
Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze

Massachusetts Institute of Technology

http://navion.mit.edu/
Motivation: Autonomous Navigation

Self Driving Cars

UAVs: Unmanned Aerial Vehicles

Robots

[Images] Electrek, Amazon, Knightscope, Boston Dynamics
How Does Autonomous Navigation Work?

Perception

Motion Planning

Control

Where to Go?
How Does Autonomous Navigation Work?

Perception

Motion Planning

Control

Perception is the computation bottleneck

[Kanellakis et al., JIRS 2017]
Challenges: High Dimensionality

- Large amount of data
  - Sensors data: High resolution & frame rates
  - Data expansion: Image pyramid
Challenges: High Dimensionality

• Large amount of data
  – Sensors data: High resolution & frame rates
  – Data expansion: Image pyramid

• Growing map size

[T. Pire et al., 2017]
Challenges: Low Power Budget

- Big battery
- Mobile CPU, GPU
Challenges: Low Power Budget

For example:

<table>
<thead>
<tr>
<th>Lifting</th>
<th>Cameras</th>
<th>CPU, GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 mW</td>
<td>100 mW</td>
<td>10 - 100 W</td>
</tr>
</tbody>
</table>
Navion: Energy-Efficient Visual-Inertial Odometry

- Energy-efficient & real-time localization and mapping
- Process stereo images at up to 171 fps
- 24 mW average power consumption
Outline

• Localization & Mapping: Visual-Inertial Odometry (VIO)
• Chip Architecture
• Main Contributions
• Chip Specifications and Comparisons
• Summary
Localization and Mapping Using VIO

Image sequence

IMU
Inertial Measurement Unit

Visual-Inertial Odometry (VIO)
Localization and Mapping Using VIO

Image sequence → Visual-Inertial Odometry (VIO) → Localization

IMU (Inertial Measurement Unit) → VIO

Mapping
Localization and Mapping Using VIO

Visual-Inertial Odometry (VIO)

Image sequence

IMU
Inertial Measurement Unit

Localization

Mapping

Subset of SLAM algorithms
(Simultaneous Localization And Mapping)
VIO: Frontend

Process mono/stereo Images
Process mono/stereo Images
- Detect & track features ($L_i$)

VIO: Frontend

Camera

Vision Frontend (VFE)

KF_1  KF_2  KF_3  KF_4

Stereo Images

KF: Keyframe
**VIO: Frontend**

**Process mono/stereo Images**
- Detect & track features ($L_i$)
- Generate *Feature Tracks* $\rightarrow$ (keyframe IDs & feature coordinates)
VIO: Frontend

- Camera
- Vision Frontend (VFE)
- Feature Tracks
- IMU Frontend (IFE)
- IMU

VFE:
- Measurements
- Gyro. & Acc. Measurements
- Stereo Images
- KF: Keyframe
- KF
- Feature Tracks
- Stereo Images
VIO: Frontend

**Camera**

**Vision Frontend (VFE)**

**Feature Tracks**

**Estimated States**

**IMU Frontend (IFE)**

**IMU Measurements**

**Gyro. & Acc. Measurements**

**Stereo Images**

**Feature Tracks**

**Estimated States**

**IMU Measurements**

**Gyro. & Acc. Measurements**

**KF: Keyframe**
**VIO: Frontend**

- **Camera**
- **Vision Frontend (VFE)**
  - Feature Tracks
  - Estimated States
- **IMU Frontend (IFE)**
  - IMU

---

**KF1**, **KF2**, **KF3**, **KF4**

**IMU12**: \( \{ \Delta R_{12}, \Delta T_{12} \} \)

**IMU23**: \( \{ \Delta R_{23}, \Delta T_{23} \} \)

**State**: Pose (Rotation \( R \))

Location (Translation \( T \))

**Gyro. & Acc. Measurements**

---

**Symposia on VLSI Technology and Circuits**
VIO: Backend

- Camera
- IMU
- Vision Frontend (VFE)
- IMU Frontend (IFE)
- Feature Tracks
- Estimated States
- Backend (BE)
Camera

Vision Frontend (VFE)

Feature Tracks

Estimated States

IMU Frontend (IFE)

IMU

Backend (BE)

Update states \( (x_i) \) to minimize inconsistencies between measurements across time
Symposia on VLSI Technology and Circuits

