Algorithms for controlling and tracking UAVs in indoor scenarios

Andrea Nisticò

Supervised by: Marco Baglietto, Fulvio Mastrogiovanni
Co-supervised by: Tommaso Falchi Delitalia

DIBRIS - Department of Informatics, Bioengineering, Robotics, and Systems Engineering
University of Genova

Degree in *Robotics Engineering*

September 18, 2015

Overview

1. Introduction
2. System setup
3. Modeling, estimation and control
4. Software architecture
5. Landing on a mobile platform
6. Conclusions

The quadrotor

The number of applications in which UAVs (Unmanned Aerial Vehicles) are involved is exponentially increasing, especially in indoor scenarios.

The most popular architecture is the **quadrotor**: a helicopter with four rotors. The main features are:

- Good stability
- Omni-directional kinematics
- Hovering on a fixed point
- Medium payload

Due to its properties, the quadrotor is the main chosen architecture for indoor flight.

Problem statement

Goal

We want to design a set of algorithms and SW components enabling a quadrotor to achieve stability, through position feedback given from a motion capture system, and perform different tasks in an indoor scenario.

The work of this thesis produced:

- A correct integration between the on board autopilot and the mocap system
- A functioning experimental setup mounted in the new laboratory
- A software architecture capable of letting the robot execute tasks sequentially from a list
- A procedure for landing on a moving platform

Work flow

The total work is divided in main steps:

1. Analysis of state of the art approaches to UAV control and modeling
2. Technical analysis and documentation on the given tools (Autopilot, Motion Capture, Quadcopter)
3. Integration between mocap and the quadrotor
   - Integration and testing between mocap and onboard estimation modules
   - Integration and testing of onboard controller through set point control
4. Relocation of the equipment in the new laboratory
5. Design of the high level software architecture
6. Testing the software, coding and debugging different kind of tasks
7. Design and testing an algorithm for landing on a mobile platform

1 Introduction

2 System setup

3 Modeling, estimation and control

4 Software architecture

5 Landing on a mobile platform

6 Conclusions

Setup

We can identify five distinct parts:

- IRIS quadcopter from 3D robotics
- Motion Capture system from Optitrack
- Linux workstation
- Windows machine
- Flight arena

IRIS

The IRIS quadcopter is flying robot manufactured by 3DRobotics and commercially available.

Main features

- Relatively cheap
- High quality and robust
- Ready to fly
- Compatible with Android tablets

Main components

- Four 850 KV motors (PWM)
- One 11.1 V LiPo battery
- Control electronics
- Auto pilot software

IRIS

IRIS - PixHawk / PX4

The brain of the robot is the **PixHawk** board.

- Two embedded processors
- IMU, barometer, magnetometer and GPS receiver
- ESCs (Electronic Speed Controllers)
- RC Radio interface
- Telemetry module through MAVLink protocol

Featuring the **PX4** open source autopilot.

PX4 runs on NuttX, an open source RTOS, and includes different modules executing in parallel.

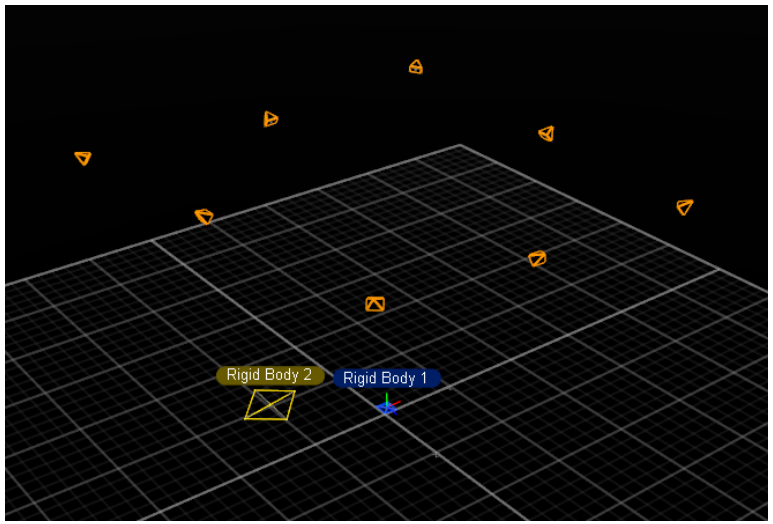
Modules implement: estimation and control, low level management, failsafe.

