EuRoC Project

UAV control application for an European Challenge

Bartolomeo Della Corte

Marco Ferro

What is EuRoC

SAPIENZA
UNIVERSITÀ DI ROMA

Reconfigurable Interactive Manufacturing Cell

Shop Floor Logistics and Manipulation

Plant Inspection And Servicing

European Robotics Challenges

KUKA
DLR
ASCENDING TECHNOLOGIES
Fraunhofer IPA
LAAS-CNRS
ETH zürich
ALSTOM Inspection Robotics
INNOCENTIVE

Stages of the Challenge

European Robotics Challenges

- **Stage I – QUALIFYING: Simulation Contest**

  The simulation contests are ranked according to objective metrics (criteria and grading system). The best 45 contestants (3 × 15) are selected based on their scores in the tests, and they become **Perspective Challengers**.
  Prospective challengers are given an opportunity to form teams with system integrators and end users and submit short proposals, of which the best 3 x 5 will be selected to become the official Challenger Teams (03/2015).

- **Stage II – REALISTIC LABS: Benchmarking, free-style and showcase**

  Round A (benchmarking + free-style).
  Round B (showcase).
  Challenger Teams will be ranked according to objective metrics (criteria and grading system). 3 x 2 Challenge Finalists will be selected for the Field Tests stage of each challenge (12/2016).

- **Stage III – FIELD TESTS: Pilot Experiments**

  This last stage involves much engineering effort because the general solutions developed during the Realistic Labs stage will be customised for end users and tested on the field. A EuRoC Winner will be selected by the BoJ (12/2017).

Challenge 3 Contestants

- **35 partecipants (9 italian Universities/Laboratories)**
  - ACTLAB (*Università di Parma*)
  - ARS (*Università del Salento*)
  - CASY (*Università di Bologna*)
  - Laborics PSI (*private*)
  - PEGASUS (*Scuola Superiore Sant'Anna*)
  - Polibrì (*Politecnico di Milano*)
  - Robo-Team *(Campus-Bio-Medico di Roma)*
  - RomaUno *(Università La Sapienza di Roma)*
  - UNIPI (*Università di Pisa*)

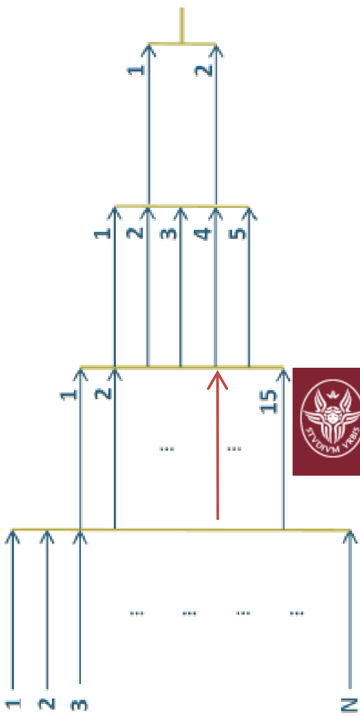
- **21 of them submitted a solution**

Challenge 3 Contestants

1. ACTLAB
2. Attempto Tuebingen
3. CVG-UPM
4. Eiffel Team
5. Eyefly
6. First ROS Team of Kosice
7. Graz Griffins
8. GRVC-CATEC
9. LEO
10. MIRIAMM
11. NimbRo Copter
12. Polibrì
13. proaut-autec
14. Robo-Team
15. RomaUno
16. RPG
17. TUM Flyers
18. Unikorn
19. UNIPI
20. UNIZG-FER
21. Vicomtech-IK4

SAPIENZA
UNIVERSITÀ DI ROMA

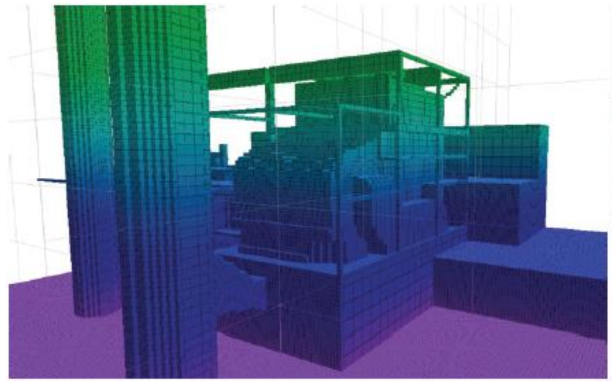
The Challenge Chart

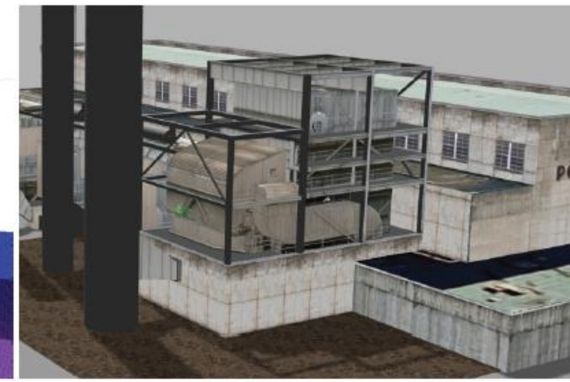


Challenge 3

Rank	Team Name	Total Score
1	TUM Flyers	90.5
2	UNIZG-FER	82.5
2	Eiffel Team	82.5
4	NimbRo Copter	77.0
5	RPG	75.0
6	Graz Griffins	74.5
7	MIRIAMM	73.5
8	Attempto Tuebingen	73.0
9	GRVC-CATEC	46.5
10	TU-Chemnitz Proaut	45.0
11	RomaUno	43.0
12	LEO	37.5
13	Polibrì	37.0
14	Unikorn	35.0
15	ACTLAB	25.0

Challenge 3: Plant Servicing and Inspection





- Type of robot: Hexacopter MAV
- Track 1: Vision-based localization and reconstruction
  - Task 1
  - Task 2
- Track2: State estimation, control and navigation
  - Task 3
  - Task 4

Task 3

- Subtask 3.1: simply keep **hovering** at the starting point
- Subtask 3.2: keep hovering with a **constant wind** applied
- Subtask 3.3: keep hovering with a **wind gust** applied

- ❑ Some **benchmarks** are defined in each subtask, in order to assign a **score** to the designed solution

- ❑ Contestants' solutions are designed under **ROS framework** (**R**obot **O**perating **S**ystem)

ROS

Outlines

- MAV System Analysis
- Designed Solution Description
- Simulations & Results

- The robot architecture provided for the Challenge is an **Hexacopter MAV**
- Structure and model comparable to the well-known Quadcopter (**six** blades instead of **four** ...)
  - **Control inputs** are the same: **Thrust** + **torques** on RPY angles
  - **Motor velocities mapping** differs because of the number of blades:

Quadcopter

$$f_i = b\,\omega_i^2$$

$$\tau_{R,i} = d\,\omega_i^2$$

$$T = f_1 + f_2 + f_3 + f_4$$

$$\tau_\varphi = l(f_2 - f_4)$$

$$\tau_\vartheta = l(f_1 - f_3)$$

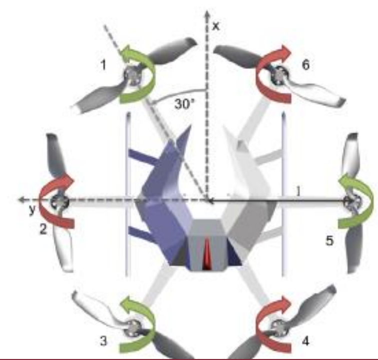
$$\tau_\psi = -\tau_{R,1} + \tau_{R,2} - \tau_{R,3} + \tau_{R,4}$$

Hexacopter

$$\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ T \end{bmatrix} = \mathrm{diag}\left(\begin{bmatrix} b \cdot l \\ b \cdot l \\ d \\ b \end{bmatrix}\right) \cdot \begin{bmatrix} s & 1 & s & -s & -1 & -s \\ -c & 0 & c & c & 0 & -c \\ -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_1^2 \\ \omega_5^2 \\ \omega_6^2 \end{bmatrix}$$

$$s = \sin(30°);\ c = \cos(30°)$$

$$l = 0.215\,\mathrm{m}$$



MAV Sensory Equipment

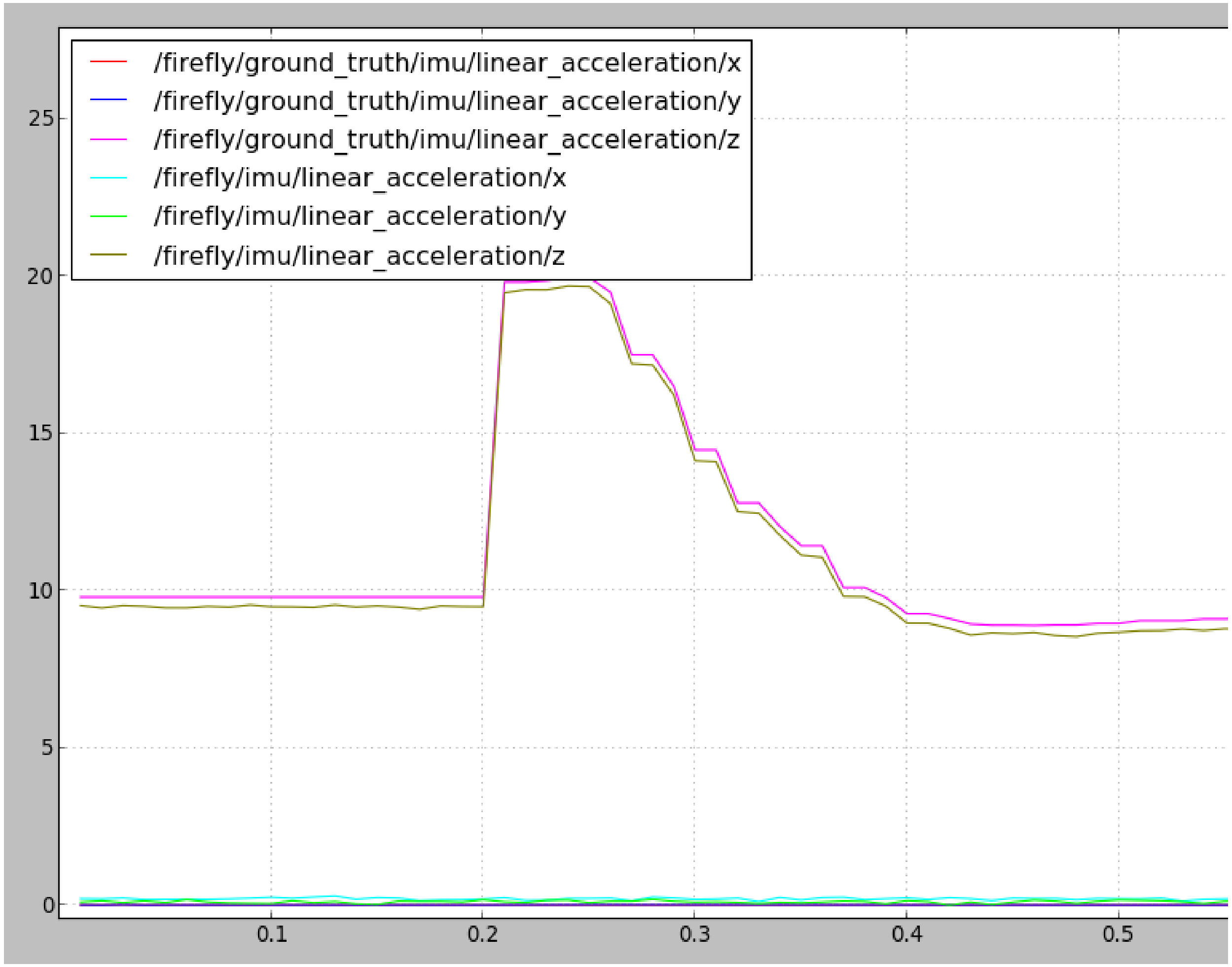
- The MAV model provided is equipped with two **sensors**:
  - A **noisy IMU sensor** providing:
    - *MAV Orientation*
    - *MAV linear acceleration*
    - *MAV angular rate*
  - A **6-DoF Pose sensor** providing:
    - Sensor position
    - Sensor orientation

