Localizzazione Cooperativa Multi-Robot con EKF

Manuale d'Uso e Relazione Tecnica (versione centralizzata, 2 robot)

Lorenzo Nobili Leonardo Bacciocchi Cosmin Alessandro Minut
Dipartimento di Ingegneria

Ottobre 2025

Indice

1 Obiettivi del progetto **2**

2 Modelli di stato e di moto **2**
2.1 Stato . . . 2
2.2 Predizione con *pose increment* . . . 2

3 Dataset UTIAS / MRCLAM **3**
3.1 Panoramica . . . 3
3.2 Struttura dei file . . . 3
3.3 Formati record (indicativi) . . . 3
3.4 Uso nel progetto . . . 3
3.5 Subset e parametri . . . 3

4 Modello sensoriale **4**
4.1 Assunzioni e convenzioni . . . 4
4.2 Tipi di misura . . . 4
4.3 Osservazione robot → landmark . . . 4
4.4 Osservazione robot → robot . . . 4
4.5 Rumori di misura e gating . . . 4

5 EKF centralizzato **4**
5.1 Predizione . . . 4
5.2 Correzione singola e in *batch* . . . 5

6 Architettura software **5**
6.1 Componenti principali . . . 5
6.2 Dettagli su topic e messaggi (ROS 2) . . . 5
6.3 API indicative (allineate al codice) . . . 6
6.4 Dipendenze esterne . . . 6

7 Istruzioni per l'uso **7**
7.1 Prerequisiti . . . 7
7.2 Build . . . 7
7.3 Lancio (centralizzato, 2 robot) . . . 7
7.4 Guida passo–passo all'esecuzione . . . 7

8 Valutazione e diagnostica **8**
8.1 Metriche (`evaluate_accuracy.cpp`) 8
8.2 Andamento covarianze 9

1 Obiettivi del progetto

Il progetto sviluppa un localizzatore *cooperativo* basato su *Extended Kalman Filter* (EKF) in configurazione **centralizzata**, progettato per stimare **la posizione di due robot alla volta**. Lo stato congiunto è

$$\mathbf{x} = \begin{bmatrix} x_1,\ y_1,\ \theta_1,\ x_2,\ y_2,\ \theta_2 \end{bmatrix}^\top.$$

Il filtro esegue la **predizione** a partire dai dati di odometria e la **correzione** utilizzando misure robot→landmark e robot→robot. Le correzioni possono riguardare esclusivamente la posa dell'osservatore (landmark) oppure coinvolgere **entrambe le pose** nel caso di misura robot–robot. L'implementazione è pensata per l'uso con dati reali (dataset UTIAS/MRCLAM) ed è integrata in ROS 2.

Punti salienti

(1) Predizione attraverso la modalità *pose increment* $(\Delta x, \Delta y, \Delta\theta)$ in modo da poter gestire accuratamente i problemi di sincronizzazione temporale tra computer e controllore motori.

(2) Generalizzazione del tipo di misura con flag `is_landmark`/`type`, svincolata dalla convenzione UTIAS, fornendo maggiore flessibilità al sistema.

(3) Possibilità di aggiornamento in *batch* con vettore unico di misure, evitando così di dover fare una correzione EKF una misura alla volta.

(4) Monitoraggio dell'andamento delle covarianze $(\sigma_x, \sigma_y, \sigma_\theta)$ nel tempo.

2 Modelli di stato e di moto

2.1 Stato

Per N robot monocili lo stato è $\boldsymbol{x} = [\, x_1\ y_1\ \theta_1\ x_2\ y_2\ \theta_2\ \ldots\ x_N\ y_N\ \theta_N\,]^\mathsf{T} \in \mathbb{R}^{3N}$.

2.2 Predizione con *pose increment*

Quando il driver fornisce lo spostamento locale $\Delta\mathbf{p}_i = [\Delta x, \Delta y, \Delta\theta]^\top$, gli incrementi lineari vengono ruotati nel frame mondo tramite l'angolo corrente θ_i e composti con la posa:

$$\begin{bmatrix} \Delta x^{\mathrm{G}} \\ \Delta y^{\mathrm{G}} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}, \qquad x_i^{k+1} = x_i^k + \Delta x^{\mathrm{G}},\ \ y_i^{k+1} = y_i^k + \Delta y^{\mathrm{G}},\ \ \theta_i^{k+1} = \mathrm{wrap}(\theta_i^k + \Delta\theta). \tag{1}$$

