Cyber-physical systems programming

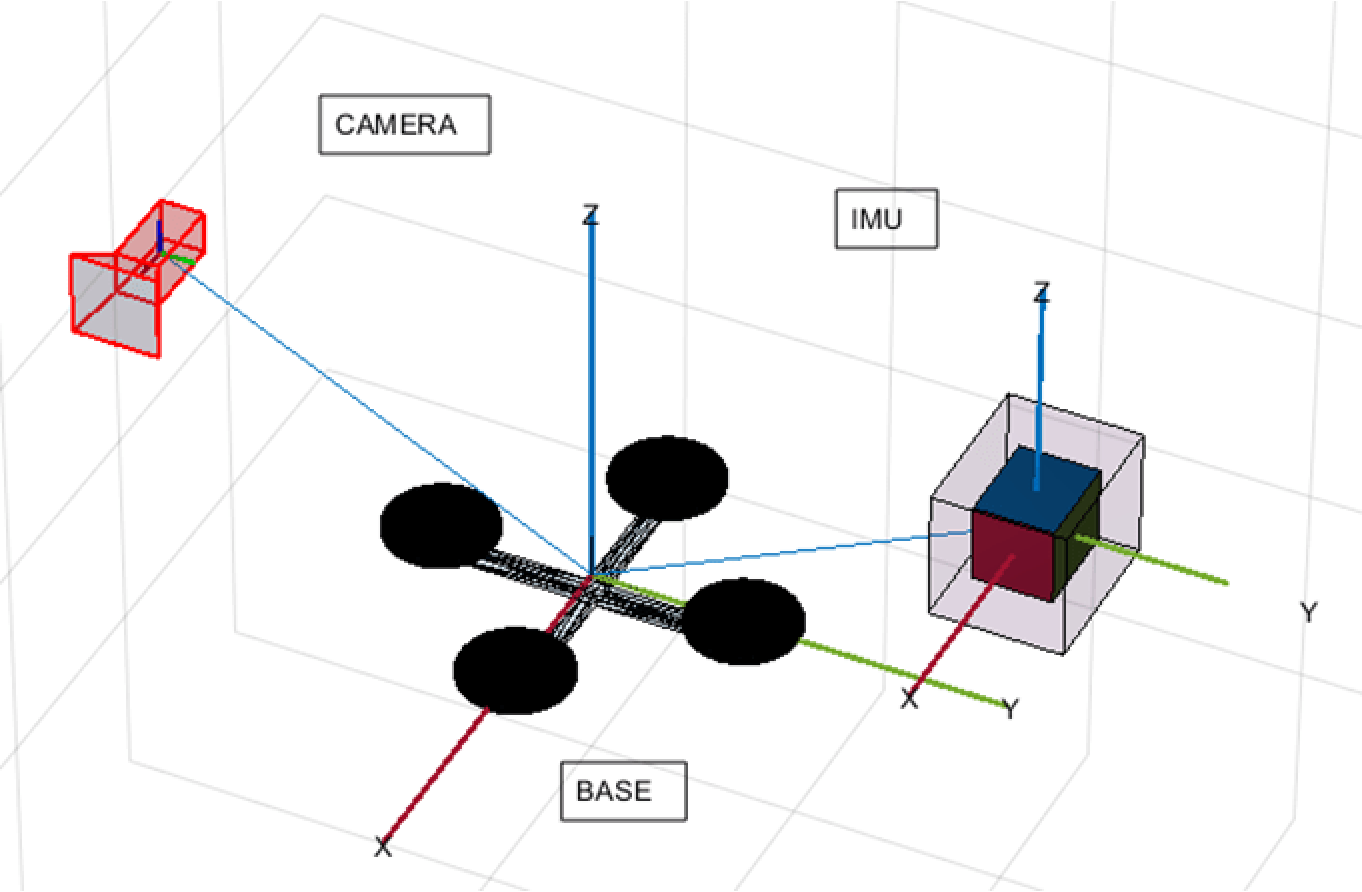
Visual Inertial Odometry for a Crazyflie drone

Group components:

Crivellari Daniele
Samorì Filippo
Ugolini Filippo

Visual Inertial Odometry Algorithm

- The Visual Inertia Odometry (VIO) is a **localization technique** for the estimation in **real time** of the position and orientation of an autonomous device in an environment.
- It combines **visual information** acquired throughout cameras and **inertial data** provided by IMU sensors.



From VIO to Real-World Applications

TRACKING OF COMPLEX TRAJECTORIES

Enabling high-accuracy flight paths for advanced robotics and drone research.

AUTONOMOUS FLIGHT

Laying the groundwork for missions where drones can fly without human intervention.

COMPUTER VISION

Using onboard cameras to perceive the surroundings and avoid obstacles in real time.