*Not a **real** sensor! It abstracts a vision-based localization approach*

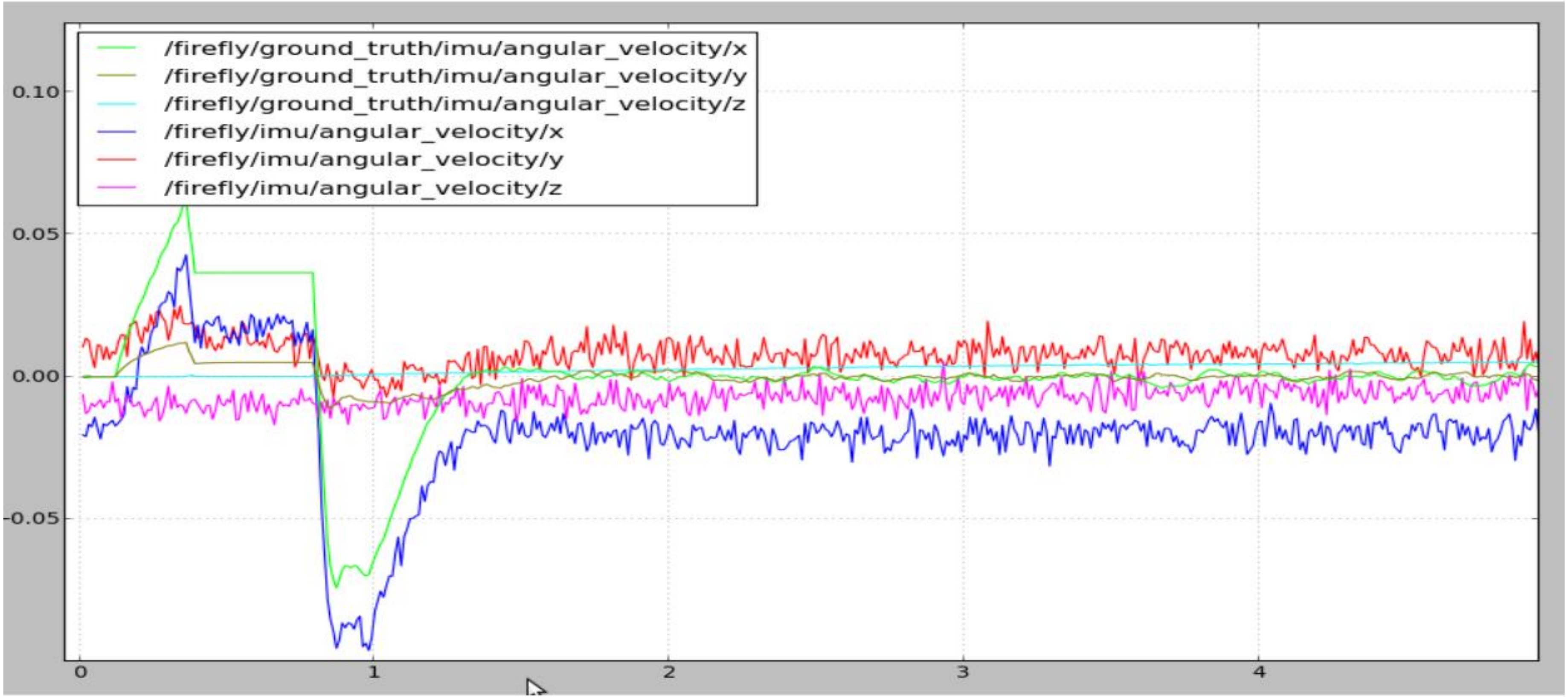
*MAV and Sensor frames do **not** coincide!*