Jacobian locale. Il Jacobiano della transizione rispetto allo stato locale $\mathbf{x}_i = [x_i, y_i, \theta_i]^\top$ è

$$\mathbf{F}_i = \begin{bmatrix} 1 & 0 & -\sin\theta_i\,\Delta x - \cos\theta_i\,\Delta y \\ 0 & 1 & \cos\theta_i\,\Delta x - \sin\theta_i\,\Delta y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

3 Dataset UTIAS / MRCLAM

3.1 Panoramica

Utilizziamo il dataset MRCLAM dell'UTIAS, composto da **9 set** indipendenti. Ogni set fornisce, per una piattaforma indoor con *motion capture* a 10 telecamere, i dati di **5 robot** monocicli mobili: odometria, misure *range–bearing* e *ground truth* delle pose dei robot, oltre alle posizioni dei **15 landmark** statici. Il sistema di riferimento del *motion capture* (Vicon) coincide con il frame inerziale del dataset; i ground truth sono a $\sim$100 Hz con accuratezza millimetrica.

3.2 Struttura dei file

Ogni set contiene 17 file di testo organizzati come segue:

- `Barcodes.dat`: associazione *barcode* $\rightarrow$ *subject_id*;
- `Landmark_Groundtruth.dat`: posizioni (x, y) dei 15 landmark (statici);
- `Robot[#]_Groundtruth.dat` (5 file): pose x, y, θ con timestamp per ciascun robot;
- `Robot[#]_Odometry.dat` (5 file): comandi di velocità con timestamp (v, ω);
- `Robot[#]_Measurement.dat` (5 file): misure *range–bearing* (r, ϕ) con timestamp.

Per convenzione, gli *ID* assegnati nel dataset sono: **1–5** per i robot, **6–20** per i landmark; la tabella dei *barcode* consente di distinguere in modo sistematico robot e landmark.

3.3 Formati record (indicativi)

I file sono in ASCII (una riga per misura). Un formato tipico è:

```
   Robot[#]_Odometry: timestamp v w|
Robot[#]_Measurement:
Robot[#]_Groundtruth: timestamp x y theta|
Landmark_Groundtruth:
```

I timestamp sono espressi in secondi; le unità sono metri e radianti.

3.4 Uso nel progetto

Nel nostro sistema (EKF centralizzato per **due robot alla volta**):

- un **DataLoader** legge `Barcodes.dat`, i file di odometria, misure e ground truth;
- le misure sono marcate come *robot→landmark* o *robot→robot* usando la mappa *barcode→subject_id*;
- costruiamo una **coda di eventi unica** (odometria + misure) ordinata per timestamp, così da rispettare l'**asincronia** tra i vari robot;
- i **RobotNode** riproducono il dataset pubblicando su `/odom`, `/landmark`, `/robot`; il **CentralizedEkfNode** consuma gli eventi e applica predizione/correzione dell'EKF.

3.5 Subset e parametri

Per le prove presentate abbiamo selezionato (a titolo d'esempio) uno dei 9 set e considerato **2 robot** (ID 1 e 2). I parametri principali di esecuzione sono: `dataset_path` (cartella MRCLAM), `num_robots`=2, `playback_speed` (fattore temporale), `use_groundtruth_init` (inizializzazione dallo stato reale).

4 Modello sensoriale

4.1 Assunzioni e convenzioni

Le misure sono espresse in metri/radianti. Il *bearing* è definito nel frame del robot *osservatore*; tutti gli angoli vengono normalizzati con $\mathrm{wrap}(\cdot) \in (-\pi, \pi]$.

4.2 Tipi di misura

Ogni misura è una tupla (t, o, s, r, ϕ, `type`) composta da: tempo t, *osservatore* o, *soggetto* s (landmark o altro robot), range r, bearing ϕ e un campo `type`/`is_landmark` che distingue in modo univoco robot→landmark da robot→robot.

