RISE: Real-Time Iteration Scheme for Estimation applied to Visual-Inertial Odometry

Philipp Foehn, Davide Scaramuzza

***Abstract*—Real-time state estimation is the cornerstone for autonomous mobile robots, with visual-inertial odometry being the prominent solution for size and weight constrained platforms. Being based on maximum-likelihood Bayesian-inference through optimization, such estimators use complex non-linear (e.g. visual and inertial) measurement modalities, which are computationally intense. To render the problem computationally tractable on constrained platforms, approximations along two dimensions are applied: (i) measurement sparsification through extraction of visual features and preintegrated inertial measurements, and (ii) temporal sparsification through marginalization up to a sliding-window of measurements, or even up to the latest measurement in the extreme case of a filter-based approach. We propose to use a further approximation exploiting the iterative convergence and similarity of subsequent sliding windows, where most of the optimization variables are already close to optimal or accurately initialized. Our proposed method differs from existing approaches by utilizing each iteration result as successively refined state estimate without waiting for convergence, resulting in a real-time iteration scheme for estimation (RISE). This enables three main advantages: (i) reaction latency to new observations is drastically reduced, (ii) the provided estimates are as accurate as possible given the computation budget, (iii) measurements can be incorporated at high frequency without waiting for convergence. In the context of a VIO pipeline, our preliminary results show an average RMSE reduction of 20%, while reducing the latency by roughly a factor of 10x, allowing more than 100fps throughput.**

I. Introduction

State estimation is the cornerstone for autonomous mobile robots such as drones, which often underlie size, weight, cost, and environmental constraints, necessitating small, independent sense and compute solutions.

Meanwhile, visual-inertial odometry (VIO) has become the state-of-the-art onboard localization solution for mobile robots, enabled by its complementary measurement nature, and attributed to its size, weight, and cost-effectiveness. However, the problem of fusing these two measurement modalities is highly non-linear and requires a considerable amount of computational resources, which mobile (aerial) platforms struggle to deliver [1].

To deploy such Bayesian methods efficiently on size and weight constrained mobile robots, many approaches make severe simplifications. In the context of VIO pipelines, these simplifications mainly come in two flavors: (i) measurement sparsification, such as the compression of visual images into feature tracks or the preintegration of IMU measurements, and (ii) temporal sparsification reducing global problems into sliding-window formulations through marginalization. In the extreme case of marginalizing down to single state-pairs, one arrives at filter-based formulations such as the MSCKF[2],

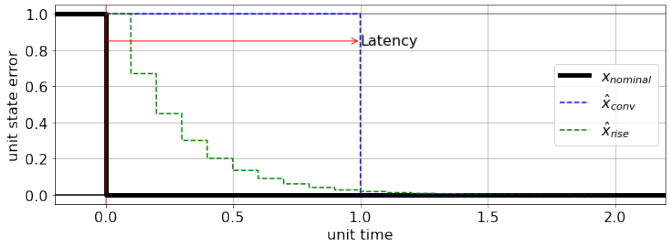

Fig. 1: Visualized comparison of a typical solver iterating up to convergence introducing latency $\hat{x}_{conv}$, and our proposed RISE method $\hat{x}_{rise}$. Note that in the case of solving up to convergence, a new estimate becomes available with significant latency, while RISE provides updated estimates at high rates, iteratively converging towards zero error.

which are computationally lean and efficient. However, spanning a sliding-window over multiple measurement periods often enhances robustness, accuracy, and versatility in describing complex non-linear measurement modalities. This lead to the popular field of sliding-window VIO algorithms, such as [3, 4], and frameworks for modelling such problems, like [5, 6].

Unfortunately, optimization-based sliding-window approaches for VIO quickly outgrow real-time applicability by their computational requirements. In fact, there is large disconnect between the optimization, typically reaching around $30\,\mathrm{fps}$ on modern mobile compute architectures [1], and the possible sensor rates which easily exceed $100\,\mathrm{fps}$ for cameras and $1\,\mathrm{kHz}$ for IMUs. Especially under Gaussian noise assumptions, higher measurement frequency allows for better noise suppression and overall better estimation performance.

On the other hand, many real-time capable approaches often deploy KLT tracking [7], which benefits from higher frame rates, due to the lower apparent motion. To take full advantage of this, the sliding-window optimization needs to be capable to process the high rate feature tracks, e.g. given by [8].

Interestingly, in the field of model predictive control (MPC), similar problems are met when computing optimal control inputs. State-of-the-art approaches to optimization-based MPC apply a real-time iteration scheme [9], updating control inputs and initial state between each iteration, instead of only after convergence. The intention behind this scheme is that the iterations converge faster than the systems states change, where the problems contractivity can be proven [9].