Sensor Noise Corruption

- IMU sensor: *MAV linear acceleration*



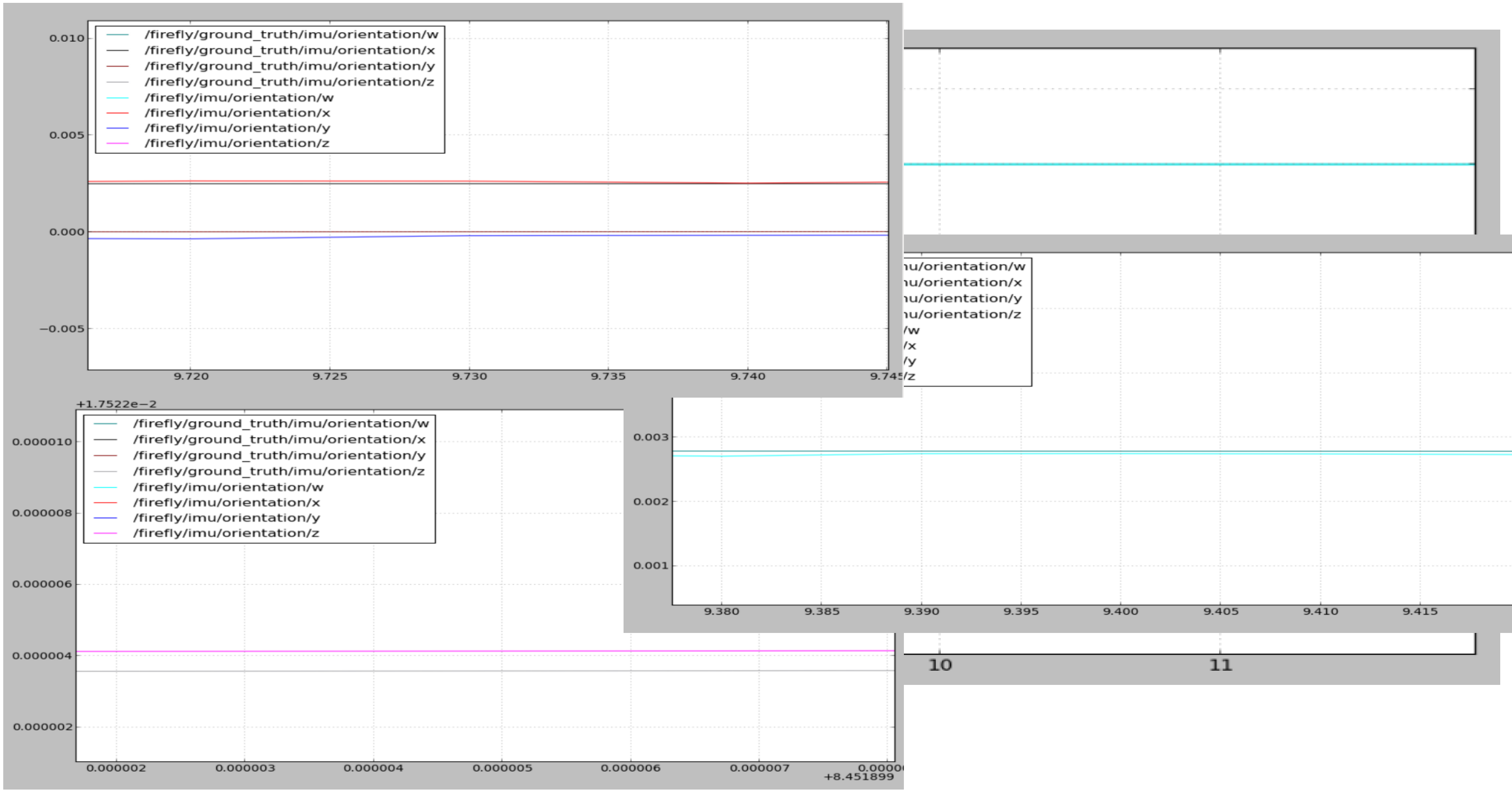
- IMU sensor: *MAV angular rate*



Sensor Noise Corruption

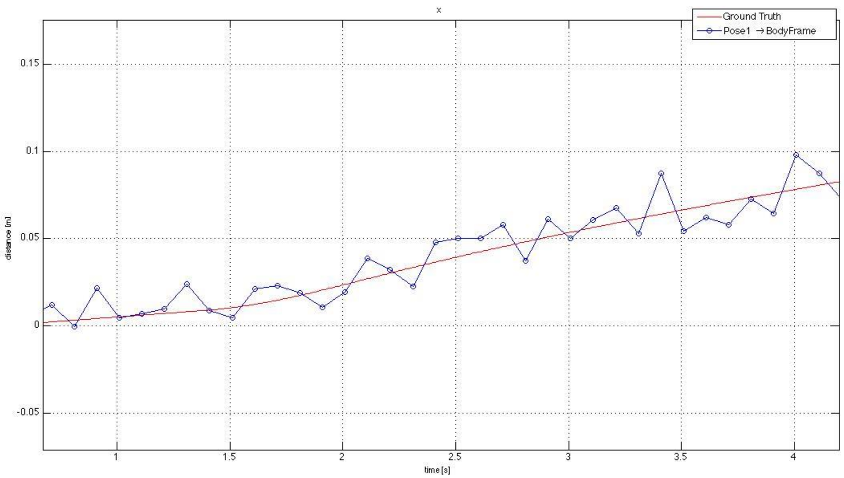
SAPIENZA
UNIVERSITÀ DI ROMA

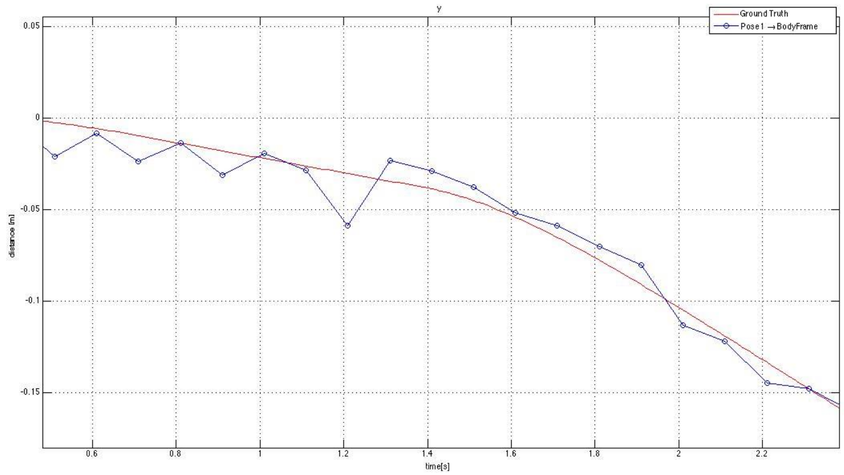
- IMU sensor: *MAV orientation*

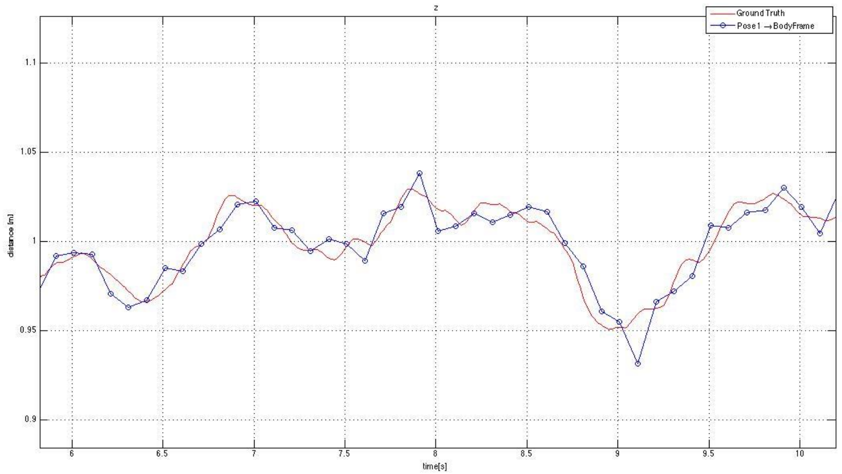


Sensor Noise Corruption

- Pose Sensor







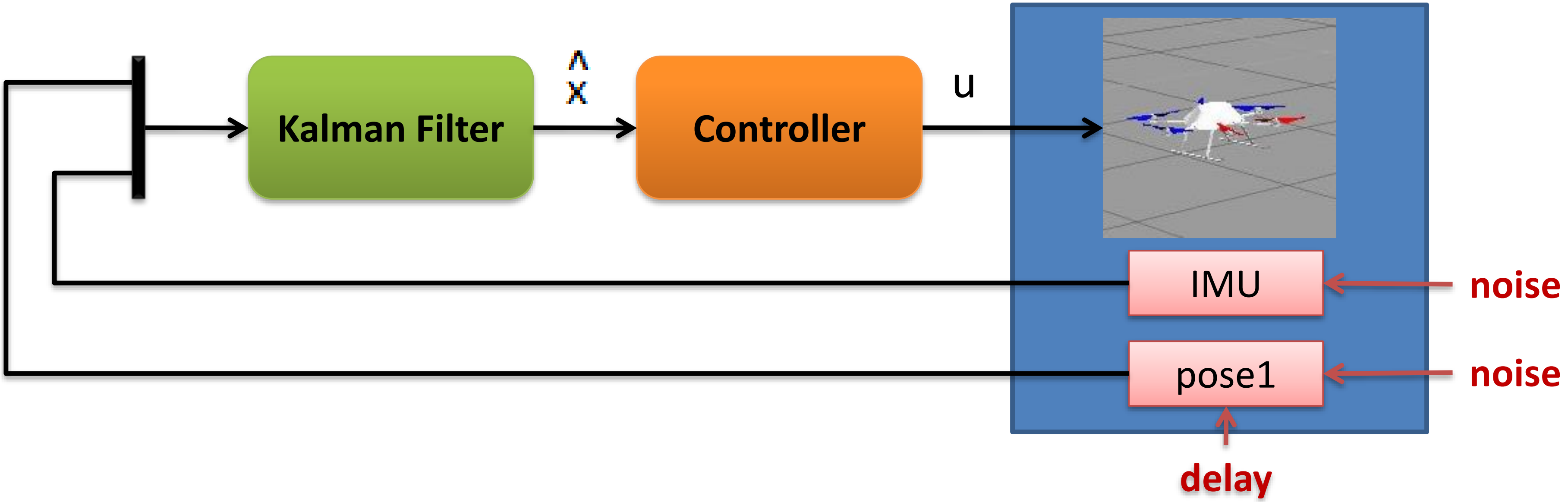
Outlines

- MAV System Analysis
- Designed Solution Description
- Simulations & Results

Filtering to reject noise

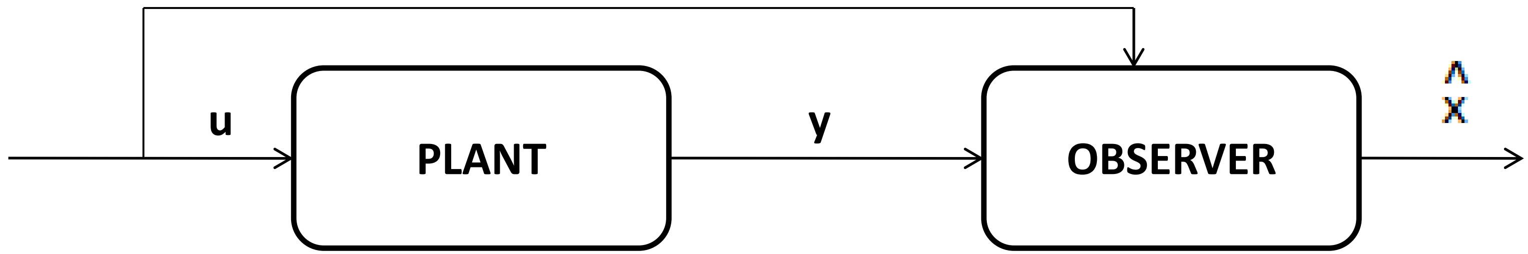
- The highly noisy nature of the sensor data prevents us to rawly use them in order to accomplish the assigned control tasks

- A **filter** is usually adopted in order to reject noise coming with corrupted data, so that the control modules are fed with more reliable inputs
- We choose to implement an **Extended Kalman Filter**



Kalman Filter

- A **Kalman Filter** is an observer that **estimates** the state of a dynamic system, if not directly available



- Built in two steps:
  Prediction step: *process dynamics* is used in order to generate an intermediate estimate of the state
  Update step: the intermediate estimate is corrected according to the *measured output*

Extended Kalman Filter

- An **Extended Kalman Filter (EKF)** is an observer for a **non-linear discrete-time system with noise**, with dynamics:

$$\begin{cases} \mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{v}_k \\ \mathbf{y}_k = \mathbf{h}_k(\mathbf{x}_k) + \mathbf{w}_k \end{cases} \qquad \begin{aligned} \mathbf{v}_k &\sim \mathcal{N}(0, \mathbf{V}_k) \\ \mathbf{w}_k &\sim \mathcal{N}(0, \mathbf{W}_k) \end{aligned}$$

- **State and Covariance Prediction**:

$$\hat{\mathbf{x}}_{k+1|k} = \mathbf{f}_k(\hat{\mathbf{x}_k}, \mathbf{u}_k)$$
$$\mathbf{P}_{k+1|k} = \mathbf{F}_k \mathbf{P}_k \mathbf{F}_k^T + \mathbf{V}_k$$

$$\mathbf{F}_k = \left. \frac{\partial \mathbf{f}_k}{\partial \mathbf{x}} \right|_{\mathbf{x} = \hat{\mathbf{x}}_k}$$

- **State and Covariance Update:**

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_{k+1|k} + \mathbf{R}_{k+1} \nu_{k+1}$$
$$\mathbf{P}_{k+1} = \mathbf{P}_{k+1|k} - \mathbf{R}_{k+1} \mathbf{H}_{k+1} \mathbf{P}_{k+1|k}$$

$$\mathbf{H}_{k+1} = \left. \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{x}} \right|_{\mathbf{x} = \hat{\mathbf{x}}_{k+1|k}}$$

innovation $\nu_{k+1} = \mathbf{y}_{k+1} - \mathbf{H}_{k+1} \hat{\mathbf{x}}_{k+1|k}$

Kalman Gain $\mathbf{R}_{k+1} = \mathbf{P}_{k+1|k} \mathbf{H}_{k+1}^T (\mathbf{H}_{k+1} \mathbf{P}_{k+1|k} \mathbf{H}_{k+1}^T + \mathbf{W}_{k+1})^{-1}$

EKF in our problem

SAPIENZA UNIVERSITÀ DI ROMA

- The state to be estimated for the hexacopter system is given by:

$$\mathbf{x} = [{}^w\mathbf{p}, {}^w\mathbf{v}, \mathbf{b}_a]^T$$

where:

- ${}^w\mathbf{p}$ is the *MAV position in the world frame;*
- ${}^w\mathbf{v}$ is the *MAV velocity in the world frame;*
- $\mathbf{b}_a$ is the *accelerometer bias*

- **Initialization**:

$$\mathbf{x}_0 = \Big[\underbrace{0, 0, 0.08}_{{}^w\mathbf{p}}, \underbrace{0, 0, 0}_{{}^w\mathbf{v}}, \underbrace{0.2, 0.1, -0.3}_{\mathbf{b}_a}\Big]^T$$

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0}_{3\times6} \\ \mathbf{0}_{6\times3} & \mathbf{0}_{6\times6} \end{bmatrix}$$

$$\mathbf{P}_{0|0} = \mathbf{0}_{9\times9}$$

$$\mathbf{W}_k = \begin{bmatrix} 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix}$$

$$\mathbf{V}_k = diag(0.00000000016 * \mathbf{I}_{3\times3}, diag(0.0000000016 * \mathbf{I}_{3\times3}), diag(0.00000016 * \mathbf{I}_{3\times3}))$$

Prediction

- An **IMU-based propagation model** has been used: *IMU linear acceleration* and *angular rate* are used as *system inputs* in the **prediction step** (actually only linear acceleration)

$$a = a_{imu} - \hat{\mathbf{b}}_a$$

State Prediction

$$\hat{\mathbf{x}}_{k+1|k} = \mathbf{f}_k(\hat{\mathbf{x}_k}, \mathbf{u}_k)$$

$${}^w\hat{\mathbf{p}}_{k+1|k} = {}^w\hat{\mathbf{p}}_k + T_{imu}\,{}^w\hat{\mathbf{v}}_k + \frac{1}{2}T_{imu}^2(R(\mathbf{q})a + g)$$

$${}^w\hat{\mathbf{v}}_{k+1|k} = {}^w\hat{\mathbf{v}}_k + T_{imu}(R(\mathbf{q})a + g)$$

$$\hat{\mathbf{b}}_{ak+1|k} = \hat{\mathbf{b}}_{ak}$$

$\mathrm{R}(\mathbf{q})$ is the **Rotation matrix** expressing the orientation of the **body frame** wrt the **world frame**

$\mathbf{F}_k$ **non-zero entries**

$$T_{imu} = 0.01s$$

$$\frac{\partial^w\hat{\mathbf{p}}_{k+1|k}}{\partial^w\hat{\mathbf{p}}_k} = \mathbf{I}_{3\times3}, \quad \frac{\partial^w\hat{\mathbf{p}}_{k+1|k}}{\partial^w\hat{\mathbf{v}}_k} = T_{imu}\mathbf{I}_{3\times3}, \quad \frac{\partial^w\hat{\mathbf{p}}_{k+1|k}}{\partial^w\hat{\mathbf{b}}_{ak}} = \text{-}\ \frac{1}{2}T_{imu}^2\mathbf{R}(\mathbf{q}),$$

$$\frac{\partial^w\hat{\mathbf{v}}_{k+1|k}}{\partial^w\hat{\mathbf{v}}_k} = \mathbf{I}_{3\times3}, \quad \frac{\partial^w\hat{\mathbf{v}}_{k+1|k}}{\partial^w\hat{\mathbf{b}}_{ak}} = -T_{imu}\mathbf{R}(\mathbf{q}), \quad \frac{\partial^w\hat{\mathbf{b}}_{ak+1|k}}{\partial^w\hat{\mathbf{b}}_{ak}} = \mathbf{I}_{3\times3},$$

Covariance Prediction

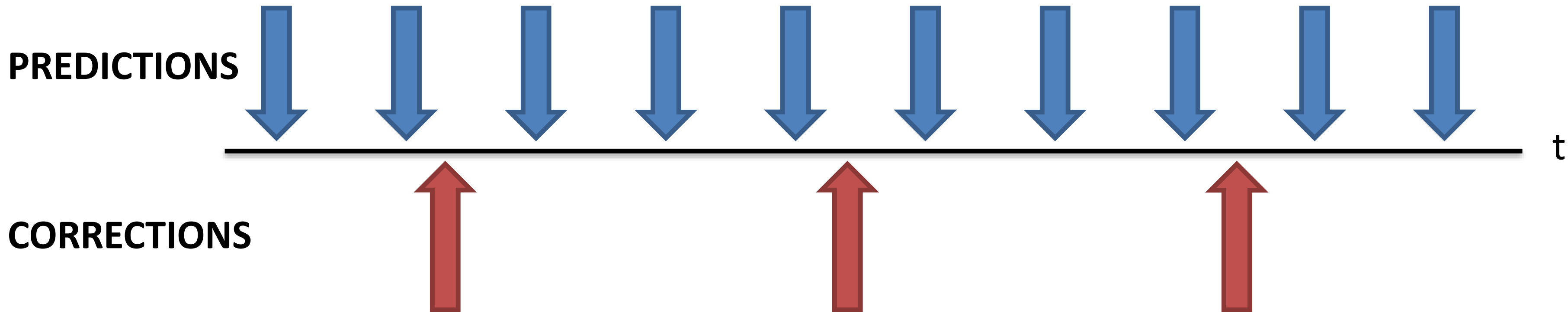
$$\mathbf{P}_{k+1|k} = \mathbf{F}_k\mathbf{P}_k\mathbf{F}_k^T + \mathbf{V}_k$$

Correction

- The Correction step is performed with data coming from the Pose Sensor, where

$$T_{pose} = 0.1s \quad (\neq T_{imu} = 0.01 \ !!!)$$

- The Prediction and Correction steps rates are clearly different: the result is that the filter applies a certain number of **predictions** before a new measurement arrives (and so, a **correction** is performed)



- The equations shown before (see State and Covariance Update) are then applied whenever a new Pose sensor message arrives

Filter Validation

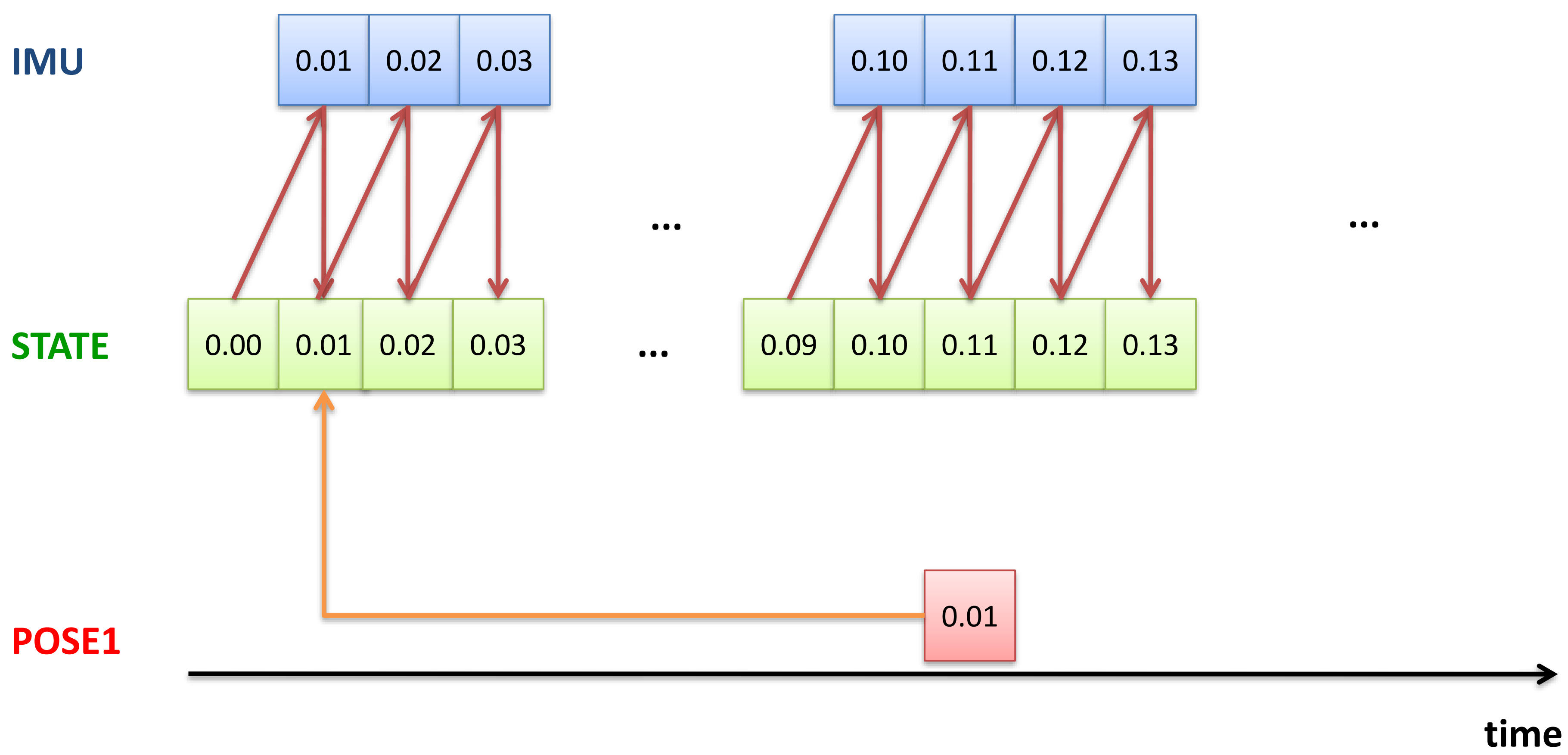
- In this way, **noise** and **low-rate** issues can be handled in the sensors...
- ... but Pose Sensor is still **delayed**!!
- In fact, Pose Sensor messages contain a *timestamp* field referring to a previous time instant, so the corresponding correction has to be applied on a properly previous prediction

- This does not cause so many troubles while working *off-line*, since a simple *timestamp comparison* is enough in order to apply the correction to the proper intermediate estimate

- On the other hand, when everything needs to work *on-line*, the **validation** on the virtual environment **Gazebo** has to include a **synchronization mechanism ...**

