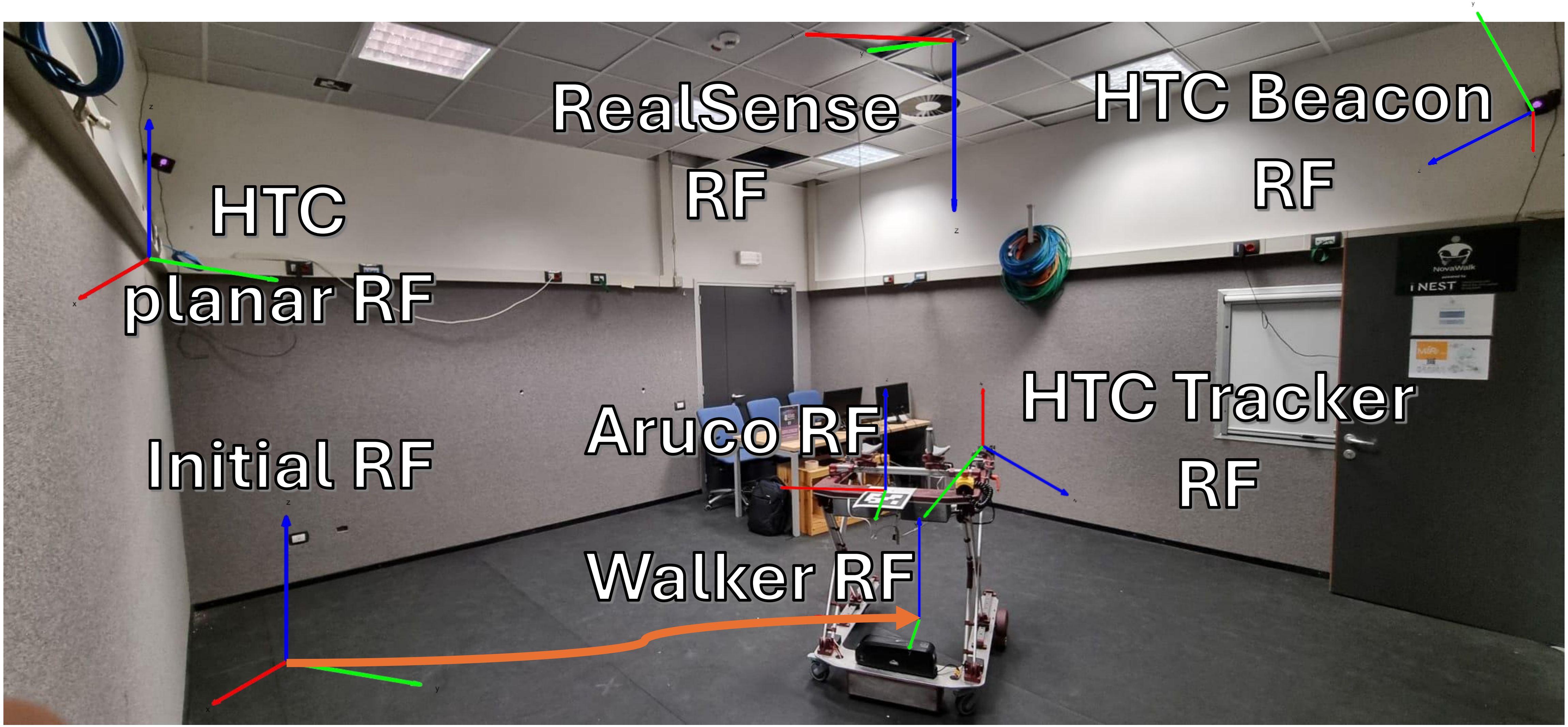
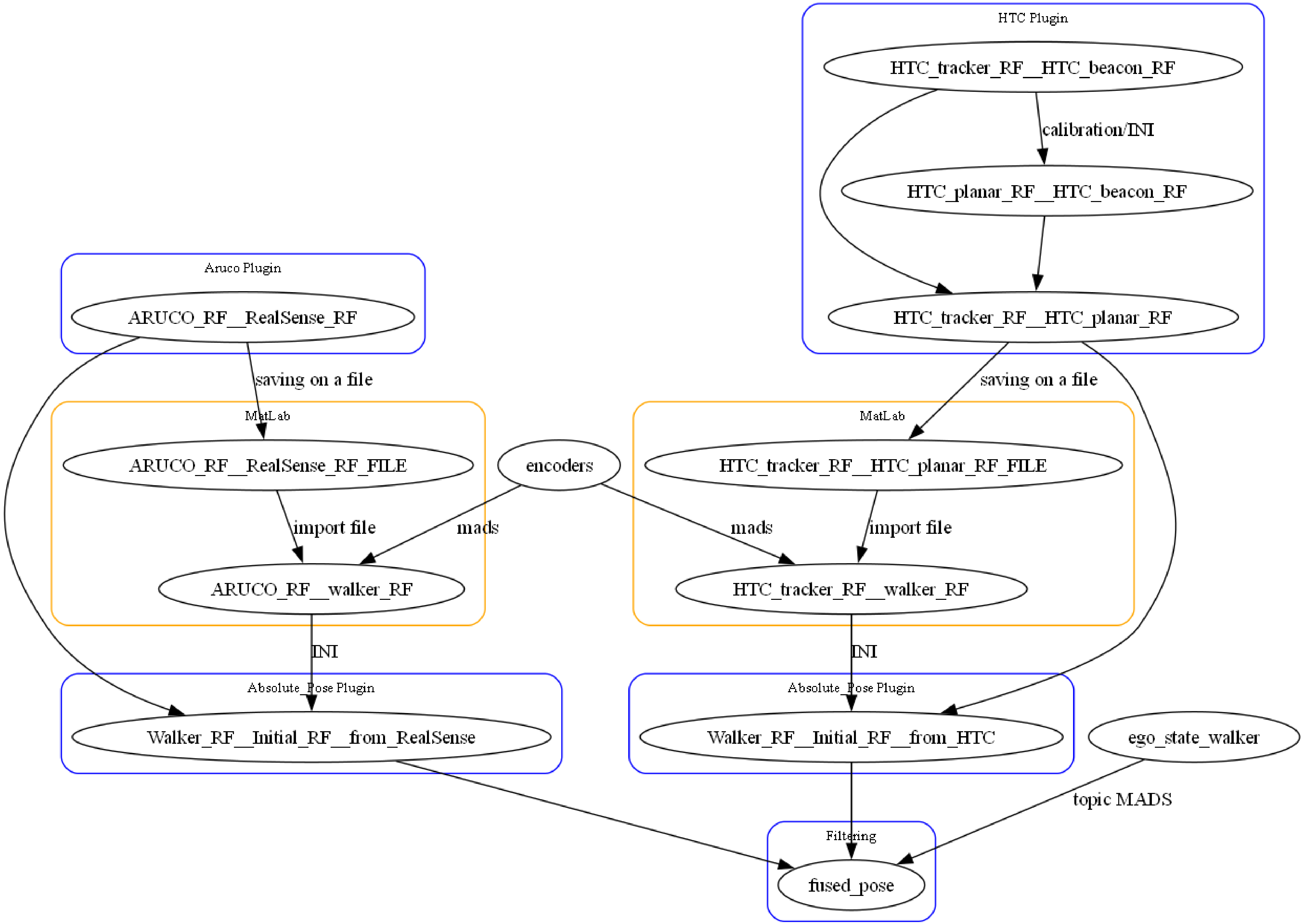
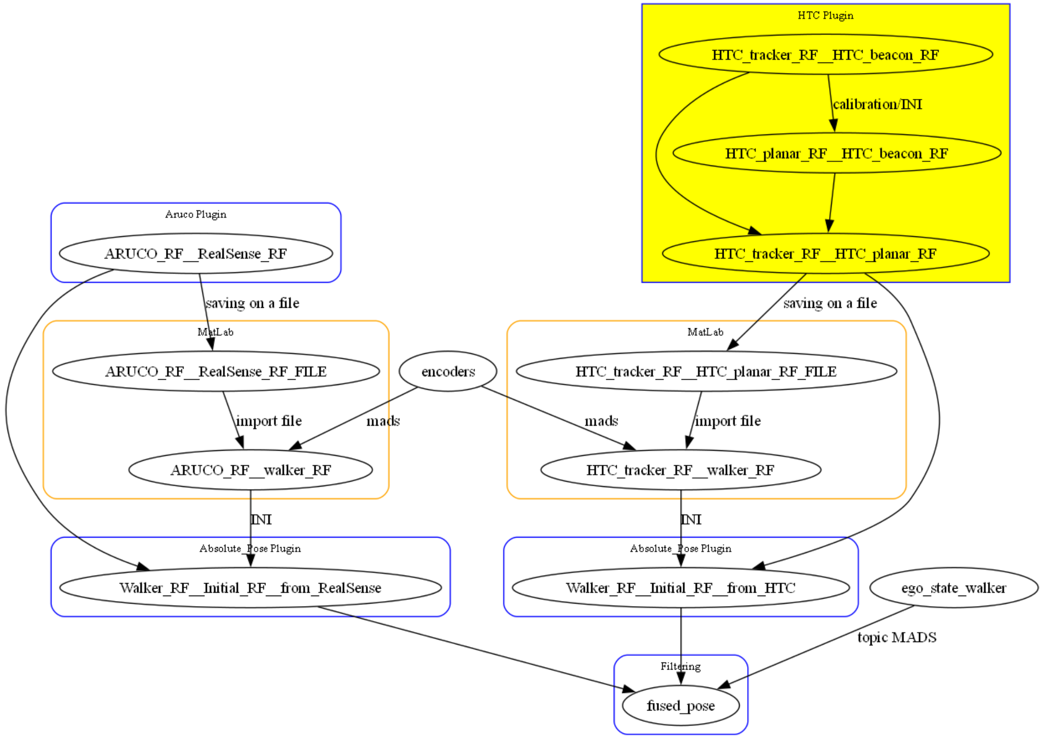
Calibration

