Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach

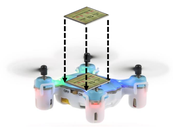
Zhengdong Zhang*, Amr Suleiman*, Luca Carlone, Vivienne Sze, Sertac Karaman
Massachusetts Institute of Technology

Motivation


Consumer Electronics


Search and Rescue

Fully-autonomous navigation without a map is essential to these applications



Goal: Running fully-autonomous navigation without a map LOCALLY

Challenge: **Power and Speed**


Lifting	1 W
Cameras	1 W

Bottle-cap-sized **nano** UAV


	Goal	Desktop CPU	Embedded CPU
Keyframe rate	> 5 fps	**8.4 fps**	**2 fps**
Power	< 2 W	**28.2 W**	**2.5 W**
		Too high power	**Too slow**

General Purpose Computing not good enough!

Low-power Specialized Hardware

FPGA

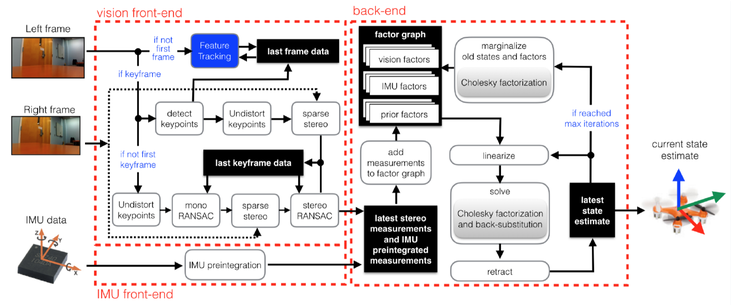

ASIC

	Energy Cost
On-chip compute	1× (Reference)
On-chip Memory: kB – MB	6×
Off-chip Memory: DRAM (GB)	200×

Low power if only use on-chip memory (e.g., **3MB** on FPGA)

Standard VIO algorithms do not fit, we need an **algorithm-and-hardware co-design approach**

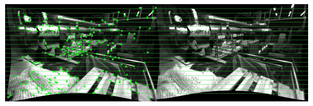
Visual Inertial Odometry (VIO) Algorithm

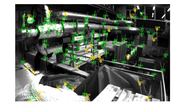


$$\min_{x} \sum_{(i,j)\in\mathcal{F}} \left\| r_{\text{IMU}}\left(x, \Delta\tilde{R}_{ij}, \Delta\tilde{p}_{ij}, \Delta\tilde{v}_{ij}\right) \right\|^2 + \sum_{k\in\mathcal{L}} \sum_{i\in\mathcal{F}_k} \left\| r_{\text{CAM}}\left(x, l_k, u_{ik}^l, u_{ik}^r\right) \right\|^2 + \left\| r_{\text{PRIOR}}(x) \right\|^2$$

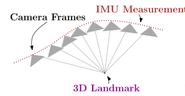
$x_i = (R_i, p_i, v_i, b_i)$: state consisting of robot poses, velocities and IMU bias
$x = \{x_{i-h}, x_{i-h+1}, \ldots, x_i\}$: extended state within the smoothing horizon
$r_{\text{IMU}}\left(x, \Delta\tilde{R}_{ij}, \Delta\tilde{p}_{ij}, \Delta\tilde{v}_{ij}\right)$: negative log-likelihood of the IMU measurements
$r_{\text{CAM}}\left(x, l_k, u_{ik}^l, u_{ik}^r\right)$: negative log-likelihood of the vision measurements

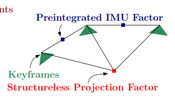
Vision Frontend


Process stereo frame


Robust tracking

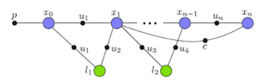
IMU Frontend





IMU Preintegration by Forster, et, al.

