Technical Report

IRI-TR-19-03

Robots and IoT devices for assistive automation IRI Technical Report

Kévin Bedin
Sergi Foix
Guillem Alenyà

August, 2019

Institut de Robòtica i Informàtica Industrial

Abstract

In recent years, we live every day in an ever more connected world. This phenomenon has developed the technologies of the Internet of Things (IoT). At the same time, advances in robotics now make it possible to have autonomous service robots in their mobility and in the accomplishment of their missions. So far the question is whether it might be useful to unite these two areas.
Just as humans do, how do these service robots integrate the objects of IoT into their environment to facilitate the fulfillment of their mission?
This report focuses on how it is possible to integrate IoT objects into the environment of a PAL Robotics robot, TIAGo. It is the result of twelve weeks of internship within the laboratory of Perception and Manipulation of the Institut de Robòtica i Informàtica Industrial of Barcelona.

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
`http://www.iri.upc.edu`

Corresponding author:
Guillem Alenyà
tel: +34 93 401 5751
`galenya@iri.upc.edu`
`http://www.iri.upc.edu/staff/galenya`

1 Introduction

"The Perception and Manipulation Group investigates in service robotics and has two TIAGo mobile manipulation robots and a laboratory simulating an apartment. This project wants to investigate the benefits of adding smart devices to this environment and making them available to the robot. In recent years, a wide variety of IoT (Internet of Things) domestic devices have appeared that can monitor environmental variables and control active parts, as well as several initiatives that try to standardize protocols and access methods to these devices. On the other hand, service robotics has advanced enormously thanks to the existence of robust solutions for critical parties, such as voice recognition or navigation avoiding obstacles. Now is the propitious moment to unite these two worlds and interconnect the robot with the environment. The robot would not only multiply its action capabilities but also increase its data acquisition capacity. For example, if you do not have enough information with your own sensors and cameras or it is ambiguous, you could connect to the surrounding network of cameras and obtain images from different points of view and places."[1]
The purpose of my internship was to know if it was possible to integrate in the Lab's apartment smartdevices using various protocols within the environment of a TIAGo robot so that it can use it to enrich its range of actions.
The first necessity was to define the subject. It was necessary that I highlight various problems in order to better understand the expectations of the project as well as the exploitable technical solutions. The first problem was:

- How to use these sensors?

Indeed before being able to connect it to the robots, you must know how to extract the information. The next question is to know if you can use a single tool that can integrate a wide variety of IOT protocols, ie the existence of a hub for connected objects.

- Is it possible to standardize these protocols ?

After that it would be a question of how the robot interacts with its environment:

- What environment does the robot use?

And then of course it would be necessary to know how to connect the world of IoT via a potential HUB and the environment interpretable by the robot:

- How to integrate these sensors into the robot environment?

Finally, in order not to lose any functionalities as well on the side of the robot as on the side of the connected objects the last question is to know if these two systems can evolve independently while having a capacity of communication:

- Is it possible to decouple the IOT part and the robot while having an activatable link if necessary?

So after having arisen this set of problems I was able to focus on researching existing technical solutions by making a state of the art. The world of IoT being born by definition on the Internet, it is relatively easy to find many software allowing to use various protocols of the IoT as well as having a user-friendly Human Machine Interaction (HMI). My choice fell on a very popular software in the world of IoT: OpenHAB version 2 (OH2). This software acts as a hub, being able to use most of the domotic protocols currently used. I have integrated various sensors and actuators from the world of home automation as well as a choosen IP camera from Amcrest company. Subsequently I looked at how to connect the environment of the robot to OpenHAB.

It is through the Robot Operating System (ROS) middleware and the iot_bridge package, that I have upgraded for this implementation, that this link was set up. Then I developed algorithms, in the form of a ROS package, for the camera: amcrest_ip_camera. The main algorithm allows to scan the apartment in search of a binary image (Aruco) and provides its location in the apartment. Finally a mission to find an Aruco has been tested with TIAGo. To do this, it can use both his own camera but also use the IP camera with the developed package. So, the main solutions used will be explained in this technical report.

2 Architecture overview

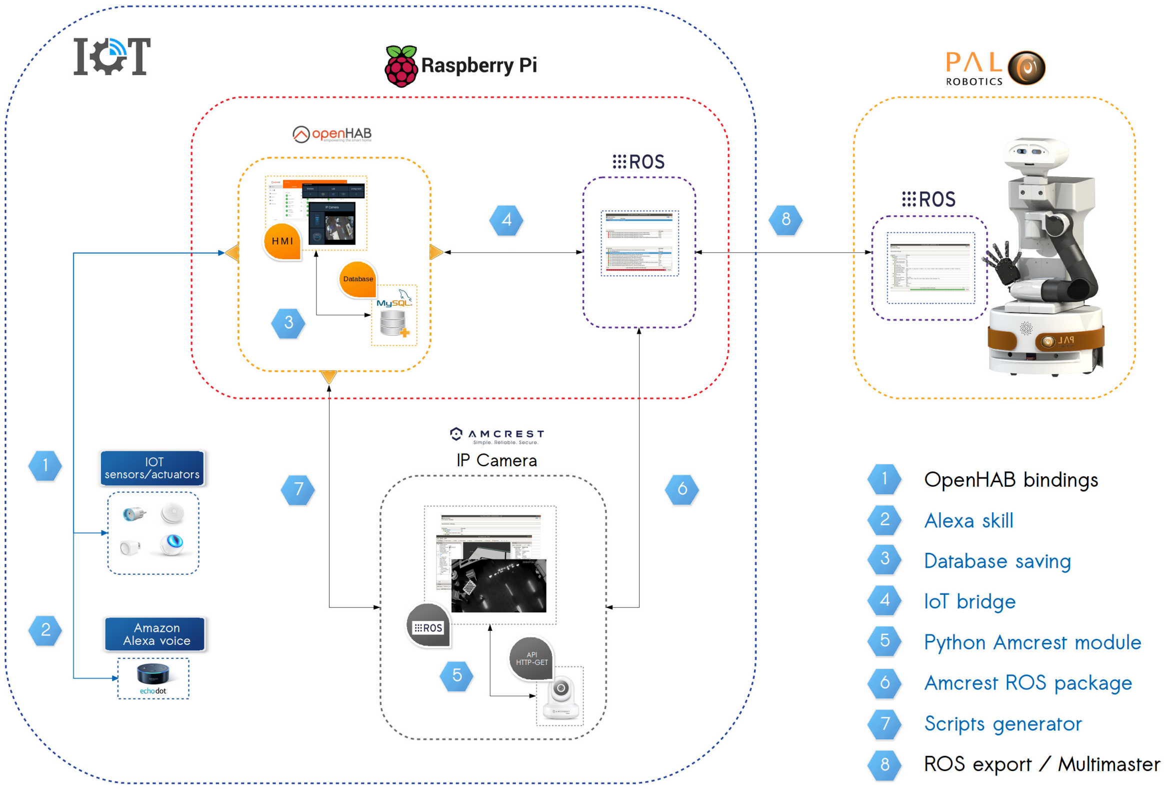


Figure 1: Global IOT architecture developed

Each blue description of the legend is a project or tutorial developed available on Gitlab and / or Github:

- Alexa skill: `alexa_openhab_turorial`
  - This tutorial explains how to link an IoT device to an echodot using OpenHAB. I explain in this technical report how to do that - see **Link OpenHAB device to Alexa - [TUTORIAL]**
  - `https://gitlab.iri.upc.edu/perception/lab/iot_apartment/alexa_openhab`
- Database saving: `OpenHab-MySQL project`
  - This project shows how to use the MySQL persistence from OpenHAB to save the data from IoT devices into a MySQL Database. I explain in this technical report how I used it - see **Data saving**
  - `https://gitlab.iri.upc.edu/perception/lab/iot_apartment/openhab-mysql`
- IoT bridge: `iot_bridge_upgrade project`
  - This project is an upgrade of the existing *iot_bridge* ROS package that makes the bridge between ROS and OpenHAB. I explain in this technical report how it works - see **iot_bridge: link the devices to ROS**
  - `https://gitlab.iri.upc.edu/perception/lab/iot_apartment/iot_bridge_upgrade`
- Python Amcrest module: `python-amcrest ros_used branch`

  - This project is an upgrade of the existing *python-amcrest* module that implementes the Amcrest's API in Python. I explain in this technical report what I have added and how to install it: see **IP Camera strategy redefinition**
  - `https://github.com/KevinBdn/python-amcrest/tree/ros_used`

- Amcrest ROS package: `amcrest_ip_camera project`
  - This project is a ROS package which allows to use an Amcrest's IP Camera its API through ROS. I explain in this technical report how it works - see **amcrest_ip_camera ROS package architecture** - and how to use it - see **Package use -** [**TUTORIAL**].
  - `https://gitlab.iri.upc.edu/perception/lab/iot_apartment/amcrest_ip_camera`

- Scripts generator: `ip_camera_openhab project`
  - This project allows to generate the necessary scripts to integrate the camera in OpenHAB. I explain in this technical report how it works - see **OpenHAB integration**.
  - `https://gitlab.iri.upc.edu/perception/lab/iot_apartment/ip_camera_openhab`

In addition, some tests and simulators used are available here:

- Tests performed: `Tests_performed project`
  `https://gitlab.iri.upc.edu/perception/lab/iot_apartment/tests_performed`

And the global view of my work is available in the main project:

- Global view: `main`
  `https://gitlab.iri.upc.edu/perception/lab/iot_apartment/main`

A description of each project and tutorial is made in the README.md on the corresponding Gitlab/Github repository.