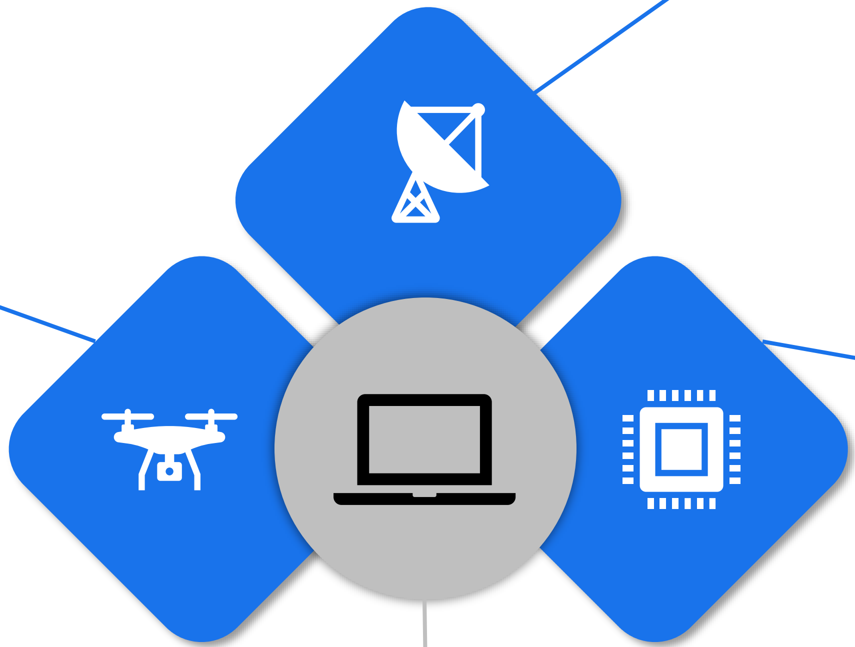
MULTI-DRONE COOPERATION

Exploring swarm behaviors and collaborative tasks among distributed autonomous systems.

Hardware components

Crazyradio 2.0:

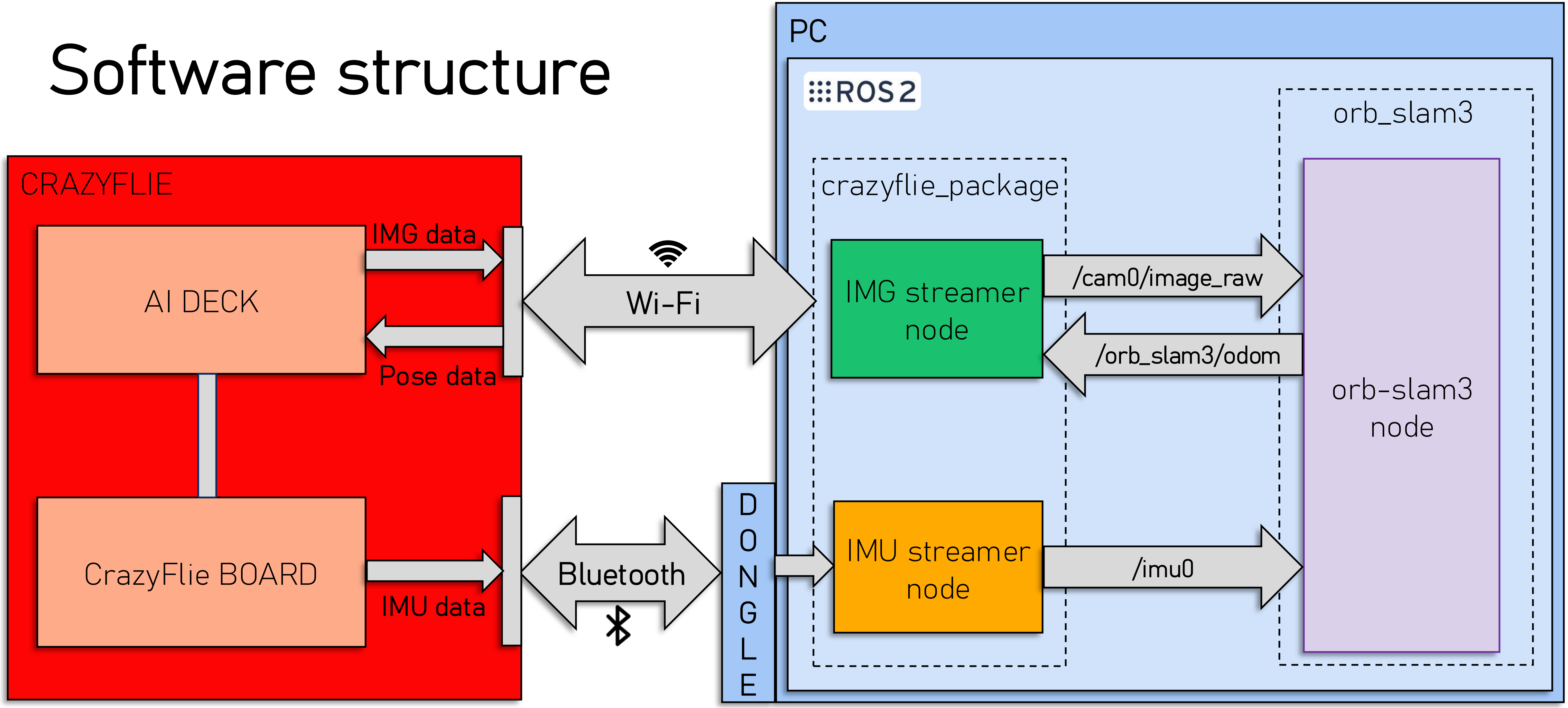
Crazyflie 2.1 drone:



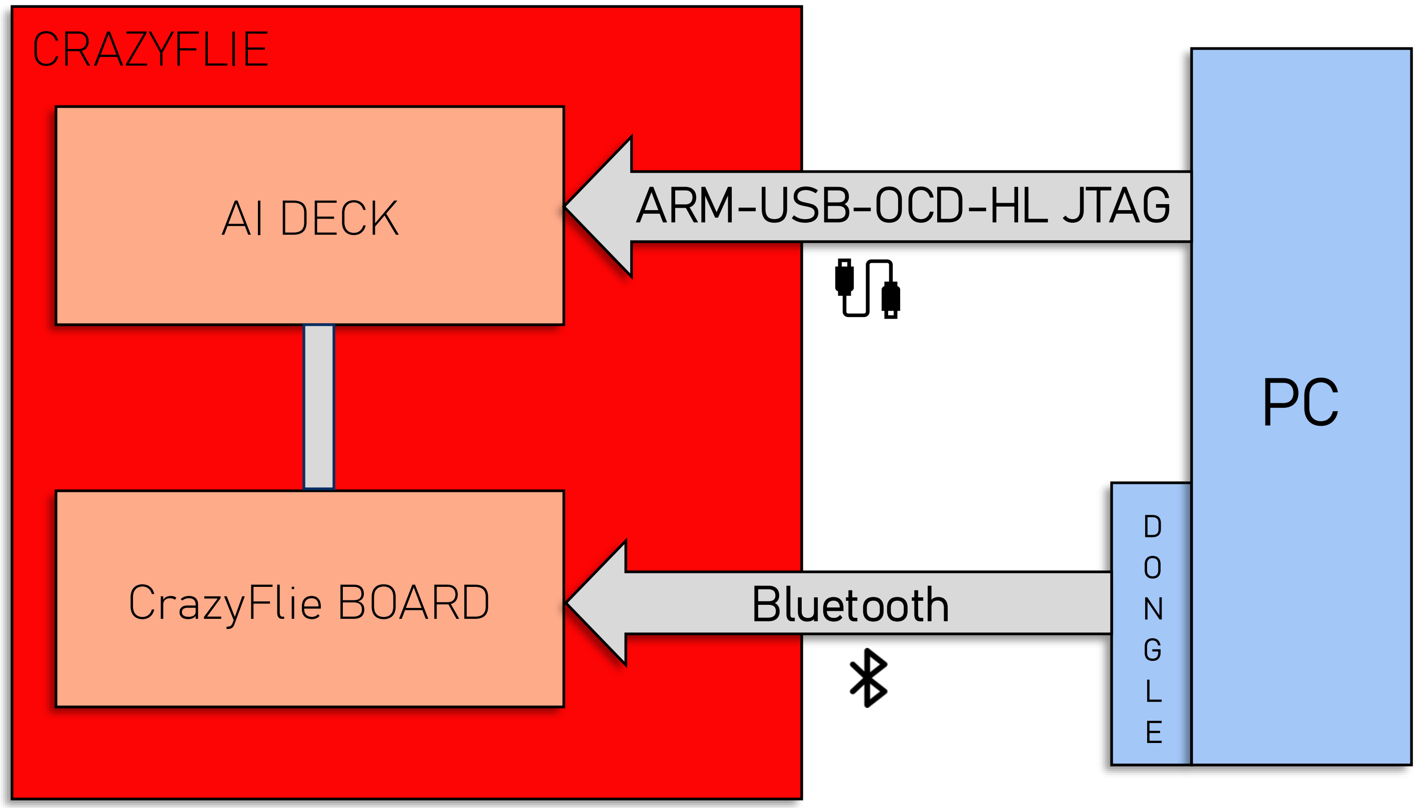
AI-deck:

PC for computations

Software structure



Section 1: Drone Setup



Firmware Setup

BUILD AND FLASH OF THE GAP8 BOOTLOADER WITH AN **OLIMEX ARM-USB-OCD-HL JTAG**

FLASHING OF THE FIRMWARE AND THE DEVELOPED APPLICATION FOR THE GAP8 VIA THE CRAZYRADIO DONGLE

GAP8 WiFi application

The application consists of three concurrent FreeRTOS tasks:

RX_TASK

Manages WiFi communications and connection status.

RX_FROM_WIFI

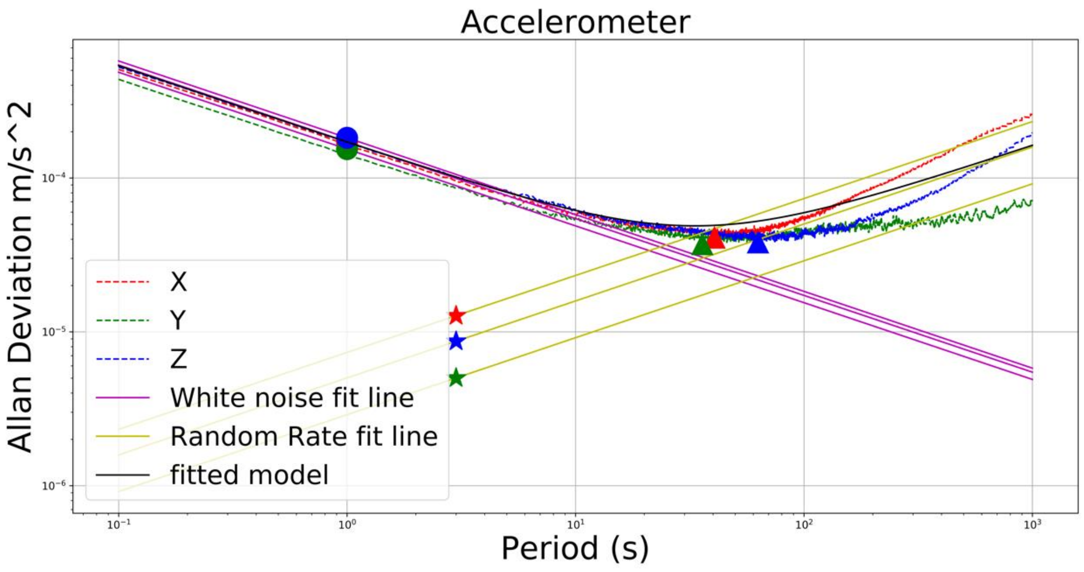
Receives estimated position from PC and computes communication latency.