We propose to use a similar scheme for non-linear optimization-based estimation, where we exploit the fact that

each new measurement only changes a subset of the states and residuals in a sliding window. Therefore, successive windows are similar, and given an initial guess for the changed subsets, iterations can converge extremely fast. Furthermore, each iteration can be verified by checking the reduction in the quadratic cost, allowing to reject unsuccessful iterations. Finally, this enables the retrieval of refined estimates after each iteration, and updating the window with new measurements as soon as they arrive, rather than after convergence. In contrast to filter-based method, it does neither require measurements to be added in temporal order, nor fix the linearization point of measurements within the sliding window, and allows multiple concurrent measurement residuals over temporally distributed states (the sliding-window) to be optimized simultaneously. In contrast to existing optimization-based methods, the real-time iteration scheme drastically reduces observation-to-reaction latency and enables incrementally adding new measurements as they arrive, without having to batch multiple measurements that are held up by waiting for convergence. The later point minimizes the subset of exchanged state variables in a sliding-window, resulting in good initialization and fast convergence.

Our real-time iteration scheme for estimation (RISE) enables three main advantages:

1) reduced reaction latency to new observations,
2) the provided estimates are as accurate as possible given the computation budget,
3) measurements can be incorporated at high frequency without waiting for convergence.

Our preliminary experiments indicate the effectiveness of this approach by reducing the positional tracking RMSE by 20% over up-to-convergence solving, reducing latency by a factor of $10\times$, and allowing to incorporate frame measurements at up 100fps.

II. Methodology

A. Nomenclature

We write scalars as s, vectors as $\boldsymbol{v}$ and matrices as $\boldsymbol{M}$. We denote the inertial frame by I, body frame B, and camera frame C. To write the translation from frame B to C expressed in I we use ${}_I\boldsymbol{t}_{BC}$, while a rotation matrix $\boldsymbol{R}_{BC} = \boldsymbol{R}(\boldsymbol{q}_{BC})$ accompanies its quaternion description $\boldsymbol{q}_{BC}$. Additionally, for the quaternion conjugate we use $\bar{\boldsymbol{q}}$, so that $\boldsymbol{R}^{-1}(\boldsymbol{q}) = \boldsymbol{R}(\bar{\boldsymbol{q}})$. For readability, a quantity expressed in its origin frame drops the first frame index $\boldsymbol{p}_{IB} = {}_I\boldsymbol{p}_{IB}$, and relations between inertial and body frame drop all indices, as in $\boldsymbol{p} = {}_I\boldsymbol{p}_{IB}$.

B. Visual-Inertial Estimation as Non-Linear Optimization

The VIO problem in its basic description can be summarized as finding the optimal state $\boldsymbol{x}^*$ that minimizes a weighted quadratic cost $\chi^2 = \|\boldsymbol{e}(\boldsymbol{x})\|^2_{\boldsymbol{W}}$ with weight matrix $\boldsymbol{W}$ on the residuals $\boldsymbol{e}(\boldsymbol{x}) = \boldsymbol{y}(\boldsymbol{x}) - \hat{\boldsymbol{y}}$, where $\hat{\boldsymbol{y}}$ is the measurement:

$$\boldsymbol{x}^* = \min_{\boldsymbol{x}} \chi^2(\boldsymbol{e}(\boldsymbol{x})) = \min_{\boldsymbol{x}} \|\boldsymbol{e}(\boldsymbol{x})\|^2_{\boldsymbol{W}} \tag{1}$$

which is equal to the maximum a posterior estimate when weighted with the inverse covariance $\boldsymbol{W} = \boldsymbol{\Sigma}^{-1}$ under Gaussian noise assumptions.

In the context of VIO, the residuals are typically comprised of visual factors $\boldsymbol{e}_V$, inertial factors $\boldsymbol{e}_I$, and prior factors $\boldsymbol{e}_P$, so that we can rewrite the problem as

$$\boldsymbol{x}^* = \min_{\boldsymbol{x}} \boldsymbol{e}_P^\intercal \boldsymbol{W}_P \boldsymbol{e}_P + \boldsymbol{e}_V^\intercal \boldsymbol{W}_V \boldsymbol{e}_V + \boldsymbol{e}_I^\intercal \boldsymbol{W}_I \boldsymbol{e}_I \tag{2}$$

In the following sections we introduce the state space descriptions and all residual formulations.

1) State Space: To formulate the VIO sliding window problem, we first define the the IMU state at time t_i as

$$\boldsymbol{x}_i = \left[\boldsymbol{p}_i, \boldsymbol{q}_i, \boldsymbol{v}_i, \boldsymbol{b}_{ai}, \boldsymbol{b}_{\omega i}\right] \tag{3}$$

with the position $\boldsymbol{p}_i$, orientation $\boldsymbol{q}_i$, velocity $\boldsymbol{v}_i$ and accelerometer and gyroscope bias $\boldsymbol{b}_{ai}$ and $\boldsymbol{b}_{\omega i}$. The orientation is further decomposed by $\boldsymbol{q}_{IB} = \boldsymbol{q}_{IR} * \boldsymbol{q}_{RB}$ where R is a local reference frame. Therefore, we can use a minimal quaternion representation $\boldsymbol{q}_{RB} = \left[\sqrt{1 - \boldsymbol{q}_x^\intercal \boldsymbol{q}_x}, \boldsymbol{q}_x^\intercal\right]^\intercal$ with $\boldsymbol{q}_x \in \mathcal{R}^3$ and linearize the state by $\boldsymbol{q}_{IR} \Longleftarrow \boldsymbol{q}_{IR} \cdot \boldsymbol{q}_{RB}$ and $\boldsymbol{q}_{RB} \Longleftarrow \begin{bmatrix}1 & \boldsymbol{0}\end{bmatrix}^\intercal$.

