *IEEE Position Location and Navigation Symposium*, 2004.

[19] K. Konolige, G. Grisetti, R. Kümmerle, B. Limketkai, and R. Vincent. Efficient sparse pose adjustment for 2D mapping. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2010.

[20] E. Kraft. A quaternion-based unscented Kalman filter for orientation tracking. In *Proc. of the Sixth Int. Conf. on Information Fusion*, volume 1, pages 47–54, 2003.

[21] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2011.

[22] J. Kurlbaum and U. Frese. A benchmark dataset for data association. Technical Report 017-02/2009, SFB TR/8, 2009.

[23] J. Lobo and J. Dias. Relative pose calibration between visual and inertial sensors. *Int. Journal of Robotics Research*, 26(6): 561–575, 2007.

[24] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333 – 349, 1997.

[25] F. Mirzaei and S. Roumeliotis. A Kalman filter-based algorithm for IMU-camera calibration: Observability analysis and performance evaluation. *IEEE Trans. on Robotics*, 24(5): 1143–1156, 2008.

[26] H. B. Nielsen. immoptibox – A Matlab toolbox for optimization and data fitting. `imm.dtu.dk/~hbn/immoptibox`, 2010.

[27] E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2006.

[28] V. Pradeep, K. Konolige, and E. Berger. Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach. In *International Symposium on Experimental Robotics (ISER)*, New Delhi, India, December 2010.

[29] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes, Third Edition*, pages 799 – 806. Cambridge University Press, Cambridge, 2007.

[30] S. Thrun and J. J. Leonard. Simultaneous localization and mapping. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, chapter 37. Springer, New York, 2008.

[31] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[32] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, 3(4):323–344, 1987.

[33] A. Ude. Nonlinear least squares optimisation of unit quaternion functions for pose estimation from corresponding features. In *Proc. of the Int. Conf. on Pattern Recognition*, 1998.

[34] A. Ude. Filtering in a unit quaternion space for model-based object tracking. *Robotics and Autonomous Systems*, 28(2–3): 163–172, 1999.

[35] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.