3 Rapsberry PI configurations

3.1 Hardware

I decided to use a Raspberry Pi 3B+ to be the IoT system. In this Rapsberry is installed:

- Ubiquity Robotics Raspberry Pi Image: Ubuntu 16.04.6 LTS (GNU / Linux 4.14.98-v7 + armv7l) + ROS
- OpenHAB version 2.4 as software

Currently the IP address of the Raspberry in the local network is `172.16.0.3` . You can access to the Rapsberry Pi via SSH:

```
ssh ubuntu@172.16.0.3 #password= ubuntu
```

By default the Raspberry with Ubiquity OS shares its own Wifi Network

```
SSID: ubiquityrobotXXXX  #Where XXXX is part of the MAC address
Password: robotseverywhere
```

I disabled it to avoid external connection to the Lab network, you can manage it using pifi.

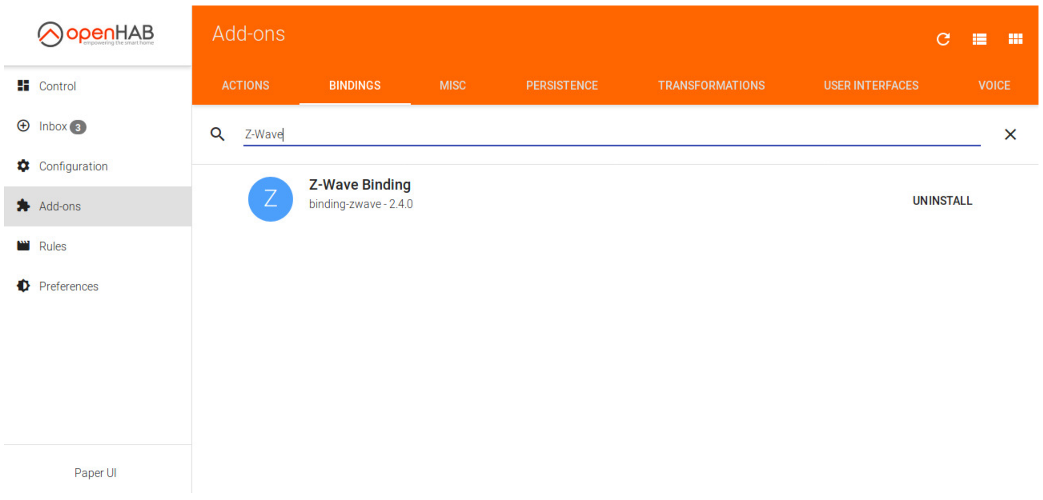
3.2 Add the devices to OpenHAB - [TUTORIAL]

You can manage OpenHAB through the local Network using the port number 8080:

```
RASPBERRY_IP_ADDRESS:8080
```

You can add new devices using the corresponding Binding from OpenHAB and Paper UI. Lot of tutorials are available in the official website. To add a Z-Wave Plus IoT device you can follow this tutorial.

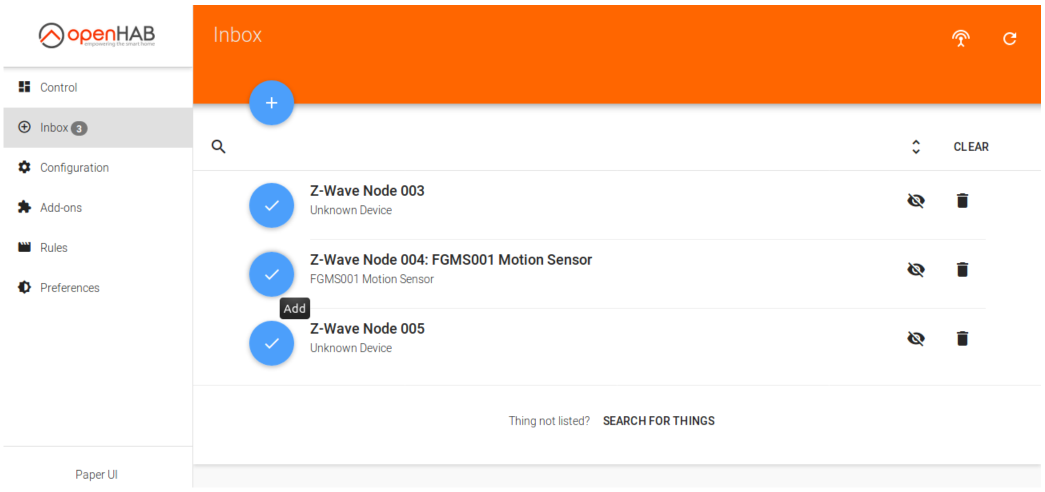
1. Make sure that the Z-Wave Plus stick is well plugged to the Raspberry Pi
2. Using `Paper UI` check if the Z-Wave Binding is well installed



Z-Wave

Z-Wave Binding
binding-zwave - 2.4.0

3. Launch the paired mode of your Z-Wave devices
4. Check in the `Inbox` if the new device appeared and add it

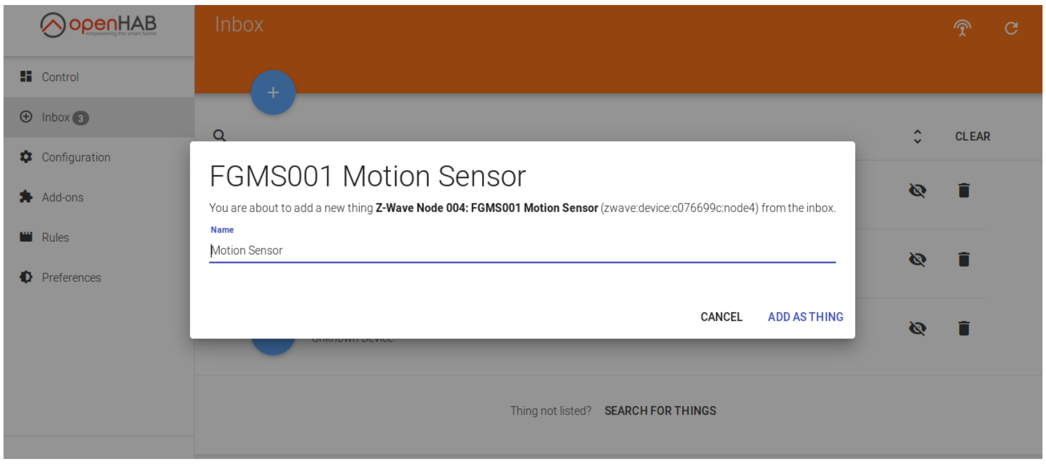


Z-Wave Node 003
Unknown Device

Z-Wave Node 004: FGMS001 Motion Sensor
FGMS001 Motion Sensor

Z-Wave Node 005
Unknown Device

Thing not listed? SEARCH FOR THINGS



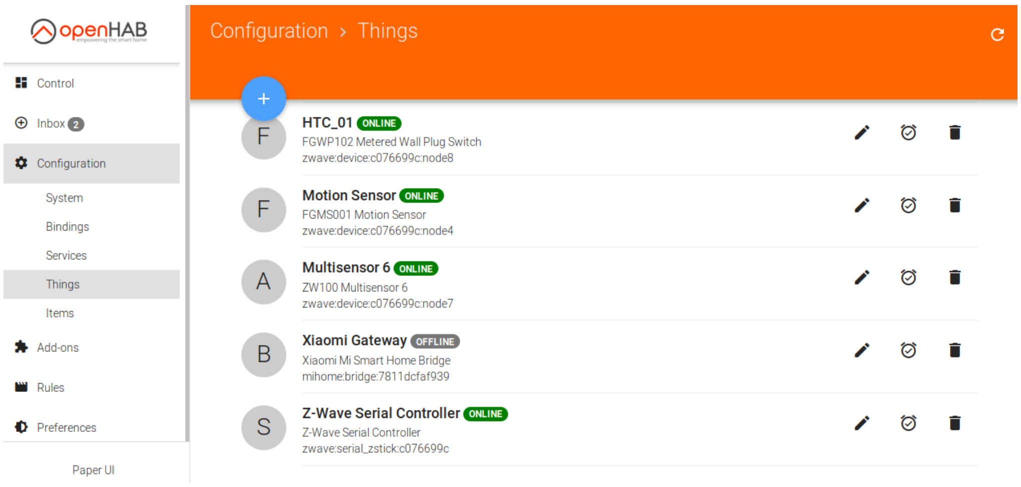
FGMS001 Motion Sensor

You are about to add a new thing **Z-Wave Node 004: FGMS001 Motion Sensor** (zwave:device:c076699c:node4) from the inbox.

