A General Framework for Mobile Robot Pose Tracking and Multi Sensors Self-Calibration

Davide Cucci, Matteo Matteucci
{cucci, matteucci}@elet.polimi.it
Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
Outline

- Motivations
- The ROAMFREE Project
  - Target Platforms
  - The *Logical Sensor* paradigm
  - Middlewares & Software Architecture
- Sensor fusion techniques
  - Extended Kalman Filter
  - Gauss-Newton
- Conclusions & Future work
Motivations

• **Position tracking** module is a fundamental component in autonomous robotics architectures

• Too often **ad-hoc methods** are employed
  ▪ Platform dependent
  ▪ Cumbersome calibration procedures
  ▪ Limited reusability

• Sensor fusion is a **mature field**
  ▪ Build frameworks upon established techniques for robust sensor fusion and calibration
The ROAMFREE\textsuperscript{1} project aims at developing:

\begin{itemize}
  \item Ready to use library of sensors and kinematics
  \item 6-DOF accurate and robust pose tracking module
  \item Calibration suite for intrinsic sensor parameters (e.g.: sensors gains, biases, displacements, misalignments)
\end{itemize}

Core concepts:

\begin{itemize}
  \item Independence from physical hardware and robotic platform
  \item Turn-on-and go but flexible and extensible
  \item Optimized C++ core libraries / Python bindings
  \item Interfaces to ROS.org
\end{itemize}

\textsuperscript{1} Italian Ministry of University and Research (MIUR) through the PRIN 2009 grant “ROAMFREE: Robust Odometry Applying Multi-sensor Fusion to Reduce Estimation Errors”.
Target Platforms: “One Size Fits Them All”

- Complex autonomous robotic applications
- Multiple, different sensors available
- Robust and accurate pose tracking needed
- Cumbersome/impractical parameter calibration
Complex robotic applications often employ a distributed architecture with a middleware connecting multiple nodes transparently.

The ROAMFREE toolsuite integrates without changes to existing software architecture.
Logical Sensors

- To abstract from physical hardware we deal with *logical sensors*
  - **Black-box** odometry information sources
  - Defined by the type of measurement they provide
    (e.g.: angular velocity, absolute position, …)
- The measurement is a function of a set of calibration parameters estimated by ROAMFREE
Logical Sensors - Examples

Sensor displacement wrt robot reference frame

- GPS
  - \( \hat{x}(W) \)
  - ROAMFREE

Wheel radius, distance, ...

- Wheel encoders
  - \( \{\hat{v}, \hat{\omega}\}(R) \)
  - ROAMFREE

Camera calibration, scale factors, ...

- Visual odometry Algorithm
  - \( \{\hat{v}, \hat{\omega}\}(R) \)
  - Camera a
  - Camera b
  - ROAMFREE
ROAMFREE provides default *logical sensor* wrappers to handle

- **Position and velocity in world frame:** $x^{(W)}, v^{(W)}$
  - e.g., Global Positioning System

- **Linear and angular velocity in sensor frame:** $v^{(S)}, \omega^{(S)}$
  - e.g., Visual odometry, Gyros

- **Acceleration in sensor frame:** $\alpha^{(S)}$
  - e.g., Accelerometers

- **Vector field in sensor frame:** $\vec{h}^{(S)}$
  - e.g., Earth Magnetic Field, Gravity Field
More predefined *logical sensors* to handle *common kinematics*

- Differential drive
- Ackermann
- Omnidirectional

The user can easily add *custom logical sensors*
We consider all the common sources of distortion, bias and noise

- For each *logical sensor*
  - Displacement and misalignment wrt Robot reference frame
- For angular velocity, acceleration, vector field
  - Gains
  - Biases
  - Non-orthogonality of sensor axes
- Kinematic models
  - Wheel radius, distance (Differential Drive)
  - Wheelbase (Ackermann)
  - ...
- User can develop in Python or C++
- Uses g2o and BFL external libraries
Framework Architecture

Online:
- Pose tracking
- Very flexible interface!
Framework Architecture

Offline:
- Parameters Calibration

- GUI
- Calibration suite
- Sensor Fusion
  - Gauss Newton
  - g2o
  - SE(3) EKF
  - BFL

