


Learning Trajectories for Visual-Inertial System Calibration via Model-based Heuristic Deep Reinforcement Learning

Conference Paper**Author(s):**

Chen, Le; Ao, Yunke; Tschopp, Florian; Cramariuc, Andrei; Breyer, Michel; [Chung, Jen Jen](#) ; Siegart, Roland; Cadena, Cesar

Publication date:

2021

Permanent link:

<https://doi.org/10.3929/ethz-b-000605726>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)

Originally published in:

Proceedings of Machine Learning Research 155

Learning Trajectories for Visual-Inertial System Calibration via Model-based Heuristic Deep Reinforcement Learning

Le Chen*
ETH Zurich
lechen@ethz.ch

Yunke Ao*
ETH Zurich
yunkao@ethz.ch

Florian Tschopp
ETH Zurich
ftschoep@ethz.ch

Andrei Cramariuc
ETH Zurich
crandrei@ethz.ch

Michel Breyer
ETH Zurich
mbreyer@ethz.ch

Jen Jen Chung
ETH Zurich
chungj@ethz.ch

Roland Siegwart
ETH Zurich
rsiegwart@ethz.ch

Cesar Cadena
ETH Zurich
cesarc@ethz.ch

Abstract: Visual-inertial systems rely on precise calibrations of both camera intrinsics and inter-sensor extrinsics, which typically require manually performing complex motions in front of a calibration target. In this work we present a novel approach to obtain favorable trajectories for visual-inertial system calibration, using model-based deep reinforcement learning. Our key contribution is to model the calibration process as a Markov decision process and then use model-based deep reinforcement learning with particle swarm optimization to establish a sequence of calibration trajectories to be performed by a robot arm. Our experiments show that while maintaining similar or shorter path lengths, the trajectories generated by our learned policy result in lower calibration errors compared to random or handcrafted trajectories.¹

Keywords: Visual-Inertial, Calibration, Model-based Deep Reinforcement Learning, Markov Decision Process, Particle Swarm Optimization

1 Introduction

In recent years visual-inertial (VI) sensors, which consist of one or more cameras and an inertial measurement unit (IMU), have become increasingly popular for robust high frequency motion estimation [1, 2, 3, 4, 5]. Before use, VI sensors need to be calibrated, which implies obtaining the parameters for the camera intrinsics, the camera-IMU extrinsics, and the time offset between the different sensors [6, 7]. The performance of VI systems is highly dependent on the quality and accuracy of the calculated calibration parameters [8]. Precise calibrations are usually obtained offline in controlled environments, following sophisticated motion routines ensuring observability [9, 10]. This makes the entire process non-trivial for an inexperienced operator [8, 11]. Additionally, the motion primitives that most effectively render the best calibration results are unknown. So instead of performing this task by hand or with a manually programmed operator, we propose the use of model-based deep reinforcement learning (RL) to learn the best motion primitives and perform them on a robotic arm.

Previous works in automatic calibration use trajectory optimization and reinforcement learning to address this problem. For the class of optimization methods that maximize the observability Gramian of the trajectories on the calibration parameters [12, 13], it remains challenging to model and include other practical optimization objectives, such as maximizing target point coverage for camera calibration and trajectory length minimization for efficiency. A learning-based approach was proposed by Nobre *et al.* [11], where a set of trajectories were pre-defined empirically and Q-learning was applied to choose a sequence of those trajectories that renders sufficient observability of the

*equal contribution

¹The code is publicly available <https://github.com/ethz-asl/Learn-to-Calibrate>

calibration problem. A disadvantage of this approach is that it remains restricted to the collection of pre-defined trajectories and therefore the possible range of movements is not fully explored. We conclude that there are two aspects that could be further studied based on existing previous works:

- **Multiple objectives:** It is not obvious whether the trajectory that renders sufficient observability of states also provides the best calibration. Besides, other empirical requirements such as path length and camera coverage could also be included in the optimized objectives.
- **Learn the trajectory:** A more general learning problem could be posed, where the pre-defined trajectories and their selection are learned jointly in a single optimization problem.

We aim to address both points by designing a RL-based method to obtain the best sequence of VI calibration trajectories that fulfill multiple objectives, including both observability and practical requirements. The approach will be applied separately to calibrate both camera intrinsics and camera-IMU extrinsics, as these two steps have very different motion requirements. The trajectories are executed by a robot arm during training in a realistic simulation for both dynamics and photorealism, thus guaranteeing the feasibility of execution by a real robot arm. Additionally, we simulate different VI sensor configurations to model the variability that also exists in the real world.

To solve the proposed problem, we model calibration as a Markov decision process (MDP) and use model-based RL [14] to establish the sequence of motion trajectories that optimizes sensor calibration accuracy. Compared with other solutions the action space of our model has fewer constraints, making it possible to learn a more general policy for calibration. In addition, we do not only consider different information-theoretic metrics of the trajectories, but also take camera coverage and path length into account, in the reward of our MDP model. For the learning algorithm, we propose a sample efficient model-based RL method using the particle swarm optimization (PSO) [15] algorithm to search for the optimal open-loop control sequence. We adapt the PSO algorithm to RL by utilizing gradient information and memorizing previous optimization results for initialization. This method addresses difficulties in searching for action sequences in a high dimensional space and the high time cost of performing a calibration at each step.

The main contributions of this work are as follows:

- Our approach models the entire calibration process as a MDP and to the best of our knowledge, we are the first to solve it using model-based deep reinforcement learning.
- Our proposed model-based heuristic RL algorithm with adapted PSO satisfies different practical requirements such as the capability of solving problems with high dimensional action space with high sample efficiency.
- The evaluation shows that the learned trajectories deliver more accurate calibrations compared to handcrafted or random ones. We enable easy transferability to real scenarios by also simulating the robot arm together with different sensor configurations.