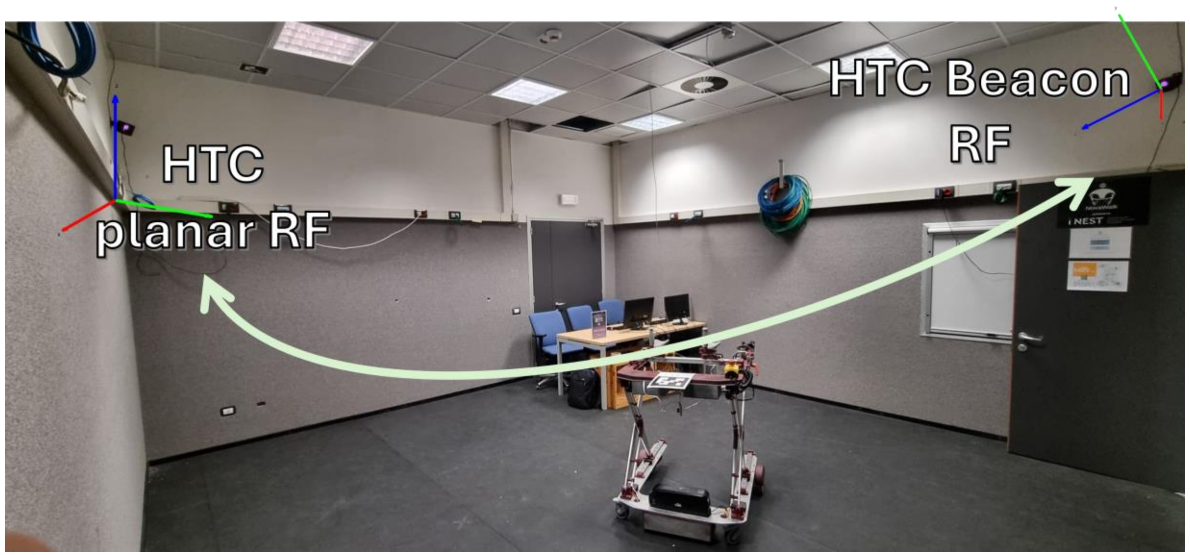
Matteo Bonetto, Prof. Mariolino De Cecco, Alessandro Luchetti
Email: matteo.bonetto-2@unitn.it, Alessandro.Luchetti@unitn.it, mariolino.dececco@unitn.it.


RealSense
RF
HTC Beacon
RF
HTC
planar RF
Aruco RF
HTC Tracker
RF
Initial RF
Walker RF


HTC Plugin
HTC_tracker_RF__HTC_beacon_RF
calibration/INI
HTC_planar_RF__HTC_beacon_RF
HTC_tracker_RF__HTC_planar_RF
Aruco Plugin
ARUCO_RF__RealSense_RF
saving on a file
saving on a file
MatLab
ARUCO_RF__RealSense_RF_FILE
encoders
MatLab
HTC_tracker_RF__HTC_planar_RF_FILE
import file
mads
mads
import file
ARUCO_RF__walker_RF
HTC_tracker_RF__walker_RF
INI
INI
Absolute_Pose Plugin
Walker_RF__Initial_RF__from_RealSense
Absolute_Pose Plugin
Walker_RF__Initial_RF__from_HTC
ego_state_walker
topic MADS
Filtering
fused_pose