**VIO: Backend**

\[
\min_x \sum_{(i,j) \in F} \left\| r_{IMU}(x, \Delta \tilde{p}_{ij}, \Delta \tilde{v}_{ij}) \right\|^2 + \sum_{k \in \mathcal{E}} \sum_{l \in F_k} \left\| r_{CAM}(x, l_k, u_{ik}, u_{lk}) \right\|^2 + \left\| r_{PRIOR}(x) \right\|^2
\]

- IMU Factors
- Vision Factors
- Other Factors

**Factor Graph**

- Feature Tracks
- Estimated States

- IMU Frontend (IFE)
- Vision Frontend (VFE)
- Camera

**Backend (BE)**

4000+ factors

Horizon at time \( t_k \)

KF_1 \quad KF_2 \quad KF_3
VIO: Backend

\[
\min_x \sum_{(i,j) \in F} \| r_{IMU}(x, \Delta \tilde{r}_{ij}, \Delta \tilde{p}_{ij}, \Delta \tilde{v}_{ij}) \|^2 + \sum_{k \in \mathcal{K}} \sum_{i \in F_k} \| r_{CAM}(x, t_k, u^1_{ik}, u^r_{ik}) \|^2 + \| r_{PRIOR}(x) \|^2
\]

**IMU Factors**  **Vision Factors**  **Other Factors**

**Factor Graph**

Updated States \((x_i)\) & Sparse 3D Map

Camera ➔ Vision Frontend (VFE)

Feature Tracks ➔ Backend (BE) ➔ Estimated States

IMU Frontend (IFE)

IMU

Symposia on VLSI Technology and Circuits  Slide 23
Outline

• Localization & Mapping: Visual-Inertial Odometry (VIO)

• Chip Architecture

• Main Contributions

• Chip Specifications and Comparisons

• Summary
Navion Chip Architecture

Vision Frontend (VFE)
- Line Buffers
- Feature Detection (FD)
- Previous Frame
- Undistort & Rectify (UR)
- Current Frame
- Feature Tracking (FT)
- Left Frame
- Right Frame
- Sparse Stereo (SS)
- Vision Frontend Control
- Data & Control Bus
  - RANSAC
  - Fixed Point Arithmetic
  - Point Cloud

Backend (BE)
- Backend Control
  - Build Graph
  - Linearize
  - Linear Solver
  - Marginal
  - Retract
- Data & Control Bus
  - Graph
  - Linear Solver
  - Horizon States
  - Shared Memory
  - Register File
- IMU Frontend (IFE)
  - Floating Point Arithmetic
  - Pre-Integration
  - IMU memory

No off-chip storage or processing
VFE: All Image Processing

- Fixed point arithmetic
- Parallel/pipelined image processing
- Mono & stereo cameras
- Runs at the sensor rate (up to 171 fps)

- Outputs at keyframe rate: Feature tracks
IFE: IMU Preintegration

- Double precision arithmetic
- Low cost: 2.4% area & 1.2% power
- Runs at the sensor rate (up to 52 kHz)

- Outputs at keyframe rate: Estimated state
BE: Fusing Sensors Data

- Double precision arithmetic
- Complex Finite State Machine (FSM)
- Runs at the keyframe rate (up to 90 fps)

- Outputs at keyframe rate: Updated state & 3D map
VIO Full Integration Challenges

• Vision Frontend (VFE)
  – Heterogeneous computation modules
    • Feature detection
    • Feature tracking
    • Stereo matching
    • Outliers rejection using RANSAC
    • ...
VIO Full Integration Challenges

• Vision Frontend (VFE)
  – Heterogeneous computation modules
    • Feature detection
    • Feature tracking
    • Stereo matching
    • Outliers rejection using RANSAC
    • …

• Backend (BE)
  – High dimensional and complex data structures
    • Large optimization problem (more than 4000 factors)
    • Dynamically changing factor graph
    • High computation precision (64-bit floating point)
Outline

• Localization & Mapping: Visual-Inertial Odometry (VIO)
• Chip Architecture

• Main Contributions
• Chip Specifications and Comparisons
• Summary
Enabling Full Integration

**Vision Frontend (VFE)**
- Line Buffers
  - Feature Detection (FD)
  - Undistort & Rectify (UR)
- Previous Frame
- Current Frame
- Left Frame
- Right Frame
- Sparse Stereo (SS)