2 Related Work

The most popular method for camera calibration during the last decades is to use a known calibration pattern and apply nonlinear regression to obtain the parameters. This method has been successfully used both for camera intrinsic [16] and extrinsic calibration [17]. For VI sensor calibration the most reliable and precise approaches are also based on the use of a calibration board. A method presented in [18] applies an Extended Kalman Filter to estimate the relative poses between sensors jointly. A parametric method proposed in [9, 10] represents the pose and bias trajectories using B-splines and introduces a batch estimator in continuous-time. While those methods are relatively efficient, they still require expert knowledge to obtain the required sensor data for good accuracy. In addition, the optimal calibration movements remain unknown, especially when aiming to increase time efficiency and calibration accuracy. The problem of designing the best motion primitive was first addressed by methods based on trajectory optimization to find the trajectory that renders the highest observability of calibration parameters.

A method proposed in [13] optimizes the expanded empirical local observability Gramian of unknown parameters based on the measurement model, to solve for the best state and input trajectories. Preiss *et al.* [12] further extended the method to be obstacle-free and more balanced among multiple objectives. Both of the approaches only optimize observability by using the observability Gramian of

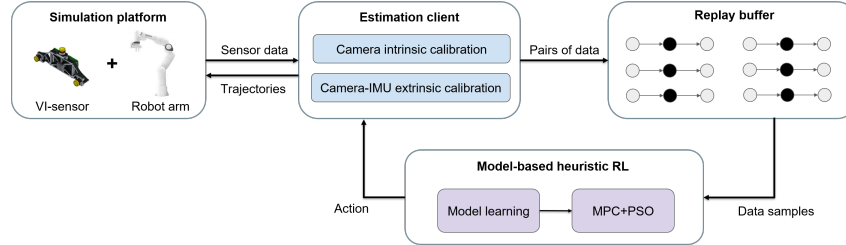


Figure 1: Overview of our calibration framework. The trajectories (actions) the agent chooses to take and their simulated calibration performance are recorded in real-time. This data is then used to train the agent with model-based RL.

the trajectories, where other empirical and general requirements are difficult to model and cannot be easily included. Our framework combines different evaluation metrics of information gain for variables in the reward design and learns to obtain the highest reward using model-based RL.

Another class of methods applies RL to get the best sequence of trajectories selected from a library of pre-designed trajectories. Nobre *et al.* [11] modeled the calibration process as an MDP, where the states are estimated parameters and the actions are choices of trajectories from the library at each step. The MDP planning problem is solved using Q-learning. However, the method only yields suggestions on predefined motions to choose from, rather than exploring new possible motion primitives. In contrast, our method includes the trajectory parameters in the action design and solves the entire calibration problem in an end-to-end fashion.

For such complex sequential decision-making problems, recent works have shown that deep RL algorithms are capable of learning policies that render high performance [19, 20, 21, 22, 23]. Model-based and model-free deep RL are two classes of deep RL algorithms. Model-free algorithms are capable of learning a wide range of sequential decision problems, but they require a large number of samples to achieve good performance [23, 24, 25]. Nagabandi *et al.* [14] combined neural network model learning with sample-based model predictive control (MPC) to improve sample efficiency, and the policy is further fine-tuned with model-free algorithms. Because of the high time cost to perform a calibration at each training step, model-based algorithms are suitable to reduce the number of required episodes to learn a good action sequence.

However, sample-based MPC performs relatively poorly in high dimensional action space, therefore we substitute random sampling in [14] with a metaheuristic algorithm. PSO [15] is such a metaheuristic algorithm that has been widely used in the last two decades due to its good performance in complex high-dimensional problems, which cannot be solved using traditional deterministic algorithms. Hein *et al.* [26] reformulated RL problems as optimization tasks and applied PSO to search for optimal solutions. In this paper, we combine MPC with a modified PSO in a model-based framework to search for optimal action sequences.

3 Method

An overview of the proposed learning framework is shown in Figure 1. Given a set of trajectory parameters, the simulation platform executes the trajectory and records the resulting sensor data. The estimation client then computes the calibration parameters based on the sensor data and transforms all the results into training data according to the MDP formulation of the problem. Finally, the agent samples from the recorded training data to learn the optimal trajectories using model-based heuristic RL and chooses an action to execute.

3.1 Visual-Inertial Calibration

Our estimator follows the Kalibr framework [9, 10], where the Levenberg-Marquardt algorithm is applied to minimize the loss between the obtained and predicted measurements to maximize the likelihood of the unknown parameters $Pr(X, \theta | D, L)$. Here, X is the estimated pose trajectory, θ depicts the calibration parameters, D are the measurements of a VI sensor consisting of images and inertial measurements and L is the known position of landmarks on the calibration target. As explained in detail in [27], the covariance matrix of the known parameters $\Sigma_{X\theta}$ can be obtained from the Jacobian of all error terms and the stacked error covariances. The covariance of the calibration

parameters Σ_θ can be extracted from Σ_{X_θ} and further normalized to $\bar{\Sigma}_\theta$. The information gain can then be evaluated with the following metrics:

- **A-Optimality:** $H_{A_{opt}} = \text{trace}(\bar{\Sigma}_\theta)$
- **D-Optimality:** $H_{D_{opt}} = \det(\bar{\Sigma}_\theta)$
- **E-Optimality:** $H_{E_{opt}} = \max(\text{eig}(\bar{\Sigma}_\theta))$

Minimizing these H metrics leads to the maximization of information gain and furthermore can be used for the reward design of our proposed method.

3.2 MDP Model for Learning to Calibrate

The whole calibration process is modeled as an MDP. The process description includes a state S_t , action A_t , transition model $S_{t+1} = f(S_t, A_t)$, and reward R_t at each time step t . We define the action A_t at each time as a looped parameterized trajectory with the same start and terminal pose. Each action represents a trajectory subsequence, and these subsequences are concatenated to form the final calibration trajectory. At each step, the calibration process needs to be rerun over the entire sequence and cannot be run only over the newly acquired measurements. The trajectory poses $\{[x_j, y_j, z_j, \alpha_j, \beta_j, \gamma_j]^\top\}_{j=1:J}$ are parameterized by $\{\sum_{q=1,2,4} \mathbf{a}_q (1 - \cos \frac{2q\pi j}{J}) + \mathbf{b}_q \sin \frac{2q\pi j}{J}\}_{j=1:J}$, where J is the number of waypoints inside one action. \mathbf{a}_q and \mathbf{b}_q are 6×1 vectors. Thus, for one trajectory 36 parameters are needed, which is the action space dimension of the MDP. The reason to choose sin and cos as basis functions is that they are capable of representing many good empirical trajectories for calibration.