$$H^{HTC\ planar\ RF}_{HTC\ tracker\ RF}$$





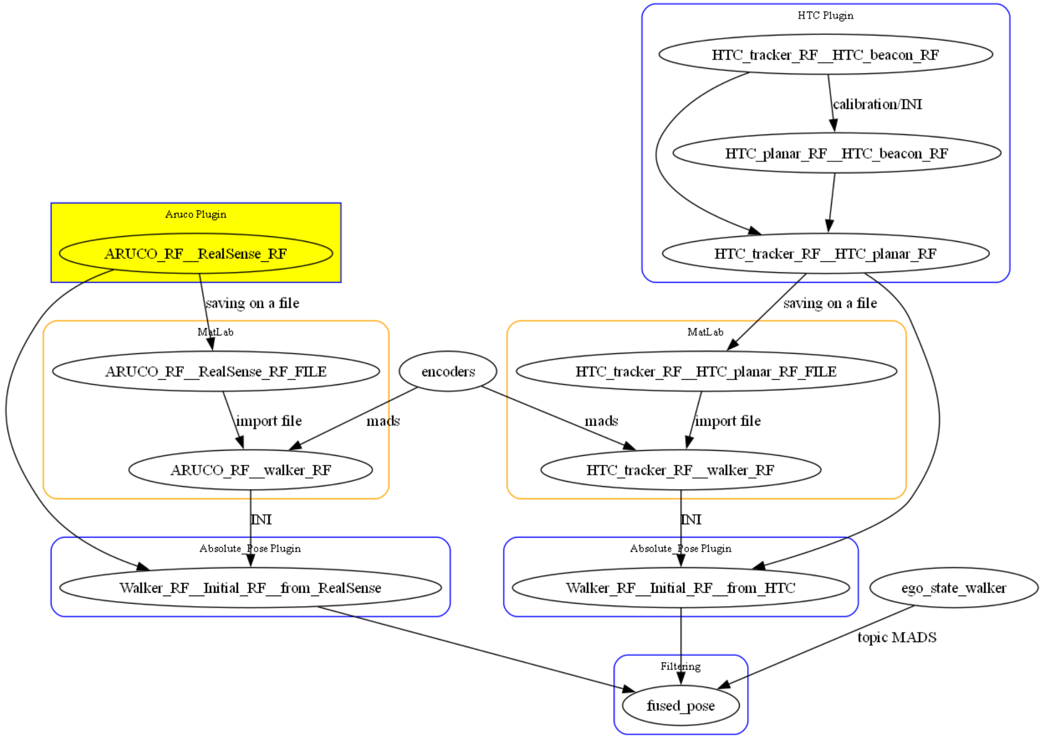
$$H_{HTC\ tracker\ RF}^{HTC\ planar\ RF}$$

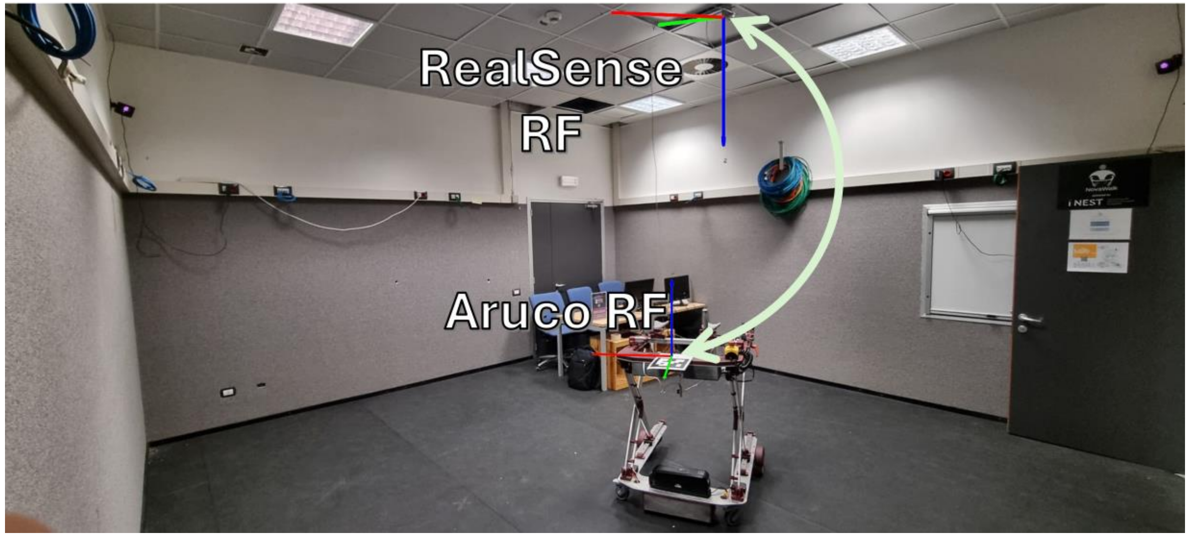
- Acquire different poses on the plane with respect to the **beacon RF** during calibration step
- Find rotation matrix R_{planar}^{beacon}:
  - Compute PCA
  - Choose the direction of the last principal component, knowing that the points acquired are lower than beacons.
  - Choose the other versor to complete a right-handed basis
- Find homogeneous matrix H_{planar}^{beacon}
- Find homogeneous matrix $H_{tracker}^{planar}$ for every pose acquired $H_{tracker}^{beacon}$

$$H_{planar}^{beacon} = \begin{bmatrix} & R_{planar}^{beacon} & & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_{tracker}^{planar} = (H_{planar}^{beacon})^{-1} * H_{tracker}^{beacon}$$