Together with the visual features parameterized by their position $\boldsymbol{l}_j \in \mathcal{R}^3$, the full state space $\boldsymbol{x}$ becomes

$$\boldsymbol{x} = [\boldsymbol{x}_0, ..., \boldsymbol{x}_{N-1}, \boldsymbol{l}_0, \ldots, \boldsymbol{l}_{M-1}] \tag{4}$$

with a sliding window of N states and M features.

2) Projection Measurements: The camera is described by a pinhole model characterized by projection matrix $\boldsymbol{K}$, and the camera to body transformation $\boldsymbol{t}_{BC}$, $\boldsymbol{q}_{BC}$. The projection of a feature $\boldsymbol{l}$ into image coordinates $\boldsymbol{s}$ follows as

$$\begin{aligned} \boldsymbol{s} &= \begin{bmatrix} 1/t_{CLz} & 0 & 0 \\ 0 & 1/t_{CLz} & 0 \end{bmatrix} \cdot \boldsymbol{K} \cdot \boldsymbol{t}_{CL} \\ \boldsymbol{t}_{CL} &= \boldsymbol{R}_{BC}^{-1}(\boldsymbol{R}_{IB}^{-1}(\boldsymbol{l} - \boldsymbol{p}_{IB}) - \boldsymbol{t}_{\mathcal{BC}}) \end{aligned} \tag{5}$$

with $\boldsymbol{t}_{CL}$ the landmark location in camera frame, and t_{CLz} its depth. We can then write the measurement residual for an observation $\hat{\boldsymbol{s}}_{ij}$ of feature j at state i as $\boldsymbol{e}_{ij} = \boldsymbol{s}_{ij} - \hat{\boldsymbol{s}}_{ij}$.

3) Preintegrated IMU Measurements: The acceleration and gyroscope measurements $\hat{\boldsymbol{a}}$, $\hat{\boldsymbol{\omega}}$ are assumed to be biased by $\boldsymbol{b}_a$, $\boldsymbol{b}_g$ and corrupted by zero-mean Gaussian noise $\eta \sim \mathcal{N}(0, \sigma)$, as

$$\begin{aligned} \hat{\boldsymbol{a}} &= \boldsymbol{a} + \boldsymbol{b}_a - \boldsymbol{g} + \boldsymbol{\eta}_a & \dot{\boldsymbol{b}}_a &= \boldsymbol{\eta}_{ba} \\ \hat{\boldsymbol{\omega}} &= \boldsymbol{\omega} + \boldsymbol{b}_\omega + \boldsymbol{\eta}_\omega & \dot{\boldsymbol{b}}_\omega &= \boldsymbol{\eta}_{b\omega} \end{aligned} \tag{6}$$

where the biases $\boldsymbol{b}_a$, $\boldsymbol{b}_\omega$ follow a random walk, and gravity is $\boldsymbol{g} = \begin{bmatrix}0 & 0 & -9.81\text{m/s}^2\end{bmatrix}^\intercal$.

To reduce many IMU samples into one measurement residual between two camera frames $i, i+1$ and their respective states $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}$, we perform IMU preintegration. Multiple IMU samples are collected in factors on position $\boldsymbol{\alpha}_i$, velocity $\boldsymbol{\beta}_i$, and orientation $\boldsymbol{\gamma}_i$, as relative motion constraints expressed in the bodyframe B_i of state $\boldsymbol{x}_i$. This avoids repeated integration of the IMU measurements, and can be precomputed by

$$\begin{aligned} {}_{k+1}\boldsymbol{\alpha}_i &= {}_k\boldsymbol{\alpha}_i + \delta t \cdot {}_k\boldsymbol{\beta}_i + \delta t^2/2 \cdot \boldsymbol{R}({}_k\boldsymbol{\gamma}_i)(\hat{\boldsymbol{a}}_i - \boldsymbol{b}_a) \\ {}_{k+1}\boldsymbol{\beta}_i &= {}_k\boldsymbol{\beta}_i + \delta t \boldsymbol{R}({}_k\boldsymbol{\gamma}_i)(\hat{\boldsymbol{a}}_k - \boldsymbol{b}_{ai}) \\ {}_{k+1}\boldsymbol{\gamma}_i &= {}_k\boldsymbol{\gamma}_i \cdot \begin{bmatrix} 1 \\ \frac{\delta t}{2}(\hat{\boldsymbol{\omega}}_k - \boldsymbol{b}_{gi}) \end{bmatrix} \end{aligned} \tag{7}$$

with the accelerometer and gyroscope measurements $\hat{\boldsymbol{a}}_k$, $\hat{\boldsymbol{\omega}}_k$ respectively, at time t_k and with $\delta t = t_k - t_{k-1}$, initialized as ${}_0\boldsymbol{\alpha}_i = \mathbf{0}$, ${}_0\boldsymbol{\beta}_i = \mathbf{0}$, and ${}_0\boldsymbol{\gamma}_i = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^\intercal$.