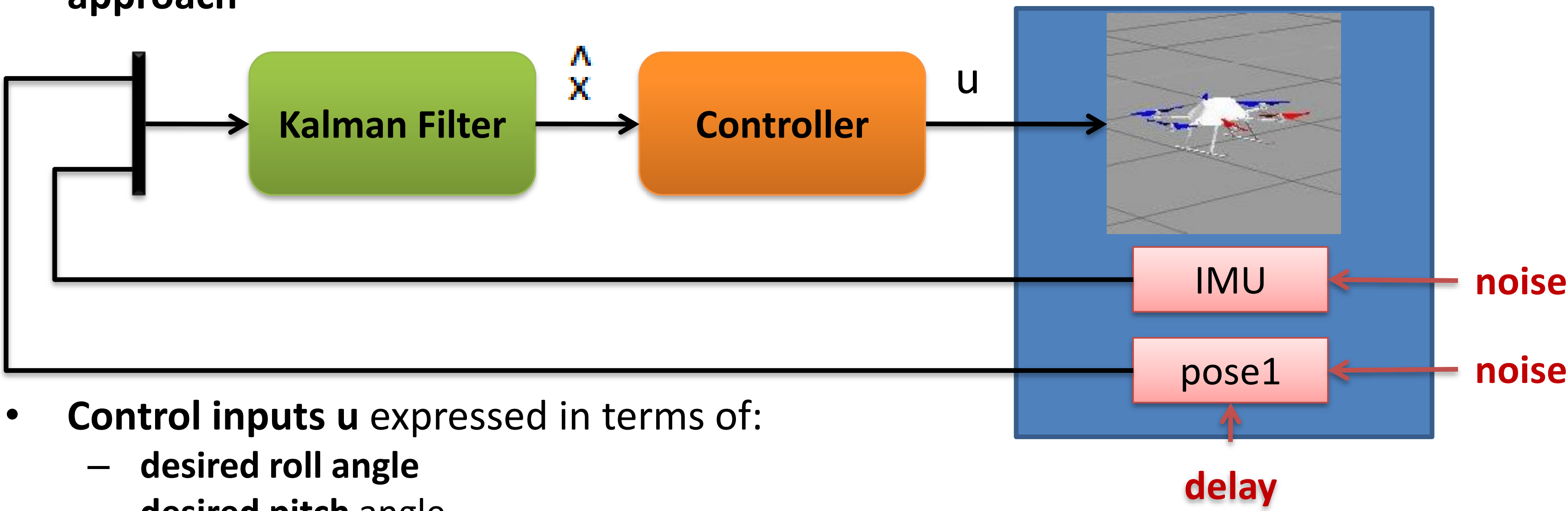
Synchronisation

SAPIENZA
UNIVERSITÀ DI ROMA



From filter ...
To control

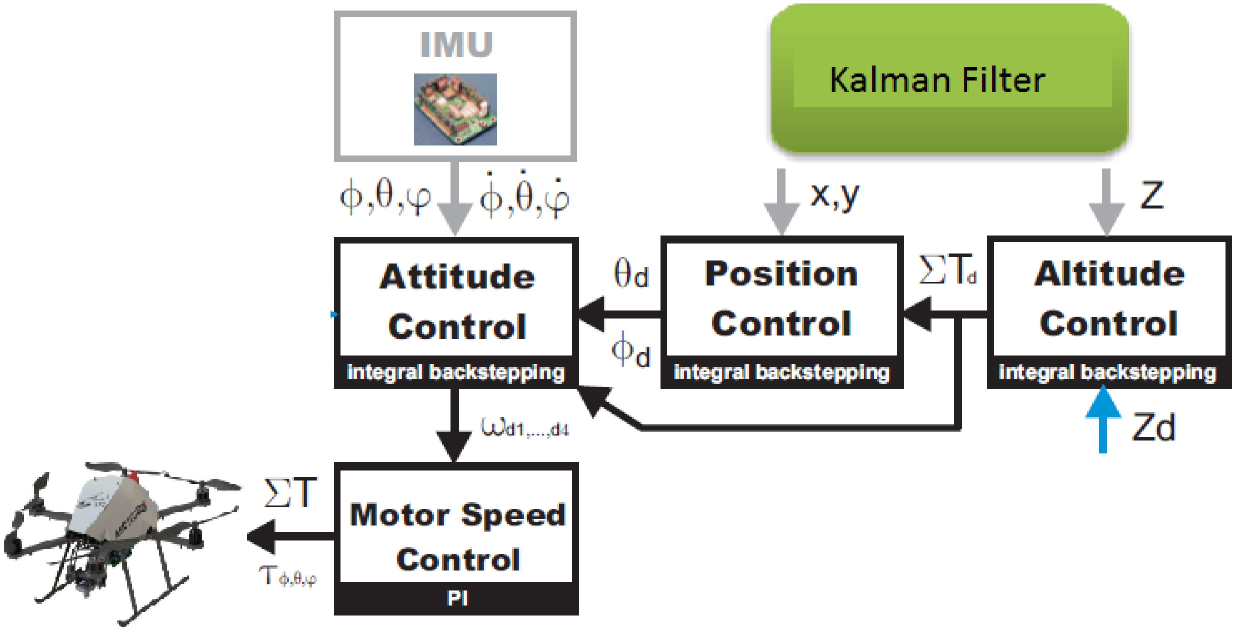
- Data coming from the Kalman Filter module are more reliable to be managed than noisy sensor data
- These data are used in order to feed the **Controller** module that allows the MAV to behave in a desired way by computing proper **control inputs**
- The chosen control paradigm for this application is an **Integral Backstepping approach**



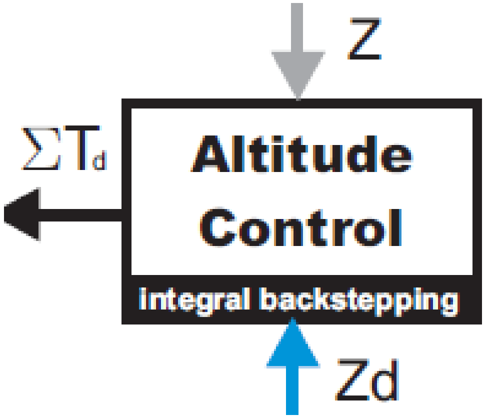
- **Control inputs u** expressed in terms of:
  - **desired roll angle**
  - **desired pitch** angle
  - **desired yaw rate**
  - **Thrust**

SAPIENZA UNIVERSITÀ DI ROMA

- **Integral Backstepping** paradigm differentiates three **control modules** for **altitude**, **position** and **attitude**
- In our node:
  - **Altitude** and **Position controller** have been implemented
  - **Attitude controller** is an inner module of the MAV provided by the organizers



Altitude Controller



- Altitude tracking error

$$e_z = z_d - z$$

- Altitude speed tracking error

$$e_{\dot{z}} = c_z e_z + \dot{z}_d + \lambda_z \chi_z - \dot{z}$$

$$\chi_i = \int_0^t e_i(\tau)\, d\tau$$
$$c_z, c_{\dot{z}}, \lambda_z > 0$$

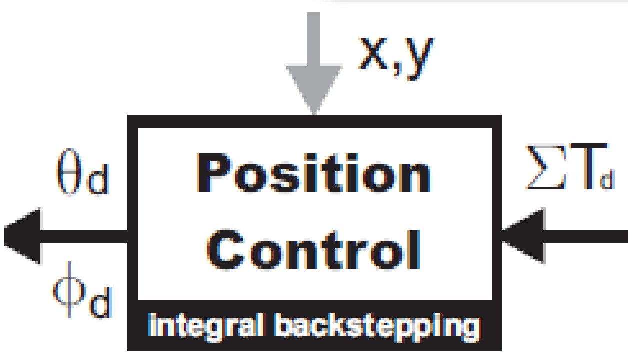
- Thrust control input

$$T = \frac{m}{\cos\phi\cos\theta} = \left[g + \left(1 - c_z^2 + \lambda_z\right) e_z + \left(c_z + c_{\dot{z}}\right) e_{\dot{z}} - c_z \lambda_z \chi_z\right]$$

Position Controller

- x- and y-tracking errors

$$e_x = x_d - x$$
$$e_y = y_d - y$$



- Speed tracking errors

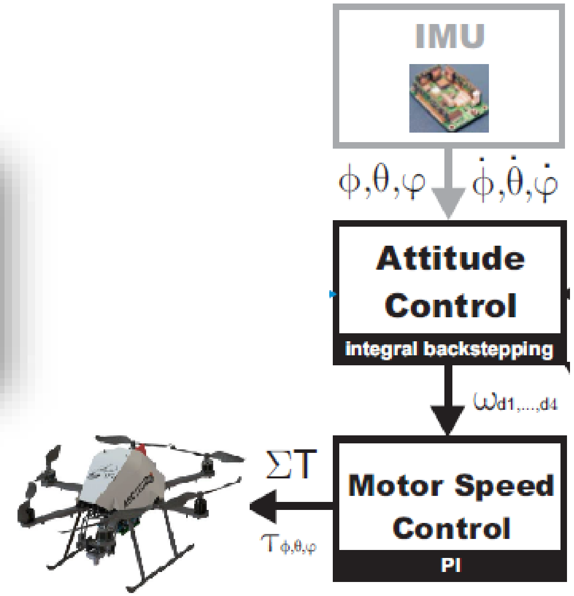
$$e_{\dot{x}} = c_x e_x + \dot{x}_d + \lambda_x \chi_x - \dot{x}$$
$$e_{\dot{y}} = cy e_y + \dot{y}_d + \lambda_y \chi_y - \dot{y}$$

- desired roll and pitch angles control inputs

$$\theta_d = \frac{m}{T}\left[\left(1 - c_x^2 + \lambda_x\right) e_x + \left(c_x + c_{\dot{x}}\right) e_{\dot{x}} - c_x \lambda_x \chi_x\right]$$
$$\phi_d = -\frac{m}{T}\left[\left(1 - c_y^2 + \lambda_y y\right) e_y + \left(c_y + c_{\dot{y}}\right) e_{\dot{y}} - c_y \lambda_y \chi_y\right]$$

Attitude Controller

- Attitude Controller has been used as provided in the Simulation VM
- **Inputs:** $T, \phi_d, \theta_d, \dot{\psi}_d$