Name
Motion Sensor

Thing not listed? SEARCH FOR THINGS

5. You can know manage the device in the `Configure -> Thing` window



HTC_01 ONLINE
FGWP102 Metered Wall Plug Switch
zwave:device:c076699c:node8

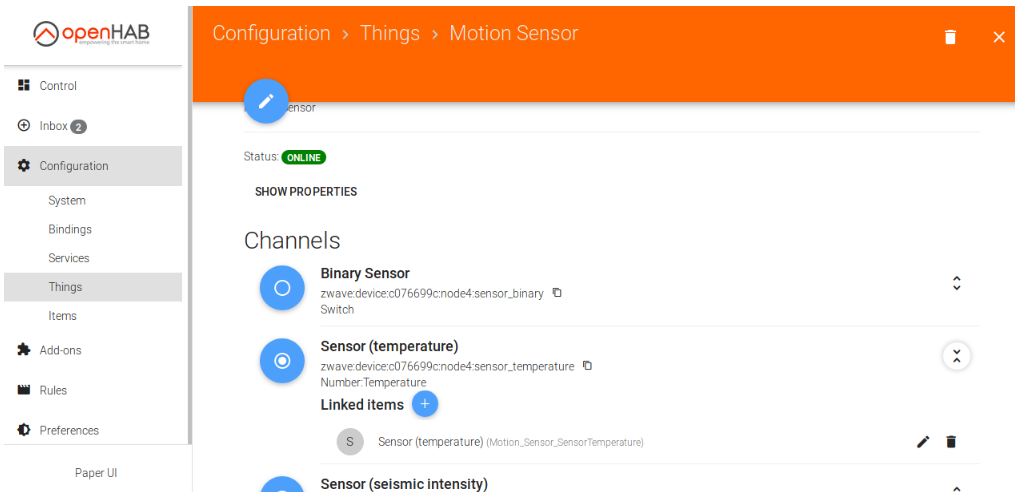
Motion Sensor ONLINE
FGMS001 Motion Sensor
zwave:device:c076699c:node4

Multisensor 6 ONLINE
ZW100 Multisensor 6
zwave:device:c076699c:node7

Xiaomi Gateway OFFLINE
Xiaomi Mi Smart Home Bridge
mihome:bridge:7811dcfaf939

Z-Wave Serial Controller ONLINE
Z-Wave Serial Controller
zwave:serial_zstick:c076699c

6. You have to associate an `Item` for each `Channel` that interests you



Status: ONLINE

Channels

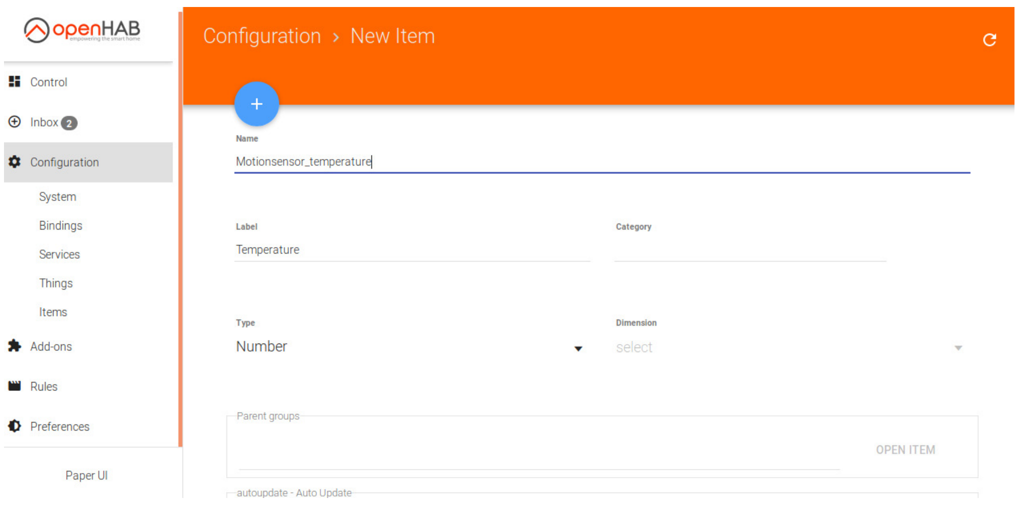
Binary Sensor
zwave:device:c076699c:node4:sensor_binary
Switch

Sensor (temperature)
zwave:device:c076699c:node4:sensor_temperature
Number:Temperature

Sensor (temperature) (Motion_Sensor_SensorTemperature)

Sensor (seismic intensity)

7. To create an `Item` you can use the `Configure -> Item` window



Name: Motionsensor_temperature

Label: Temperature

Category:

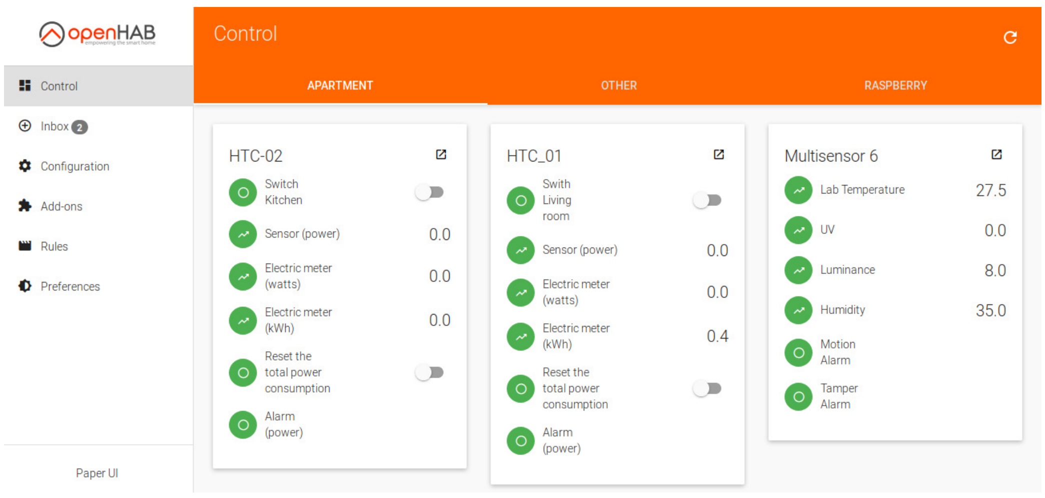
Type: Number

Dimension: select

Parent groups

autoupdate - Auto Update

8. Then you can manage the device using the `Control` window and through the other available UI.



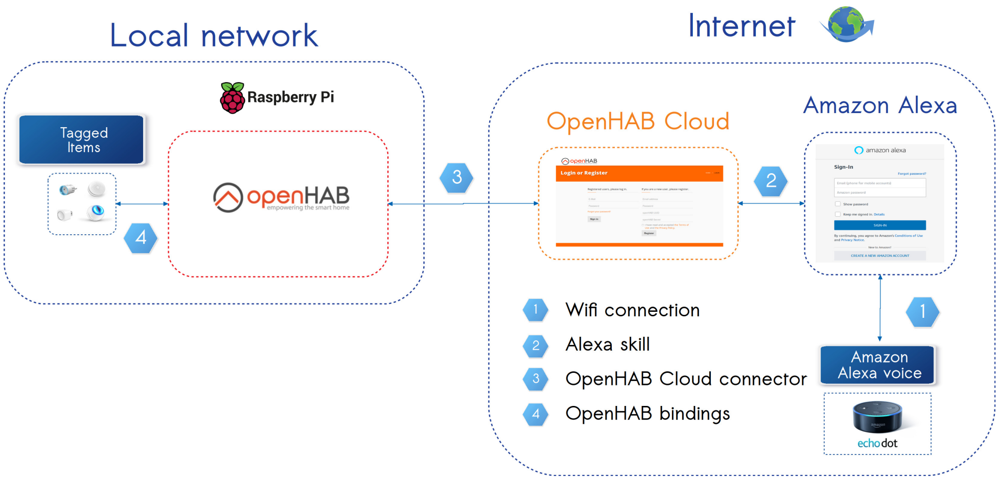
HTC-02
Switch Kitchen
Sensor (power) 0.0
Electric meter (watts) 0.0
Electric meter (kWh) 0.0
Reset the total power consumption
Alarm (power)