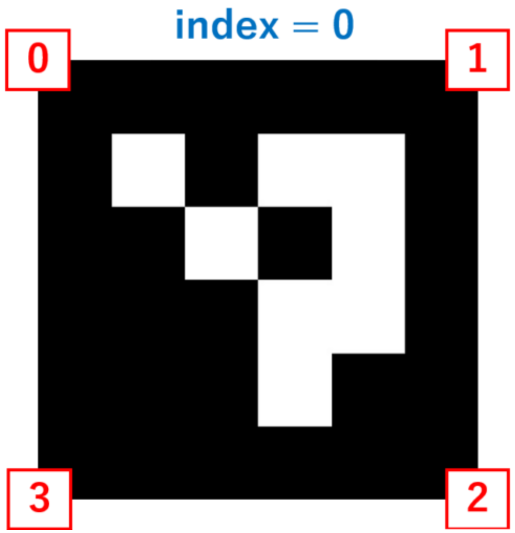
$$H^{RealSense\ RF}_{Aruco\ RF}$$



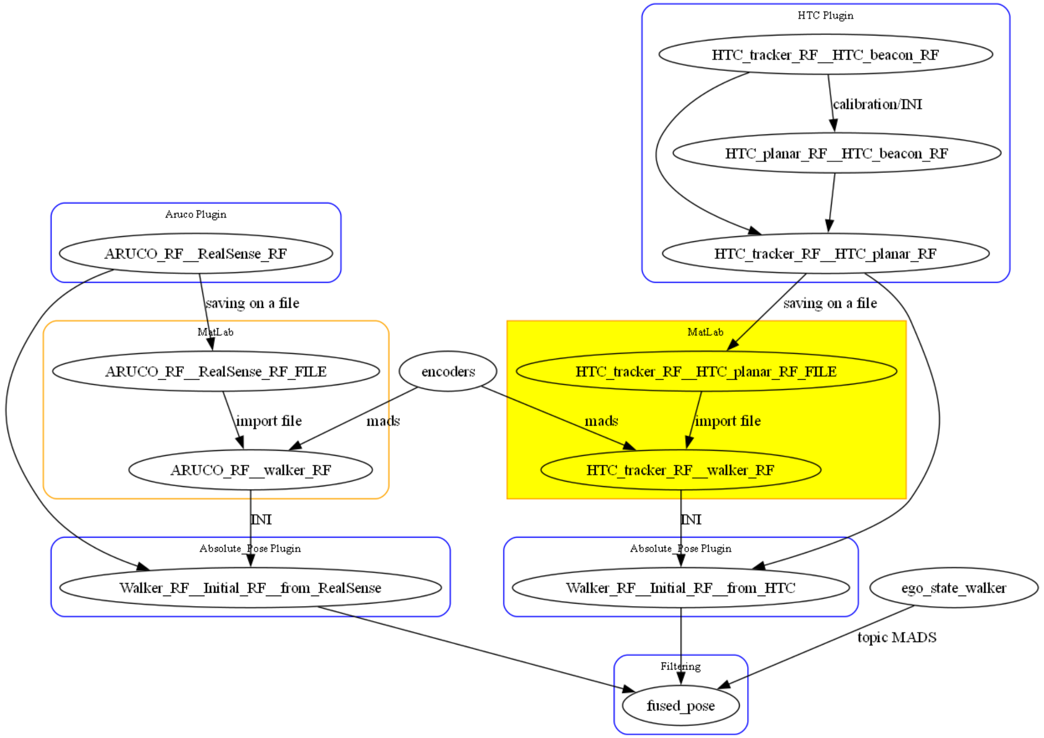


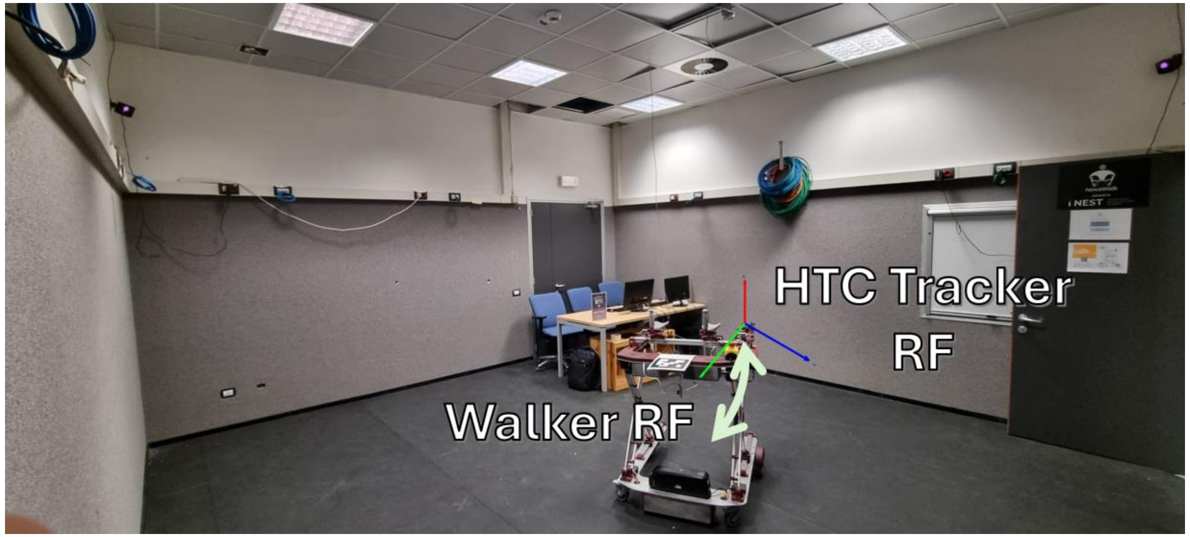
$$H_{Aruco\ RF}^{RealSense\ RF}$$

- acquired $H_{tracker}^{beacon}$
- Identify the Aruco
- Extract the 4 corner positions $(\overline{p_0}, \overline{p_1}, \overline{p_2}, \overline{p_3})$
- Generate the versor given by $\overline{p_0}$ and $\overline{p_1}$ → $\overline{v}$ = $\overline{p_1}$ - $\overline{p_0}$
- Assuming the RealSense perfectly oriented orthogonal to the floor, the rotation's angle aroud *z* is given by $atan2(v_y, v_x)$