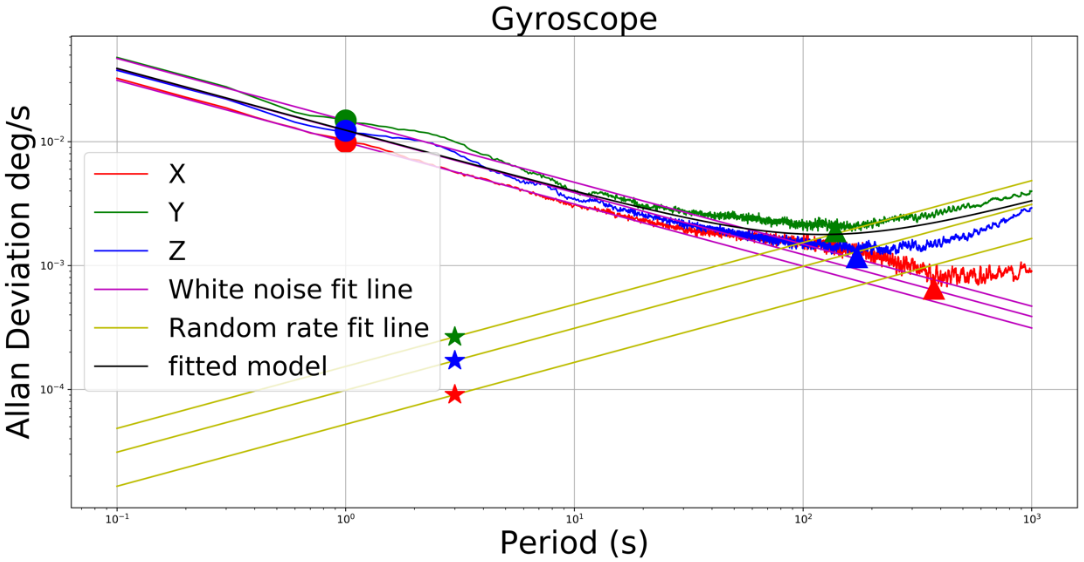
CAMERA_TASK

Continuously captures 324x244 grayscale images from the Himax camera and transmits them to the PC via WiFi together with the associated timestamp. The task operates at the maximum frequency allowed by communication bandwidth and hardware performance constraints.

IMU Calibration

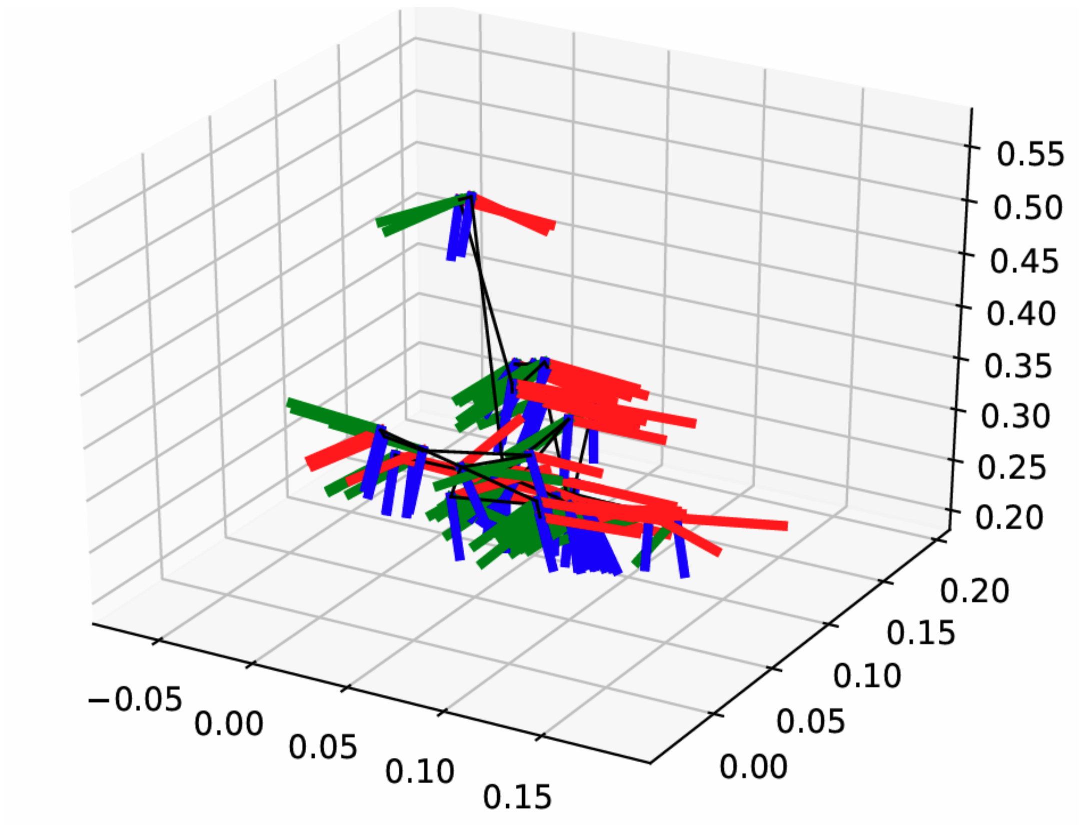
Sensor	Noise Density	Random Walk
Accelerometer	$0.000176\frac{m}{\sqrt{s^3}}$	$3.4328\times10^{-5}\frac{m}{\sqrt{s^5}}$
Gyroscope	$0.000260\frac{rad}{\sqrt{s}}$	$2.0043\times10^{-5}\frac{rad}{\sqrt{s^3}}$