HTC_01
Swith Living room
Sensor (power) 0.0
Electric meter (watts) 0.0
Electric meter (kWh) 0.4
Reset the total power consumption
Alarm (power)

Multisensor 6
Lab Temperature 27.5
UV 0.0
Luminance 8.0
Humidity 35.0
Motion Alarm
Tamper Alarm

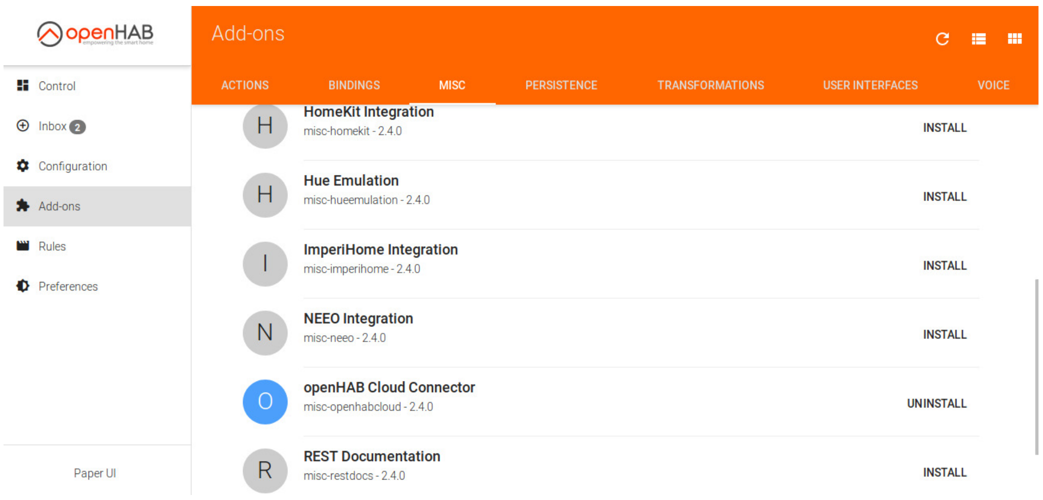
3.3 Link OpenHAB device to Alexa - [TUTORIAL]

The link between OpenHAB's paired devices and Alexa is made through this process:



So you can link devices to Alexa following these steps:

1. Install the `OpenHAB Cloud connector`



HomeKit Integration
misc-homekit - 2.4.0 INSTALL

Hue Emulation
misc-hueemulation - 2.4.0 INSTALL

ImperiHome Integration
misc-imperihome - 2.4.0 INSTALL

NEEO Integration
misc-neeo - 2.4.0 INSTALL

openHAB Cloud Connector
misc-openhabcloud - 2.4.0 UNINSTALL

REST Documentation
misc-restdocs - 2.4.0 INSTALL

2. Get the `UUID` and the `Secret` generated using this location:

File	Location
UUID	`/var/openhab2/uuid`
Secret	`/var/lib/openhab2/openhabcloud/secret`

3. Create an account on `https://myopenhab.org/` using the `UUID` and the `Secret`

4. On the Amazon Alexa account used by the echodot add `Alexa skill` and configure item

5. Now you can tag in OpenHAB the desired Items. Currently few tags are available on the official `Alexa skill` but a new version is developed. It increases the variety of Items that can be used. Here are the current tags:

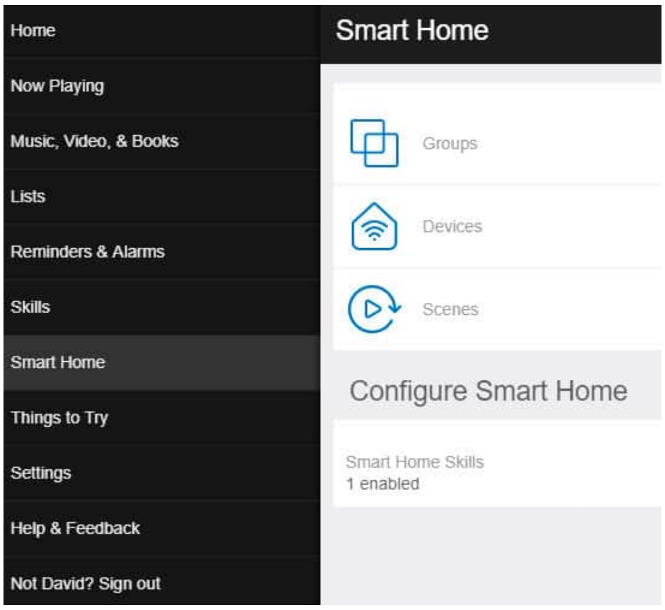
Tag
Lighting
Switchable
CurrentTemperature
Thermostat
CurrentTemperature
HeatingCoolingMode
TargetTemperature

To add a tag the easier solution is to use the OpenHAB's console on the Raspberry:

```
$ openhab-cli console
$ openhab> items list #In order to remind the items names
$ openhab> items addTag item_name the_tag
$ openhab> items list #In order to check if the tag has been well added

```

6. On Amazon Alexa using `Smart Home` you can now search the new devices and add it to appropriate groups



Once these steps are done, it is possible to interact with the objects connected via Alexa. Here are some sentences that can be used:

- `Alexa, switch on the Kitchen.`
- `Alexa, what is the temperature in the Lab ?`

For more details see my tutorial on Gitlab.
In the appendix you can have the current accounts and configurations used.

3.4 iot_bridge : link the devices to ROS

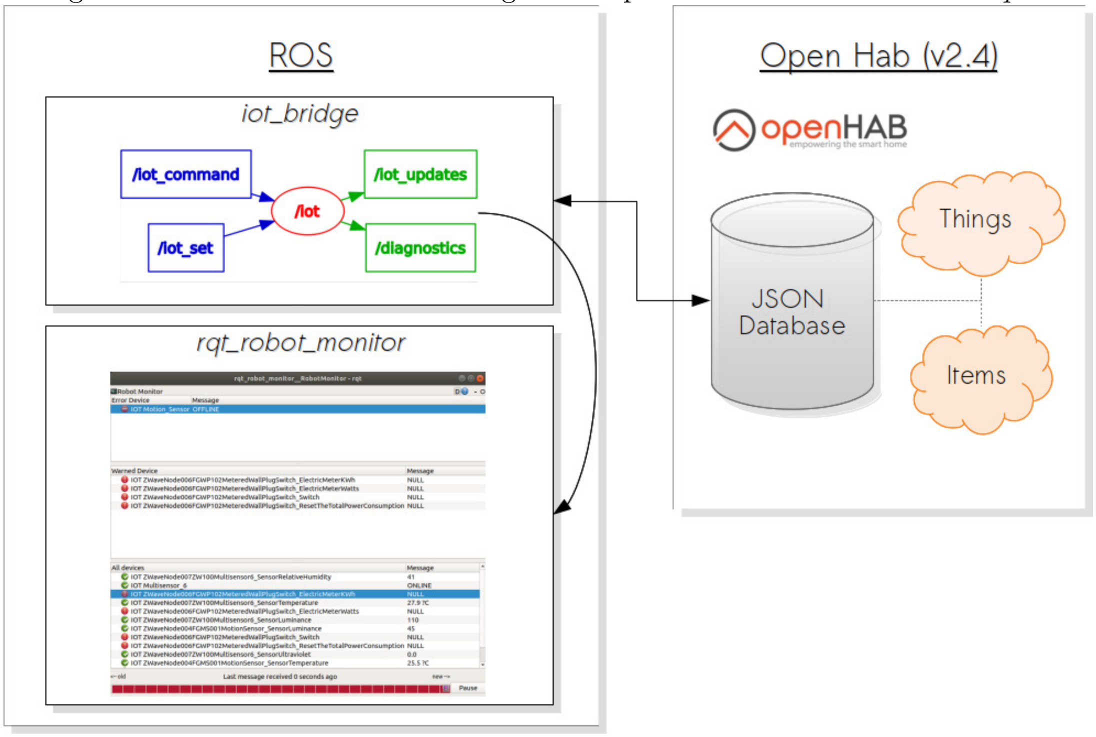
This ROS package written in python uses the OpenHAB JSON database accessible on the local network to read and write the data of the various integrated devices. This data is then distributed via topics on the ROS middleware.