$H^{HTC\ tracker\ RF}_{walker\ RF}$

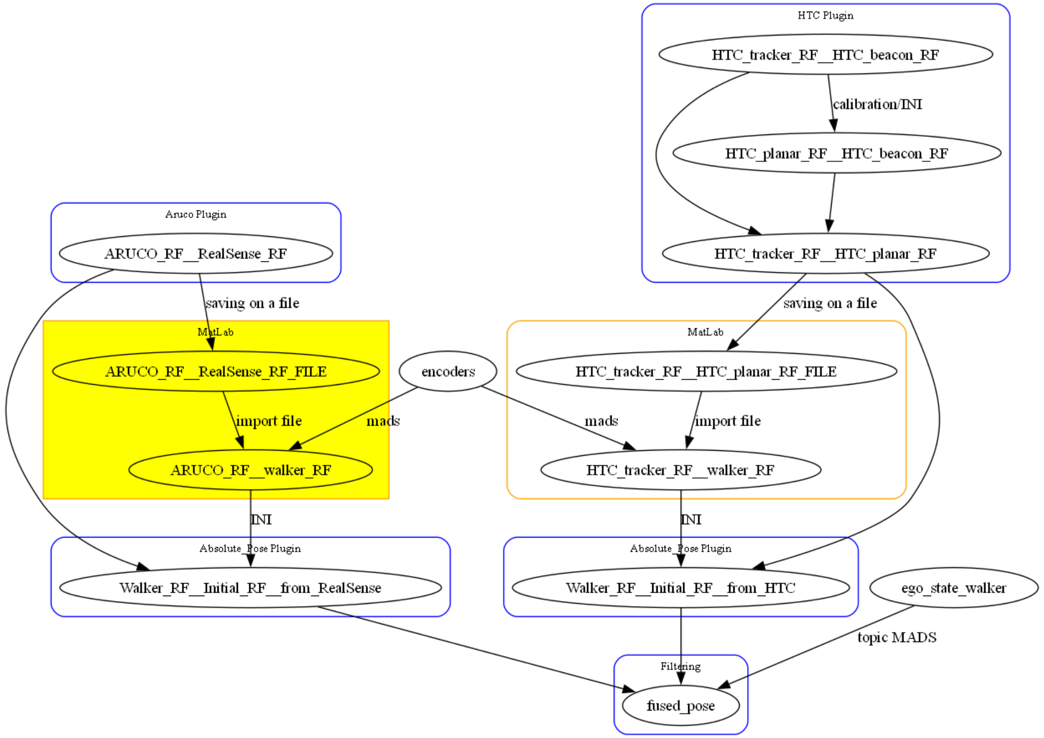


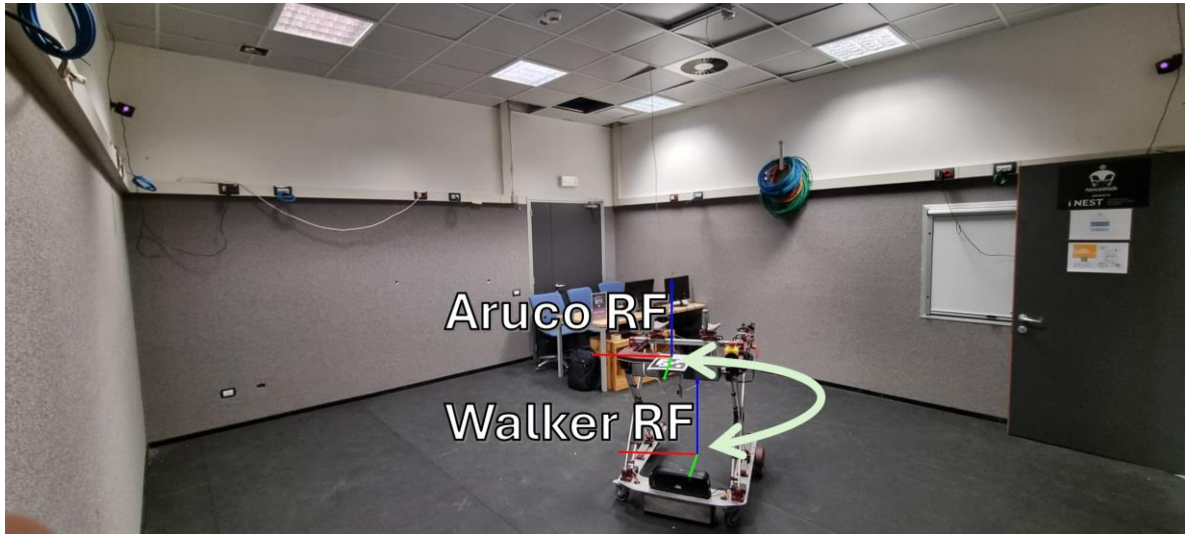


$$H_{walker\ RF}^{HTC\ tracker\ RF}$$

- Assuming infinitesimally small rotations about the plane's defining axes, it is reasonable to extract from $H_{tracker}^{beacon}$ just the angle around *z axis*, passing form 3D domain into 2D domain ($T\_real_{tracker}^{beacon}$).
- Reconstruct the robot's path using odometry derived from wheel encoder tick counts.
- Apply a generic roto translation matrix from walker pose to HTC tracker pose ($T\_simulation_{tracker}^{beacon}$)
- By means of an optimization, minimize the difference between $T\_real_{tracker}^{beacon}$ and $T\_simulation_{tracker}^{beacon}$