- Tracking module
- Roscpp node wrappers
- Rospy node

Python
Example: ROS integration – 1/3

- Sensor readings usually available through standard ROS msg (e.g.: sensor_msgs/Imu, gps_common/GPSFix) possibly logged with rosbags
Example: ROS integration – 2/3

- ROAMFREE rosny node
  1) subscribes to sensors topics and provides sensor parameters
  2) drives the main sensor fusion library with sensor readings
  3) publishes the resulting pose estimate

```
ROS.org

ROAMFREE rosny node

1) subscribes to sensors topics and provides sensor parameters
2) drives the main sensor fusion library with sensor readings
3) publishes the resulting pose estimate
```
Example: ROS integration – 3/3

- **Offline** calibration case:
  - Rosbags are employed to generate a ROAMFREE dataset
  - The dataset is fed into the calibration GUI to estimate unknown sensor parameters

![Diagram showing the process of offline calibration using Rosbags and the ROAMFREE dataset.]

**Diagram Notes:**
- rosbag
- sensor_msgs/Imu, gps_common/GPSFix
- Rospy node
- ROAMFREE dataset
- GUI
- Calibration suite
- Sensor Fusion
Two modules available for sensor fusion

- **Graph-based Gauss-Newton optimization**
  - Online tracking and calibration
  - Tracks the trajectory of the robot over a finite time window (instead of the single pose at time $t$)
  - Parameters calibration → long time window

- **Extended Kalman Filter in the SE(3) Lie Group**
  - Online tracking
  - Work in progress
Gauss-Newton Pose Tracking

Maintains and optimizes online a measurements hyper-graph

- Nodes: last robot poses or unknown parameters
- Edges: measurement constraints

Maximize the likelihood of the nodes given the sensor readings

- **Gauss-Newton** (or Levenberg–Marquardt) optimization algorithm

\[
P : \arg \min_x \sum_{i=1}^{N} e_i(x_i) \Omega_{e_i} e_i(x_i)
\]
Graph construction: Example – 1/8

6-DOF Pose at time $t_0$: $\Gamma^W_O = \begin{bmatrix} x \\ y \\ z \\ q_w \\ qx \\ q_y \\ q_z \end{bmatrix}$
Graph construction: Example – 2/8

Predict the next pose using measurement

e.g.: \( x_{t_1} = x_{t_0} + v_{t_1} \Delta t \)

New odometry measure at time \( t_1 \), \((v, \omega)\)
Add an odometry constraint between poses

\[ err = x_{t_1} - (x_{t_0} + v_{t_1}) \Delta t \]

Gauss-Newton tries to minimize these error functions.

New odometry measure at time \( t_1 \), \((v, \omega)\)
Graph construction: Example – 4/8

New odometry measure at time $t_2$, $(v, \omega)$
Graph construction: Example – 5/8

New GPS measure at time $t_2$

Sensor displacement wrt robot odometric center

Absolute position constraint
e.g. $err = x_{t_2} + R_{O_{t_2}}^{W} S^{(O)} - z$
Parameter vertices are shared by edges of the same type

Another absolute position constraint
Graph construction: Example – 7/8

Edges in the past → **out-of-order measurements**
e.g.: acceleration measure at time $t_2$
Online case:
Old poses and constraints are discarded as time passes
Nice Video to Entertain Attendees
Lots of PROs
• Iterative optimization algorithm (possibly) yields higher tracking performances wrt other Bayesian approaches
• Very flexible formulation thanks to the hyper-graph approach
  ▪ Arbitrary number of sensors, possibly added at runtime
  ▪ Asynchronous sensors, different sampling frequencies
• Straightforward management of out-of-order measurements
• Outliers handled robustifying error functions
• Manifold optimization through encapsulation (Hertzberg et al., 2013)

With some CONs
• Increased time complexity, lower operation frequency
Further work

- Run benchmarks on a variety of robotic platforms (in progress)
- Extend the sensor library and the error models
- Extend framework to handle multi-body platforms (e.g.: vehicles with coachwork linked to body by suspensions)
- Compare the EKF and the Gauss-Newton approaches
GPS displacement and IMU misalignment can be estimated employing only their noisy measurement (simulated dataset)
GPS displacement and IMU misalignment can be estimated employing only their noisy measurement (simulated dataset).