The measurement residual between state i and $i+1$ can now be written as

$$\boldsymbol{e}_i = \begin{bmatrix} \boldsymbol{R}_{IBi}^{-1}\left(\boldsymbol{p}_{i+1} - \boldsymbol{p}_i - \delta t \cdot \boldsymbol{v}_i - \delta t^2/2 \cdot \boldsymbol{g}\right) - {}_i\boldsymbol{\alpha} \\ \boldsymbol{R}_{IBi}^{-1}\left(\boldsymbol{v}_{i+1} - \boldsymbol{v}_i - \delta t \cdot \boldsymbol{g}\right) - {}_i\boldsymbol{\beta} \\ 2\left[\bar{\boldsymbol{q}}_i \cdot \boldsymbol{q}_{i+1} \cdot {}_i\bar{\boldsymbol{\gamma}}\right]_{xyz} \\ \boldsymbol{b}_{ai+1} - \boldsymbol{b}_{ai} \\ \boldsymbol{b}_{\omega i+1} - \boldsymbol{b}_{\omega i} \end{bmatrix} \tag{8}$$

where the first three terms correspond to the relative position residual using $\boldsymbol{\alpha}$, velocity residual using $\boldsymbol{\beta}$, orientation residual using $\boldsymbol{\gamma}$, and the last two terms are the constant-expectation of the random-walk biases.

4) Prior Residual and Marginalization: When a new frame arrives we either decide to replace the latest frame or shift the latest frame to add it as a new keyframe in the window. Adding new frames in a fixed size sliding window estimator requires marginalizing the oldest state $\boldsymbol{x}_m = \boldsymbol{x}_0$, and representing its information in the form of a prior residual on states it is related to i.e. $\boldsymbol{x}_r = [\boldsymbol{x}_1, \boldsymbol{l}_0, \ldots, \boldsymbol{l}_{M-1}]$. To eliminate $\boldsymbol{x}_0$, we employ the Schur complement on a sub-system of factors connected to $\boldsymbol{x}_m$:

$$\mathbf{A}_{prior}\Delta\mathbf{x}_r = \mathbf{b}_{prior} \tag{9}$$

$$\text{where} \quad \mathbf{A}_{prior} = \left(\mathbf{A}_{rr} - \mathbf{A}_{rm}\mathbf{A}_{mm}^{-1}\mathbf{A}_{mr}\right) \tag{10}$$

$$\boldsymbol{b}_{prior} = \mathbf{b}_r - \mathbf{A}_{rm}\mathbf{A}_{mm}^{-1}\mathbf{b}_m \tag{11}$$

where $\mathbf{A}_{ij} = \mathbf{J}_i^T\mathbf{W}\mathbf{J}_j$ and $\mathbf{b}_i = \mathbf{J}_i^T\mathbf{W}\mathbf{e}$ involve jacobians with respect to states i and j and the indices ${}_m$ and ${}_r$ stand for the marginalized and related states, respectively. The resulting prior residual on $\boldsymbol{x}_r$ is formulated as:

$$\mathbf{e}_{prior} = \Delta\mathbf{x}_r - \mathbf{A}_{prior}^{-1}\mathbf{b}_{prior} \qquad \mathbf{W}_{prior} = \mathbf{A}_{prior} \tag{12}$$

where $\Delta\mathbf{x}_r$ is the deviation of the state $\mathbf{x}_r$ from its estimate $\mathbf{x}_{r0}$ at the time of marginalization. In our implementation, we maintain sparsity of our problem by dropping priors on features and only keep the prior on $\boldsymbol{x}_1$.

5) Feature Track Selection: The given formulation of the VIO problem is implemented most efficiently when an upper limit on the state and residual size can be set. This implies that the sliding window length and the number of feature tracks is limited, requiring to select a subset $\mathcal{A}$ of active tracks to use as residuals, and a passive (larger) subset $\mathcal{P}$ to consider as replacements if tracks in $\mathcal{A}$ become obsolete. However, track selection has to be done carefully, to guarantee (i) a good distribution of tracks throughout the window, (ii) a minimum number of observations per frame, and (iii) no disconnect between states in the window. Many pipelines apply heuristics for this track selection, which need to be tuned to or just require a massive number of features tracks to have a high probability of good problem conditioning. We employ a simple scheme to compare weighted lengths of feature tracks, and maximize them throughout the sliding window.