$H^{Aruco\ RF}_{walker\ RF}$

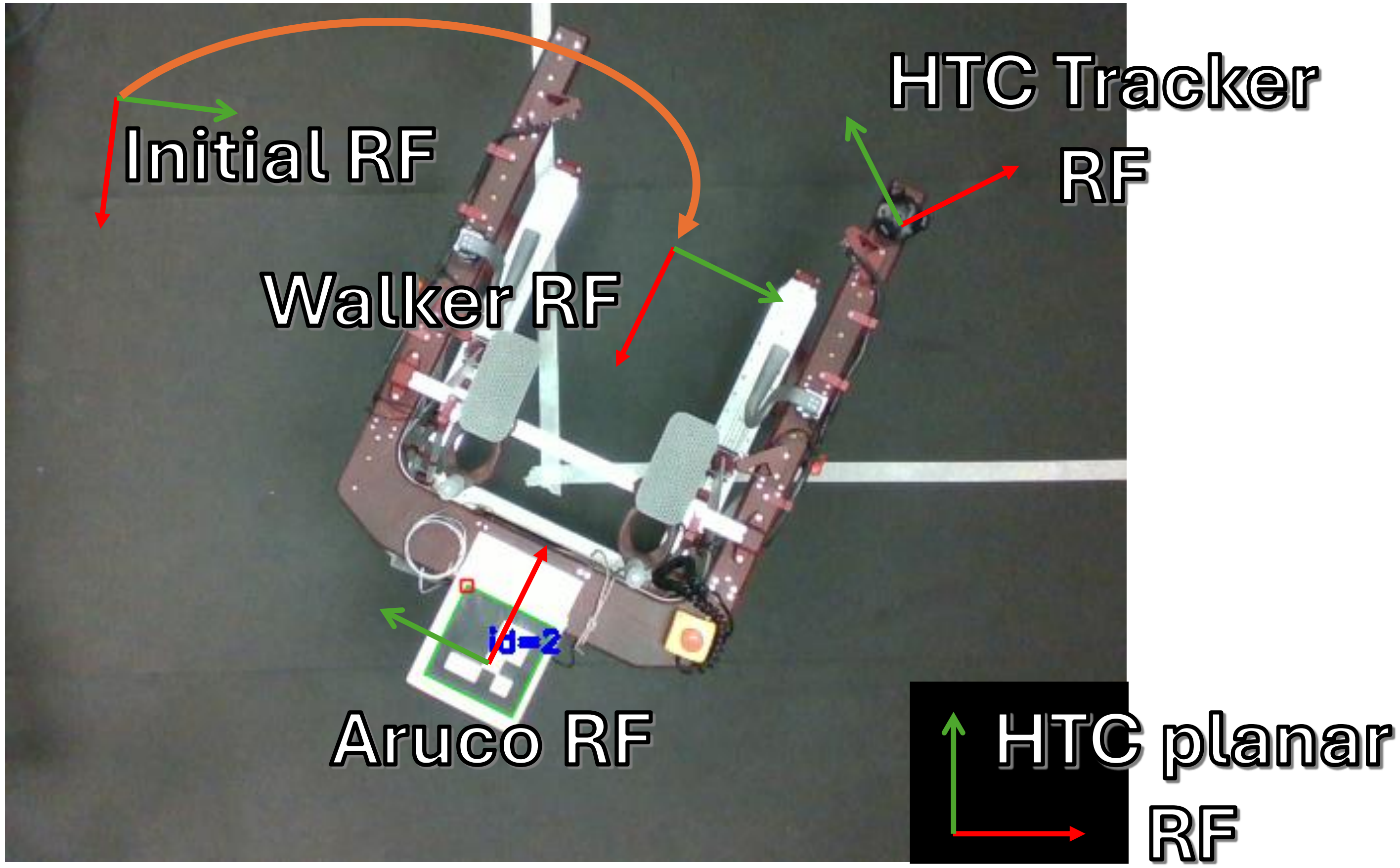




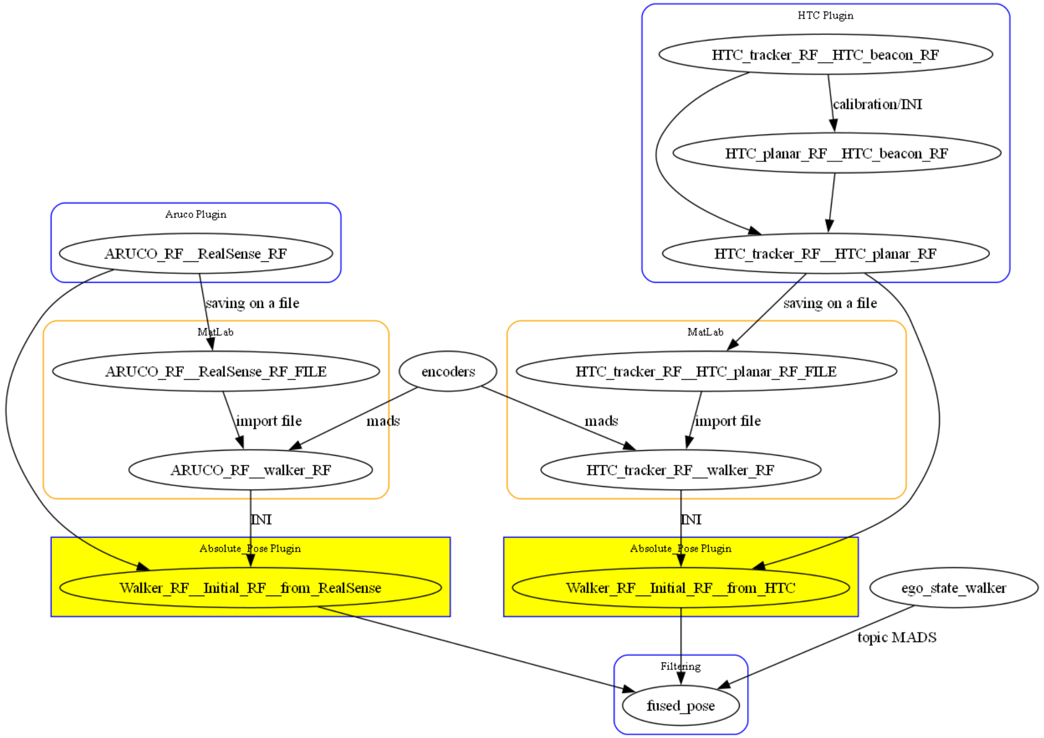
$$H_{walker\ RF}^{Aruco\ RF}$$

Applying the same procedure to the aruco's pose instead of the HTC tracker in the MatLab optimization, $H_{walker\ RF}^{Aruco\ RF}$ is achieved.

$$H_{walker\ RF}^{Aruco\ RF} \ \& \ H_{walker\ RF}^{HTC\ tracker\ RF}$$



$$H^{\text{Initial RF}}_{\text{walker RF}}$$



$$H_{walker\ RF}^{Initial\ RF}$$

- Retrieve from the INI file the matrix $H_{walker\ RF}^{Aruco\ RF}$.
- When a trigger is received to indicate the initiation of odometry, the transformation $H_{RealSense\ RF}^{Initial\ RF}$ is computed.
- For each pose at time t_i of the Aruco ($H_{Aruco\ RF}^{RealSense\ RF}$), the transformation $H_{walker\ RF}^{Initial\ RF}{}_{t_i}$ is obtained.

$$H_{RealSense\ RF}^{Initial\ RF} = (H_{walker\ RF}^{Aruco\ RF})^{-1} * (H_{Aruco\ RF}^{RealSense\ RF}{}_{t_0})^{-1}$$

$$H_{walker\ RF}^{Initial\ RF}{}_{t_i} = H_{RealSense\ RF}^{Initial\ RF} * H_{Aruco\ RF}^{RealSense\ RF}{}_{t_i} * H_{walker\ RF}^{Aruco\ RF}$$

$$H_{walker\ RF}^{Initial\ RF}$$

The procedure is applied to the HTC data as well in order to obtain the pose of the walker with respect to its initial reference system, exploiting the data of another environment referred sensor.

How to import acquire data in your MADS framework to compute your **Sensor Fusion Plugin**?

Requirements

💻 Windows

✅ Install Git

1. Go to the official Git website: https://git-scm.com/download/win ↗
2. The download will start automatically.
3. Run the installer and follow the setup instructions:
   - Use default options unless you have specific requirements.
   - Make sure "Git from the command line" is selected.
4. Verify the installation:

bash Copy code

```bash
git --version
```

✅ Install CMake

1. Visit the CMake download page: https://cmake.org/download ↗
2. Download the **Windows x64 Installer** (e.g., `cmake-<version>-windows-x86_64.msi`).
3. Run the installer:
   - Select **"Add CMake to system PATH for all users"** during setup.
4. Verify the installation:

bash Copy code

```bash
cmake --version
```

🐧 Ubuntu Linux

✅ Install Git

Open a terminal and run:

bash Copy code

```bash
sudo apt update
sudo apt install git
```

Verify installation:

bash Copy code

```bash
git --version
```

✅ Install CMake

Install CMake using `apt`:

bash Copy code

```bash
sudo apt update
sudo apt install cmake
```

Verify installation:

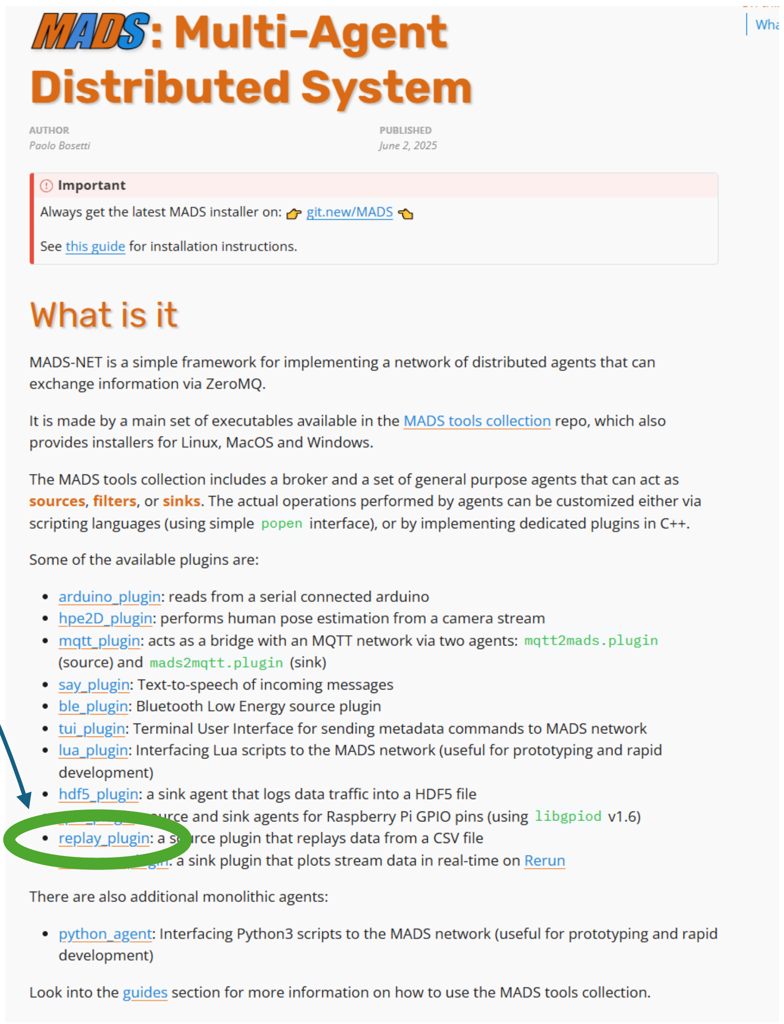
bash Copy code

```bash
cmake --version
```

> Note: The version from Ubuntu's default repositories may not be the latest. If you need a newer version, consider building from source (not covered in this guide).