4.3 Osservazione robot → landmark

Dato l'osservatore o con posa (x_o, y_o, θ_o) e un landmark noto $\boldsymbol{\ell} = [x_\ell, y_\ell]^\mathsf{T}$, poniamo

$$\Delta x = x_\ell - x_o, \qquad \Delta y = y_\ell - y_o, \qquad r = \sqrt{\Delta x^2 + \Delta y^2}, \qquad \phi = \mathrm{wrap}(\mathrm{atan2}(\Delta y, \Delta x) - \theta_o)\,.$$

La funzione di misura è $\boldsymbol{h}_{o\ell}(\boldsymbol{x}) = [\,r \;\; \phi\,]^\mathsf{T}$. Il Jacobiano rispetto allo *slice* di stato del solo osservatore $\boldsymbol{x}_o = [x_o, y_o, \theta_o]^\mathsf{T}$ è

$$\boldsymbol{H}_o = \begin{bmatrix} -\Delta x/r & -\Delta y/r & 0 \\ \Delta y/r^2 & -\Delta x/r^2 & -1 \end{bmatrix}, \qquad \boldsymbol{H} = \big[\, \cdots \underbrace{\boldsymbol{H}_o}_{o} \cdots \,\big].$$

4.4 Osservazione robot → robot

Con osservatore i e soggetto j (pose (x_i, y_i, θ_i) e (x_j, y_j, θ_j)):

$$\Delta x = x_j - x_i, \qquad \Delta y = y_j - y_i, \qquad q = \Delta x^2 + \Delta y^2, \qquad r = \sqrt{q}, \qquad \phi = \mathrm{wrap}(\mathrm{atan2}(\Delta y, \Delta x) - \theta_i)\,.$$

La misura è $\boldsymbol{h}_{ij}(\boldsymbol{x}) = [\,r \;\; \phi\,]^\mathsf{T}$. Il Jacobiano ha due blocchi non nulli, sull'osservatore i e sul soggetto j:

$$\boldsymbol{H}_i = \begin{bmatrix} -\Delta x/r & -\Delta y/r & 0 \\ \Delta y/q & -\Delta x/q & -1 \end{bmatrix}, \qquad \boldsymbol{H}_j = \begin{bmatrix} \Delta x/r & \Delta y/r & 0 \\ -\Delta y/q & \Delta x/q & 0 \end{bmatrix}, \qquad \boldsymbol{H} = \big[\, \cdots \; \boldsymbol{H}_i \; \cdots \; \boldsymbol{H}_j \; \cdots \,\big].$$

4.5 Rumori di misura e gating

Si assumono rumori gaussiani a media nulla:

$$\boldsymbol{R}_{\mathrm{lm}} = \mathrm{diag}(\sigma^2_{r,\mathrm{lm}}, \sigma^2_{\phi,\mathrm{lm}}), \qquad \boldsymbol{R}_{\mathrm{rr}} = \mathrm{diag}(\sigma^2_{r,\mathrm{rr}}, \sigma^2_{\phi,\mathrm{rr}}).$$

Prima della correzione, ogni misura è validata tramite distanza di Mahalanobis $d^2 = (\boldsymbol{z} - \boldsymbol{h})^\mathsf{T} \boldsymbol{S}^{-1} (\boldsymbol{z} - \boldsymbol{h})$, con $\boldsymbol{S} = \boldsymbol{H}\boldsymbol{P}\boldsymbol{H}^\mathsf{T} + \boldsymbol{R}$; se d^2 supera una soglia, la misura è scartata. Gli angoli (bearing e innovazione) sono sempre normalizzati con $\mathrm{wrap}(\cdot)$.

5 EKF centralizzato

5.1 Predizione

Per ogni odometria si aggiorna lo *slice* di stato del robot coinvolto e il blocco di covarianza corrispondente.

5.2 Correzione singola e in *batch*

Si supportano update singoli e un aggiornamento con vettore $\boldsymbol{z} \in \mathbb{R}^{2m}$ (stack di m misure) e jacobiano $\boldsymbol{H} \in \mathbb{R}^{2m \times 3N}$. L'innovazione $\boldsymbol{dz} = \boldsymbol{z} - \hat{\boldsymbol{z}}$ normalizza gli angoli. L'aggiornamento di stato e covarianza è

$$\boldsymbol{K} = \boldsymbol{P}\,\boldsymbol{H}^{\mathsf{T}}\,(\boldsymbol{H}\,\boldsymbol{P}\,\boldsymbol{H}^{\mathsf{T}} + \boldsymbol{R})^{-1}, \quad \boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{K}\,\boldsymbol{dz}, \quad \boldsymbol{P} \leftarrow (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\,\boldsymbol{P}\,(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})^{\mathsf{T}} + \boldsymbol{K}\,\boldsymbol{R}\,\boldsymbol{K}^{\mathsf{T}}. \tag{1}$$

Gating: le misure sono validate via distanza di Mahalanobis; gli outlier sono scartati.