Parameter Initialization

K_p: attitude gain

K_d: angular rate gain

I: inertia matrix

$$b = 8.54858 \cdot 10^{-6} \left[\frac{kg \cdot m}{s^2}\right]$$

$$d = 1.3677 \cdot 10^{-7} \left[\frac{kg \cdot m^2}{s^2}\right]$$

This makes gain tuning independent of inertia matrix

$$K_p \leftarrow K_p / I$$
$$K_d \leftarrow K_d / I$$

$$A = \begin{bmatrix} s & 1 & s & -s & -1 & -s \\ -c & 0 & c & c & 0 & -c \\ -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} : \text{allocation matrix}$$

$$u = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ T \end{bmatrix} \quad K = diag\left(\begin{bmatrix} b \cdot l \\ b \cdot l \\ d \\ b \end{bmatrix}\right)$$

$$u = KA\omega^2 \rightarrow \omega^2 = (KA)^{-1}u = (KA)^{-1} \begin{bmatrix} I & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_\tau \\ T \end{bmatrix}$$

$$\omega^2 = A^T(AA^T)^{-1}K^{-1}\bar{I}a_\tau = \tilde{A}\begin{bmatrix} a_\tau \\ T \end{bmatrix}$$

Attitude Controller

Compute Angular Acceleration

1. $\psi \rightarrow R\left(\mathbf{q}\right).\mathrm{get\_yaw}()$

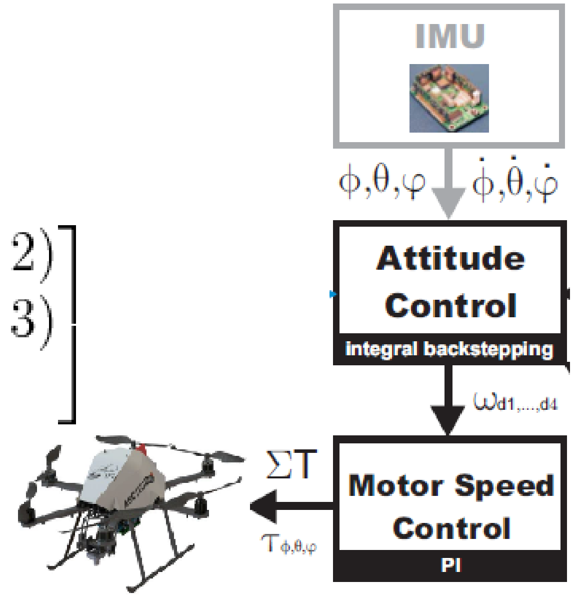
2. $R_d = R(\psi, \vec{z})R(\phi_d, \vec{x})R(\theta_d, \vec{y})$

3. $e_R = \frac{1}{2}\left(R_d^T R - R^T R_d\right) \rightarrow e_{angle} = \begin{bmatrix} e_R(3,2) \\ e_R(1,3) \\ 0 \end{bmatrix}$

4. $\omega_d = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi}_d \end{bmatrix}$

5. $e_\omega = \omega - R^T R_d \omega_d$

6. $a_\tau = -K_p e_{angle} - K_d e_\omega + \omega \times \omega$



Outlines

- MAV System Analysis
- Designed Solution Description
- Simulations & Results

Provided Infrastructure for Challenge 3

- **Two virtual machines** (running Ubuntu 12.04) are provided to each Contestant:
  - A **Simulation VM**, containing the virtual environment (**Gazebo**) for simulations.
  - A **Contestant VM**, containing the Contestant's solution for the assigned tasks.
- The VMs communicate via **Client/Server Protocol**:
  - The **Simulation VM** acts as a **Server** and must not be modified
  - The **Contestant VM** acts as a **Client** submitting the solution to the Server
- **ROS** is installed on both VMS

ROS in a nutshell

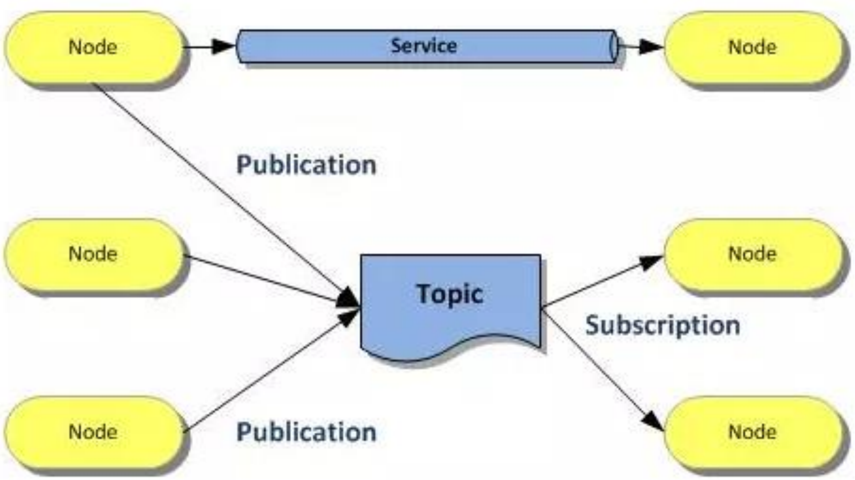
- ROS framework is an Operating System for robots (a *meta-operating system*)
  - **standard OS functionalities** offered:
    - Hardware abstraction
    - Processes handling and message-passing mechanism
    - ...
  - Development environments provided, with client **API libraries**:
    - C++
    - Python
    - Java
    - Lisp

further info: www.wiki.ros.org

ROS in a nutshell (2)

SAPIENZA UNIVERSITÀ DI ROMA

- Processes running under ROS are called **nodes**
- **Nodes** communicate through **message-passing mechanism**
- A **message** is a data structure with some designed typed fields (integer, float, boolean, array, ...)
- Nodes can write a message and **publish** it on a **topic**, or it can **subscribe** to a topic in order to read the corresponding message
- A special node, called ***roscore***, acts as a **master node** and needs to be run first before any other node.



- ROS usually works on a single machine; but so ...
- ... How can we make VMs to communicate between them?

- EuRoC partners have set a “*network bridging*” mechanism between the two machines, so that the ***roscore*** master node of the Server machine is shared with the Client Machine
- On the Contestant VM:

```
  i. <host machine’s IP>            eurocsimserv
 ii. <Contestant VM adapter 3 IP>   eurocsimclient
```

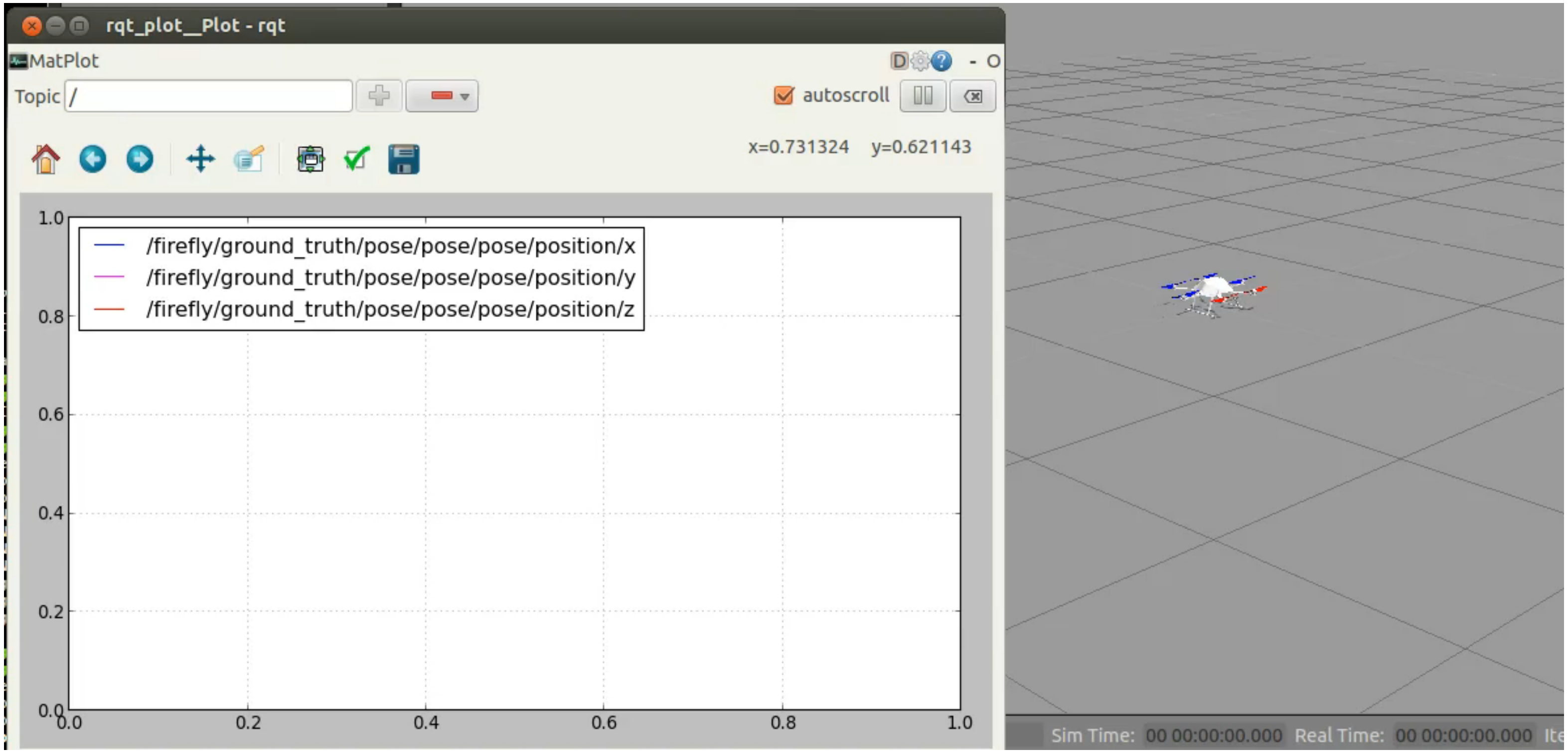
```
  i. export ROS_MASTER_URI=http://eurocsimserv:11311
 ii. export ROS_HOSTNAME=eurocsimserv
```