Motion Capture

Responsible for the localization of the robot.

It relies on eight OptiTrack cameras detecting IR beams reflected by passive markers.

Provides 6D (translation and rotation) position of rigid bodies defined by at least three markers. Position is provided by a local frame predefined by the user.



Adapting reference frames

Mocap reference frame

- X and Z axis on the horizontal plane
- Y axis vertical pointing up

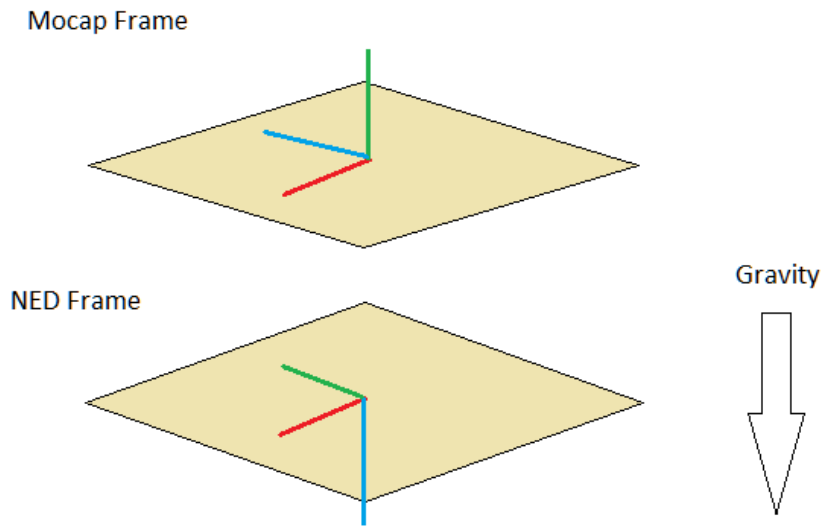
Earth reference frame

- X and Y axis on the horizontal plane
- Z axis vertical pointing down
- When mag is on, X is North, Y East and Z is down

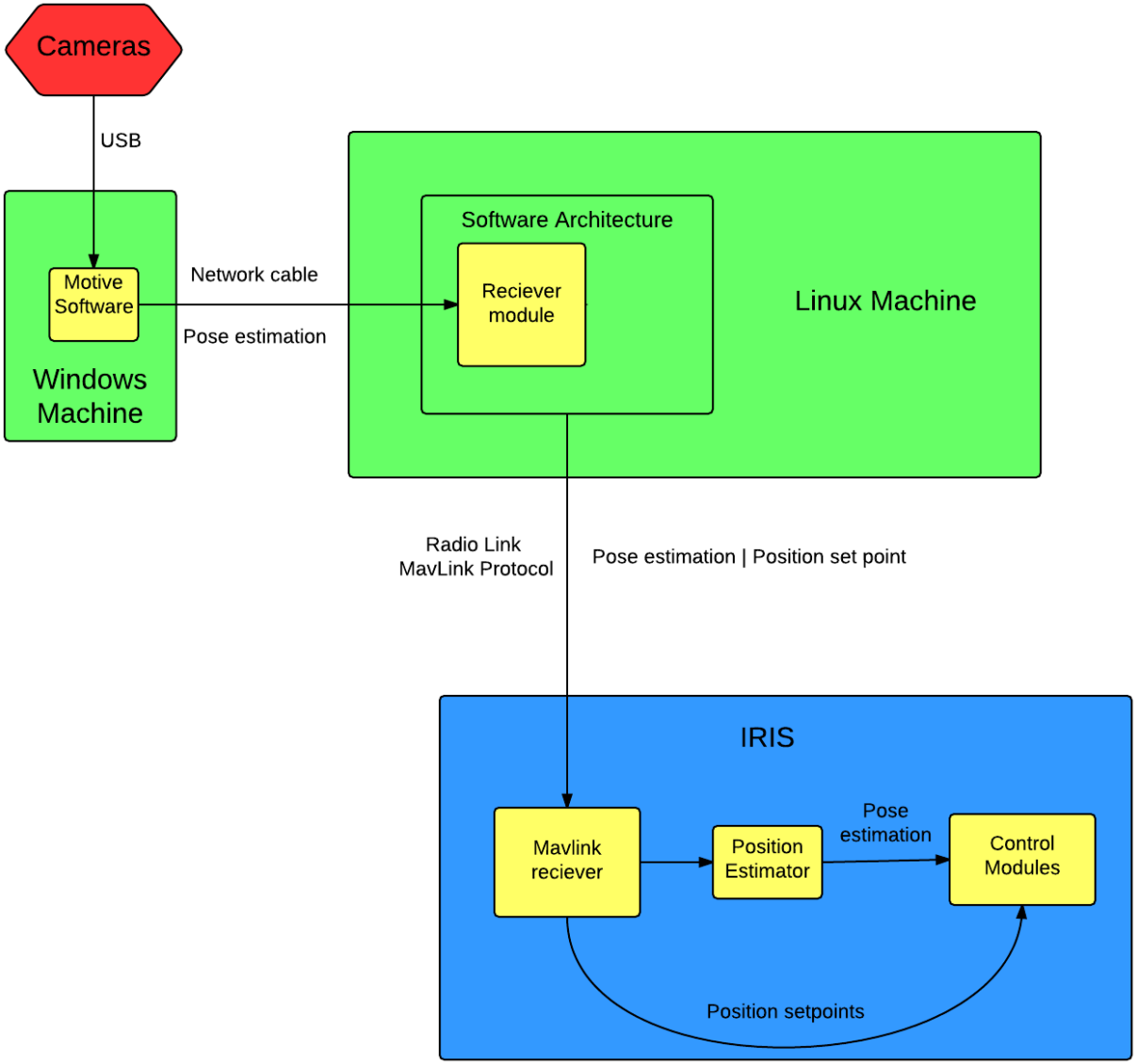
$$P^{\mathrm{E}} = \begin{bmatrix} x & z & -y \end{bmatrix}^T$$