6 Architettura software

6.1 Componenti principali

- `centralized_node.cpp` — Nodo ROS 2 centrale: carica parametri e (opz.) stato iniziale da ground truth, legge i landmark, si sottoscrive a `/odom`, `/landmark`, `/robot`, `/dataset_status`, esegue i passi EKF di predizione/correzione e stampa/salva diagnostica.
- `data_loader.cpp` — Caricatore del dataset: legge `Barcodes.dat`, `Landmark_Groundtruth.dat`, odometria e misure; converte barcode→ID soggetto; marca le misure come robot→landmark o robot→robot; produce una coda eventi ordinata per timestamp.
- `ekf_centralized.cpp` — Implementazione dell'EKF centralizzato: stato congiunto, due modelli di predizione (unicycle e *pose increment*), correzioni singola/batch, gating Mahalanobis, normalizzazione degli angoli e update in forma di Joseph.
- `evaluate_accuracy.cpp` — Valutazione offline: carica ground truth di due robot, rigioca gli eventi (predizione/correzione), confronta la stima finale con il GT e calcola metriche di accuratezza (errori su posizione/orientamento).
- `groundtruth_loader.cpp` — Utility ground truth: lettura dei file `Robot#_Groundtruth.dat` e costruzione dello *state* iniziale (prima posa di ciascun robot) per l'inizializzazione dell'EKF.
- `robot_node.cpp` — Nodo ROS 2 che simula un robot: legge il dataset del robot, costruisce una coda eventi e ripubblica `/odom`, `/landmark`, `/robot` e `/dataset_status` con timing realistico (scalato da `playback_speed`).
- `test_dataset.cpp` — Test rapido: carica un sottoinsieme del dataset (2 robot), inizializza EKF e processa i primi eventi per verificare parsing e aggiornamenti, stampando stato e conteggi di odometria/misure.
- `cooperative_localization_launch.py` — Launch ROS 2: definisce argomenti (`dataset_path`, `playback_speed`, `print_interval`), avvia il nodo EKF centralizzato e due `RobotNode` (ID 1 e 2).

6.2 Dettagli su topic e messaggi (ROS 2)

- `/odom` (`nav_msgs/msg/Odometry`): contiene v, ω in `twist`; `child_frame_id` indica l'ID del robot.
- `/landmark`, `/robot` (`geometry_msgs/msg/PointStamped`): `frame_id = "robot_<id_osservatore>"`; `point.x` = $r\cos\phi$, `point.y` = $r\sin\phi$, `point.z = subject_id`. Il nodo centrale ricostruisce (r, ϕ) da (x, y) e usa `point.z` come identificativo del soggetto (landmark o robot).
- `/dataset_status` (`std_msgs/msg/Bool`): segnala la fine della riproduzione del dataset per ogni robot.

6.3 API indicative (allineate al codice)

- `data_loader.h` — Modulo di **ingestione e normalizzazione** del dataset MRCLAM che fornisce ai nodi del progetto un'unica sorgente di dati *temporale* e coerente. In particolare:
  - definisce i **tipi** delle misure (`OdometryData`, `PoseIncrement`, `MeasurementData`) e un `Event` (con relativo `EventComparator`) per gestire odometria e misure in un'unica **coda ordinata per timestamp**;
  - costruisce la mappa **barcode** $\rightarrow$ **subject_id** a partire da `Barcodes.dat` per distinguere in modo univoco *robot* e *landmark*;
  - carica `Landmark_Groundtruth.dat` ed espone l'accesso alle posizioni dei landmark;
  - legge `Robot#_Odometry.dat` e `Robot#_Measurement.dat`, convertendo ogni riga in un *evento tipizzato* (con timestamp, osservatore/soggetto, range–bearing, ecc.);
  - offre un'interfaccia di **iterazione** sugli eventi che garantisce un flusso temporale coerente anche in presenza di asincronie tra robot.

  In questo modo `RobotNode` e `CentralizedEkfNode` consumano gli stessi dati "ripuliti" e già *fusi* in ordine cronologico.

- `ekf_centralized.h` — Dichiara la classe `EkfCentralized` che implementa un **EKF centralizzato** per localizzazione cooperativa con:
  - **stato congiunto** $[x_1, y_1, \theta_1, x_2, y_2, \theta_2, \dots]$;
  - **predizione** basata su modello cinematico *unicycle* e (nelle note) supporto a *pose increment*;
  - **correzioni** con misure landmark (assolute) e robot–robot (relative);
  - **validazione** delle misure via distanza di Mahalanobis; gestione di $\mathbf{Q}$, $\mathbf{R}$;
  - metodi di **diagnostica** (stampa stato/incertezza, storico covarianze).