**Vision Frontend Control**

**Data & Control Bus**
- Frame buffers: 1,410 kB

**Backend (BE)**
- Backend Control
  - Build Graph
  - Linear Solver
  - Linearize
  - Horizon States
  - Shared Memory
  - Register File
- Graph memory (Feature tracks): 962 kB
- Linear solver memory: 703 kB

**IMU Frontend (IFE)**
- RANSAC
- Fixed Point Arithmetic
- Point Cloud
- Floating Point Arithmetic
- Pre-Integration
- IMU memory

Symposia on VLSI Technology and Circuits
Slide 33
Enabling Full Integration

Frame buffers 1,410 kB

Use compression and exploit sparsity

Symposia on VLSI Technology and Circuits
Method 1
Data Compression
Frame Buffer: Image Compression

- Block-wise Lossy Image Compression

8 bit/pixel

Original
(352.5 kB)

Find Min. & Max.

4x4 pixels example

Compress

1 bit/pixel

Frame Memory

Min.

Max.

≥ 7

>>1

11

4

≥ ?
Frame Buffer: Image Compression

- Block-wise Lossy Image Compression

<table>
<thead>
<tr>
<th>Compress</th>
<th>Decompress</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4 pixels example</td>
<td>1 bit/pixel</td>
</tr>
<tr>
<td>Find Min. &amp; Max.</td>
<td>Frame Memory</td>
</tr>
<tr>
<td>4 6 6 4</td>
<td>4 4 4 4</td>
</tr>
<tr>
<td>6 8 6 5</td>
<td>4 7 4 4</td>
</tr>
<tr>
<td>6 8 10 11</td>
<td>4 7 7 7</td>
</tr>
<tr>
<td>8 5 5 6</td>
<td>7 4 4 4</td>
</tr>
</tbody>
</table>

Min. & Max. | Thresh | >>1 | + |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7</td>
<td>?</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Threshold: 7

8 bit/pixel

Original | Compressed |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(352.5 kB)</td>
<td>(79.4 kB)</td>
</tr>
</tbody>
</table>

1.625 bit/pixel
Frame Buffer: Image Compression

- Block-wise Lossy Image Compression

Lossy Image Compression:
4.4x Memory size reduction

Used only in Feature tracking & Sparse stereo

8 bit/ pixel
Original (352.5 kB)  Compressed (79.4 kB)  1.625 bit/ pixel

Compress 4 6 6 4 1 bit/pixel

Min. 4 Min.

8 bit/ pixel

Symposia on VLSI Technology and Circuits
Method 2
Exploit Sparsity
(Structured & Unstructured)
Linear Solver memory: Structured Sparsity

\[
\min_x \sum_{(i,j) \in E} \|r_{IMU}(x, \Delta R_{ij}, \Delta \tilde{p}_{ij}, \Delta \tilde{v}_{ij})\|^2 + \sum_{k \in L} \sum_{i \in E_k} \|r_{CAM}(x, l_k, u_{ik}^l, u_{ik}^r)\|^2 + \|r_{PRIOR}(x)\|^2
\]

\[
H\delta = \epsilon
\]

Solve a large linear system for \(\delta\)
Linear Solver memory: Structured Sparsity

\[
\min_x \sum_{(i,j) \in F} \| r_{IMU}(x, \Delta \tilde{R}_{ij}, \Delta \tilde{p}_{ij}, \Delta \tilde{v}_{ij}) \|^2 + \sum_{k \in L} \sum_{i \in F_k} \| r_{CAM}(x, l_k, u_{ik}^l, u_{ik}^r) \|^2 + \| r_{PRIOR}(x) \|^2
\]

\[H\delta = \epsilon\]

Solve a large linear system for \(\delta\)

\(H\) is:
- Symmetric

Memory size (kB)

<table>
<thead>
<tr>
<th>Full</th>
<th>Sym</th>
</tr>
</thead>
<tbody>
<tr>
<td>703</td>
<td>353</td>
</tr>
</tbody>
</table>

2x
Linear Solver memory: Structured Sparsity

\[
\min_x \sum_{(i,j) \in E} \| r_{IMU}(x, \Delta \tilde{R}_{ij}, \Delta \tilde{p}_{ij}, \Delta \tilde{v}_{ij}) \|^2 + \sum_{k \in L} \sum_{i \in F_k} \| r_{CAM}(x, l_k, u_{ik}, u_{ik}^T) \|^2 + \| r_{PRIOR}(x) \|^2
\]

\[H\delta = \bar{\epsilon}\]