$$Q^{\mathrm{E}} = \begin{bmatrix} w & x & z & -y \end{bmatrix}^T$$

with $[x, y, z]$ and $[w, x, y, z]$ in mocap frame



Overall integration



1 Introduction

2 System setup

3 Modeling, estimation and control

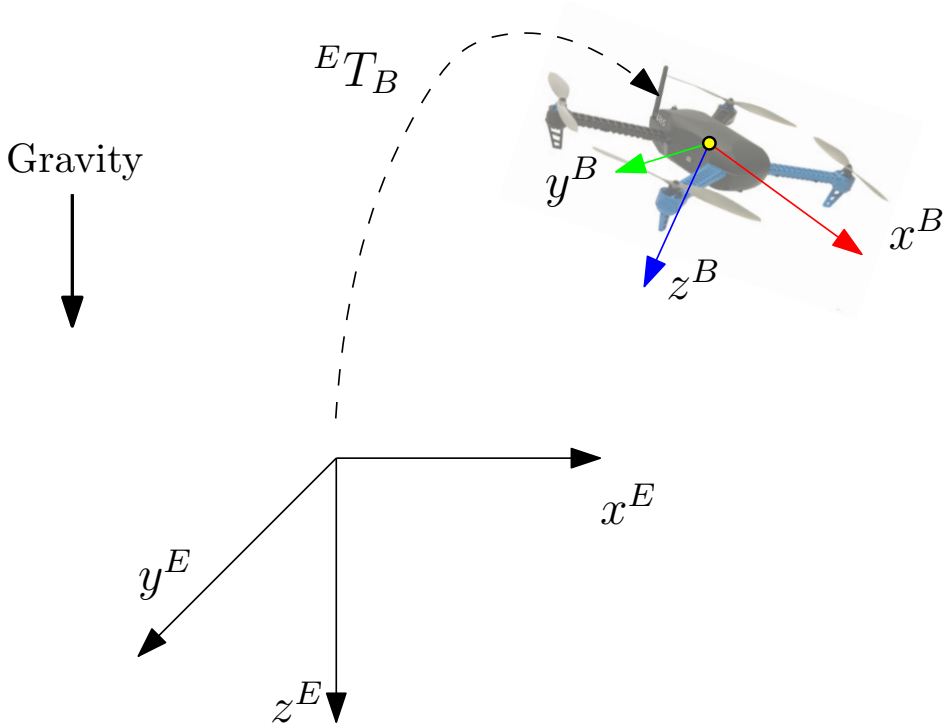
4 Software architecture

5 Landing on a mobile platform

6 Conclusions

Reference frames

- $\Re^B$ body frame, attached to the robot center of mass
- $\Re^E$ earth frame, placed on the floor at the center of the room



${}^E T_B$ defines the transformation of $\Re^B$ respect to $\Re^E$ as base frame.