- `groundtruth_loader.h` — Utility per la valutazione dei **ground truth** MRCLAM:
  - inizializzazione dello stato dell'EKF con la **prima posa** di ciascun robot (da `RobotX_Groundtruth.dat`);
  - uso dei GT per **valutazione** (metriche di errore) e **debugging**;
  - funzione `getInitialStateFromGroundtruth(...)` che costruisce il vettore $[x_1, y_1, \theta_1,\ x_2, y_2, \theta_2,\ \dots]$ di dimensione $3 \cdot$ `num_robots`, con **fallback** a $(0, 0, 0)$ se i file mancano.

6.4 Dipendenze esterne

- **ROS 2** (`rclcpp`, `nav_msgs`, `geometry_msgs`, `std_msgs`)
  Framework di comunicazione per robotica usato per implementare i nodi, i topic e i tipi di messaggio:
  - `rclcpp`: API C++ per creare nodi, publisher/subscriber, timer e parametri;
  - `nav_msgs/msg/Odometry`: odometria su `/odom` (contiene v, ω in `twist`);
  - `geometry_msgs/msg/PointStamped`: misure su `/landmark` e `/robot` con `frame_id = "robot_<id>"` e `point = (`$r\cos\phi$, $r\sin\phi$, *subject_id*`)`;
  - `std_msgs/msg/Bool`: segnalazione fine dataset su `/dataset_status`.

  ROS 2 fornisce anche il sistema di logging, la gestione dei parametri (`dataset_path`, `playback_speed`, `num_robots`, ...) e l'infrastruttura di launch.

- **Eigen3**
  Libreria C++ per algebra lineare utilizzata per rappresentare lo *stato congiunto* (`Eigen::VectorXd`) e le matrici di covarianza/jacobiani (`Eigen::MatrixXd`). Supporta operazioni a blocchi

(es. selezione dei sotto-stati dei singoli robot), fattorizzazioni numericamente stabili e la forma di aggiornamento di Joseph nell'EKF.

- **Dataset UTIAS / MRCLAM**
  Sorgente dei dati sperimentali (odometria, misure range–bearing, landmark e ground truth) usati per:
  - **riproduzione** "in tempo reale" tramite i `RobotNode` (topic `/odom`, `/landmark`, `/robot`);
  - **stima** con l'EKF centralizzato (correzioni landmark e robot–robot);
  - **valutazione** rispetto al *ground truth* (inizializzazione opzionale e calcolo metriche).

7 Istruzioni per l'uso

7.1 Prerequisiti

Ubuntu con ROS 2, `colcon`, Eigen, dipendenze ROS (`rclcpp`, `nav_msgs`, `geometry_msgs`, `std_msgs`).

7.2 Build

```
# Nel tuo workspace ROS2 (es. ~/ros2_ws)
source /opt/ros/$ROS_DISTRO/setup.bash    # ambiente ROS2
colcon build --packages-select cooperative_localization
source install/setup.bash                 # sorgente l'overlay del
    workspace
```

7.3 Lancio (centralizzato, 2 robot)

```
ros2 launch cooperative_localization cooperative_localization_launch.py
     \
  num_robots:=2 \
  dataset_path:=MRCLAM1/MRCLAM_Dataset1/ \
  playback_speed:=10.0 \
  print_interval:=3.0 \
  use_groundtruth_init:=true
```

Parametri utili: `num_robots` (qui 2), `dataset_path`, `playback_speed`, `print_interval`, `use_groundtruth_init`.

7.4 Guida passo–passo all'esecuzione

1. **Scaricare il pacchetto** e copiarlo in `$WORKSPACE_PATH/src`.
2. **Entrare nella cartella del pacchetto**:
   ```
   cd $WORKSPACE_PATH/src/cooperative_localization
   ```
3. **Eseguire lo script di build** (assicurati che sia eseguibile):
   ```
   chmod +x build_package.sh
   ./build_package.sh
   ```
4. **Aggiornare le variabili d'ambiente del workspace**:
   ```
   source $WORKSPACE_PATH/install/setup.bash
   ```

5. **Lanciare il sistema** (parametri di default):

```
ros2 launch cooperative_localization
   cooperative_localization_launch.py
```

Topic ROS 2 utilizzati

Input al nodo centralizzato (`CentralizedEkfNode`):

/odom
`nav_msgs/msg/Odometry`
Dati di odometria del robot (v, ω in `twist`); `child_frame_id` contiene l'ID del robot.