Requirements

- Open MADS WebSite([Multi-Agent Distributed System]())
- Select *replay_plugin*
- Navigate to the directory where you want to install this plugin
- `git clone https://github.com/MADS-NET/replay_plugin.git`
- Follow the Instructions to install and use it correctly



MADS: Multi-Agent Distributed System

AUTHOR
Paolo Bosetti

PUBLISHED
June 2, 2025

Important

Always get the latest MADS installer on: git.new/MADS

See this guide for installation instructions.

What is it

MADS-NET is a simple framework for implementing a network of distributed agents that can exchange information via ZeroMQ.

It is made by a main set of executables available in the MADS tools collection repo, which also provides installers for Linux, MacOS and Windows.

The MADS tools collection includes a broker and a set of general purpose agents that can act as **sources**, **filters**, or **sinks**. The actual operations performed by agents can be customized either via scripting languages (using simple `popen` interface), or by implementing dedicated plugins in C++.

Some of the available plugins are:

- arduino_plugin: reads from a serial connected arduino
- hpe2D_plugin: performs human pose estimation from a camera stream
- mqtt_plugin: acts as a bridge with an MQTT network via two agents: `mqtt2mads.plugin` (source) and `mads2mqtt.plugin` (sink)
- say_plugin: Text-to-speech of incoming messages
- ble_plugin: Bluetooth Low Energy source plugin
- tui_plugin: Terminal User Interface for sending metadata commands to MADS network
- lua_plugin: Interfacing Lua scripts to the MADS network (useful for prototyping and rapid development)
- hdf5_plugin: a sink agent that logs data traffic into a HDF5 file
- …rce and sink agents for Raspberry Pi GPIO pins (using `libgpiod` v1.6)
- replay_plugin: a s…rce plugin that replays data from a CSV file
- …: a sink plugin that plots stream data in real-time on Rerun

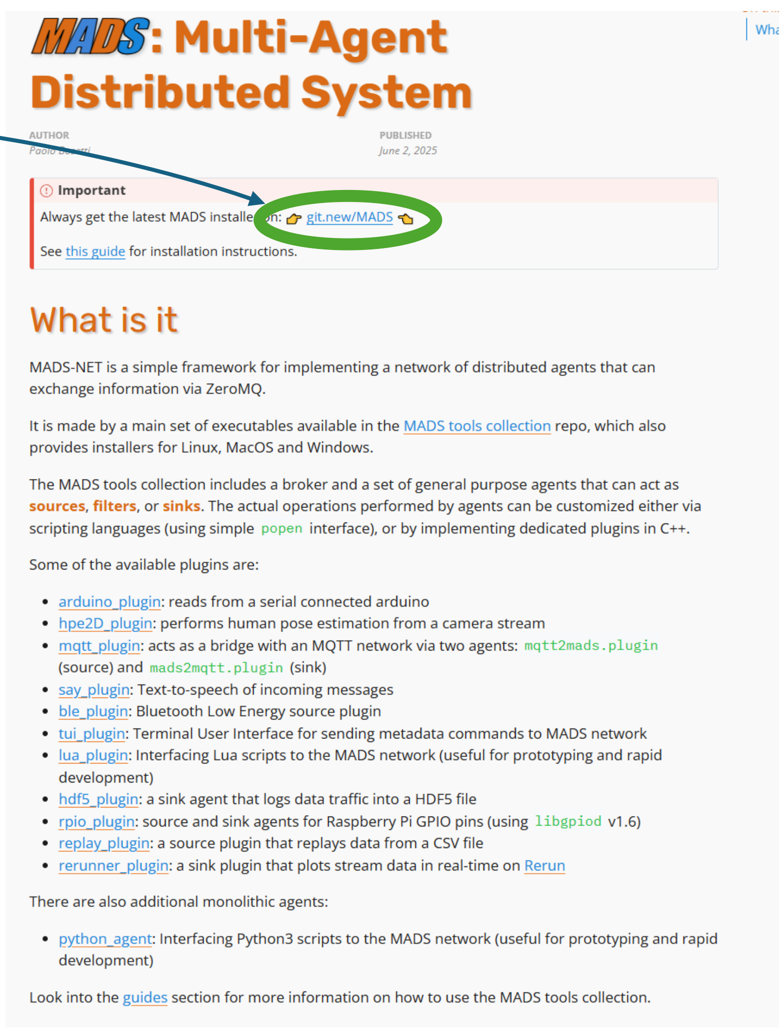
There are also additional monolithic agents:

- python_agent: Interfacing Python3 scripts to the MADS network (useful for prototyping and rapid development)

Look into the guides section for more information on how to use the MADS tools collection.

Requirements

- Install last MADS version
- With this command will be created the INI file, it will also print the path where the file is installed: `mads ini -i`



MADS: Multi-Agent Distributed System

AUTHOR
Paolo Bosetti

PUBLISHED
June 2, 2025

> **Important**
> Always get the latest MADS installer on: 👉 git.new/MADS 👈
>
> See this guide for installation instructions.

What is it

MADS-NET is a simple framework for implementing a network of distributed agents that can exchange information via ZeroMQ.

It is made by a main set of executables available in the MADS tools collection repo, which also provides installers for Linux, MacOS and Windows.

The MADS tools collection includes a broker and a set of general purpose agents that can act as **sources**, **filters**, or **sinks**. The actual operations performed by agents can be customized either via scripting languages (using simple `popen` interface), or by implementing dedicated plugins in C++.

Some of the available plugins are:

- arduino_plugin: reads from a serial connected arduino
- hpe2D_plugin: performs human pose estimation from a camera stream
- mqtt_plugin: acts as a bridge with an MQTT network via two agents: `mqtt2mads.plugin` (source) and `mads2mqtt.plugin` (sink)
- say_plugin: Text-to-speech of incoming messages
- ble_plugin: Bluetooth Low Energy source plugin
- tui_plugin: Terminal User Interface for sending metadata commands to MADS network
- lua_plugin: Interfacing Lua scripts to the MADS network (useful for prototyping and rapid development)
- hdf5_plugin: a sink agent that logs data traffic into a HDF5 file
- rpio_plugin: source and sink agents for Raspberry Pi GPIO pins (using `libgpiod` v1.6)
- replay_plugin: a source plugin that replays data from a CSV file
- rerunner_plugin: a sink plugin that plots stream data in real-time on Rerun

There are also additional monolithic agents:

- python_agent: Interfacing Python3 scripts to the MADS network (useful for prototyping and rapid development)

Look into the guides section for more information on how to use the MADS tools collection.