Although functional, this package is based on version 1.0 of OpenHAB - based only on the concept of Item (an actuator or sensor) - that became almost obsolete in favor of version 2.0 - that incorporates the concept of Thing (a device) on which several Items are connected.

In order to better meet the needs of the laboratory and to be consistent with the version of OpenHAB used (2.4), I therefore upgraded this package creating a new project: **iot_bridge_upgrade**.

3.4.1 Package architecture

I integrated the notion of OH2 and a diagnostic topic to monitor all devices via *rqt_robot_monitor*.



```
rqt_robot_monitor__RobotMonitor - rqt
Robot Monitor
Error Device          Message
  IOT Motion_Sensor   OFFLINE

Warned Device                                                                    Message
  IOT ZWaveNode006FGWP102MeteredWallPlugSwitch_ElectricMeterKWh                  NULL
  IOT ZWaveNode006FGWP102MeteredWallPlugSwitch_ElectricMeterWatts                NULL
  IOT ZWaveNode006FGWP102MeteredWallPlugSwitch_Switch                            NULL
  IOT ZWaveNode006FGWP102MeteredWallPlugSwitch_ResetTheTotalPowerConsumption     NULL

All devices                                                                      Message
  IOT ZWaveNode007ZW100Multisensor6_SensorRelativeHumidity                       41
  IOT Multisensor_6                                                              ONLINE
  IOT ZWaveNode006FGWP102MeteredWallPlugSwitch_ElectricMeterKWh                  NULL
  IOT ZWaveNode007ZW100Multisensor6_SensorTemperature                            27.9 ?C
  IOT ZWaveNode006FGWP102MeteredWallPlugSwitch_ElectricMeterWatts                NULL
  IOT ZWaveNode007ZW100Multisensor6_SensorLuminance                              110
  IOT ZWaveNode004FGMS001MotionSensor_SensorLuminance                            45
  IOT ZWaveNode006FGWP102MeteredWallPlugSwitch_Switch                            NULL
  IOT ZWaveNode006FGWP102MeteredWallPlugSwitch_ResetTheTotalPowerConsumption     NULL
  IOT ZWaveNode007ZW100Multisensor6_SensorUltraviolet                            0.0
  IOT ZWaveNode004FGMS001MotionSensor_SensorTemperature                          25.5 ?C
<-- old                 Last message received 0 seconds ago                 new -->
                                                                             Pause
```

Figure 2: Overview diagram of the **iot_bridge_upgrade** package

There are four topics:

- */iot_command* (diagnostic_msgs/`KeyValue`)
  When iot_bridge receives a name/value pair from the ROS iot_command topic, it publishes those to OpenHAB and OpenHAB sends that command to the device specified.

- */iot_set* (diagnostic_msgs/`KeyValue`)
  When the iot_bridge receives a name/value pair from the ROS iot_set topic, it publishes those to OpenHAB and OpenHAB updates the status for the item specified (e.g. indicates that a switch is now ON).

- */iot_updates* (diagnostic_msgs/`KeyValue`)
  The IoT bridge receives updates from OpenHAB and publishes those as name/value pairs to the iot_updates ROS topic.

- */diagnostics* (diagnostic_msgs/`DiagnosticArray`)
  The IoT bridge receives updates from OpenHAB and publishes those under a DiagnosticArray message to the /diagnostic_agg ROS topic to be monitored by rqt_robot_monitor.

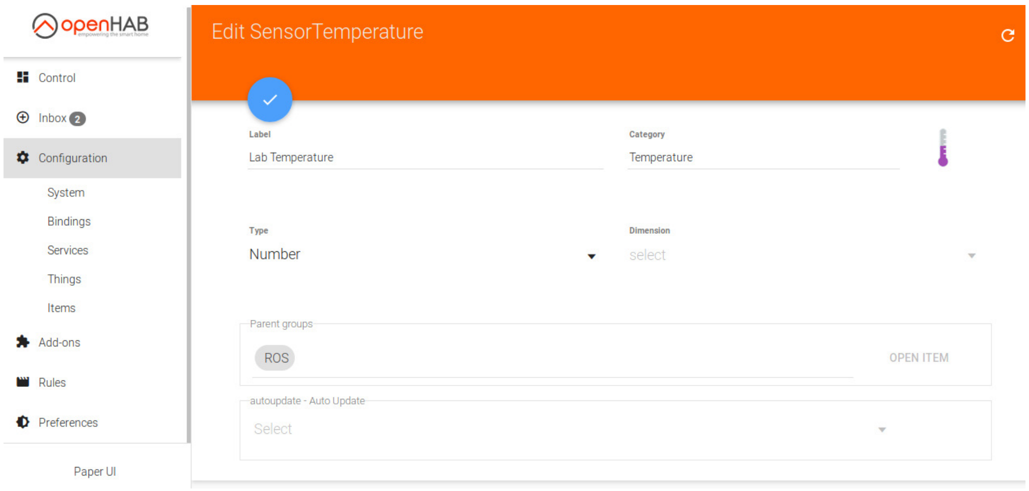
You can see the wiki for more explanations about the topics.

3.4.2 Use the package

The configuration of OpenHAB is made in the `config/items.yaml` file from the package.
The managed Items must be part of the ROS group.
To create the ROS group in OpenHAB you have to create the file `/etc/openhab2/items/ros.items` and add:

```
      Group ROS (All)
      String ROS\_Status "ROS [%s]"
```

Then you can attach the ROS group to the desired Items using PaperUI for instance. They will then be available on ROS.



Edit SensorTemperature

Label: Lab Temperature
Category: Temperature
Type: Number
Dimension: select
Parent groups: ROS
autoupdate - Auto Update: Select

To use the package you can:

- Launch it:

  ```
  roslaunch iot_bridge iot.launch
  ```

  It will start the *iot* node and *rqt_robot_monitor* to monitor the linked Items.

- Run it:

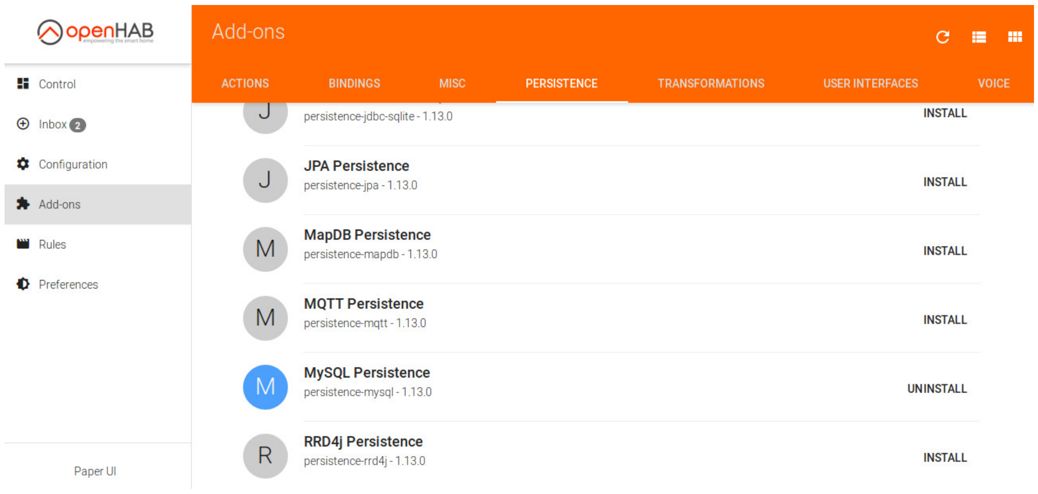
  ```
  rosrun iot_bridge iot_bridge
  ```