Let $\boldsymbol{T} \in \{0,1\}^{M\times N}$ be a binary matrix of size $M \times N$ with M the number of tracks and N the number of sliding window states. Each entry of $\boldsymbol{T}_\mathcal{A}$ (for the $M_\mathcal{A}$ active tracks) and $\boldsymbol{T}_\mathcal{P}$ (for the $M_\mathcal{P}$ passive tracks) indicates whether a feature at e.g. $T_{\mathcal{A}ij} \in \{0,1\}$ is set or not. Thanks to the matrix notation, we can now compute the length $\boldsymbol{\zeta} \in \mathbb{R}^M$ of all feature tracks as $\boldsymbol{\zeta} = \boldsymbol{T}\mathbf{1}$. However, this weights all entries by the vector of ones $\mathbf{1} \in \mathbb{R}^N$, whereas it is desirable to have an uneven weighting over the sliding window, defined by the weight vector $\boldsymbol{w}_\zeta \in \mathbb{R}^N$ as

$$w_{\zeta i} = \eta^i \qquad \forall i \in [0, N) \tag{13}$$

$$\boldsymbol{\zeta}_\mathcal{A} = \boldsymbol{T}_\mathcal{A}\boldsymbol{w}_\zeta \qquad \boldsymbol{\zeta}_\mathcal{B} = \boldsymbol{T}_\mathcal{B}\boldsymbol{w}_\zeta \tag{14}$$

with $\eta \in (0.5, 2)$ a tuning parameter, where $\eta = 1$ is uniform weight, $\eta = 2$ prioritizes recent tracks, and $\eta = 0.5$ prioritizes older tracks.

We compute $\boldsymbol{\zeta}_\mathcal{A}$ and $\boldsymbol{\zeta}_\mathcal{P}$ and then form the matrix of their element-wise difference $\delta\boldsymbol{\zeta} \in \mathbb{R}^{M_\mathcal{A}\times M_\mathcal{P}}$, as

$$\delta\zeta_{ij} = \zeta_{\mathcal{P}j} - \zeta_{\mathcal{A}i} \qquad \forall i \in [0, M_\mathcal{A}), j \in [0, M_\mathcal{P}) \tag{15}$$

where the elements $\delta\zeta_{ij}$ intuitively represent the change in weighted track length, if an active feature i is exchanged with passive feature j. We can now simply search for the biggest elements in $\delta\boldsymbol{\zeta}$ which correspond to the best replace candidates, maximizing the weighted track lengths.

C. Non-Linear Least-Squares Optimization

To minimize the cost $\chi^2(\boldsymbol{e}(\boldsymbol{x}_k))$ and find the optimal value $\boldsymbol{x}_k^*$ of the estimation problem at time t_k, we use a Levenberg-Marquardt solver, which compromises between gradient descent and the Gauss-Newton method. We iterate over ${}^i\boldsymbol{x}_k$ with iteration counter i by ${}^{i+1}\boldsymbol{x}_k = {}^i\boldsymbol{x}_k + {}^i\boldsymbol{h}_k$, where ${}^i\boldsymbol{h}_k$ is the update step according to

$$(\boldsymbol{J}^\intercal\boldsymbol{W}\boldsymbol{J} + \lambda \cdot diag(\boldsymbol{J}^\intercal\boldsymbol{W}\boldsymbol{J}))\boldsymbol{h} = \boldsymbol{J}^\intercal\boldsymbol{W}(\boldsymbol{y}(\boldsymbol{x}) - \hat{\boldsymbol{y}}) \tag{16}$$

with the Jacobian $\boldsymbol{J} = \frac{\partial \boldsymbol{y}(\boldsymbol{x})}{\partial \boldsymbol{x}}$ and measurements $\hat{\boldsymbol{y}}$, where the indices k and i have been dropped for readability. Note that for large values of ${}^i\lambda$, the update step approximates a gradient descent, and for small ${}^i\lambda$ a Gauss-Newton step. Eqn. (16) is repeated until the update step falls below a certain magnitude $\|{}^ih_k\| < \epsilon_h$ or the change in $\delta^i\chi_k^2 < \epsilon_\chi$ becomes small enough and the optimal solution is declared to be $\boldsymbol{x}_k^* = {}^{i_F}\boldsymbol{x}_k$ of the final iteration i_F.

While iterating, ${}^i\lambda$ is refined by some heuristics, often chosen by lowering ${}^{i+1}\lambda = \alpha_l{}^i\lambda$ by a factor α_l when the update step was successful with ${}^{i+1}\chi_k^2 < {}^i\chi_k^2$, and otherwise increasing ${}^{i+1}\lambda = \alpha_u{}^i\lambda$ by a factor α_u.

In most real-time estimation problems, the iterations run until the convergence criteria is met, only after which a new estimate $\boldsymbol{x}_k^* = {}^{i_F}\boldsymbol{x}_k$ becomes available. Note that the computational latency $\delta_c t_k = {}_c^F t_k - {}_c^0 t_k$ between the time when the optimization for $\boldsymbol{x}_k^*$ is started at ${}_c^0 t_k$ and the final solution is available at ${}_c^F t_k$ is the cumulative computation time for all iterations $\delta_c t_k = \sum_{i=0}^{i_F} {}_c^i t_k$. Consequently, all information (i.e. feature tracks) received in the period $[{}_c^0 t_k, {}_c^F t_k)$ is incorporated

in batch after ${}^{F}_{c}t_k$, and the solution process is restarted with the latest solution serving as the initial guess ${}^{0}\boldsymbol{x}_{k+1} = {}^{i_F}\boldsymbol{x}_k$.