Let D_t be the measurements acquired with action A_t , and Y_t be the vector that stacks all the calibration parameters and their information gain status, then $Y_t = [\theta_t^*, O_t]^\top = \text{Cal}(\bigcup_{i=0}^{t-1} D_i)$, where Cal represents the calibration process that returns the predicted calibration parameters θ^* and their respective information gain status O_t . For camera calibration O_t contains the progress of the coverage of the horizontal axis, the vertical axis, size and skew. For camera-IMU calibration, O_t is composed of the eigenvalues of the covariance matrix for extrinsics, used in the optimization process of the calibration tool.

Ignoring the sensor noise, we assume the action history sequence together with the calibration status determine the measurement sequence: $\bigcup_{i=0}^{t-1} D_i = h(A_{0:t-1}, Y_{0:t-1})$, where h represents an unknown mapping. In this way, the state S_t is defined as the concatenation of all actions and calibration results of previous time steps: $S_t = [A_{0:t-1}, Y_{0:t}]$. The state transition satisfies the Markov property and the transition model becomes

$$S_{t+1} = A_{0:t} \cup Y_{0:t+1} = A_{0:t-1} \cup Y_{0:t} \cup Y_{t+1} \cup A_t = S_t \cup \text{Cal}(h(A_{0:t}, Y_{0:t})) \cup A_t \quad (1)$$

where the right hand side of (1) only depends on S_t and A_t .

Finally, the reward at each step is composed of four parts: empirical reward e_t , information gain for the calibration parameters o_t , trajectory length l_t and relative calibration error d_t . The empirical reward encodes intuitive requirements such as image view coverage with target observations. The information gain includes different evaluation metrics such as the determinant, trace, and eigenvalues of the aforementioned covariance matrix for the extrinsics. The trajectory length l is computed by summing up the position and Euler angle distances between each two neighboring waypoints

$$l = \sum_{j=1}^J (\|x_j - x_{j-1}, y_j - y_{j-1}, z_j - z_{j-1}\|_2 + C \|\alpha_j - \alpha_{j-1}, \beta_j - \beta_{j-1}, \gamma_j - \gamma_{j-1}\|_2), \quad (2)$$

where C is a tuneable weighting factor to balance the importance between rotation and translation. The calibration errors are defined as the Euclidean distance between the calibration result θ^* and ground truth θ . The relative calibration error is further normalized by the norm of the ground truth. In a real world scenario, where the ground truth calibration parameters are not available, this reward term can instead be computed using the reprojection error of the calibration process, as it directly measures how close to the ground truth calibration we currently are. Finally, the reward is the weighted sum of increments of each term at each time step

$$R_t = \eta_1 \Delta e_t + \eta_2 \Delta o_t - \eta_3 \Delta d_t - \eta_4 \Delta l_t, \quad (3)$$

where η_1, \dots, η_4 are positive weights that can be manually tuned separately.

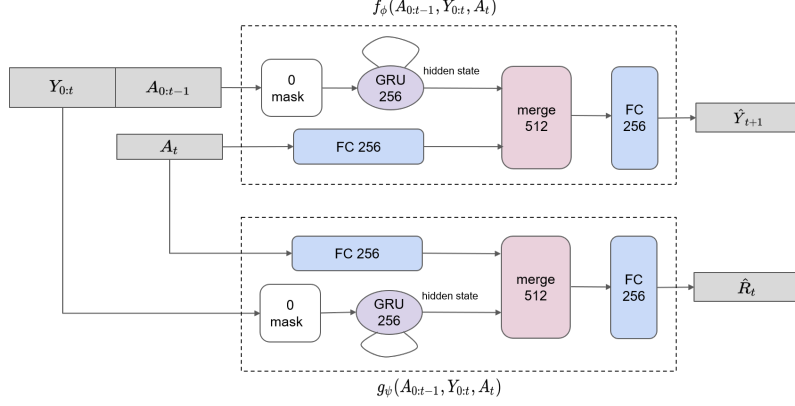


Figure 2: Network architecture of the dynamics model (top) and reward model (bottom).

3.3 Model-based Heuristic Reinforcement Learning with PSO

Neural network dynamics and reward functions: We parameterize both the dynamics model and reward model as neural networks. The network structures are shown in Figure 2. Each model contains a recurrent neural network [28] to encode a hidden state to compress the information acquired so far, given the action history sequence $A_{0:t-1}$ and the calibration status sequence $Y_{0:t}$ as inputs. The reward model then feeds the hidden state and current action input A_t to a fully-connected network to predict the current reward $\hat{R}_t = g_\psi(A_{0:t-1}, Y_{0:t}, A_t)$. With the same input, the dynamics model predicts the next calibration status $\hat{Y}_{t+1} = f_\phi(A_{0:t-1}, Y_{0:t}, A_t)$, where ψ and ϕ are the weights of the two neural networks. Given a batch of training data $\{Y_{0:t}^i, A_{0:t-1}^i, A_t^i, Y_{t+1}^i, R_t^i\}_{i=1}^N$, both models are trained using stochastic gradient descent (SGD) to minimizing the mean squared dynamics and reward errors

$$\epsilon_{dyn} = \frac{1}{N} \sum_{i=1}^N \|Y_{t+1}^i - \hat{Y}_{t+1}^i\|^2, \quad (4)$$

$$\epsilon_{reward} = \frac{1}{N} \sum_{i=1}^N \|R_t^i - \hat{R}_t^i\|^2 \quad (5)$$

respectively. A higher model prediction accuracy is fundamental to the performance of the learned action sequence.