Launch files

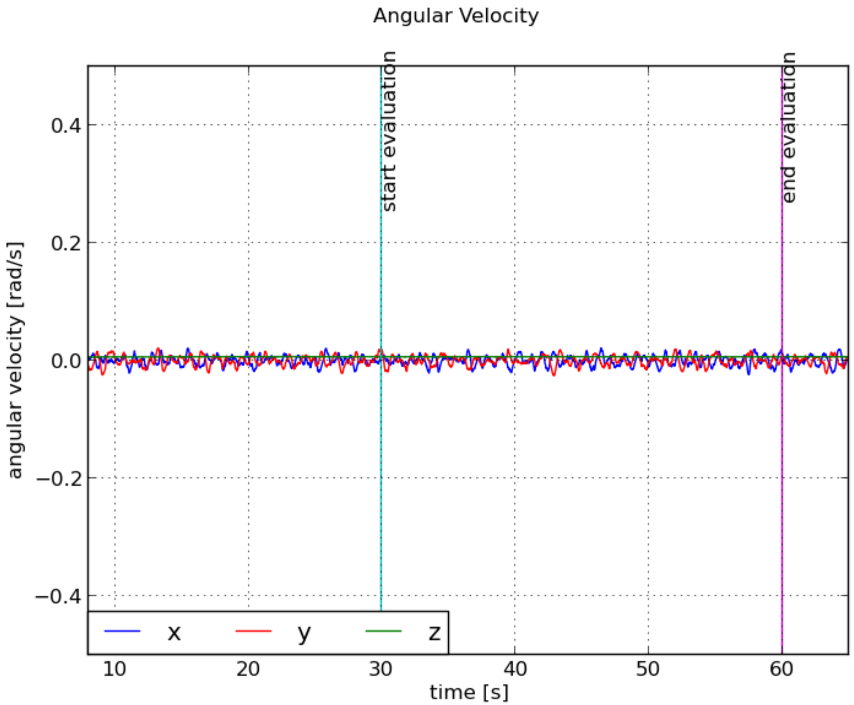
- The source code is the same for each subtask...
- ... But virtual scenarios change!
- Some common parameters need to be modified so that each subtask can be properly satisfied (mainly **control gains** and **flags**)
- These parameters may be set by defining a **launch file** for each subtask
- Solution node can then be run through the ROS command *roslaunch*

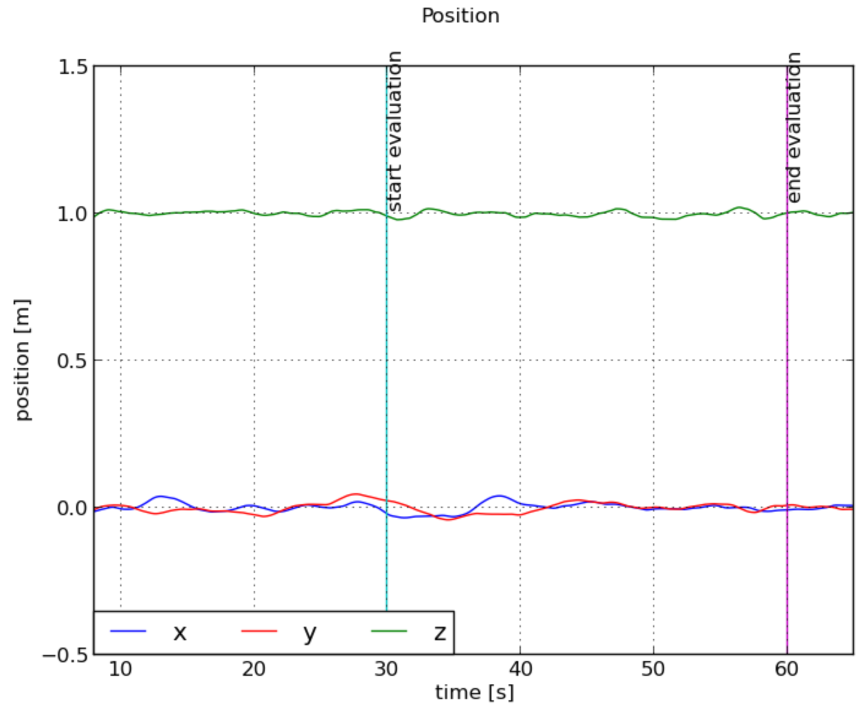
```
<launch>
  <group ns="firefly">

    <!-- Launch your nodes here. Extend / adapt for the subtask at hand, if necessary.-->

    <node name="euroc_solution_t3"  pkg="euroc_solution_t3" type="euroc_solution_t3" output="screen"/>
  <param name="c10" value="0.5"/>
  <param name="c12" value="0.5"/>
  <param name="alfa" value="0.3"/>
  <param name="beta" value="1.5"/>
  <param name="lambda4" value="1.0"/>
  <param name="lambda5" value="1.0"/>
  <param name="lambda6" value="1.0"/>
  <param name="sub_task2or3" value="1"/>
  </group>
</launch>
```

