MBMF Combination Via Value Expansion

Le Chen[*]
D-ITET
ETH Zurich
lechen@ethz.ch

Yunke Ao[*]
D-MAVT
ETH Zurich
yunkao@ethz.ch

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 [1]. 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 sensor [2, 3]. The performance of VI systems is highly dependent on the quality and accuracy of the calculated calibration parameters [4]. Precise calibrations are usually obtained offline in controlled environments, following sophisticated motion routines ensuring observability [5, 6]. This makes the entire process non-trivial for an inexperienced operator [4, 7]. 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.

In previous work, we introduced a novel framework that uses model-based deep RL with an adapted version of particle swarm optimization (PSO) for sampling, to generate trajectories for efficiently collecting measurements to calibrate both camera intrinsic and VI extrinsic parameters. While the proposed model-based RL framework is able to achieve good performance in our Markov decision process (MDP) model for VI calibration with short horizons, an interesting improvement is to integrate our approach with model-free learners. When solving problems with a longer horizon, this would help to overcome potential shortcomings, including vanishing gradients and a too large dimensional space for optimization.

Combining model-based RL with model-free learning has been investigated by [8, 9], where learned models are utilized for improving value learning through Model-based Value Expansion (MVE). In our framework, learned models are further used to speed up policy learning progress, by introducing model predictive control (MPC) to make decisions based on the model and enforcing the policy to imitate the MPC policy at the beginning of training. Additionally, the number of planning steps in MPC is reduced gradually as the learned value functions have less and less error. In this way, the final decision process is transformed into a model-free case which always performs not worse than original model-free methods.

2 MDP Model for Visual-Inertial Calibration

An overview of the previous 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.

2.1 Visual-Inertial Calibration

Our estimator follows the Kalibr framework [5, 6], where the Levenberg-Marquardt algorithm is applied to minimize the loss between the obtained and predicted measurements to maximize the

[*]equal contribution

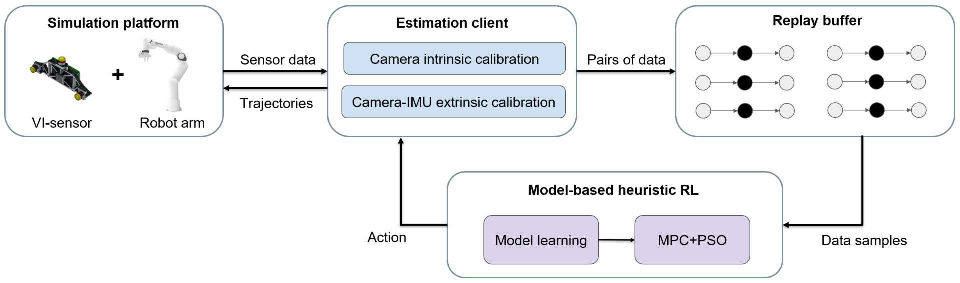

Figure 1: Overview of our previous calibration framework. The trajectories (actions) the agent chooses to take and their simulated calibration performance are recorded in real-time. This data is then used to train the agent with model-based RL.

likelihood of the unknown parameters $Pr(X, \theta|Z, L)$. Here, X is the estimated pose trajectory, θ depicts the calibration parameters, Z is the measurements of VI sensors and L is the known position of landmarks on the calibration target. As explained in detail in [10], 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 $\Sigma_{X\theta}$ and further normalized to $\overline{\Sigma}_\theta$. The information gain can then be evaluated with the following metrics in [10]. Minimizing the H metrics in [10] leads to the maximization of information gain and furthermore can be used for the reward design of our proposed method.

2.2 MDP Model for Learning to Calibrate

The whole calibration process is modeled as a MDP. The process description includes a state S_t, action A_t, transition model $Pr(S_{t+1}|S_t, A_t)$, 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 can not 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 $\{\sum_{q=1,2,4} \boldsymbol{a}_q(1-\cos\frac{2q\pi j}{J}) + \boldsymbol{b}_q \sin\frac{2q\pi j}{J}\}_{j=1:J}$, where J is the number of waypoints inside one action. $\boldsymbol{a}_q$ and $\boldsymbol{b}_q$ are 6×1 vectors. Thus, for one trajectory 36 parameters are needed, which is the action space dimension of MDP. The reason to choose $\sin$ and $\cos$ as basis functions is that they are capable of representing many good empirical trajectories for calibration.