Model-based open-loop optimization: With the learned reward and dynamics model, we use a model predictive control (MPC) to control the agent. In the MPC framework, an open-loop action sequence $A_{t:T}^*$ from the current time step t to the end time T is first optimized to maximize the predicted future sum of rewards. Then only the first action A_t is executed and the corresponding new states and rewards are obtained. The optimal action sequence until the end $A_{t+1:T}^*$ is then recalculated and the next action is executed. For our problem, the open-loop optimization to solve the optimal future action sequence $A_{t:T}^*$ at each time step t is formalized as

$$A_{t:T}^* = \arg \max_{A_{t:T}} \sum_{\tau=t}^T g_\psi(A_{0:\tau-1}, Y_{0:t}, \hat{Y}_{t+1:\tau}, A_\tau), \quad (6)$$

where $\hat{Y}_{t+1:\tau}$ is predicted using the dynamics model f_ϕ for each time step.

This non-linear optimization problem is solved using a modified version of the PSO algorithm. For a particle swarm with M particles in total, each particle position $P_{i \in \{1, 2, \dots, M\}}^t$ represents a potential solution of actions $A_{t:T}^i$. For the position update, the velocities v of particles of the original PSO algorithm include 3 components: social component, cognitive component, and inertia [15]. In our method, the cognitive component is modified to be the gradient of the optimization function, rather than the direction towards the local best position the particle has ever visited. This is because the gradients are more informative to indicate the local optimal position. Furthermore, the gradients can be obtained from the learned model as $\mu_i = \nabla_{A_{t:T}} \sum_{\tau=t}^T g_\psi(A_{0:\tau-1}, Y_{0:t}, \hat{Y}_{t+1:\tau}, A_\tau)|_{A_{t:\tau} = P_i^t}$. In

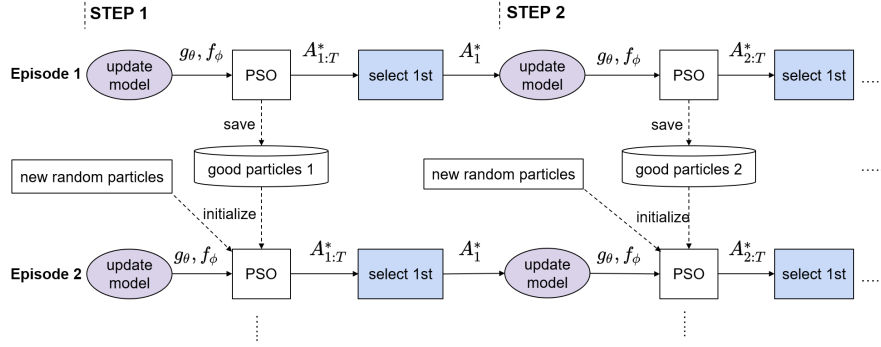


Figure 3: Particle initialization and action selection in subsequent episodes.

this way, the particles move according to the following rules at each optimization iteration ι

$${}^{\iota}v_i = \omega_0 {}^{\iota-1}v_i + c_1(G_{best} - {}^{\iota-1}P_i^t) + c_2 {}^{\iota-1}\mu_i \quad (7)$$

$${}^{\iota}P_i^t = {}^{\iota-1}P_i^t + {}^{\iota}v_i, \quad (8)$$

where G_{best} represents the global best position all the particles have covered. The first two terms in Equation (7) represent the inertia and social components, the last one is the modified cognitive component. The parameters ω_0, c_1, c_2 are coefficients that can be tuned to trade-off between exploration and exploitation.

Due to high computational costs, at each time step, only a small number I of iterations are used for PSO. A good solution cannot be guaranteed with a limited number of iterations and random initialization, but the performance can be improved by using previous optimization results to initialize new particles. Although the optimization problems are different between different time steps in the same episode (the lengths of open-loop action sequences to optimize are different), they are similar between the same time step of different episodes. Therefore, we save the positions of the top K particles after each time step as P_{best}^t , for initialization of new particles of the same time step in subsequent episodes. The other $M - K$ new particles are randomly initialized to enable exploration and avoid local minima.

Model predictive control: After I iterations and obtaining the final position of all particles $P_{1:M}^t = \{A_{t:T}^i\}_{i=1}^M$ at each time step, a particle $A_{t:T}^*$ is selected as the open-loop control solution. Different policies are applied to select the particle for training and testing in our method. For training episodes, in order to encourage exploration, the action sequence is randomly selected from the top W particles with the the highest predicted reward sum, rather than always selecting the globally best particles. To prevent the top W particles from converging to the same point, $W > K$ should be satisfied. In this way, the W particles contain not only the K particles that have been optimized continuously among episodes but also some new particles that are initialized randomly in the current episode. For testing, only the particle with the highest predicted sum of rewards is selected. Following the MPC rule, we only extract the first action A_t^* from $A_{t:T}^*$ for execution. An overview of the particle initialization and action selection framework is shown in Figure 3. The entire algorithm is summarized in Algorithm 1.

4 Implementation

We evaluate our method by performing experiments in a simulation platform based on Gazebo [29]. This includes a checkerboard, and a VI sensor consisting of a pinhole camera and an IMU mounted on the end-effector of a FRANKA EMIKA Panda robot arm. To increase robustness and generalization, both the camera intrinsics and camera-IMU extrinsics are sampled from Gaussian distributions in every episode. Each interaction episode is an independent calibration process without sharing measurement with other episodes. However, the training process is off-policy so that all recorded interaction data is utilized to update the model to improve sample efficiency. For the camera intrinsic calibration, we limit the maximum time step size T to be 4 (run at most 4 looped trajectories) in each training episode. The empirical reward is the coverage of the image view with target observations. The accuracy reward is computed by dividing the decrease in Euclidean distance from the ground truth by the norm of the ground truth. The reward and dynamic models for intrinsic calibration are trained in total for 1000 episodes.