You can both use it on the Raspberry Pi or on a computer connected to the same network. To do that you can simply export the ROS Master of the Raspberry to the computer and then launch it. Start a `roscore` on the computer and on the Raspberry run:

```
    $ export ROS_MASTER_URI=http://COMPUTER_IP_ADDRESS:11311
    $ roslaunch iot_bridge iot.launch
```

3.5 Data saving

The sensors and actuators now integrated into the OpenHAB2 software and available on ROS, it seems so worthwhile to save the sensor data so that they can be used by the robot if necessary.
To do this I created a project on the gitlab of the laboratory **OpenHab-MySQL**.
The MySQL persistence is an openHAB plugin that allows you to save data from Items.



persistence-jdbc-sqlite - 1.13.0 INSTALL

JPA Persistence
persistence-jpa - 1.13.0 INSTALL

MapDB Persistence
persistence-mapdb - 1.13.0 INSTALL

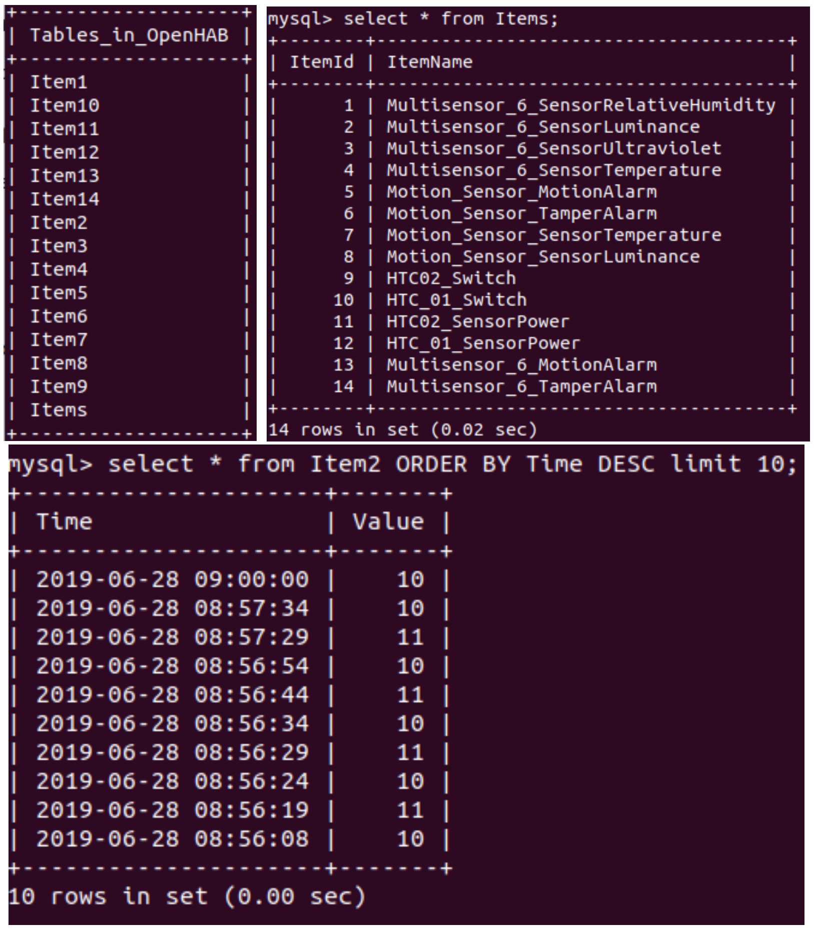
MQTT Persistence
persistence-mqtt - 1.13.0 INSTALL

MySQL Persistence
persistence-mysql - 1.13.0 UNINSTALL

RRD4j Persistence
persistence-rrd4j - 1.13.0 INSTALL

The OpenHAB database created contains two kind of tables:

- Items: links an item with a primay key ID
- Item*: where * is the item's ID from the Items table, contains all the values and the corresponding date from the item.



```
+-------------------+
| Tables_in_OpenHAB |
+-------------------+
| Item1             |
| Item10            |
| Item11            |
| Item12            |
| Item13            |
| Item14            |
| Item2             |
| Item3             |
| Item4             |
| Item5             |
| Item6             |
| Item7             |
| Item8             |
| Item9             |
| Items             |
+-------------------+
```

```
mysql> select * from Items;
+--------+-------------------------------------+
| ItemId | ItemName                            |
+--------+-------------------------------------+
|      1 | Multisensor_6_SensorRelativeHumidity |
|      2 | Multisensor_6_SensorLuminance       |
|      3 | Multisensor_6_SensorUltraviolet     |
|      4 | Multisensor_6_SensorTemperature     |
|      5 | Motion_Sensor_MotionAlarm           |
|      6 | Motion_Sensor_TamperAlarm           |
|      7 | Motion_Sensor_SensorTemperature     |
|      8 | Motion_Sensor_SensorLuminance       |
|      9 | HTC02_Switch                        |
|     10 | HTC_01_Switch                       |
|     11 | HTC02_SensorPower                   |
|     12 | HTC_01_SensorPower                  |
|     13 | Multisensor_6_MotionAlarm           |
|     14 | Multisensor_6_TamperAlarm           |
+--------+-------------------------------------+
14 rows in set (0.02 sec)
```

```
mysql> select * from Item2 ORDER BY Time DESC limit 10;
+---------------------+-------+
| Time                | Value |
+---------------------+-------+
| 2019-06-28 09:00:00 |    10 |
| 2019-06-28 08:57:34 |    10 |
| 2019-06-28 08:57:29 |    11 |
| 2019-06-28 08:56:54 |    10 |
| 2019-06-28 08:56:44 |    11 |
| 2019-06-28 08:56:34 |    10 |
| 2019-06-28 08:56:29 |    11 |
| 2019-06-28 08:56:24 |    10 |
| 2019-06-28 08:56:19 |    11 |
| 2019-06-28 08:56:08 |    10 |
+---------------------+-------+
10 rows in set (0.00 sec)
```

Figure 3: OpenHAB database from MySQL Persistence

So I created a configuration that records all sensor and actuator data associated with the ROS group at each new data and every 30 minutes. This configuration comes from the file `mysql.persist` located in the `/etc/openhab2/persistence/` folder.

```
Strategies {
everyMinutes: "0 */1 * * * ?"
every10Minutes: "0 */10 * * * ?"
every30Minutes: "0 */30 * * * ?"
default = everyChange
}

Items
{
ROS* : strategy = everyChange, every30Minutes, restorOnStartup
}
```

This database being local, I made it accessible for all IPv4 addesses to the network changing into `/etc/mysql/mysql.conf.d/mysqld.cnf` the `bind_address`:

~~~
    ```bind\_address : 0.0.0.0 ```
~~~

Then I created a user for the external connections:

```
sudo mysql -u root -p
mysql> CREATE USER 'new_user' @ 'IP_address' IDENTIFIED BY 'password';
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON OpenHAB. * TO 'new_user' @ '
    IP_address';
mysql> FLUSH PRIVILEGES;
```

E.g. for all local IP addresses:

```
sudo mysql -u root -p
mysql> CREATE USER 'openhab'@'172.16.0.%' IDENTIFIED BY 'MySQLOH2';
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON OpenHAB.* TO ' openhab'@'
    172.16.0.%';
mysql> FLUSH PRIVILEGES;
```

Here are the created accounts:

```
Id: root
Pwd: OpenHAB2MySQL

Id: openhab
Pwd: MySQLOH2
```

Now the data acquired via OpenHAB are now saved in the OpenHAB local MySQL database and accessible from the local network. By this way, the robot increased its sensor data and it can acquire information from devices that are present in the apartment.

4 IP Camera

My biggest project was to integrate an IP Camera in the environment of the robot so that the robot can fulfill missions such as locating an object in the apartment using its own cameras and the IP camera in place. The IP camera was positioned on the ceiling above the apartment.

4.1 Choosen IP Camera

Having chosen OpenHAB as an IoT controller, the choice of the camera was made in the interests of the software integration. A plugin - IpCamera - , in development but usable, allows to connect some IP cameras to OpenHAB2. Here is the list of cameras tested and approved by this plugin:

Brand	Amcrest	Dahua	Foscam	Hikvision	Instar
Model	IP2M-841EW	IPC-HDBW4433R-AS	FI9831W	DS-2CD2385FWD-I	IN-8015 Full HD
	IP2M-841B	IIPC-HFW4431R-Z	FI9821P	DS-2CD2042WD-I	
	IP2M-844	IPC-HDW4421E-AS	FI9900P	DS-2CD2142FWD-IWS	
	IP3M-943B	IPC-HDW2431R-ZS	Fosbaby P1	DS-7208HUI-K2	
	IP8M-2493EW	DH-SD22404T-GN PTZ	C1 Lite	DS-7208HQHI-F1 / N	
			C1	DS-2CD2383G0-I	
			C2	DS-7616NI-K2 / 16P	
				DS-2DE3304W-DE	

Table 1: IP Camera used by the OpenHAB IP Camera addon

Having not found a more robust solution already existing we chose a cheap camera in terms of price / quality ratio to perform integration tests.
The camera we bought is an Amcrest IP3M-941W IP camera at a cost of $ 75. The features of the camera are available in the **annex**.