Propulsion

Each rotor generates:

- A force $f_i = k\omega_i^2$
- A torque $\tau_i = k_d\omega_i^2$

Performing forces and torques balance:

$$\begin{bmatrix} T^B \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} k & k & k & k \\ kd_iS(30) & -kd_iS(30)k & -d_iS(30) & d_iS(30) \\ -kd_iC(30) & -kd_iC(30)k & d_iC(30) & d_iC(30) \\ k_d & -k_d & -k_d & k_d \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}^2$$

T^B: thrust along z^B

τ : torques on the three axis

Rigid body equations

The model is described by the well known rigid body equations of motion:

$$\begin{aligned} \dot{\mathbf{r}} &= \mathbf{v} \\ m\dot{\mathbf{v}} &= \mathbf{F} \\ \dot{R} &= R\Omega_{\times} \\ I\dot{\mathbf{\Omega}} &= -\mathbf{\Omega} \times I\mathbf{\Omega} + \boldsymbol{\tau} \end{aligned}$$

Define the input vector as : $\mathbf{U} = \begin{bmatrix} T^B & \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$

then: $\dot{\mathbf{v}} = \frac{1}{m}\left(\begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}^E + R\begin{bmatrix} 0 \\ 0 \\ U_1 \end{bmatrix}^B\right)$ and $\dot{\mathbf{\Omega}} = I^{-1}\left(\begin{bmatrix} U_2 \\ U_2 \\ U_4 \end{bmatrix} - \mathbf{\Omega} \times I\mathbf{\Omega}\right)$

State estimation

Position estimator: Fixed gain observer

- Accelerometers (input)
- GPS
- Barometer
- Mocap

Attitude estimator: Extended Kalman Filter

- Accelerometer
- Gyroscope
- Magnetometer

Position estimator model:

$$\mathbf{x_k}(\mathbf{x_{k-1}}, dt, \mathbf{a}) = \mathbf{x_{k-1}} + \mathbf{v_k} dt + \frac{1}{2}\mathbf{a} dt^2$$

$$\mathbf{v_k}(\mathbf{v_{k-1}}, \mathbf{a}, dt) = \mathbf{v_{k-1}} + \mathbf{a} dt$$

The magnetometer gives the value of the magnetic field in robot frame along three axis. The yaw is normally calculated respect to the North-South line.

Yaw correct estimation

Measured mag. field in body frame:

$$\mathbf{mag}^B = \begin{bmatrix} mag_x \\ mag_y \\ mag_z \end{bmatrix}$$

Fake mag. field in earth frame:

$$\mathbf{mag}^{\mathbf{E}} = \begin{bmatrix} 1 \\ 0 \\ 0.4 \end{bmatrix}^T$$

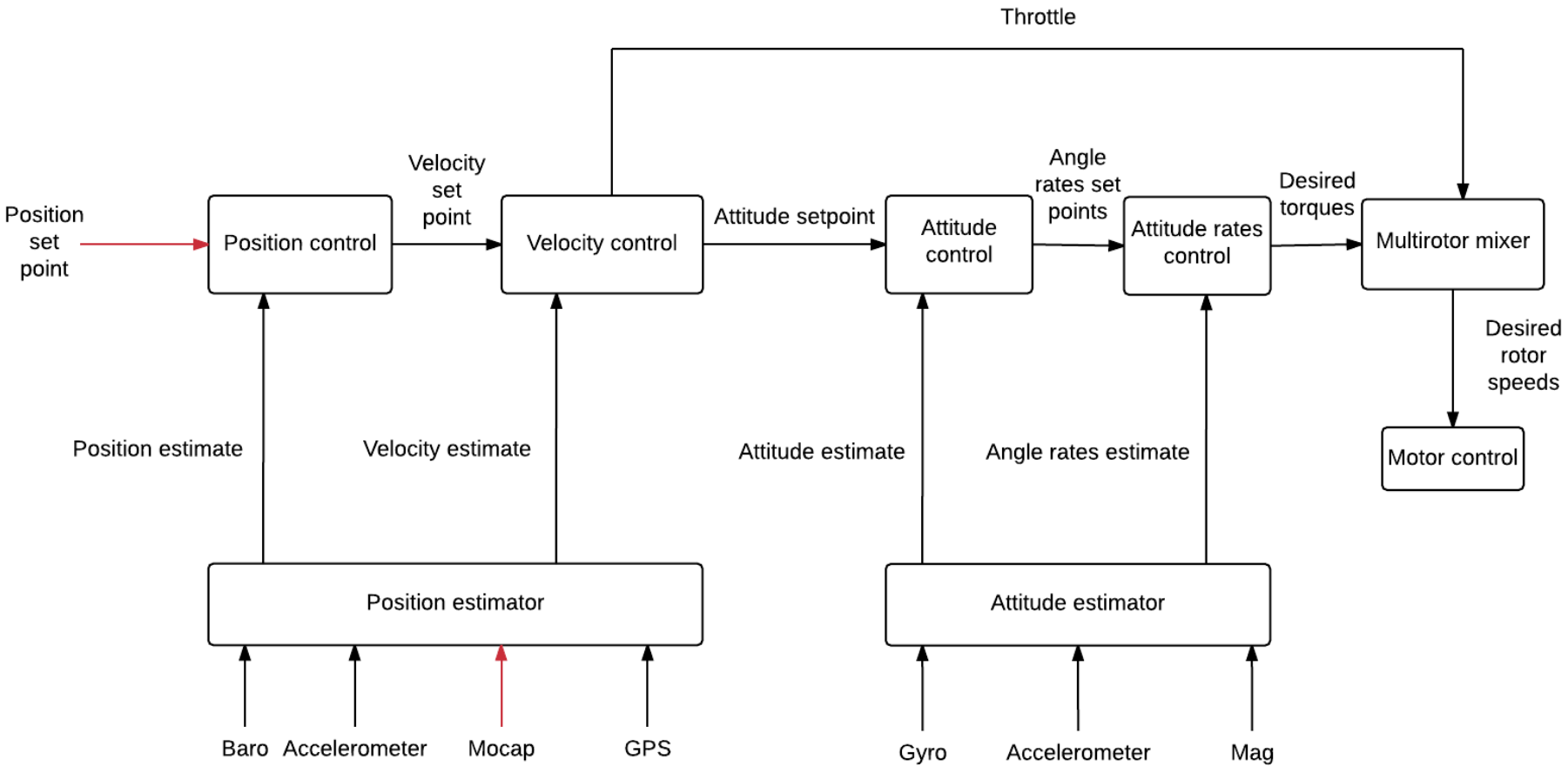
Then we cut of the magnetometer sensor and replace $\mathbf{mag}^B$ with:

$$\mathbf{mag}^{\mathbf{B}} = R^T \mathbf{mag}^{\mathbf{E}} \tag{1}$$

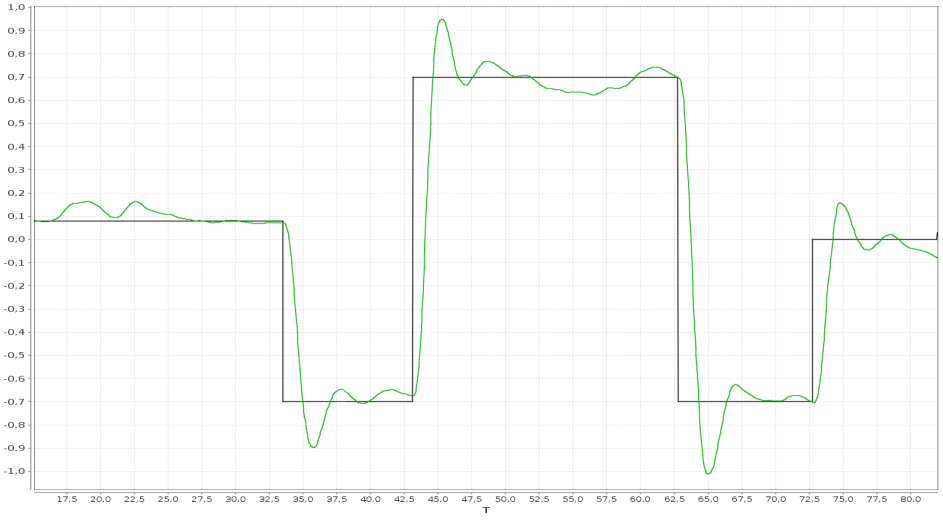
Roll and *Pitch* angles are estimated by the EKF. By passing the value of the *Yaw* calculated by the mocap we are able to construct the R matrix and inject the value of $\mathbf{mag}^{\mathbf{B}}$ inside the estimator.