Github repository: [allan_variance_ros]()

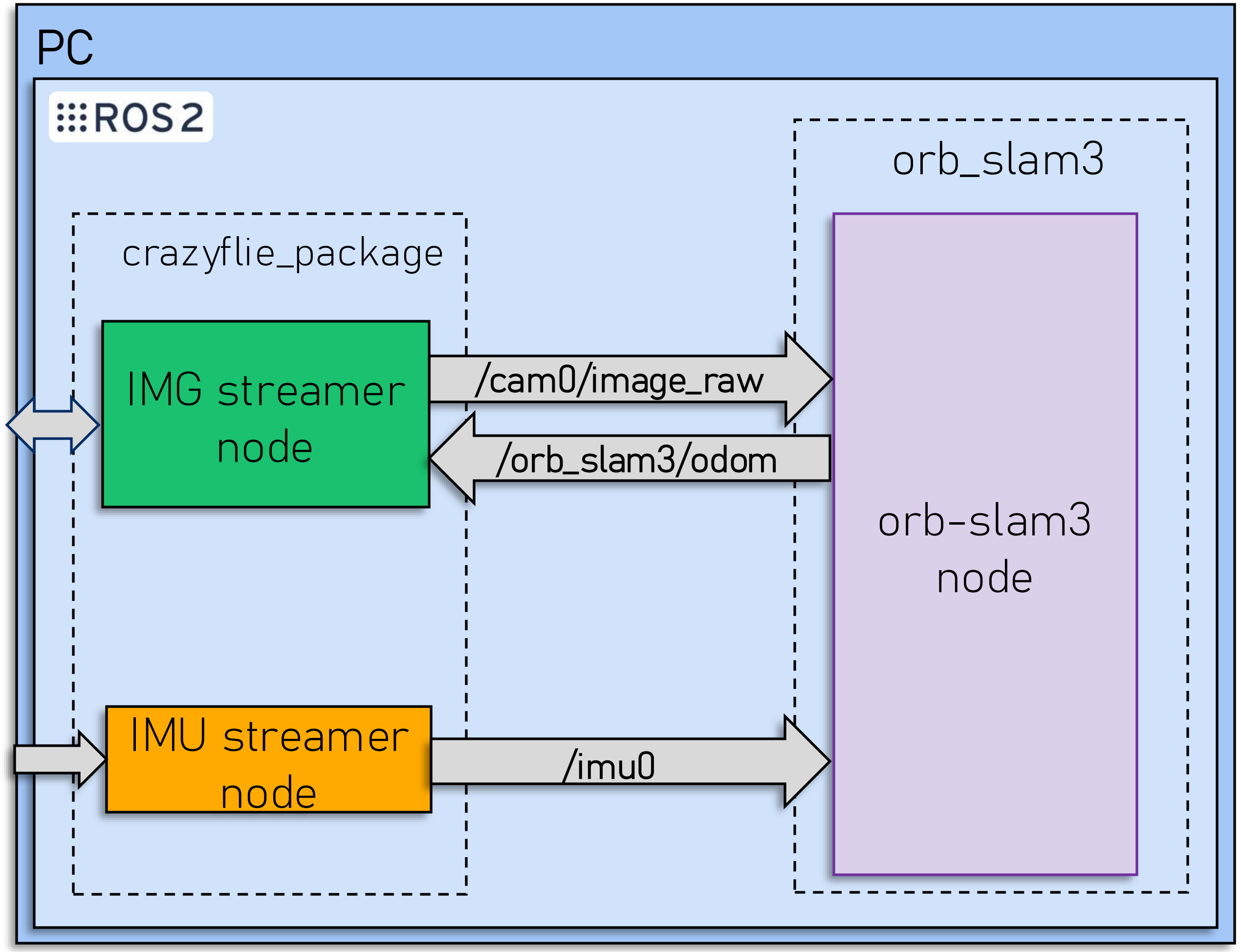
IMU + Camera Calibration



Parameter	Value
Focal length	$[182.040108, 181.960582]$
Principal point	$[161.919551, 153.772955]$
Radial distorsion coefficients	$[-0.072991,\ -0.005429]$
Tangential distorsion coefficients	$[-0.000866,\ 0.000639]$
Rotation matrix	$\begin{bmatrix} -0.00658 & -0.99998 & -0.00051 \\ -0.22460 & 0.00198 & -0.974445 \\ 0.97443 & -0.00629 & -0.22461 \end{bmatrix}$
Translation vector	$[-0.0292 \quad 1.1811 \quad 0.6743]^T$