Algorithm 1 Model-based Heuristic Reinforcement Learning

```
1: set  $T, M, K, W$  and  $I$ , where  $K, W < M$ 
2: initialize  $g_\psi, f_\phi$  and randomly initialize  $\{P_{\text{best}}^t\}_{t=0}^T$ 
3: gather dataset  $D_{RL}$  with random trajectories
4: for each episode do
5:   reinitialize the MDP
6:   for  $t=0$  to  $T$  do
7:     train  $g_\psi$  and  $f_\phi$  by applying SGD to (4)-(5) with  $D_{RL}$ 
8:     update  $Y_{0:t}, A_{0:t-1}$ 
9:     initialize  $P_{1:M}^t$  by assigning  $P_{1:M}^t = \{P_{\text{best}}^t, \text{randomly sampling } P_{K+1:M}^t\}$ 
10:    for  $\iota=0$  to  $I$  do
11:      update  $P_{1:M}^t$  using (7)-(8)
12:    end for
13:    sort  $P_{1:M}^t$  descendingly by the predicted reward sum
14:    update the best  $K$  particles  $P_{\text{best}}^t = P_{1:K}^t$ 
15:    randomly select  $A_{t:T}^*$  from  $P_{1:W}^t$ 
16:    execute  $A_t^*$  and obtain  $Y_{t+1}, R_t$ 
17:    add  $\{Y_{0:t}, A_{0:t-1}, A_t, Y_{t+1}, R_t\}$  to  $D_{RL}$ 
18:  end for
19: end for
20: return  $g_\psi$  and  $f_\phi$ 
```

For the camera-IMU extrinsic calibration, each training episode contains 3 time steps. The empirical reward includes the entropy of IMU measurement data in 6 dimensions and number of target observations captured per image. The model for extrinsic calibration is first trained for 900 episodes. Subsequently, we fine-tune the agent for another 650 episodes by integrating Kalibr [9, 10] into the framework. Two items are added to the reward. One is the information gain reward, which is the negative A-optimality H_{Aopt} computed based on the covariance matrix. The other one is the calibration accuracy itself. For PSO, we use $M = 15, K = 5, W = 5$ and $I = 5$. Finally, it should be noted that when training on real systems, the ground truths are not available. In this case calibration error could be replaced by the reprojection error given from the calibration process (see Appendix C).

5 Experiments

In this section, we conduct two groups of experiments to evaluate the learned trajectories for camera intrinsic calibration and camera-IMU extrinsic calibration. The learned trajectories are extracted from the result actions using an MPC framework on the final reward model and dynamics model. We compare our learned trajectories with empirically handcrafted trajectories and ones with random parameters as baselines. The path lengths and calibration errors are computed as stated in Section 3. In addition, we also verified that a favorable policy can be learned when calibration error is substituted by the reprojection error. These results are shown in Appendix C.

5.1 Camera Intrinsic Calibration

	Camera Intrinsic Calibration		Camera-IMU Extrinsic Calibration		
	Mean error	Path length	Mean error	Path length	Mean A-optimality
Random trajectory	0.560 %	9.627 m	0.396 %	7.331 m	$4.16 \cdot 10^{-08}$
Handcrafted trajectory	0.196 %	11.116 m	0.340 %	3.306 m	$1.97 \cdot 10^{-07}$
Learned trajectory	0.159 %	11.037 m	0.265 %	4.367 m	$5.83 \cdot 10^{-08}$
Fine-tuned trajectory	—	—	0.214 %	3.241 m	$9.83 \cdot 10^{-08}$

Table 1: Comparison of mean relative calibration error, path length, and A-optimality between random, handcrafted, learned, and fine-tuned trajectories. Note that the handcrafted and learned trajectories are different for intrinsic and extrinsic calibration. The means are averaged over all intrinsics and extrinsics settings for evaluation. The path lengths are computed by Equation 2 where $C = 1$ m/rad.

We evaluate our policy on cameras with the same image size (640 px \times 480 px) but different FoVs. The relative errors of the intrinsic calibration results w.r.t. the ground truth in different settings of horizontal FoVs are reported in Table 1 and Figure 4. For smaller FoVs, the handcrafted trajectory

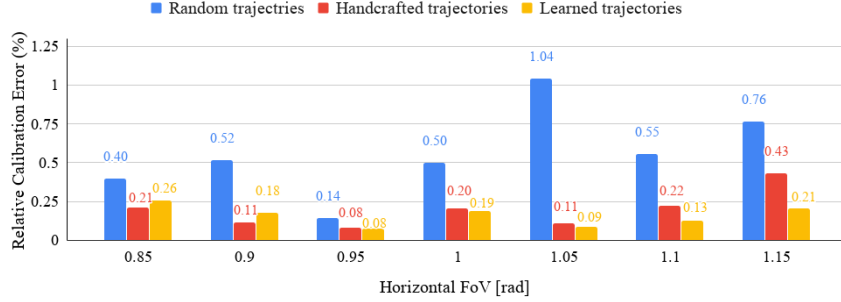


Figure 4: Relative errors of the intrinsic calibration results w.r.t. the ground truth for cameras with different horizontal field of views (FoVs). For each pair of camera setting and policy, we perform the calibration experiments 5 times and calculate the average relative calibration errors w.r.t the ground truth.

renders slightly higher calibration accuracy than learned trajectories. This may be because, with smaller FoVs, the target appears larger in the image view, which makes it easier for both the learned and handcrafted trajectories to achieve high calibration accuracy. However, with larger FoVs which make the target smaller in the image view and harder to achieve a high image coverage with target observations, the learned trajectory shows better performance. Table 1 shows that under similar path lengths, our framework could learn how to perform favorable motion trajectories and collect enough measurements efficiently that yield the desired camera intrinsic parameters.

5.2 Camera-IMU Extrinsic Calibration

In this experiment, we train an agent to learn a policy for camera-IMU extrinsics calibration. As shown in Figure 5 and Table 1, the trajectories generated by our learned policy without fine-tuning can achieve higher calibration accuracies compared with the random and handcrafted trajectories. The low mean A-optimality of learned trajectories can be interpreted as a confidence of Kalibr about the calibration results. The fine-tuned trajectories perform the best as they achieve the lowest mean error and the shortest path length. The higher mean A-optimality of the fine-tuned trajectory compared with the learned trajectory is possibly caused by the lower path length. A lower path length provides less data, which results in Kalibr being less confident about the calibrations.