The state S_t contains all measurements acquired so far and the current calibration parameters obtained from the full trajectory. Based on this the next action and state transition would be determined entirely from the current state. 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 $Y_t = [\theta_t^*, O_t]^T = Cal(\bigcup_{i=0}^{t-1} D_i)$ and $S_t = Y_t \cup \bigcup_{i=0}^{t-1} D_i$, where Cal represents the calibration process that returns the predicted calibration parameters θ^* and their respective information gain status O_t. In this way, the state transition satisfies the Markov property and the transition model becomes

$$S_{t+1} = Y_{t+1} \cup (\bigcup_{i=0}^{t-1} D_i) \cup D_t = (S_t - Y_t) \cup h(A_t) \cup Cal((S_t - Y_t) \cup h(A_t)), \tag{1}$$

where h represents a mapping from the trajectory parameters to the acquired measurements.

For camera calibration O_t contains the progress of coverage of horizontal axis, 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.

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 of the sensor. 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. 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 $\eta_1, \cdots, \eta_4$ are positive weights that can be manually tuned separately.

3 MBMF Reinforcement Learning

3.1 Overview

The whole framework is shown in Figure 2. At first, a MPC controller gives control policy for interactions with the environment based on the currently learned model. The data generated during interaction is composed of planned trajectories from the open-loop planning module of MPC and executed trajectories. They will be utilized for value learning, policy learning, and model learning in the learning module. Finally, the updated value function and model will be taken by the MPC controller for interaction again.

As the times of looping increase, the value function, as well as the policy, are getting more precise, and the number of planning steps of the MPC controller will be reduced gradually to zero following a pre-defined rule. In this way, finally, the whole training and interaction framework becomes a classical model-free framework with only value learning and policy learning, which guarantees a comparable final performance with pure model-free methods.

The whole algorithm is based on two assumptions:

1. The value functions of the states closer to the terminal states (or reach the maximum number of steps) are well-learned earlier.
2. Modeling learning is easier than value learning, which means the models are well-learned much earlier than value functions.

Here ”well-learned” means sufficiently close to the true value.

3.2 MPC for Interaction

In a traditional MPC framework, the open-loop planning step is solving a constrained optimization problem:

$$\begin{aligned} min \quad & \sum_{t=0}^{H-1} I(s_t, a_t) + I_f(s_H) \\ s.t. \quad & s_{t+1} = f(s_t, a_t), \quad t = 0, 1, \cdots, H-1 \\ & s_t \in \mathcal{S}, a_t \in \mathcal{A}, \quad t = 0, 1, \cdots, H-1 \\ & s_H \in \mathcal{S}_f \end{aligned}$$

Where $I(s_t, a_t)$ denotes the stage cost, $I_f(s_H)$ is the terminal cost that approximate the ”cost to go” after step H. $f(s_t, a_t)$ is the dynamic model. $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{S}_f$ are constraints for states and actions.

In our framework, the definition of the value function is similar to the terminal cost of MPC. Therefore,

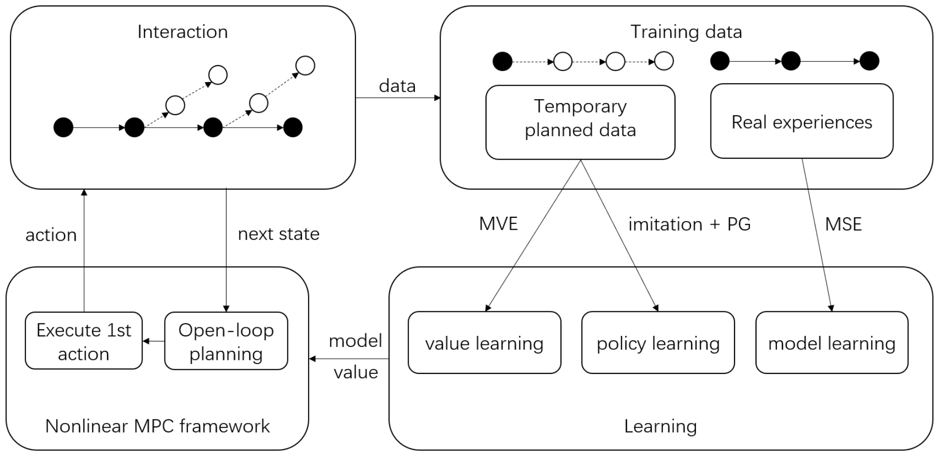

Figure 2: Overview of our methods. It consists of four parts: interaction, training data generation, learning and MPC framework.