Run Data acquired on a topic

- In the INI file create a section for each csv file

```
[replay_imu]
csv_file = "/home/alessandroluchetti/Downloads/Data_Sensor_Fusion/mads_calibration.imu.csv"
loop = false # once the end of file has been reached, start again from beginning

[replay_htc]
csv_file = "/home/alessandroluchetti/Downloads/Data_Sensor_Fusion/mads_calibration.absolute_pose_wa>
loop = false # once the end of file has been reached, start again from beginning

[replay_encoders]
csv_file = "/home/alessandroluchetti/Downloads/Data_Sensor_Fusion/mads_calibration.encoders.csv"
loop = false # once the end of file has been reached, start again from beginning

[replay_realsense]
csv_file = "/home/alessandroluchetti/Downloads/Data_Sensor_Fusion/mads_calibration.absolute_pose_wa>
loop = false # once the end of file has been reached, start again from beginning
```

```
mads_calibration.absolute_pose_walker_for_htc.csv        mads_calibration.encoders.csv
mads_calibration.absolute_pose_walker_for_realsense.csv  mads_calibration.imu.csv
```

Run Data acquired on a topic

- Launch MADS broker
- With argument –n it is possible to change the name of the plugin
- With argument –p it is possible to change the time period
- Launch the command one for each of the file you want to replay

```
alessandroluchetti@Jakku: ~/Ermes/plugin 100x53
alessandroluchetti@Jakku:~/Ermes/plugin$ mads source replay.plugin -n "replay_encoders" -p 1000
Agent: replay_encoders
  Settings file:     tcp://localhost:9092
  Pub endpoint:      tcp://localhost:9090
  Pub topic:         replay_encoders
  Sub endpoint:      tcp://localhost:9091
  Sub topics:        control
  Compression:       enabled
  Timecode FPS:      5000
  Timecode offset:   0.004 s
  Sampling period:   1000 ms (from -p option)
  Searching for installed plugin in the default location /usr/local/lib
  Plugin:            /usr/local/lib/replay.plugin (loaded as replay_encoders)
  CSV file           /home/alessandroluchetti/Downloads/Data_Sensor_Fusion/mads_calibration.encoders.
csv
  Loop               false
  Blob format:       none
Source plugin started
Messages processed: 7 total, 0 with errors
```

```
alessandroluchetti@Jakku: ~/Ermes/plugin 100x53
alessandroluchetti@Jakku:~/Ermes/plugin$ mads broker
Reading settings from /usr/local/etc/mads.ini [broker]
ioctl error: No such device
Cannot get IP address for NIC lo0
Binding broker frontend (XSUB) at tcp://*:9090
Binding broker backend (XPUB) at tcp://*:9091
Binding broker shared settings (REP) at tcp://*:9092
Timecode FPS: 5000
Settings are provided via tcp://127.0.0.1:9092
CTRL-C to immediate exit
Type P to pause, R to resume, I for information, Q to clean quit, X to restart and reload settings
Sending settings to agent replay_encoders (v1.4.0)
```

Format of the topics

What it is sent on mads can be seen with the command:

`mads feedback`

Format of the topics: **encoders**


```
replay_encoders: {
  "_id": "68f65b23c658f28f7d093773",
  "hostname": "Jakku",
  "message": {
    "currents": {
      "left": 0.0,
      "right": 0.0
    },
    "encoders": {
      "left": 34836.0,
      "right": 154840.0
    },
    "hostname": "raspberrypi",
    "timecode": 33111.89,
    "timestamp": "2025-10-06T07:11:51.890Z"
  },
  "timecode": 40164.045,
  "timestamp": {
    "$date": "2025-10-21T11:09:24.045+0200"
  },
  "warning": {
    "get_output": "timestamp field name is not allowed, it will be removed"
  }
```


Example for reading these values in your plugin:

```
if(topic == "replay_encoders") {
  if (input["message"].contains("encoders") {
    _torque_sx = input["message"]["encoders"].value<int>("left", _default_value);
    _torque_dx = input["message"]["encoders"].value<int>("right", _default_value);
  }
}
```

Format of the topics: **HTC**


```
replay_htc: {
  "_id": "68f65b24c658f28f7d0937c7",
  "hostname": "Jakku",
  "message": {
    "hostname": "raspberrypi",
    "pose": {
      "H": [
        [
          0.9999999920800665,
          -2.972301194737036e-05,
          -0.00012229639399275922,
          8.846029233594566e-06
        ],
        [
          2.9697361032496166e-05,
          0.9999999775635343,
          -0.0002097403071998661,
          -0.00020635345215325174
        ],
        [
          0.00012230262536255178,
          0.00020973667365875869,
          0.9999999705262971,
          -0.00020258953204348 27
        ],
        [
          0.0,
          0.0,
          0.0,
          0.9999999999999999
        ]
      ],
      "R": [
        [
          0.9999999920800665,
          -2.972301194737036e-05,
          -0.00012229639399275922
        ],
        [
          2.9697361032496166e-05,
          0.9999999775635343,
          -0.0002097403071998661
        ],
        [
          0.00012230262536255178,
          0.00020973667365875869,
          0.9999999705262971
        ]
      ],
```



```
      "attitude": [
        0.00020974031030612791,
        -0.00012229639429761177,
        2.9723012174071138e-05
      ],
      "attitude_along_z": 2.9697361258966928e-05,
      "attitude_deg": [
        0.012017234574305372,
        -0.007007067242920942,
        0.0017030031519902416
      ],
      "position": [
        8.846029233594566e-06,
        -0.00020635345215325174,
        -0.00020258953204348 27
      ]
    },
    "timecode": 33112.17,
    "timestamp": "2025-10-06T07:11:52.173Z"
  },
```


- *attitude_along_z* represents the angle in the odometry data format. Unlike a typical angle that resets after 360°
- *H* is the rototranslation matrix
- *R* is the rotation matrix
- *Position* is the vector of translation from the Initial RF

Format of the topics: **IMU**


```
replay_imu: {
  "_id": "68f65b24c658f28f7d0937bd",
  "hostname": "Jakku",
  "message": {
    "hostname": "raspberrypi",
    "imu": {
      "left": {
        "accelerations": {
          "x": 25.0,
          "y": 9.77,
          "z": 0.01
        },
        "gyroscopes": {
          "x": 0.02,
          "y": 0.0,
          "z": 39.19
        }
      },
      "middle": {
        "accelerations": {
          "x": 10.05,
          "y": 0.01,
          "z": -0.04
        },
        "gyroscopes": {
          "x": -0.02,
          "y": 0.11,
          "z": -0.23
        }
      },
      "right": {
        "accelerations": {
          "x": 14.92,
          "y": 39.19,
          "z": 9.27
        },
        "gyroscopes": {
          "x": -43.76,
          "y": 11.28,
          "z": 0.24
        }
      }
    },
    "timecode": 33112.14,
```

IMU position