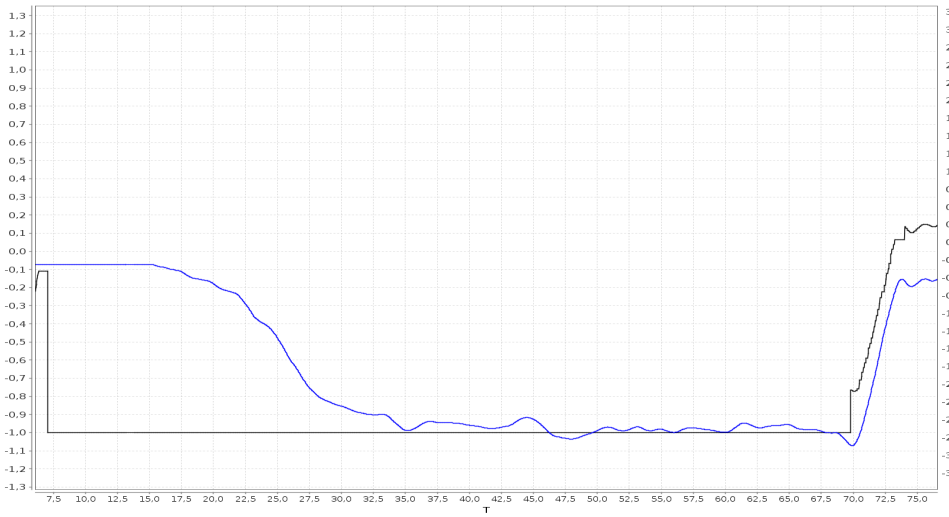
Controller structure

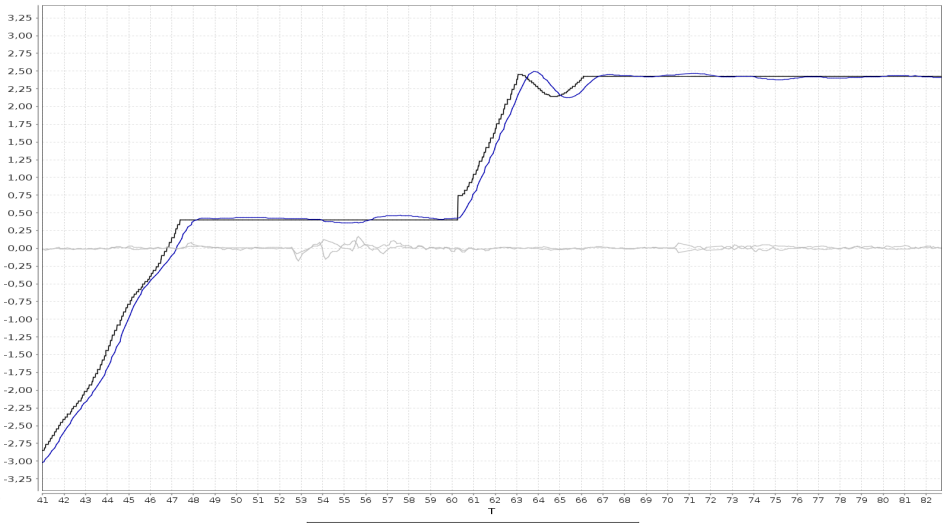
Two modules implement a cascade of P / PID controllers: **position control** module and **attitude control** module.



Results







1 Introduction

2 System setup

3 Modeling, estimation and control

4 Software architecture

5 Landing on a mobile platform

6 Conclusions

Definition

Definition

A software architecture is an abstract view of a software system distinct from the details of implementation, algorithms, and data representation.

It is represented by a block diagram. Each block is independent with known in/out relation.