Github repository: kalibr

Section 2: ROS2 environment



IMU STREAMER NODE

01 The *imu_streamer* node is initiated and it connects to the drone via the Crazyradio dongle

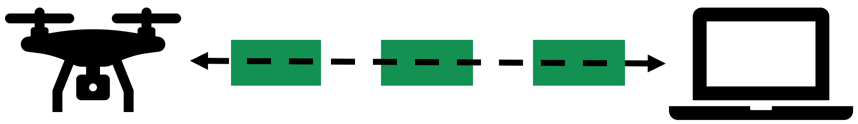
02 The node starts logging IMU data from the drone at 100Hz:
- *Accelerometer data*
- *Gyroscope data*

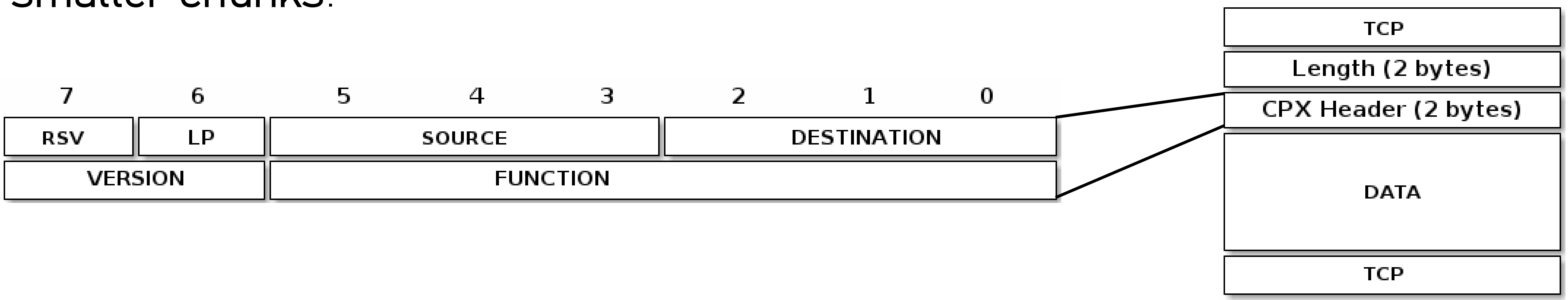
03 A ROS2 message of type *Imu()* is instantiated and filled with the logged values

04 The Imu message is then published in the */imu0* topic

WI-FI COMMUNICATION PROTOCOL

The image communication is performed using the **TCP protocol**, with data packets structured according to the custom convention CPX. Due to the images size, they are split into several **smaller chunks**.





IMG STREAMER NODE

01 The *img_streamer* node is initiated, and it connects to the AI-deck via WIFI

02 The node reconstructs the images reshaping it into a 244 x 324 grayscale image

03 A ROS2 message of type *Image()* is filled with the image (with a bridge)

04 The message is published on the topic */cam0/image_raw* with the associated TimeStamp

05 Receives estimated pose on the topic */orb_slam3/odom* and sends it back to the drone via WiFi

ORB_SLAM3 NODE

orb_slam3

orb_slam3.cpp

- Subscribes to image and camera topics.
- Manages those data through an instance of the DataGrabber class.
- Sends back to img_streamer node the estimated pose.

data_grabber library

- Receives image and IMU messages and queues them internally using buffers.
- Feed synchronized data to the ORB-SLAM3 tracking system.
- Receives estimated pose.

VIO algorithm

ORB_SLAM3 VIO ALGORITHM

ORB-SLAM3 is a real-time SLAM system that supports **visual, visual-inertial, and multi-map SLAM** using **monocular, stereo, and RGB-D cameras**, compatible with both **pin-hole and fisheye lens models**.

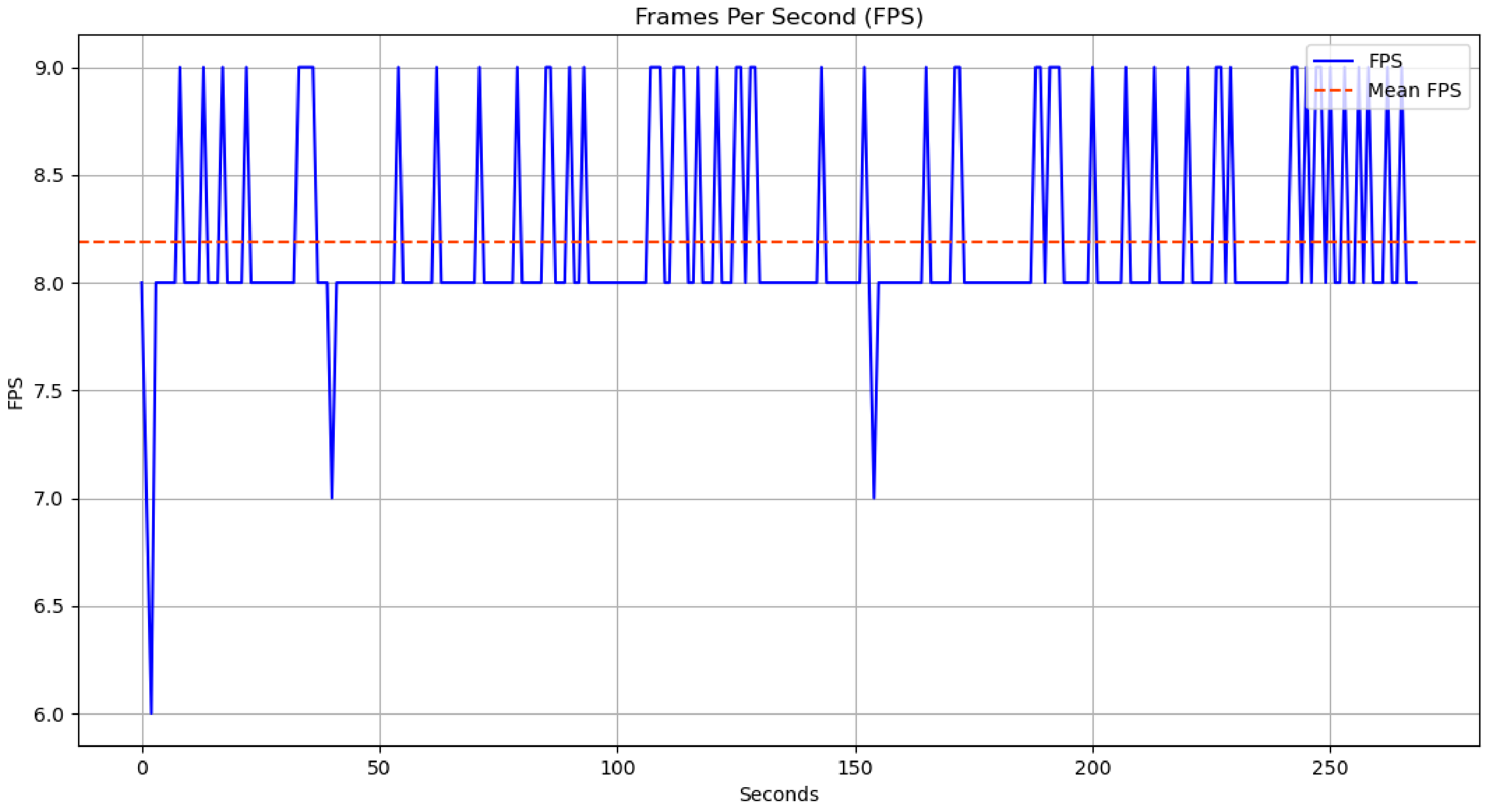
It **extracts ORB features** from the image while **integrating IMU data**, then defines the MAP-based optimization problem.