When considering consecutive optimization of a problem where the unknown variables change, e.g. when adding a new frame, sliding the window, or adding new feature tracks, only a subset of the optimization variables can be initialized from the previous solution. This becomes prominent when adding data in batch, which typically is the case when the optimization takes longer than the measurement frequency, where many variables are left without or only with an approximate initial guess.

Such a deteriorate initial guess typically leads to requiring more iterations ${}^{F}i_{k+1}$ in the next solution process, and reduces convergence rates and robustness due to poor linearization.

D. Real-Time Iteration Scheme for Estimation

To address the aforementioned problems and increase the real-time capabilities of sliding window estimators, we propose the adaption of the real-time iteration scheme [9], known from the field of model-predictive control. This scheme is intuitively based on the fact that in many optimization problems on real-time systems two consecutive problems are similar and a (large) subset of the variables is optimally constant or can be initialized close to their optimum. In the context of a sliding-window estimator for VIO, this holds well since:

- when adding visual measurements, a frame is inserted or replacing the head state ($\boldsymbol{x}_{N-1}$), which can be initialized using IMU propagation from the last available estimate,
- when marginalizing the last state, remaining states are unaffected,
- when exchanging landmarks $\boldsymbol{l}_j$, an initial guess can be computed through triangulation, if at least two corresponding camera poses have been estimated before.

However, all of these points become poor approximations if data is added in batch, exchanging multiple states of the sliding window at once and degrading the initial guess for those states. On the other hand, when deploying the real-time iteration scheme, rather than waiting for $\delta_c t_k$ until convergence, we retrieve the incrementally improved estimate ${}^{i}\boldsymbol{x}_k$ after the much smaller single-iteration latency $\delta^i_c t_k << \delta_c t_k$, and already incorporate measurements.

Furthermore, an iteration is only accepted if it reduces the square error $\chi^2(\boldsymbol{e}(\boldsymbol{x}))$, therefore guaranteeing improvement over the previous estimate.

Finally, processing higher measurement rates implies a reduction in the difference between the optimal solution of two consecutive problem windows, facilitating linearization and initialization quality of the optimization.

III. Experiments

All experiments are performed on a Jetson TX2, to keep a deployment-ready experiment environment and demonstrate the advantages of RISE on a computationally constrained platform. Both RISE and the baseline are implemented using our RISE framework, where the only change between the baseline *Solver* configuration and the proposed *RISE* configuration is that *Solver* iterates until convergence, while *RISE* allows adding and retrieving data between iterations.

Since our proposed method only considers the backend optimization and not the frontend including feature tracking and other data processing, we base our experiments on simulated feature tracks and IMU data (described in III-A), to exclude any other effect that might affect our evaluation.

We first describe our data generation process, then compare *RISE* and *Solver* with equal data rates and configuration. Finally, we demonstrate improved performance of RISE in two steps: first, by increasing the frame rate, and second by increasing the sliding window-size and and feature track count.

A. Evaluation Data Generation

To evaluate our framework, we generate data a priori. We describe trajectories as 7th-order polynomials in position and compute a sequence of them through multiple waypoints, either arranged in a Figure-8 or randomly distributed on $3\,\mathrm{m}$ intervals, see Fig. 2. Additionally, we vary the trajectory speed and add an overlaid rotation around all axes, which is also represented by the same polynomial formulation and uniformly sampled in $(\theta_{min}, \theta_{max})$

To generate IMU measurements $\hat{\boldsymbol{a}}$ and $\hat{\boldsymbol{\omega}}$, sample the trajectory at $1\,\mathrm{kHz}$ and add a random walk bias and Gaussian noise according to Eq. (6). For the feature tracks, we project features according to Eq. 5, add Gaussian noise and round the values to full integer numbers, since many feature trackers do not allow for sub-pixel accurate tracking. To create features, we uniformly sample a random image coordinate $\boldsymbol{s}$ and a uniform depth $z \in (z_{min}, z_{max})$, compute the feature location, and reproject it in subsequent frames. In reality, features (or other landmarks) are frequently occluded, lost due to appearance changes, and newly detected on the image border that reveals new scenery (along the motion vector). Therefore, we drop randomly picked features with a probability p_{drop} in every frame, and add new features with a probability p_{border} within the image border of width w_{border} facing the motion direction, and with probability $1 - p_{border}$ in the inner image region. The whole process is performed at $300\,\mathrm{Hz}$, from which only a subset is used at test time, to reflect varying frame rates such as $30\,\mathrm{fps}$ and $100\,\mathrm{fps}$ used in our tests. Therefore, the feature-drop probability p_{drop} accumulates dropouts over multiple frames when using frame rates below the nominal $300\,\mathrm{fps}$. This corresponds roughly to the effects on a real system,

Trajectory Noise level	Figure-8 medium	Figure-8 high	Random medium
σ_a $[\mathrm{m/s^2}]$	0.2	0.4	0.2
σ_ω $[\mathrm{rad\,s^{-1}}]$	0.02	0.04	0.02
σ_{ba} $[\mathrm{m/s^2}]$	2e−4	2e−4	2e−4
$\sigma_{b\omega}$ $[\mathrm{rad\,s^{-1}}]$	2e−4	2e−4	2e−4
σ_{proj} [pixel]	0.2	0.4	0.2
p_{drop} [%]	1%	1%	1%
v_{max} $[\mathrm{m\,s^{-1}}]$	$[2, 5, 8, 15]$	$[2, 5, 8, 15]$	$[2, 5, 8, 15]$