A well written software architecture should:

- Provide flexibility and adaptability
- Allow for interoperability
- Provide control on the system
- Reduce maintenance time and cost
- Help developers improving the software

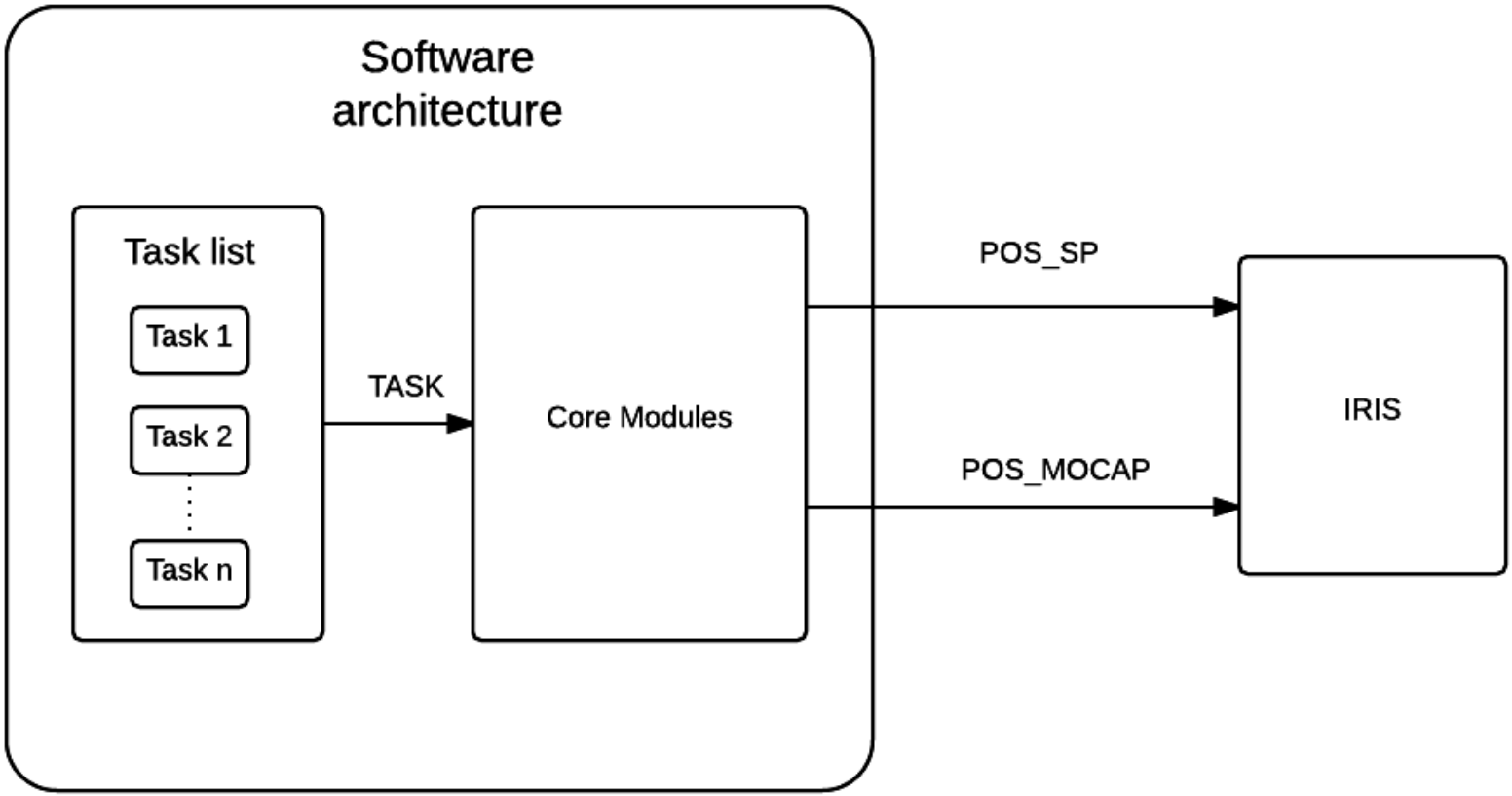
Goal

The SW must:

- Be expandable
- Be flexible
- Be able to control the robot by sending position feedback and set point
- Execute sequentially a list of tasks predefined by the user

Note: Position feedback and set point are vectors $P = \begin{bmatrix} x \\ y \\ z \\ yaw \end{bmatrix}$

Scheme


Software
architecture
Task list
Task 1
Task 2
Task n
TASK
Core Modules
POS_SP
POS_MOCAP
IRIS

Design Patterns

How do we design the core modules?