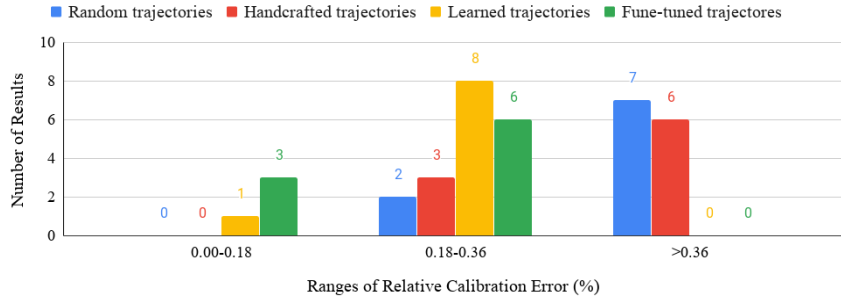


Figure 5: The distribution of the relative errors of extrinsic calibration results w.r.t. the ground truth. Each trajectory is tested with 9 different pairs of camera intrinsic and VI extrinsic configurations.

6 Conclusion

In this work, we introduced a novel framework that uses model-based deep RL with an adapted version of PSO for sampling, to generate trajectories for efficiently collecting measurements to calibrate both camera intrinsic and VI extrinsic parameters. Our experiments show that under similar or even shorter path lengths, the trajectories generated by our learned policy can lead to more accurate results and a higher calibration confidence. While the proposed model-based RL framework is able to achieve state of the art performance in our MDP model for VI calibration, an interesting avenue for further improvement is to integrate our approach with model-free learners. Additionally, model-free RL can also be applied for further fine-tuning the learned policies.

Acknowledgments

This paper was partially supported by ABB Corporate Research, Siemens Mobility GmbH, ETH Mobility Initiative under the project *PROMPT*, and the Luxembourg National Research Fund (FNR) 12571953.

References

- [1] P. Corke, J. Lobo, and J. Dias. An introduction to inertial and visual sensing. *The International Journal of Robotics Research*, 26(6):519–535, 2007.
- [2] F. Tschopp, T. Schneider, A. W. Palmer, N. Nourani-Vatani, C. Cadena, R. Siegwart, and J. Nieto. Experimental comparison of visual-aided odometry methods for rail vehicles. *IEEE Robotics and Automation Letters*, 4(2):1815–1822, 2019. doi:10.1109/LRA.2019.2897169.
- [3] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3): 314–334, 2015.
- [4] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart. Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research*, 36(10):1053–1072, 2017.
- [5] T. Qin, P. Li, and S. Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018. doi:10.1109/TRO.2018.2853729.
- [6] F. Tschopp, M. Riner, M. Fehr, L. Bernreiter, F. Furrer, T. Novkovic, A. Pfrunder, C. Cadena, R. Siegwart, and J. Nieto. Versavis—an open versatile multi-camera visual-inertial sensor suite. *Sensors*, 20(5):1439, 2020.
- [7] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 431–437. IEEE, 2014.
- [8] J. Kelly and G. S. Sukhatme. Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration. *The International Journal of Robotics Research*, 30(1):56–79, 2011.
- [9] P. Furgale, J. Rehder, and R. Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286. IEEE, 2013.
- [10] P. Furgale, T. D. Barfoot, and G. Sibley. Continuous-time batch estimation using temporal basis functions. In *2012 IEEE International Conference on Robotics and Automation*, pages 2088–2095. IEEE, 2012.
- [11] F. Nobre and C. Heckman. Learning to calibrate: Reinforcement learning for guided calibration of visual-inertial rigs. *The International Journal of Robotics Research*, 38(12-13):1388–1402, 2019.
- [12] J. A. Preiss, K. Hausman, G. S. Sukhatme, and S. Weiss. Trajectory optimization for self-calibration and navigation. In *Robotics: Science and Systems*, 2017.
- [13] K. Hausman, J. Preiss, G. S. Sukhatme, and S. Weiss. Observability-aware trajectory optimization for self-calibration with application to uavs. *IEEE Robotics and Automation Letters*, 2(3): 1770–1777, 2017.
- [14] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.
- [15] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN’95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

- [16] P. F. Sturm and S. J. Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. PR00149)*, volume 1, pages 432–437. IEEE, 1999.
- [17] Q. Zhang and R. Pless. Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2301–2306. IEEE, 2004.
- [18] F. M. Mirzaei and S. I. Roumeliotis. A kalman filter-based algorithm for imu-camera calibration: Observability analysis and performance evaluation. *IEEE transactions on robotics*, 24(5): 1143–1156, 2008.
- [19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.
- [23] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.
- [24] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.
- [25] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018.
- [26] D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. Reinforcement learning with particle swarm optimization policy (pso-p) in continuous state and action spaces. *International Journal of Swarm Intelligence Research (IJSIR)*, 7(3):23–42, 2016.
- [27] T. Schneider, M. Li, C. Cadena, J. Nieto, and R. Siegwart. Observability-aware self-calibration of visual and inertial sensors for ego-motion estimation. *IEEE Sensors Journal*, 19(10):3846–3860, 2019.
- [28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- [29] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.
- [30] G. Bradski and A. Kaehler. *Opencv. Dr. Dobb's journal of software tools*, 3, 2000.
- [31] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmman, and R. Siegwart. Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4304–4311. IEEE, 2016.

7 Appendix

A Implementation Details

Simulation settings: We use Gazebo [29] for dynamics and sensor simulation. Our simulation environment includes a checkerboard, and a VI sensor consisting of a pinhole camera and an IMU mounted on the end-effector of a FRANKA EMIKA Panda robot arm². For the robot arm, the mass of the links is computed according to the total mass of the robot with a uniform density assumption. The physical parameters for materials are set by the values for aluminum. A typical PID controller is set for each joint. The exact configuration of the nominal VI-sensors is shown in Figure 6a, which can be modified by changing the camera-IMU extrinsics. The detailed sensor settings are shown in Table 2 and 3. We include noise and drift for the IMU and distortion for the camera to achieve more realistic simulation. During training, the parameters for camera intrinsics and camera-IMU extrinsics are re-sampled from Gaussian distributions after each episode, to ensure that the model learns to generalize well also to other similar sensors. The parameters for the Gaussian distributions are shown in Table 4. Our target board is a 7×6 checkerboard with $6 \text{ cm} \times 6 \text{ cm}$ squares. The distance from the target to the robot arm’s initial pose is 2 m.