Optimization solved through tightly-coupled bundle adjustment, minimizes visual reprojection and inertial errors.

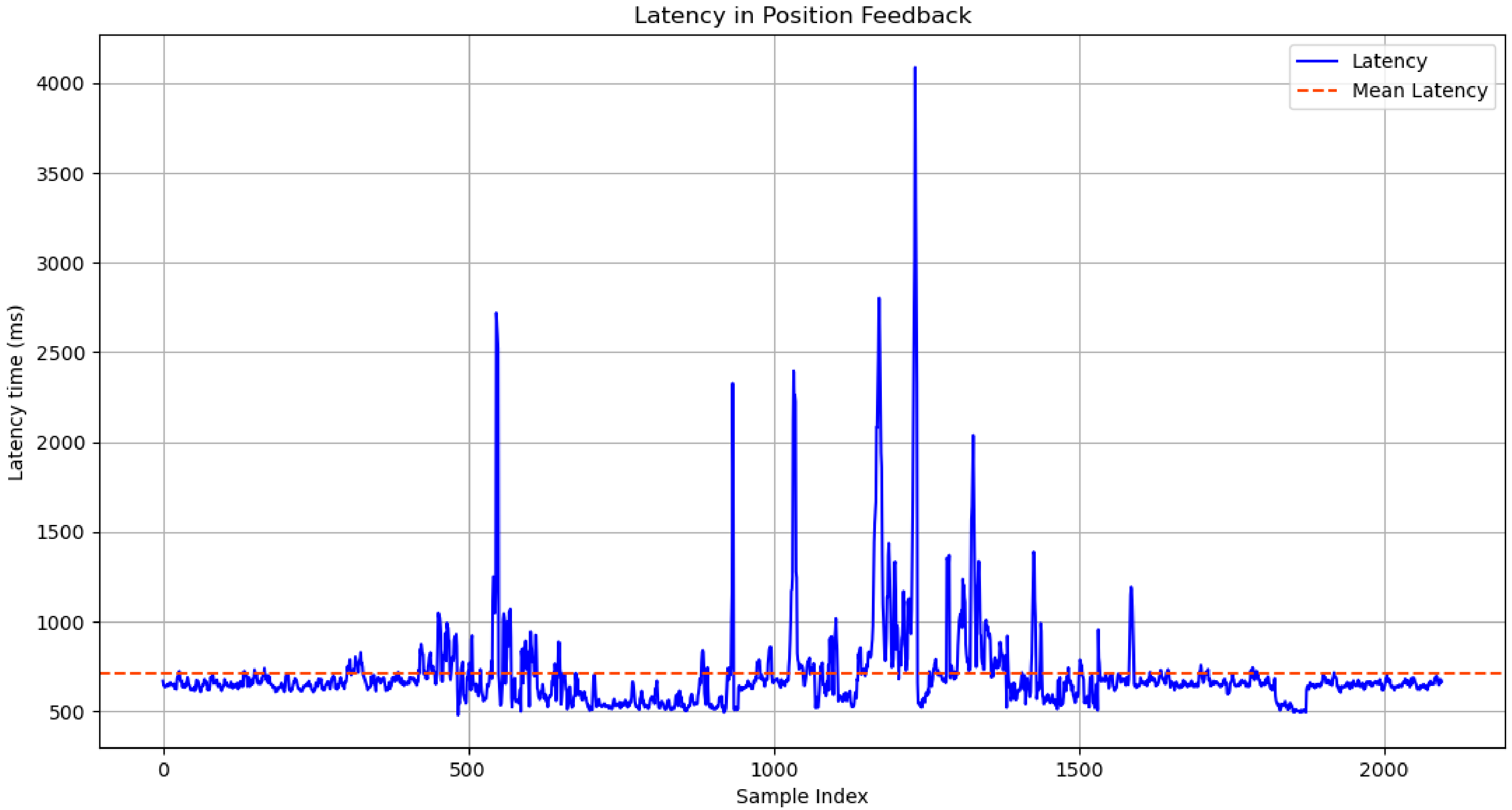
Performance enhanced by DBoW2-based place recognition for **loop closure** and **drift correction**.