Design Pattern

The pattern is a description or template for how to solve a problem that can be used in many different situations.

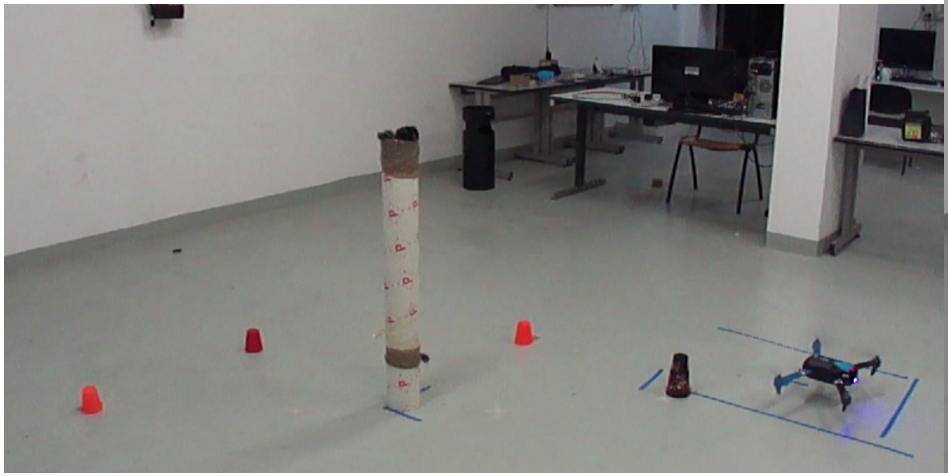
The Pattern is the architectural style chosen among the available ones depending on the problem.

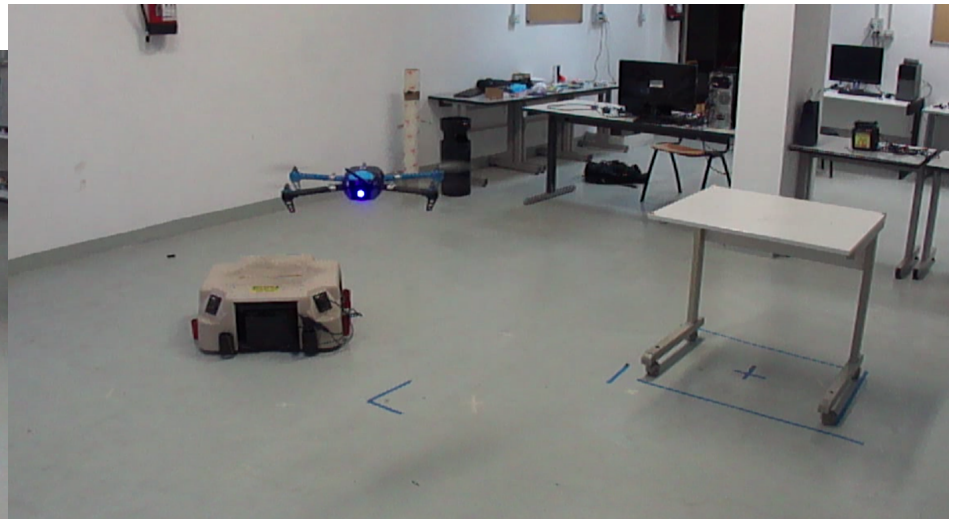
A **Behavioral design** is used for organizing each SW component.

Behavioral design in nature

Inspired from animals

Demo





Verbatim

Example (Theorem Slide Code)

```
\begin{frame}
\frametitle{Theorem}
\begin{theorem}[Mass--energy equivalence]
$E = mc^2$
\end{theorem}
\end{frame}
```

1 Introduction

2 System setup

3 Modeling, estimation and control

4 Software architecture

5 Landing on a mobile platform

6 Conclusions

1 Introduction

2 System setup

3 Modeling, estimation and control

4 Software architecture

5 Landing on a mobile platform

6 Conclusions

Thank you