Solve a large linear system for \(\delta\)

**H is:**
- Symmetric
- Sparse
  (Black: non zero)

Memory size (kB):
- Full: 703
- Sym: 353
- Sym + Sparse: 134

5.2x reduction in memory size.
Linear Solver memory: Structured Sparsity

\[
\min_x \sum_{(i,j) \in F} \| r_{MU}(x, \Delta \hat{R}_{ij}, \Delta \hat{p}_{ij}, \Delta \hat{v}_{ij}) \|^2 + \sum_{k \in \mathcal{L}} \sum_{i \in \mathcal{F}_k} \| r_{CAM}(x, l_k, u^l_{ik}, u^r_{ik}) \|^2 + \| r_{PRIOR}(x) \|^2
\]

\[H\delta = \hat{\epsilon}\]

Solve a large linear system for \(\delta\)

**H is:**
- Symmetric
- Sparse
  (Black: non zero)

<table>
<thead>
<tr>
<th>Memory size (kB)</th>
<th>Full</th>
<th>Sym</th>
<th>Sym + Sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>703</td>
<td>353</td>
<td>134</td>
</tr>
<tr>
<td></td>
<td>2x</td>
<td></td>
<td>5.2x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processing time (ms)</th>
<th>Full</th>
<th>Sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>48.2</td>
<td>6.7</td>
</tr>
<tr>
<td></td>
<td>7.2x</td>
<td></td>
</tr>
</tbody>
</table>
Linear Solver memory: Structured Sparsity

\[ \min_x \sum_{(i,j) \in \mathcal{F}} \| r_{IMU}(x, \Delta \tilde{R}_{ij}, \Delta \tilde{p}_{ij}, \Delta \tilde{v}_{ij}) \|^2 + \sum_{k \in \mathcal{L}} \sum_{i \in \mathcal{F}_k} \| r_{CAM}(x, l_k, u_{ik}, u_{ik}^T) \|^2 + \| r_{PRIOR}(x) \|^2 \]

**Storing symmetric non-zero values:**
5.2x Memory size reduction

**Skip processing zeros:**
7.2x Speed up
Feature Tracks: Unstructured Sparsity

- Feature Tracks accounts for 88% of the Graph memory
Feature Tracks: Unstructured Sparsity

- Feature Tracks accounts for 88% of the Graph memory

3D Point (3x64-bit)

<table>
<thead>
<tr>
<th>U11</th>
<th>U12</th>
<th>U13</th>
<th>U14</th>
<th>Empty</th>
<th>U21</th>
<th>U22</th>
<th>U23</th>
<th>Empty</th>
<th>U31</th>
<th>U32</th>
<th>U33</th>
<th>Empty</th>
<th>U41</th>
<th>U42</th>
<th>U43</th>
<th>Empty</th>
</tr>
</thead>
<tbody>
<tr>
<td>v11</td>
<td>v12</td>
<td>v13</td>
<td>v14</td>
<td></td>
<td>v21</td>
<td>v22</td>
<td>v23</td>
<td></td>
<td>v31</td>
<td>v32</td>
<td>v33</td>
<td></td>
<td>v41</td>
<td>v42</td>
<td>v43</td>
<td></td>
</tr>
<tr>
<td>z11</td>
<td>z12</td>
<td>z13</td>
<td>z14</td>
<td></td>
<td>z21</td>
<td>z22</td>
<td>z23</td>
<td></td>
<td>z31</td>
<td>z32</td>
<td>z33</td>
<td></td>
<td>z41</td>
<td>z42</td>
<td>z43</td>
<td></td>
</tr>
</tbody>
</table>

KF ID (5-bit)

<table>
<thead>
<tr>
<th>KF1</th>
<th>KF2</th>
<th>KF3</th>
<th>KF4</th>
<th>KF1</th>
<th>KF2</th>
<th>KF3</th>
<th>KF4</th>
<th>KF1</th>
<th>KF2</th>
<th>KF3</th>
<th>KF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

One Memory (962 kB)
Feature Tracks: Unstructured Sparsity

• Feature Tracks accounts for 88% of the Graph memory

3D Point (3x64-bit)

KF ID (5-bit)

3D Point (3x64-bit)

One Memory
(962 kB)