TABLE I: Trajectory generation parameters, with noise as standard deviation

(a) Figure-8 (b) Random-2 (c) Random-1

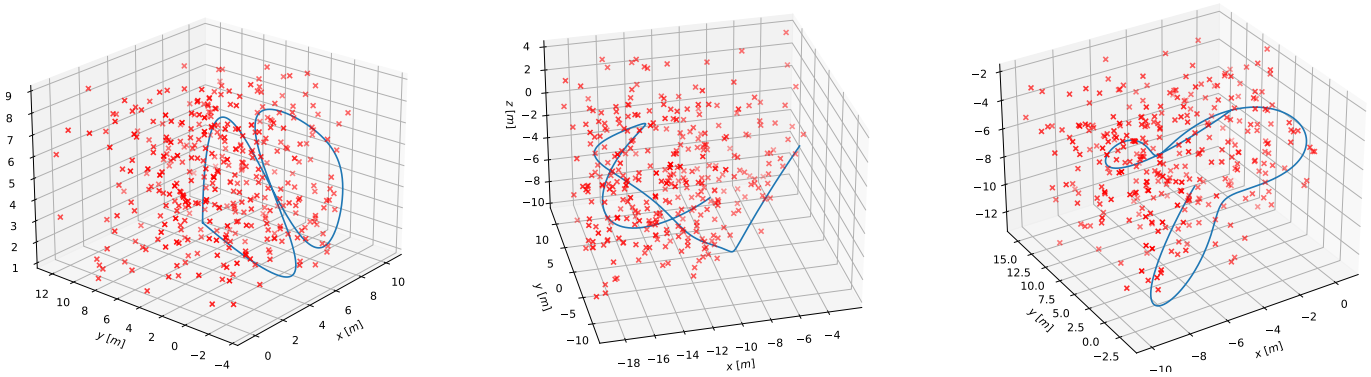

Fig. 2: Generated trajectories

TABLE II: Comparison of *RISE* agains baseline *Solver* in RMSE normalized by trajectory length

Algorithm RMS Error over trajectory length	RISE 10/60 @100fps pos [%]	 rot $[\circ/m]$	RISE 8/40 @100fps pos [%]	 rot $[\circ/m]$	RISE 8/40 @30fps pos [%]	 rot $[\circ/m]$	Solver 8/40 @30fps pos [%]	 rot $[\circ/m]$
Fig-8 $2\,\mathrm{m\,s^{-1}}$ medium noise	0.81	**0.02**	**0.63**	0.02	0.65	0.03	0.85	0.03
Fig-8 $2\,\mathrm{m\,s^{-1}}$ high noise	**0.75**	**0.03**	0.92	0.04	0.77	0.04	0.94	0.05
Fig-8 $5\,\mathrm{m\,s^{-1}}$ medium noise	**0.25**	**0.02**	0.39	0.03	0.44	0.03	0.39	0.03
Fig-8 $5\,\mathrm{m\,s^{-1}}$ high noise	**0.27**	**0.02**	0.48	0.04	0.56	0.04	0.56	0.04
Fig-8 $8\,\mathrm{m\,s^{-1}}$ medium noise	0.38	0.03	**0.34**	0.03	0.41	**0.02**	0.39	0.02
Fig-8 $8\,\mathrm{m\,s^{-1}}$ high noise	**0.44**	**0.03**	0.49	0.03	0.52	0.04	0.69	0.05
Fig-8 $15\,\mathrm{m\,s^{-1}}$ medium noise	**0.50**	0.04	0.55	**0.04**	0.53	0.04	0.58	0.04
Fig-8 $15\,\mathrm{m\,s^{-1}}$ high noise	**0.54**	0.04	0.65	**0.04**	24.86	1.16	27.86	1.13
Random 1 $2\,\mathrm{m\,s^{-1}}$ medium noise	**0.33**	**0.02**	0.49	0.02	0.40	0.02	0.41	0.03
Random 1 $5\,\mathrm{m\,s^{-1}}$ medium noise	**0.20**	**0.01**	0.32	0.02	0.40	0.02	0.33	0.02
Random 1 $8\,\mathrm{m\,s^{-1}}$ medium noise	**0.29**	**0.02**	0.33	0.02	0.33	0.02	0.34	0.02
Random 1 $15\,\mathrm{m\,s^{-1}}$ medium noise	0.44	0.03	**0.41**	0.03	0.53	**0.02**	0.54	0.03
Random 2 $2\,\mathrm{m\,s^{-1}}$ medium noise	1.09	**0.02**	1.22	0.02	**1.00**	0.03	1.05	0.03
Random 2 $5\,\mathrm{m\,s^{-1}}$ medium noise	**0.24**	**0.02**	0.36	0.02	0.40	0.03	0.42	0.03
Random 2 $8\,\mathrm{m\,s^{-1}}$ medium noise	**0.22**	**0.01**	0.31	0.02	0.28	0.02	0.31	0.02
Random 2 $15\,\mathrm{m\,s^{-1}}$ medium noise	**0.35**	**0.02**	0.46	0.03	0.49	0.02	0.50	0.02
Random 3 $2\,\mathrm{m\,s^{-1}}$ medium noise	1.17	**0.02**	0.96	0.02	**0.70**	0.03	1.07	0.03
Random 3 $5\,\mathrm{m\,s^{-1}}$ medium noise	0.29	0.02	0.30	**0.02**	0.31	0.02	**0.27**	0.02
Random 3 $8\,\mathrm{m\,s^{-1}}$ medium noise	**0.24**	0.02	0.29	**0.02**	0.33	0.02	0.44	0.02
Random 3 $15\,\mathrm{m\,s^{-1}}$ medium noise	0.50	**0.03**	**0.44**	0.03	2.31	0.10	2.49	0.11