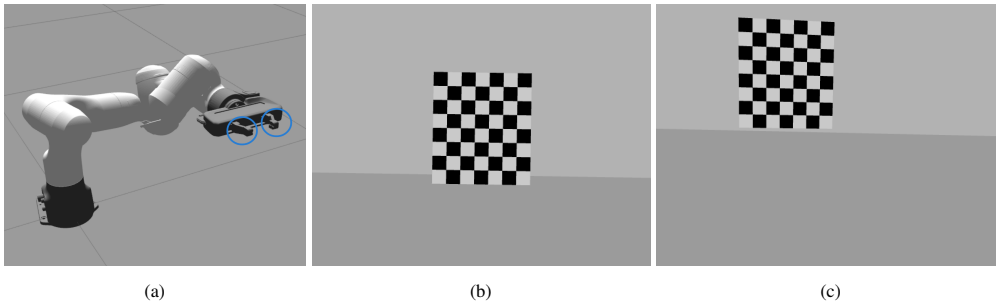


Figure 6: The position of sensors mounted on the end-effector and example image views. (a): positions of the camera (left finger) and IMU (right finger). (b)-(c): example image views from the camera, where (b) is the initial pose, (c) is for a different pose that changes the image view.

update rate	acceleration drift	acceleration noise	angular velocity drift	angular velocity noise
200 Hz	0.006 m/s^2	0.004 m/s^2	$0.000038785 \text{ rad/s}$	0.0003394 rad/s

Table 2: Simulation settings for the IMU.

update rate	width	height	nominal horizon FOV	camera model
10 Hz	640 px	480 px	1.0 rad	pinhole

Table 3: Simulation settings for the camera.

Parameters	Intrinsics		Extrinsics				
	Horizontal FOV [rad]	X [m]	Y [m]	Z [m]	Roll [rad]	Pitch [rad]	Yaw [rad]
Mean	1.00	0.06	0.00	-0.10	0.00	0.00	1.5708
Std.	0.05	0.01	0.01	0.01	0.10	0.10	0.10

Table 4: Gaussian distribution settings for intrinsics and extrinsics parameters during training.

Estimation: First, our camera intrinsic calibration is based on the OpenCV [30] calibration toolbox. The image samples are recorded from the Gazebo camera sensor data. Not all samples are included in the database in practice to avoid processing redundant data and unbalancing the calibration. Instead, given a new sample, the OpenCV calibration toolbox compares the variety of this sample with all the samples in the database and judges whether the motion speed is sufficiently low between this frame and the previous frame. Furthermore, if this sample is different enough from others and the movement speed is not too high, it would be added to the database. The algorithm also judges if the samples

²<https://erdalpekel.de/?p=55>

	Hyperparameter	Value
Intrinsic reward	empirical weight η_1	1.0
	relative error weight η_3	1.0
	path length weight η_4	0.2
	accuracy bonus	5.0
	reprojection error weight η'_3	2.0
Extrinsic reward	empirical weight η_1	1.0
	information gain weight η_2	1×10^8
	relative error weight η_3	1.0
	path length weight η_4	1.0
	reprojection error weight η'_3	2.0
Network training	reward model learning rate	1×10^{-4}
	dynamics model learning rate	1×10^{-4}
PSO settings	social component weight c_1	1×10^{-5}
	cognitive component weight c_2	1×10^{-4}
	inertia weight ω_0	1×10^{-5}
	max iterations I	5
	number of particles M	15
	number of top particles for initialization K	5
	number of top particles to select for action execution W	5

Table 5: Hyperparameter settings for reward design, network training and PSO.

in the database provide good coverage and variation of target observations to do the calibration by computing how much progress has been made toward adequate variation. The coverage of target observations in the image view is computed by the sum of ‘X’, ‘Y’, ‘size’, and ‘skew’ coverage progress. When exceeding a certain threshold, the calibration procedure is triggered and the camera parameters are estimated.

On the other hand, our camera-IMU extrinsic calibration is based on the Kalibr [9, 10, 31] toolbox. The data input for calibration are camera and IMU measurements from Gazebo, which are recorded as ROS-bag files. The information gain is computed based on the covariance matrices extracted from the linear solver of the Kalibr framework. During training, the maximum step of optimization for Kalibr is limited to 1 to reduce time cost, while during testing it is set to 10 to achieve accurate calibrations.

Trajectory planning and execution: We use the Move-It! manipulation software for planning and execution of trajectories of the end-effector. To enable sequential trajectory execution, at the beginning and end of each loop trajectory (action) at each time step, the end-effector returns to a predefined initial pose. The pose is selected to render high controllability so that the end-effector is able to move freely in a larger space around this pose. At this pose, the target board is also centered in the image view as is shown in Figure 6b. Given the current action parameters, poses of a sequence of waypoints are first computed based on the expression of the trajectory. Then the Move-It! planner computes a Cartesian path that follows all those waypoints and executes the trajectory with joint controllers. To reduce the possibility of getting stuck while executing the trajectories, we limit the maximum absolute value of each element of action to be 0.015. The trajectory parameters for roll, pitch, and yaw angle are multiplied by 2.5, 2.5, and 5 respectively to obtain reasonable scales.

Training: We use TensorFlow as our deep learning framework. For camera calibration, we train the reward and dynamics model for 900 episodes. For the camera-IMU calibration, we first train the reward and dynamics model with only empirical and path length rewards for 1000 episodes. This training is relatively fast as there is no need for interfacing with the Kalibr toolbox. Then, the model is fine-tuned for another 650 episodes by including information gain and accuracy reward obtained from Kalibr. In both cases, we directly extract the action sequence chosen by the MPC controller for the learned model as final resulting trajectories. All models were trained on a single desktop computer (Intel i7-9750H CPU @ 2.60GHZ) with an NVIDIA GTX 1660 Ti GPU.