Section 3: Results

Performances: FPS at runtime

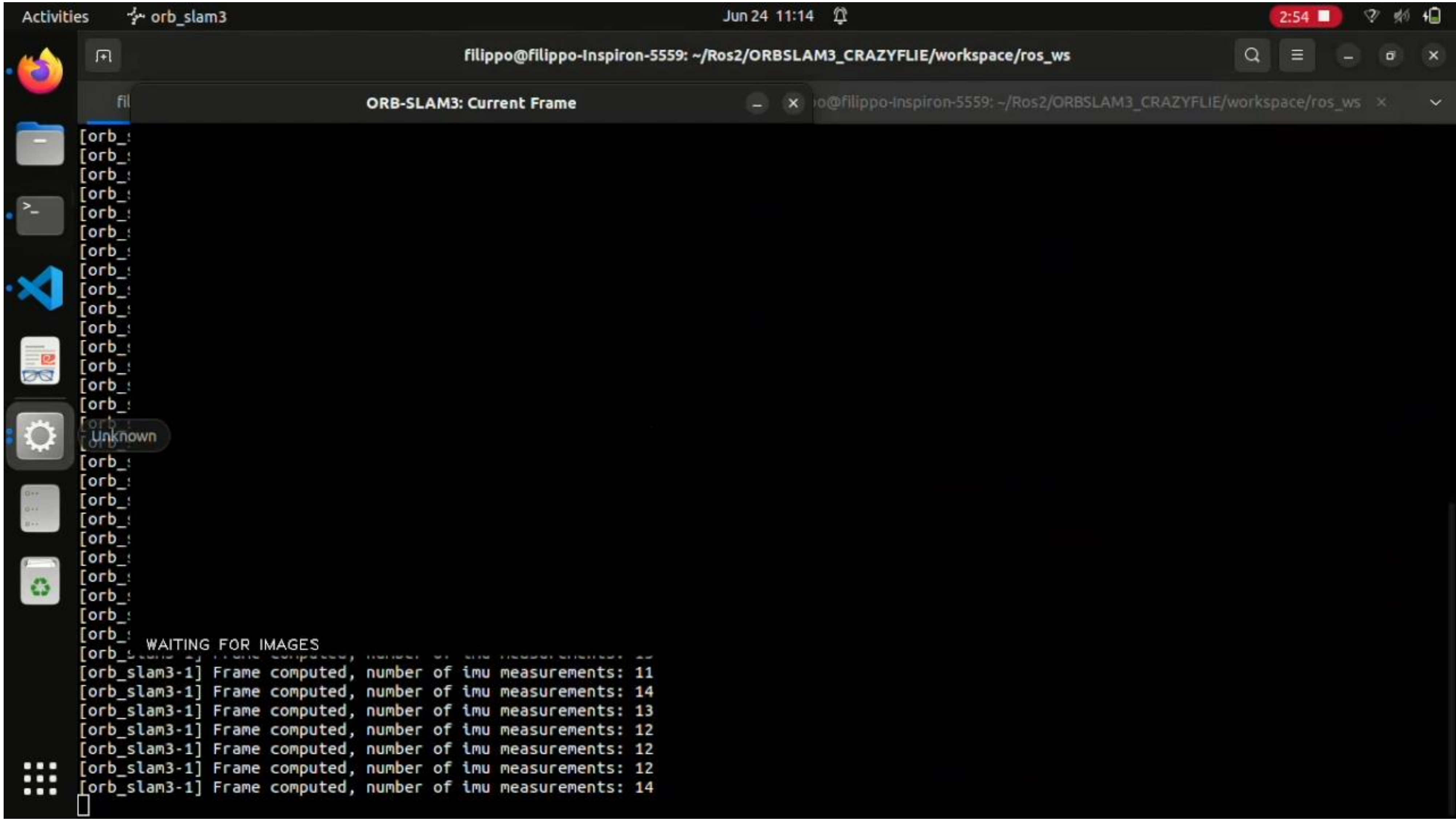


Performances: Latency at runtime



Video

Real-time demo of the algorithm running in a rectangular environment..



Conclusions

Main goal achieved: Successful integration of Crazyflie, AI-deck, and ROS2 using dual communication (Wi-Fi + Bluetooth) for a successful Visual Inertial Odometry in **real time**.

Challenges addressed:

- Logging of data from both IMU and Camera;
- Integration of different frameworks (Orb Slam 3, ROS2, CF library, GAP8 programming);
- Performance improvement (i.e. FPS).

System limitations:

- Communication constraints;
- Poor camera resolution.

Ideas for performance improvements

Develop the application outside the ROS2 framework for increased performances.

Define a better strategy to use a single channel for the communication.

Improve the camera system (higher resolution, stereo or RGB-D cameras).

THANKS FOR YOUR ATTENTION!