/landmark
`geometry_msgs/msg/PointStamped`
Misure robot→landmark: `frame_id = "robot_<id_osservatore>"`, `point.x =` $r\cos\phi$,

/robot
`geometry_msgs/msg/PointStamped`
Misure robot→robot con la stessa codifica (r, ϕ ricostruiti da (x, y); `subject_id` in `point.z`).

/dataset_status
`std_msgs/msg/Bool`
Segnala il completamento della riproduzione del dataset da parte di ciascun RobotNode.

Nota sull'output. Il sistema *non* pubblica topic di output; **stampa** periodicamente su console lo stato stimato e le deviazioni standard (parametro `print_interval`) e **salva** la diagnostica in `covariance_history.csv`.

8 Valutazione e diagnostica

8.1 Metriche (`evaluate_accuracy.cpp`)

Al termine dell'elaborazione si confrontano le *pose finali* stimate con il ground truth. Per ciascun robot $i \in \{1, 2\}$:

$$e_{x,i} = \hat{x}_i^{\text{final}} - x_{i,\text{GT}}^{\text{final}}, \quad e_{y,i} = \hat{y}_i^{\text{final}} - y_{i,\text{GT}}^{\text{final}}, \quad e_{\theta,i} = \text{wrap}\big(\hat{\theta}_i^{\text{final}} - \theta_{i,\text{GT}}^{\text{final}}\big).$$

L'**errore di posizione** è in **metri**:

$$e_{\text{pos},i} = \sqrt{e_{x,i}^2 + e_{y,i}^2} \ \ [\text{m}],$$

mentre l'**errore angolare** è calcolato in **radianti** e riportato anche in **gradi**:

$$|e_{\theta,i}| \ \ [\text{rad}], \qquad |e_{\theta,i}|_{\text{deg}} = |e_{\theta,i}| \cdot \frac{180}{\pi} \ \ [^\circ].$$

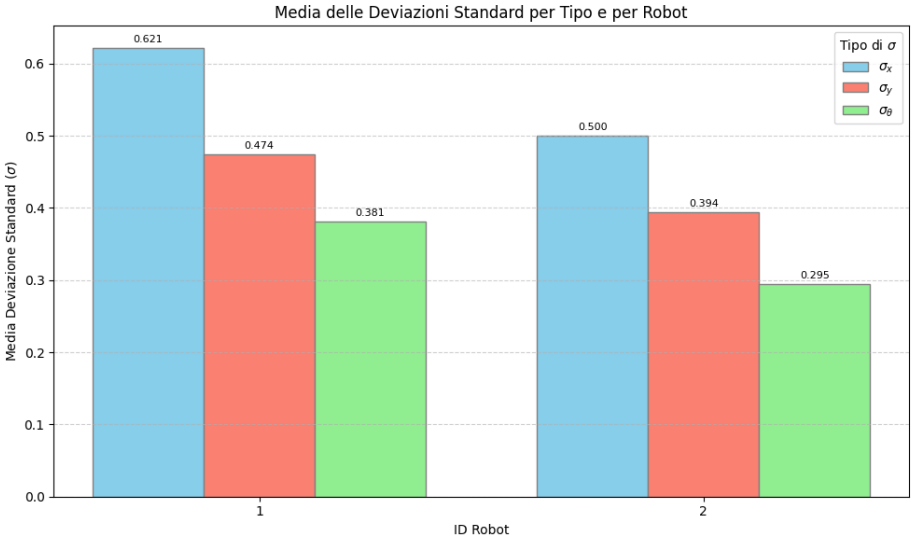
Gli indicatori globali (media sui due robot) sono:

$$\text{avg\_pos\_error} = \tfrac{1}{2}\big(e_{\text{pos},1} + e_{\text{pos},2}\big) \ [\text{m}], \qquad \text{avg\_ang\_error} = \tfrac{1}{2}\big(|e_{\theta,1}| + |e_{\theta,2}|\big) \ [\text{rad}],$$

Incertezze finali: Dalle diagonali di $\mathbf{P}$ si estraggono le deviazioni standard finali: $\sigma_{x,i}, \sigma_{y,i}$ [m] e $\sigma_{\theta,i}$ [rad].

8.2 Andamento covarianze

Il nodo centrale può salvare `covariance_history.csv`; dalla sua analisi otteniamo le deviazioni standard su posizione e orientamento.



figureMedia delle deviazioni standard (σ_x, σ_y, σ_θ) per tipo e per robot, da `covariance_history.csv`.