B Hyperparameters

As is shown in Table 5, our reward design for the intrinsic and extrinsic calibration is slightly different. For the intrinsic camera calibration, we give an extra bonus for accurate calibration if the relative error

	<i>Camera Intrinsic Calibration</i>		<i>Camera-IMU Extrinsic Calibration</i>		
	Mean error	Path length	Mean error	Path length	Mean A-optimality
Random trajectory	0.560 %	9.627 m	0.396 %	7.331 m	$4.16 \cdot 10^{-08}$
Handcrafted trajectory	0.196 %	11.116 m	0.340 %	3.306 m	$1.97 \cdot 10^{-07}$
Learned trajectory	0.159 %	11.037 m	0.265 %	4.367 m	$5.83 \cdot 10^{-08}$
Fine-tuned trajectory	—	—	0.214 %	3.241 m	$9.83 \cdot 10^{-08}$
Learned with reproj error	0.155 %	9.319 m	0.268 %	4.333 m	$8.39 \cdot 10^{-08}$

Table 6: Comparison of mean relative calibration error, path length, and A-optimality between random, handcrafted, learned, and fine-tuned trajectories when calibration error is substituted by reprojection error for training.

is less than 1 %. For the extrinsic camera-IMU calibration, as we limit the maximum optimization step of Kalibr, obtaining accurate results is not realistic. Therefore, the reward for accuracy only includes the calibration error. The weights are chosen to make the scales of different parts of the reward comparable. We use the same training setup and PSO hyperparameters for both intrinsic and extrinsic calibration. For the PSO settings, we set a low weight for the social component to avoid too early convergence. In this way, the particles will focus on exploiting their local regions, which benefits searching for the global optimum.

C Additional Experimental Results

The result outputs by the Kalibr toolbox for the handcrafted, initial learned, and fine-tuned trajectories are shown in Figure 7. High noise and drifts are imposed on the simulated IMU sensor, as can be shown in Figure 7a-7c, which causes deviation of the measurements from the predicted trajectories. Regarding re-projection errors shown in Figure 7g-7i, the fine-tuned trajectories achieve smaller and more symmetric error distribution compared to both hand-crafted and fine-tuned trajectories.

Additionally, we extend the result in the main paper by conducting experiments on using reprojection error instead of calibration error during training, as would be done in a real world setup. The result is shown in the table 6. For the camera intrinsic calibration, we achieved similar results. For camera-IMU extrinsic calibration, the trajectories using reprojection error outperform the handcrafted trajectories while being only slightly worse than the trajectories using the calibration error.

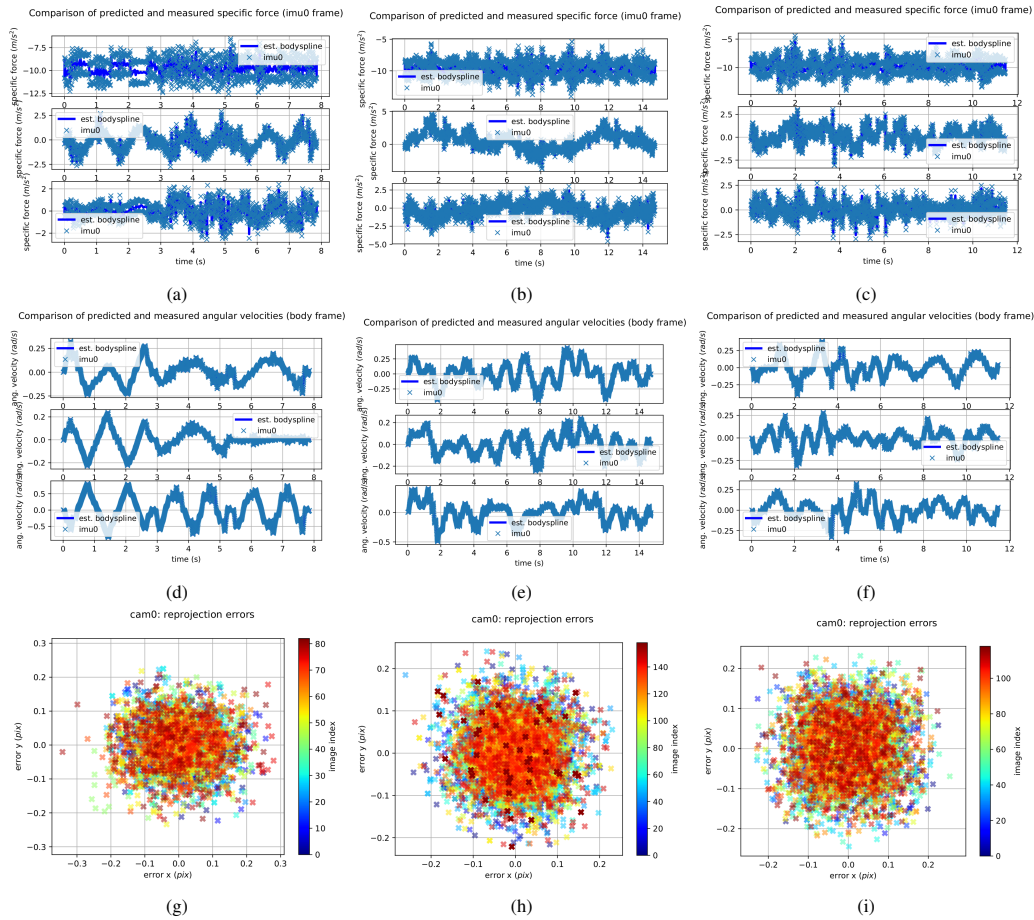


Figure 7: Extrinsic calibration results from the Kalibr toolbox. (a), (d), (g): Results for handcrafted trajectories. (b), (e), (h): Results for learned trajectories without fine-tuning. (c), (f), (i): Results for fine-tuned trajectories.