where at low frame rates larger displacements and appearance changes between two frames lead to more frequent feature track loss. The camera is parameterized with VGA resolution 640×480 at a horizontal field of view of $120°$. Features are generated at the image border of width $w_{border} = 32\,\mathrm{pixel}$ with a probability of $p_{border} = 25\%$. All other parameters can be found in Tab. I.

B. Comparison between RISE and Solver

To establish a baseline, we first execute the *Solver* configuration, running up to convergence but limited to 10 iterations, similar to many real-world applications. We then run the same setup, but with our *RISE* configuration, iterating continuously and reporting each intermediate estimate. Both setups use a sliding window over 8 states and 40 features (indicated by "8/40"), equal noise variance and parameters, and are provided with $30\,\mathrm{fps}$ visual feature tracks and $300\,\mathrm{Hz}$ IMU measurements. The results of both runs on multiple trajectories are depicted in Table II (two rightmost columns), indicating a slight improvement in positional RMSE, for the *RISE* configuration, with mostly equal rotational RMSE. In general, the highest errors occur at low speed with high noise values, where the signal-to-noise ration of the IMU measurements is the worst. Note that one configurations on the Figure-8 trajectory at the highest tested speed of $15\,\mathrm{m\,s^{-1}}$ led to a loss of tracking for both *RISE* and *Solver*.

However, Table III shows a clear advantage in terms of latency, where the first updated estimate can be retrieved within $< 4\,\mathrm{ms}$ for the *RISE* 8/40 configuration. Meanwhile, the *Solver* configuration is mostly capped by the iteration limit after 10 converging steps, increasing the latency to $> 30\,\mathrm{ms}$.

TABLE III: Latency Comparison

Timing	mean [ms]	std [ms^2]	min [ms]	min [ms]
Solver 8/40	31.7	4.77	12.9	53.6
RISE 8/40	3.47	0.842	1.55	5.81
RISE 10/60	6.34	3.01	2.16	9.98

C. Extended Capabilities of RISE

Given the reduced latency of *RISE* and it's capability to keep up well with the $30\,\mathrm{fps}$ in our baseline comparison, we can now increase the measurement frequency and the problem complexity by spanning larger windows an more feature tracks, expecting improved performance as reduced RMSE when doing so.

We first change the measurement rate to provide $100\,\mathrm{fps}$ feature tracks (and $300\,\mathrm{Hz}$ IMU) and report the results in Table II (second column), indicating a slight further reduction in RMSE on both position and rotation, and removes the failure case on the Figure-8 at $15\,\mathrm{m\,s^{-1}}$.

Lastly we increase the problem convexity by expanding to a sliding window over 10 states and 60 feature tracks (indicated as "10/60"), reported in Table II (first column). This further reduces the RMSE, leading to an overall error reduction of $\sim 20\%$. However, this also increases the latency to roughly $\sim 6\,\mathrm{ms}$ as reported in Table III.

IV. Conclusion

We proposed the novel adaption of the real-time iteration scheme for estimation, motivated by the iterative convergence of the maximum-a-posteriori solution of a Bayesian estimation problem. Our work only serves as a proof of concept, and does not yet deploy the approach on real-world systems and data, neither provides contractivity guarantees for the iterative refined estimate. However, preliminary results indicate that it can be deployed on visual-inertial odometry and increase accuracy, latency, and data throughput.

References

[1] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2018.

[2] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2007, pp. 3565–3572.

[3] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial SLAM using nonlinear optimization," *Int. J. Robot. Research*, 2015.

[4] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 1004–1020, 2018.

[5] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Institute of Technology, Tech. Rep. GT-RIM-CP&R-2012-002, Sep. 2012.

[6] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Int. J. Robot. Research*, vol. 31, no. 2, pp. 217–236, Feb. 2012.

[7] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *Int. J. Comput. Vis.*, vol. 56, no. 3, pp. 221–255, 2004.

[8] B. Nagy, P. Foehn, and D. Scaramuzza, "Faster than fast: Gpu-accelerated frontend for high-speed vio," 2020.

[9] M. Diehl, H. Bock, and J. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM J. on Control and Optimization*, 2005.