Task 3.1: results



Task 3.1: results





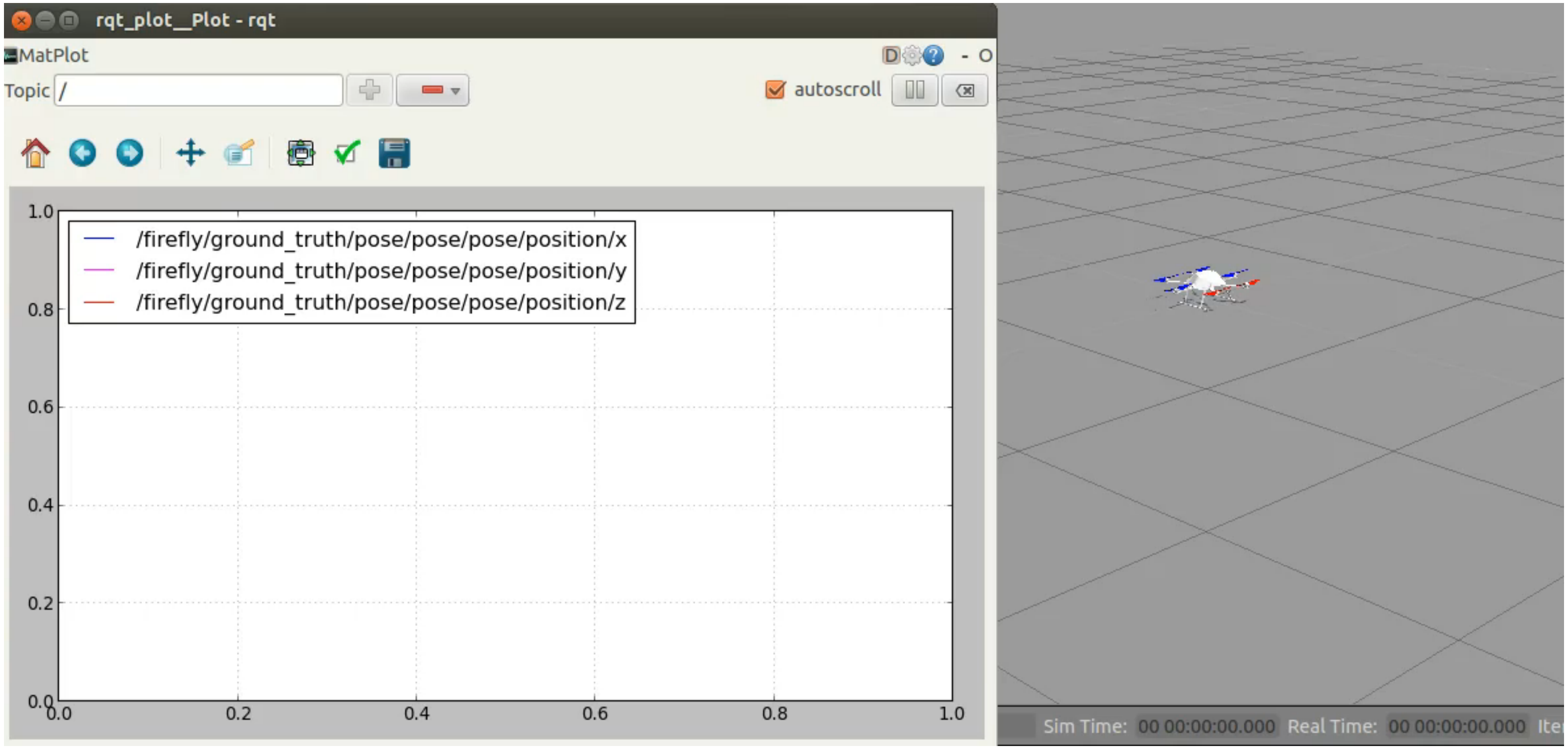
Position RMS error: 0.027 m

Angular velocity RMS error: 0.014 rad/s

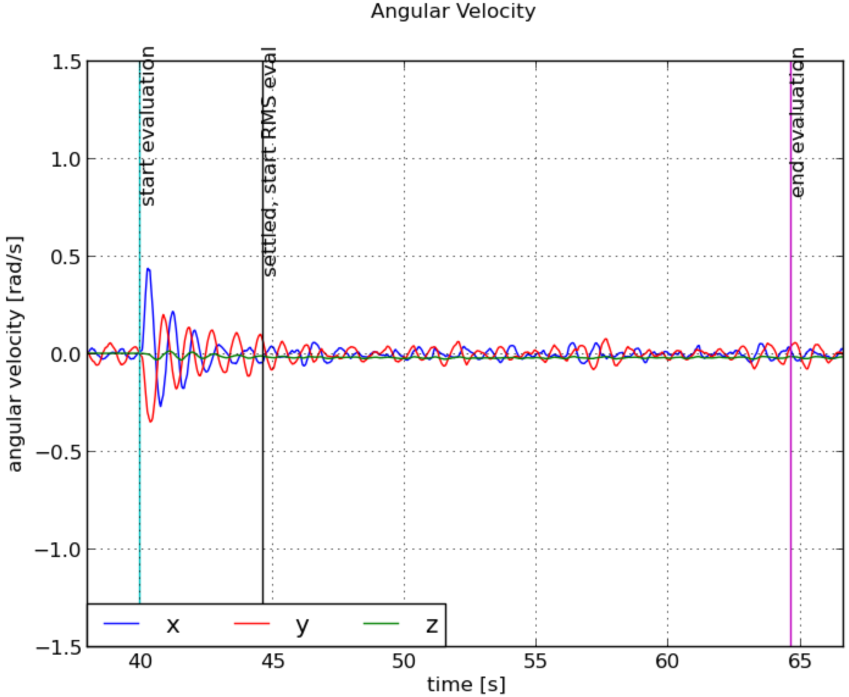
Scores: 3.0, 4.5

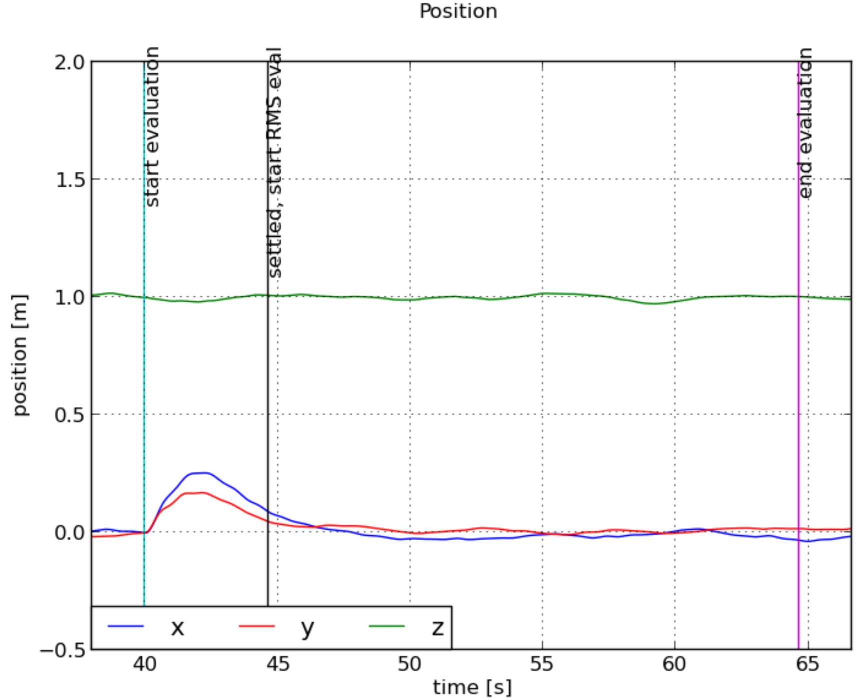
Total Score for Task 3.1 is: **7.5**

Task 3.2: results



Task 3.2: results





Settling time: 4.630 s

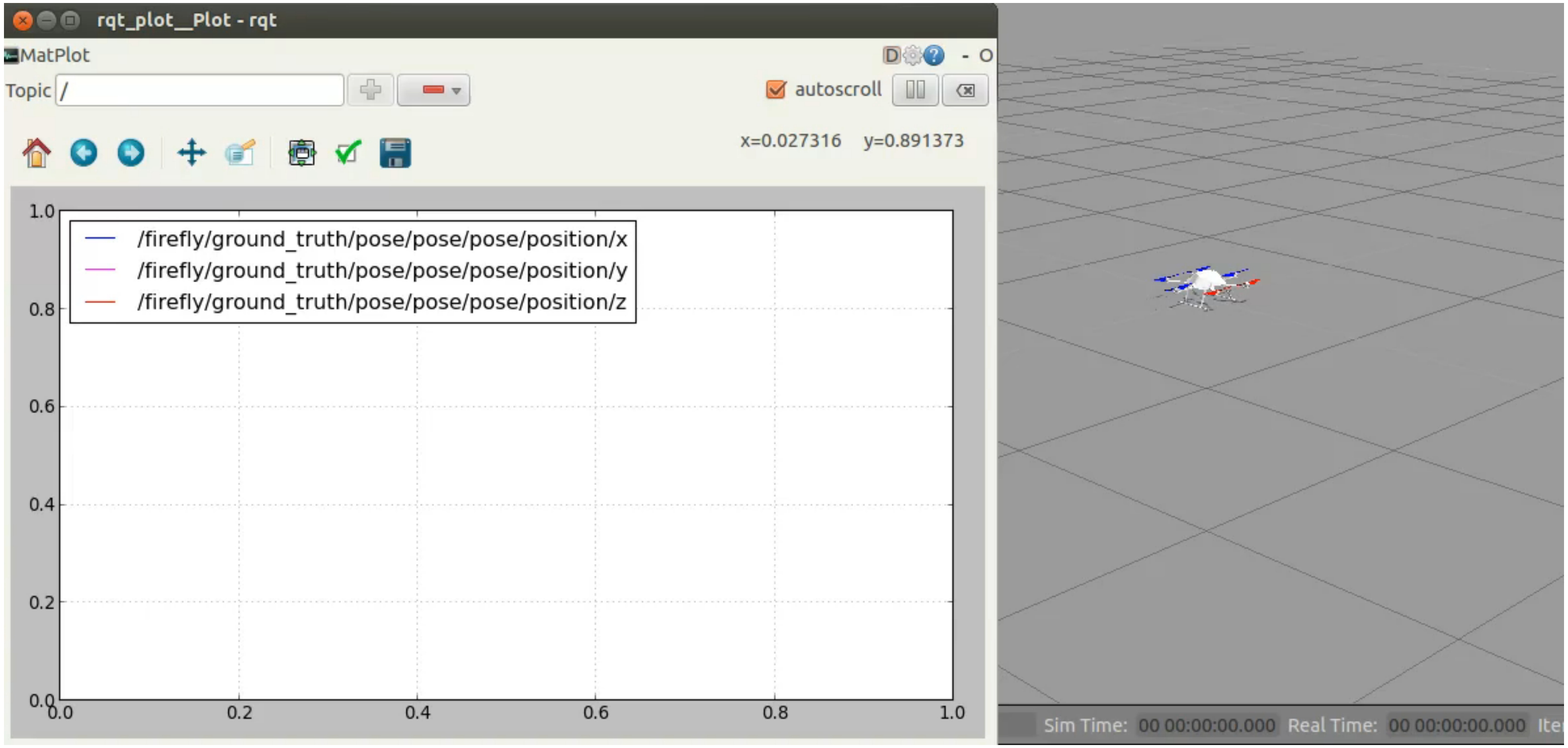
Position RMS error: 0.031 m

Angular velocity RMS error: 0.041 rad/s

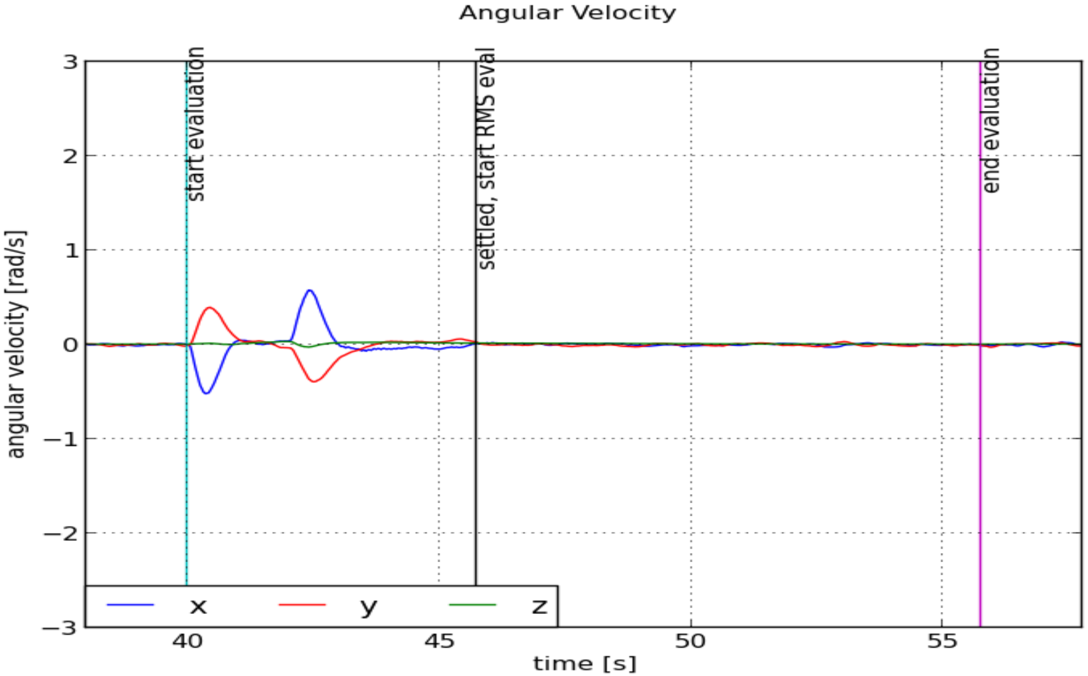
Scores: 1.0, 1.0, 1.0

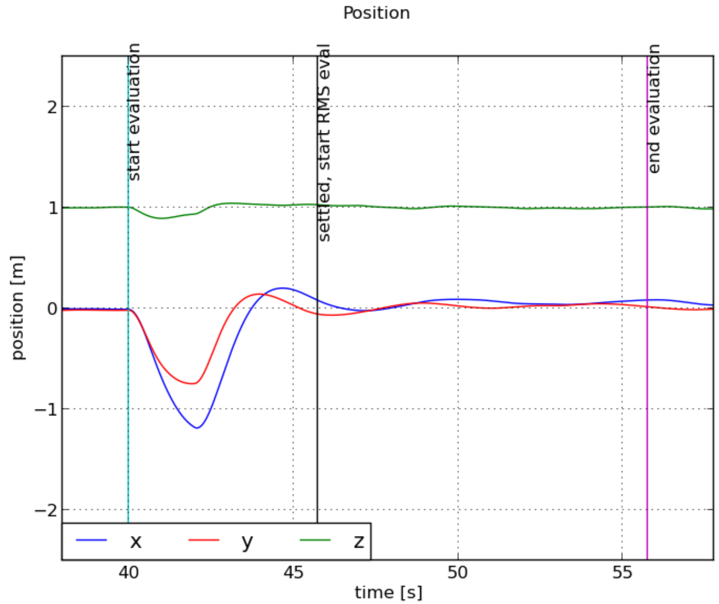
Total Score for Task 3.2 is: **3.0**

Task 3.3: results



Task 3.3: results





Settling time: 5.760 s

Position RMS error: 0.070 m

Angular velocity RMS error: 0.017 rad/s

Scores: 2.0, 2.0, 3.0

Total Score for Task 3.3 is: **7.0**

Questions?