Given learned reward model $\hat{g}_\beta$, dynamic model $\hat{f}_\alpha$, the value function $\hat{Q}_\theta$ and the policy μ_ϕ, the open-loop planning optimization actually solved in our case is:

$$\begin{aligned} min \quad & \sum_{t=0}^{H-1} \gamma^t \hat{g}_\beta(s_t, a_t) + \gamma^H \hat{Q}_\theta(s_H, \mu_{\phi'}(s_H)) && (4) \\ s.t. \quad & s_{t+1} = \hat{f}_\alpha(s_t, a_t), \quad t = 0, 1, \cdots, H-1 && (5) \\ & s_t \in \mathcal{S}, a_t \in \mathcal{A}, \quad t = 0, 1, \cdots, H-1 && (6) \\ & s_H \in \mathcal{S}_f && (7) \end{aligned}$$

where γ is the discount factor and ϕ' is the parameters of target policy network. This non-linear optimization problem can be solved by the iterative LQR or heuristic algorithms.

(***Remarks:*** *We need to find an appropriate algorithm here*). After solving this, the controller will only take the first action of the planned trajectory and then plan again at the new state. Some random noise could be imposed to the chosen action to encourage exploration.

3.3 Training

Different from the model-free algorithms, there are two classes of data to train the RL agent: imagined data (which we obtain from the planned trajectories) and interaction experiences (from the physical or simulated environment in the interaction step). They will be separated to train the value, policy, and model networks based on the above assumptions.

Neural network dynamics and reward functions: Dynamic and reward models are always trained using real experiences. We parameterize both the dynamic model and reward model as neural networks: $\hat{S}_{t+1} = f_\alpha(S_t, A_t)$, $\hat{R}_t = g_\beta(S_t, A_t)$. Given a batch of training data $\{S_t^i, A_t^i, S_{t+1}^i, R_t^i\}_{i=1}^N$, both models are trained using stochastic gradient decent (SGD) to minimizing the mean squared dynamic and reward errors:

$$\epsilon_{dyn} = \frac{1}{N} \sum_{i=1}^N \|S_{t+1}^i - \hat{S}_{t+1}^i\|^2, \tag{8}$$

$$\epsilon_{reward} = \frac{1}{N} \sum_{i=1}^N \|R_t^i - \hat{R}_t^i\|^2 \tag{9}$$

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

Learning value function with MVE: Based on assumption 2, suppose the learned model is already accurate, the Q-value functions can be trained, given an imaginary trajectory $\{s_{t'}, a_{t'}, r_{t'}\}_{t'=t}^{t+H}$ extended from a single pair of data $\{s_t, a_t, s_{t+1}, r_t\}$, with MVE loss:

$$\sum_{t=0}^{H-1} \|Q_\theta(s_t, a_t) - \sum_{\tau=t}^{H-1} \gamma^{\tau-t} g_\phi(s_\tau, a_\tau) - \gamma^{H-t} Q_\phi(s_H, a_H)\|^2 \tag{10}$$

where $a_H = \mu_\phi(s_H)$. (***Remarks:*** *Are there other ways to train Q using interaction data? For example, labeling the interaction experience according to which step the state is at, and only train the Q function with data pairs of certain labels.*) By comparison the traditional Q-learning algorithm minimize the one-step TD loss:

$$\|r_t + \gamma Q_\theta(s_{t+1}, \mu_\phi(s_{t+1})) - Q_\theta(s_t, a_t)\|^2 \tag{11}$$

This is applied in model-free algorithms such as Deep Deterministic Policy Gradient (DDPG).

Learning policy through imitation and policy gradient: We gradually reduce the number of planning steps according to the accuracy of Q-learning. So when the number of planning steps is H, we assume the $Q_\theta(s_H, \mu_\phi(s_H))$ is close to the true value for all planned trajectories. Since the deterministic policy gradient depends on the learned Q-function for deep Deterministic Policy Gradient (DPG), we only apply DPG on those more accurately learned Q-functions.