Backend

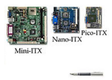


Factor graph based optimization Output trajectory Output 3D point cloud

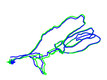
Algorithm-Hardware Co-design

Step 1: Specify Performance and Resource Goals


Battery life, endurance
Power ≤ $2W$



Form factor
Board size, weight


Accuracy
Error ≤ 25 cm


Speed, agility
Keyframe rate ≥ 5 fps

Step 2: Define Design Space, *D*

$$D = H \times A \times I \times P$$

H Hardware choices	A Algorithm choices	I Implementation choices	P Parameter choices
desktop-CPU	Tracking	Streaming	Max feature num
embedded-CPU	RANSAC	Pipelining	Template size
FPGA	Relinearization for Marginalization	Parallelism	Max tracking levels
ASIC		Reduced precision	Intra-keyframe time
		Low cost arithmetic	Nr. GN iterations

Step 3: Explore Design Space

a) Choose a hardware (H):


Desktop CPU ⇨


Embedded CPU ⇨

FPGAs ⇨


ASICs

b) Split $A \times I \times P$ into (A, P_a) and (I, P_h)

1) Choose an algorithm a, and algorithmic parameters p_a in (A, P_a) to minimize **power** , while preserving the **error**

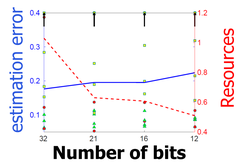
2) Choose an hardware implementation i, and hardware parameters p_h in (I, P_h) to re-establish the **speed**

c) Iterate until finding a feasible design

Iterative Splitting Co-design

Co-designing VIO System on Chip

Reduced Precision of Data Representation



Reduce vision front-end to 16 bits fixed-point for efficient **accuracy vs. memory trade-off**

Hardware Design Choices

$+, \times$ $\div, \sqrt{}$

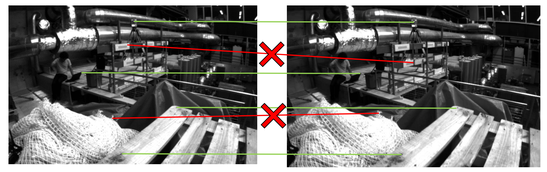
Avoid division and sqrt as much as possible

$\frac{d}{\sqrt{n}} < s$ transform $d^2 < s^2 \times n$

DIV, SQRT MULT

Parallelism and pipelining increase speed, but also increase power/resources. Use carefully!

5-point RANSAC vs 2-point RANSAC



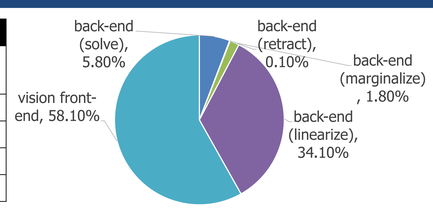
The vision front-end uses mono-RANSAC to remove outliers in the tracking

5-point RANSAC
- Complicated
- Hardware unfriendly
- 145 iterations

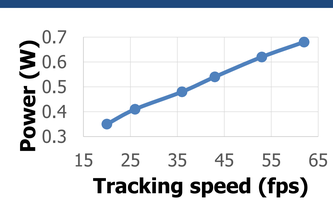
2-point RANSAC (ours)
- Use the rotation from the IMU preintegration
- Hardware friendly
- 16 iterations

Co-design Results

Platform	Design Goal	Baseline		Design (a, p)		Design (h, a, i, p)
		Desktop CPU	Embedded CPU	Desktop CPU	Embedded CPU	FPGA
Accuracy	≤ 0.25	**0.15**		**0.16**		**0.19**
Front-end Throughput (fps)	≥ 20	**15.4**	**3.9**	**20.8**	**5.2**	**20**
Back-end Throughput (fps)	≥ 5	**8.4**	**2.0**	**12.7**	**2.7**	**5**
Power (W)	≤ 2	**28.2**	**2.5**	**26.1**	**2.3**	**1.5**



VIO complexity breakdown: CPU time



Throughput versus power trade-off for the front-end, obtained by sweeping the clock frequency

Resource utilization on Xilinx Kintex-7 XC7K355T FPGA: Only **2.1 MB** memory is needed

Resource	Front-end	Linear Solver	Linearize	Marginalize	Stereo Factors	IMU & Other Factors	State Estimation	Total	Utilization
Memory	1.5 MB	355 KB	0	180 KB	86 KB	15 KB	3 KB	2.1 MB	67.3 %
Block RAM	378	80	0	41	22	4	1	0.5 K	73.5 %
Flip Flops	72 K	5 K	67 K	Shared	n/a	n/a	n/a	144 K	32.4 %
LUTs	111 K	16 K	65 K	Shared	n/a	n/a	n/a	192 K	86.2 %
DSP	607	62	102	Shared	n/a	n/a	n/a	771	53.5 %

navion.mit.edu

[1] Forster, et, al. IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation, RSS 2015
[2] Forster, et, al. On-manifold preintegration theory for fast and accurate visual-inertial navigation. IEEE Trans. Robotics, 2016