Two-stage Memory
(177 kB)
Feature Tracks: Unstructured Sparsity

- Feature Tracks accounts for 88% of the Graph memory

**Feature tracks two-stage storage:**

- **5.4x** Memory size reduction

**Overhead:**

- 1 extra cycle access latency

One Memory (962 kB) → Two-stage Memory (177 kB)
Outline

• Localization & Mapping: Visual-Inertial Odometry (VIO)
• Chip Architecture
• Main Contributions
• Chip Specifications and Comparisons
• Summary
Navion Chip

- **Technology**: 65nm CMOS
- **Chip area (mm²)**: 4.0 x 5.0
- **Logic gates**: 2,043 kgates
- **Resolution**: 752 x 480
- **SRAM**: 854 kB
- **Camera rate**: 28 - 171 fps
- **Keyframe rate**: 16 - 90 fps
- **Average Power**: 24 mW
- **GOPS**: 10.5 - 59.1
- **GFLOPS**: 1 - 5.7
Memory Optimization

5.0 mm

4.0 mm

[Diagram showing memory optimization with bars indicating different memory sizes and labels for different processes like Frame, Graph, Linear Solver, and Misc.]

Symposia on VLSI Technology and Circuits
Navion System Demo

Navion chip PCB  Xilinx Zynq FPGA Board  Results
Navion Evaluation

- EuRoC dataset
  - A very challenging, and widely used UAV dataset
  - 11 sequences with three categories: easy, medium & difficult

Examples of easy Sequences

Examples of difficult Sequences

Dark scenes
Motion blur
Navion Evaluation

- Average numbers over the 11 EuRoC dataset sequences

<table>
<thead>
<tr>
<th>Platform</th>
<th>Xeon (E5-2667)</th>
<th>ARM (Cortex A15)</th>
<th>Navion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trajectory Error (%)</td>
<td>0.22%</td>
<td>0.28%</td>
<td></td>
</tr>
<tr>
<td>Camera rate (fps)</td>
<td>63</td>
<td>19</td>
<td>71</td>
</tr>
<tr>
<td>Keyframe rate (fps)</td>
<td>12</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>Average Power (W)</td>
<td>27.9</td>
<td>2.4</td>
<td>0.024</td>
</tr>
<tr>
<td>Energy (nJ/pixel)</td>
<td>2,531</td>
<td>1,094</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Navion Evaluation

• Average numbers over the 11 EuRoC dataset sequences

<table>
<thead>
<tr>
<th>Platform</th>
<th>Xeon (E5-2667)</th>
<th>ARM (Cortex A15)</th>
<th>Navion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trajectory Error (%)</td>
<td>0.22%</td>
<td>0.28%</td>
<td></td>
</tr>
<tr>
<td>Camera rate (fps)</td>
<td>63</td>
<td>19</td>
<td>71</td>
</tr>
<tr>
<td>Keyframe rate (fps)</td>
<td>12</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>Average Power (W)</td>
<td>27.9</td>
<td>2.4</td>
<td>0.024</td>
</tr>
<tr>
<td>Energy (nJ/pixel)</td>
<td>2,531</td>
<td>1,094</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Navion Energy:
684x less than embedded ARM CPU
1,582x less than server Xeon CPU
Outline

• Localization & Mapping: Visual-Inertial Odometry (VIO)
• Chip Architecture
• Main Contributions
• Chip Specifications and Comparisons
• Summary
Summary

• First full integration of VIO pipeline on chip for robot perception
Summary

• First **full integration** of VIO pipeline on chip for robot perception

• Leverage compression and sparsity to reduce memory size
  – 4.4x reduction with image compression
  – 5.2x reduction with structured sparsity in linear solver
  – 5.4x reduction with unstructured sparsity in feature tracks
Summary

• First full integration of VIO pipeline on chip for robot perception

• Leverage compression and sparsity to reduce memory size
  – 4.4x reduction with image compression
  – 5.2x reduction with structured sparsity in linear solver
  – 5.4x reduction with unstructured sparsity in feature tracks

• Navion is 2 to 3 orders of magnitude more energy efficient than CPU
Summary

• First full integration of VIO pipeline on chip for robot perception
• Leverage compression and sparsity to reduce memory size
  – 4.4x reduction with image compression
  – 5.2x reduction with structured sparsity in linear solver
  – 5.4x reduction with unstructured sparsity in feature tracks
• Navion is 2 to 3 orders of magnitude more energy efficient than CPU

Acknowledgment

AFOSR YIP and NSF CAREER
Questions

http://navion.mit.edu/