Given a planned trajectory $\{s_t, r_t, a_t\}_{t=1}^{H}$, the policy will be trained twice: once is to imitate the MPC policy for the beginning step, and the other once is to apply policy gradient with the final Q-function:

$$\phi \leftarrow \phi - \nabla_\phi \|\mu_\phi(s_1) - a_1\|^2 \tag{12}$$

$$\phi \leftarrow \phi + \nabla_\phi \mu_\phi(s_H) \nabla_a Q_\theta(s_H, a)|_{a=\mu_\phi(s_H)} \tag{13}$$

(***Remarks:*** *Again, can we utilize interaction data here? For example, labeling the number of steps of states and only using the data of certain labels to train.*)

3.4 Automatic Transformation From Model-based RL to Model-free RL

Following assumption 1, we can reduce the number of steps intuitively based on the accuracy of Q-functions. (***Remarks:*** *Are there any other good ways?*). To make everything easier to be tracked using time steps, we first extend the length of all episodes to be same. In order to achieve this, even if an agent has arrived at a terminal state, we assume that it continuously stays at the terminal state until a maximum time step is reached. Here assume that our maximum number of time step is T, and the initial maximum number of planning is H (ideally it is equal to T but may be less considering the computation cost).

As the number of time steps is fixed for all episodes (the trajectories of which are described by $\{s_0^i, a_0^i, r_0^i, s_1^i, a_1^i, r_1^i, \cdots, s_{T-1}^i, a_{T-1}^i, r_{T-1}^i, s_T^i\}_{i=1}^N$), we can divide the interaction process into model-free part and model-based part by time steps, for example, letting the last K steps to be model-free and first T-K steps to be model based. This means, for interaction, we will apply MPC in first T-K steps and only use model free policy in last K steps. For training, we train value functions using MVE (10) and train policy using imitation (12) for recorded data tuples in first T-K steps $(s_t^i, a_t^i, s_{t+1}^i, r_t^i), t \leq T-K$, while for data from last K steps $(s_t^i, a_t^i, s_{t+1}^i, r_t^i), t > T-K$ we apply model-free Q-learning and policy gradient (13).

Now the only question remains is, how can we increase K from 0 to T as training goes on. The general ideas is, the value functions of states in model-free part should be well-learned enough so that applying policy gradient based on Q-value is more effective. Assume at some time the model free part is the last K steps of all episodes (model based part is T-K). Then we can assume that Q-values of all states at time steps higher than $T-K$ are well learned. The model-free part will only be expanded to the last K+1 steps when

$$Q_\theta(s_{T-K}^i, a_{T-K}^i) \approx r_{T-K}^i + \gamma Q_\theta(s_{T-K+1}^i, \mu_\phi(s_{T-K+1}^i)) \tag{14}$$

is mostly satisfied for tuples $\{s_{T-K}^i, a_{T-K}^i, s_{T-K+1}^i, r_{T-K+1}\}_{i=1}^N$ in the recorded interaction experiences. This means the Q-values of states at time step $T-K$ is also well-learned. As this proceeds, finally when $K = T$, the whole learning and interaction process is reduced to a model-free case, which guarantees a comparable final performance with model-free algorithms.

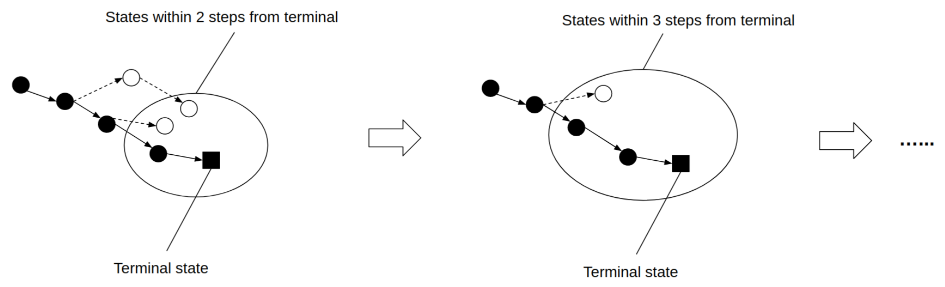
States within 2 steps from terminal
Terminal state
States within 3 steps from terminal
Terminal state
......

Figure 3: Number of Planning Steps for MPC in Interaction

The last thing to note is how to determine the number of planning step for both MPC for interaction and imaginary extension of data pairs for training. As is shown in figure 3, the planning will be stopped once it reaches the model-free part, because the future cost to go could be approximated using the value function of the last state with high confidence. More formally, the number of planning steps (imaginary extension) H_t at some state s_t at time step t is determined by:

$$H_t = \begin{cases} H, & t+H \leq T-K \\ T-K+1-t, & t+H > T-K, t \leq T-K \\ 0, & t > T-K \end{cases} \tag{15}$$

3.5 Entire Algorithm

Algorithm 1 Model-based Model-free Reinforcement Learning

```
1: //Initialization:
2: set maximum time step $T$, maximum planning steps $H$.
3: set automatic transformation accuracy criteria $c_1$ and proportion criteria $c_2$
4: initialize models $Q_\theta$, $\mu_\phi$, $g_\beta$, $f_\alpha$
5: initialize model-free, model-based training datasets $D_f$, $D_b$
6: initialize number of model-free time steps $K = 0$.
7: gather random (expert) trajectories, train $g_\beta$, $f_\alpha$
8:
9: for each episode do
10:     //Interaction:
11:     reset the state and environment
12:     for t=0 to $T$ do
13:         get current state $s_t$
14:         if $t \leq T - K$ then                                    ▷ model-based interaction
15:             compute $H_t$ with (15)
16:             solve (4), get open-looped sequence $a^*_{t:t+H_t}$
17:             apply $a_t = a^*_t$, receive $r_t$, arrive at $s_{t+1}$
18:             record $(t, s_t, a_t, r_t, s_{t+1})$ in $D_b$        ▷ If add noise, also record action without noise?
19:         else                                                     ▷ model-free interaction
20:             apply $a_t = \mu_\phi(s_t)$, receive $r_t$, arrive at $s_{t+1}$
21:             record $(t, s_t, a_t, r_t, s_{t+1})$ in $D_f$
22:         end if
23:     end for
24:
25:     //Training: (could be done once in several episodes)
26:     train $g_\psi$ and $f_\phi$ by applying SGD to (8)-(9) with data sampled from $D_f \cup D_b$
27:     randomly sample training batch $\{d_i\}_{i=0}^N$ from $D_b$           ▷ model-based training
28:     for i=0 to $N$ do
29:         extract $d_i = (t^i, s^i_{t^i}, a^i_{t^i}, s^i_{t^i+1}, r^i_{t^i})$
30:         compute $H_{t^i}$ with (15)
31:         use $g_\beta$, $f_\alpha$ to extend $d_i$ to $(s^i_{t'}, a^i_{t'}, r^i_{t'})_{t'=t^i}^{t^i+H_{t^i}}$      ▷ Here use which policy? MPC or learned
32:         set $d'_i = (s^i_{t'}, a^i_{t'}, r^i_{t'})_{t'=t^i}^{t^i+H_{t^i}}$
33:     end for
34:     train $Q_\theta$ with $\{d'_i\}_{i=0}^N$ by applying (10)
35:     train $\mu_\phi$ with $\{d'_i\}_{i=0}^N$ by applying (12)-(13)
36:     randomly sample training batch $\{d^f_i\}_{i=0}^N$ from $D_f$           ▷ model-free training
37:     train $Q_\theta$ with $\{d^f_i\}_{i=0}^N$ by applying (11)
38:     train $\mu_\phi$ with $\{d^f_i\}_{i=0}^N$ by applying (13)
39:
40:     //Automatic transformation: (could be done once in several episodes)
41:     extract $\{s^i_{T-K}, a^i_{T-K}, s^i_{T-K+1}, r_{T-K+1}\}_{i=1}^N$ from $D_b$
42:     if $c_2$ of $\{s^i_{T-K}, a^i_{T-K}, s^i_{T-K+1}, r_{T-K+1}\}_{i=1}^N$ satisfy (14) within accuracy of $c_1$ then
43:         remove $\{s^i_{T-K}, a^i_{T-K}, s^i_{T-K+1}, r_{T-K+1}\}_{i=1}^N$ from $D_b$ to $D_f$
44:         $K+ = 1$
45:     end if
46: end for
47: return $Q_\theta$, $\mu_\phi$, $g_\beta$, $f_\alpha$
```

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, 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.