4.2 IP Camera strategy redefinition

When choosing the adopted strategies and the camera, the connection between the camera and the robot should be via iot_bridge. The camera had to be linked to OpenHAB and communicates via ROS through the bridge. However, after having configured the associated blinding, the speed and the possible action panel were very bad. This did not allow to use the camera dynamically by the robot. So I had to adapt the strategy regarding the camera.

I developed a set of programs so that the chosen IP camera is both usable by the robot via ROS, but also usable directly from the network, while being integrated with OpenHAB. The advantage of the chosen camera is that it has an integrated API in the form of HTTP GET requests. This API allows you to configure videos, control the movement of the PTZ (Pan, Tilt, Zoom) camera, take snapshots, etc. After some research, a Python module - python-amcrest- already existing, I turned to use this language to create the *amcrest_ip_camera* ROS package.

The module does not include all the integrated features so I upgraded it so that it can be integrated into ROS with the necessary features. Two features have been added:

- Possibility of audio recording for a given duration
- Possibility of saving a PTZ position (preset point)

The version incorporating these changes is available in a Git repository on my account: python-amcrest ros_used branch. The second feature has resulted in a merge request with the official module. To install it you can use `pip`:

```
sudo pip install -e  git://github.com/KevinBdn/python-amcrest.git@ros_used#egg=python-amcrest
```

4.3 amcrest_ip_camera ROS package architecture

The package aims to interact with the HTTP API of the camera through different topics. It is therefore a question of making a second API usable via ROS.

Here is the class diagram of the developed package:

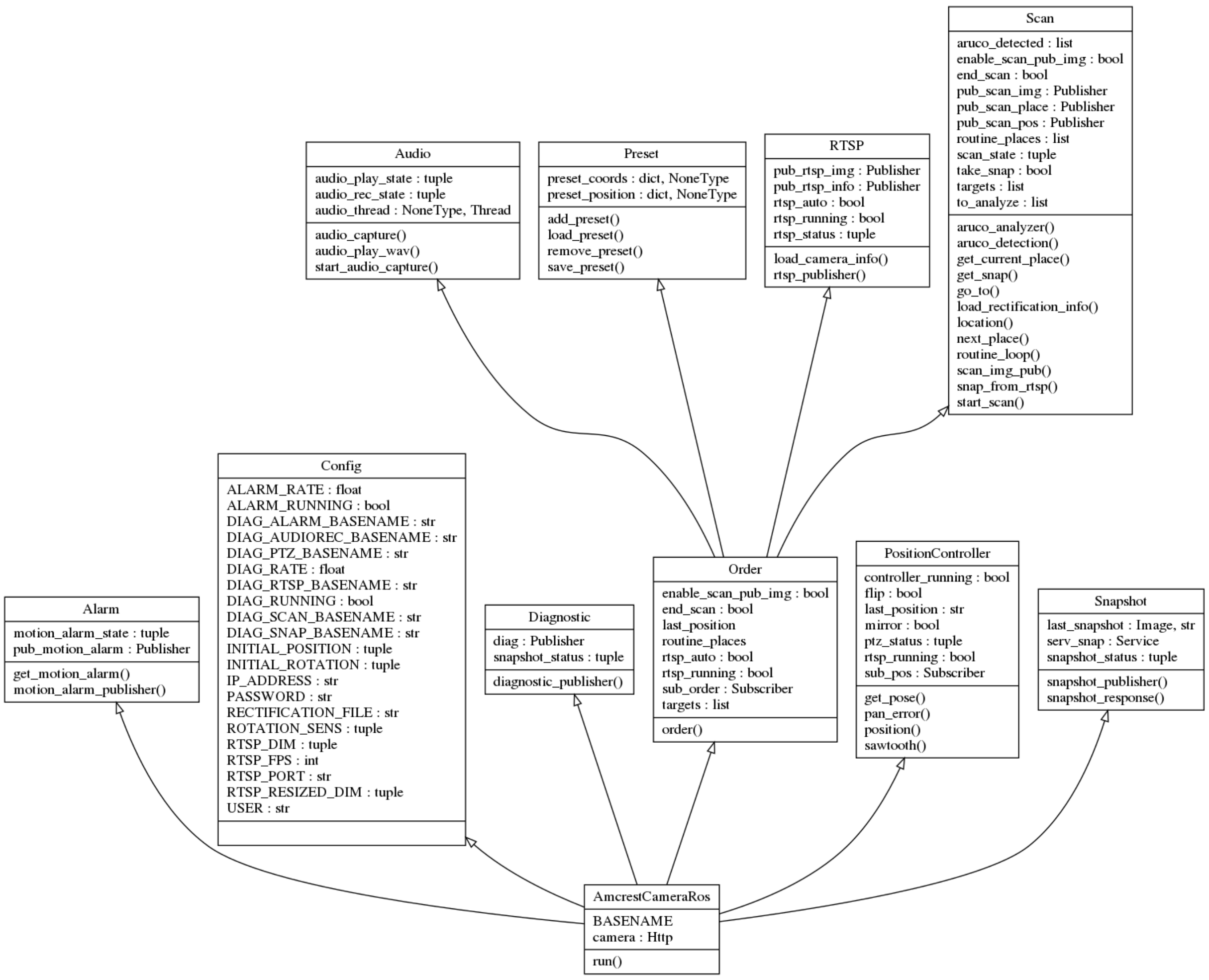


Figure 4: Class diagram of the **amcrest_ip_camera** package

Here is how you can interact with the camera via ROS:

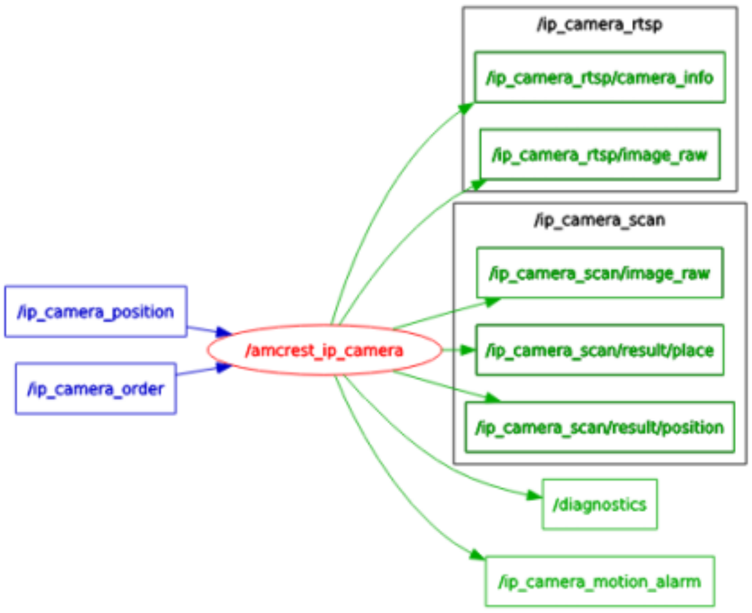

/ip_camera_rtsp
/ip_camera_rtsp/camera_info
/ip_camera_rtsp/image_raw
/ip_camera_scan
/ip_camera_scan/image_raw
/ip_camera_position
/amcrest_ip_camera
/ip_camera_scan/result/place
/ip_camera_order
/ip_camera_scan/result/position
/diagnostics
/ip_camera_motion_alarm

Figure 5: Graph node of the **amcrest_ip_camera** package

Here are short descriptions of the topics and services used:

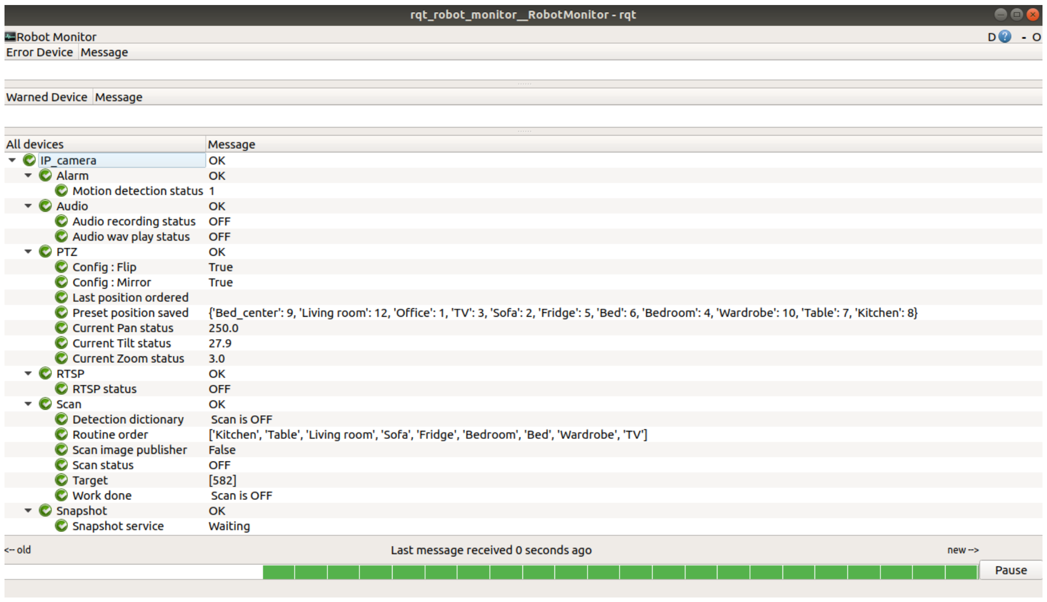
- *ip_camera_position* - [Subscriber] -`Point` message
  Move the camera to the PTZ configuration ordered using the controller
  - x : Pan angle between 0 and 360 (float)
  - y : Tilt angle between -10 and 108 (float)
  - z : Zoom level between 0 and 8 (int) [0 is zoom out maximun]
- *ip_camera_order* - [Subscriber] -`KeyValue` message

Key	**Description**
Goto	Move the camera to a saved Place
SaveAs	Save the current PTZ configuration as a Place
Remove	Remove a saved Place
RTSP	Enable/Disable the RTSP stream
AudioRec	Record audio for a predefined time
AudioPlay	Play an audio file
Mirror	Configure the Image as the mirror of the current Image
Flip	Flip the Image
Scan	Scan the apartment looking for an Aruco
ScanImgPub	Enbale/Disable the analyzed images during the scan routine
SetTarget	Define the Aruco targeted during the scan routine
SetRoutine	Define the place order during the scan routine
AddTarget	Add an Aruco to the targeted Aruco list
RemoveTarget	Remove an Aruco from the targeted Aruco list
Reboot	Reboot the camera
VideoMode	Change the video configuration
Move	Change the PTZ configuration - Up/Down/Left/Right/etc
Zoom	Zoom In or Out

For the possible values you can check the GitLab project.

- *ip_camera_rtsp/image_raw* - [Publisher] -`Image` message
  Publisher running when the associated variable is "True".
- *ip_camera_rtsp/image_raw* - [Publisher] -`CameraInfo` message
  Publisher running when the associated variable is "True".
- *ip_camera_motion_alarm* - [Publisher] -`DiagnosticStatus` message
  The possible values are:
  - 0: No detection
  - 1: Motion detected (less than 15s ago)
  - -1: Alarm trouble
- *ip_camera_scan/result/position* - [Publisher] -`Point` message
  Publisher running when scan is running and when a targeted Aruco is identified. It publishes the (x,y,z) position of the Aruco in the apartment frame.
- *ip_camera_scan/result/place* - [Publisher] -`KeyValue` message
  Publisher running when scan is running and when a targeted Aruco is identified. It publishes the Place location of the Aruco. The key is the Aruco's ID and the value is the Place.
- *diagnostics* - [Publisher] -`DiagnosticArray` message
  Messages monitored by *rqt_robot_monitor*. See below for more details.
- *snapshots_ip_camera* - [Service] -`Trigger` message
  The service gets and provides snapshots from the ip_camera as `TriggerResponse` message when it is called.

The current status of the camera is published in the *diagnostics* topic that can monitor whether or not it works via *rqt_robot_monitor*.



All devices	Message
IP_camera	OK
Alarm	OK
Motion detection status	1
Audio	OK
Audio recording status	OFF
Audio wav play status	OFF
PTZ	OK
Config : Flip	True
Config : Mirror	True
Last position ordered	
Preset position saved	{'Bed_center': 9, 'Living room': 12, 'Office': 1, 'TV': 3, 'Sofa': 2, 'Fridge': 5, 'Bed': 6, 'Bedroom': 4, 'Wardrobe': 10, 'Table': 7, 'Kitchen': 8}
Current Pan status	250.0
Current Tilt status	27.9
Current Zoom status	3.0
RTSP	OK
RTSP status	OFF
Scan	OK
Detection dictionary	Scan is OFF
Routine order	['Kitchen', 'Table', 'Living room', 'Sofa', 'Fridge', 'Bedroom', 'Bed', 'Wardrobe', 'TV']
Scan image publisher	False
Scan status	OFF
Target	[582]
Work done	Scan is OFF
Snapshot	OK
Snapshot service	Waiting

Figure 6: *rqt_robot_monitor* from **amcrest_ip_camera** package

I invite you to check the GitLab project to have more details about the displayed messages.

4.4 Meet the missions

The camera is now integrated with ROS so it is possible to meet the missions that will be entrusted to it. For that, I will go back in more details on some of the developed functionalities.

4.4.1 PTZ controller: controller.py

A proportional controller has been integrated. The manufacturer's HTTP API allows to send a single *start* or *stop* command in one direction (left, right, up, down) with a speed ranging from 1 to 8. The same applies for the zoom. The API also provides the possibility of getting the current PTZ configuration.
However, even when sending a *start* then *stop* request with a minimum speed, the camera taking a while to process the HTTP requests - the minimum step is about 0.3°for the pan or the tilt. In addition you should know that the camera has a "no-go" area at the Pan angle between 180° and 185°. Regarding the zoom, there are 5 levels of current state but the level between two levels is very wide so that for the same zoom value the actual magnification can be different. Thanks to these various elements I was able to create a regulator allowing to position the camera in a desired PTZ configuration. In reality, there are three separated regulators because the camera can only handle one type of movement at a time.
Here is the simplified algorithm implemented for the Pan angle controller:

Algorithm 1 Simplified Pan angle controller

```
1: function SAWTOOTH($\alpha$)                                        ▷ Implementation in degrees
2:     $\beta \leftarrow (\alpha + 180) \bmod (360) - 180$
3:     return $\beta$
4:
5: function PAN_ERROR($\theta, \bar{\theta}$)
6:     $\epsilon \leftarrow Sawtooth(\theta - \bar{\theta})$
7:     if $\epsilon < 0$ and *no-go zone* is between $\theta$ and $\bar{\theta}$ then
8:         $\epsilon \leftarrow \epsilon + 360$
9:     else if $\epsilon > 0$ and *no-go zone* is between $\bar{\theta}$ and $\theta$ then
10:        $\epsilon \leftarrow \epsilon - 360$
11:    return $\epsilon$
12:
13: function PAN_CONTROLLER($\bar{\theta}, \Delta_\theta, \Delta_t, K$)
14:    if $\bar{\theta}$ is in the *no-go zone* then                      ▷ $\bar{\theta}$: Goal Pan angle
15:        $\bar{\theta} \leftarrow$ 180°or 185°according to the nearest angle
16:    $\theta \leftarrow$ Reading the current Pan angle
17:    $\epsilon \leftarrow Sawtooth(\theta - \bar{\theta})$
18:    while $|\epsilon| > \Delta_\theta$ do                              ▷ $\Delta_\theta$: maximum error
19:        $u \leftarrow \lfloor min(8, K.|\epsilon|) \rfloor$             ▷ $K$: proportionality coefficient
20:        if $\epsilon < 0$ then
21:            Move the camera to the right at speed set to $u$
22:        else
23:            Move the camera to the left at speed set to $u$
24:        Wait for $\Delta_t$ seconds                                  ▷ $\Delta_t$: time interval
25:        if $|\epsilon| < 7$ then
26:            Stop the camera                                  ▷ Step by step when the error is low
27:        $\theta \leftarrow$ Reading the current Pan angle
28:        $\epsilon \leftarrow Sawtooth(\theta - \bar{\theta})$
```

Here is schematically the utility of the *Pan_error* function:

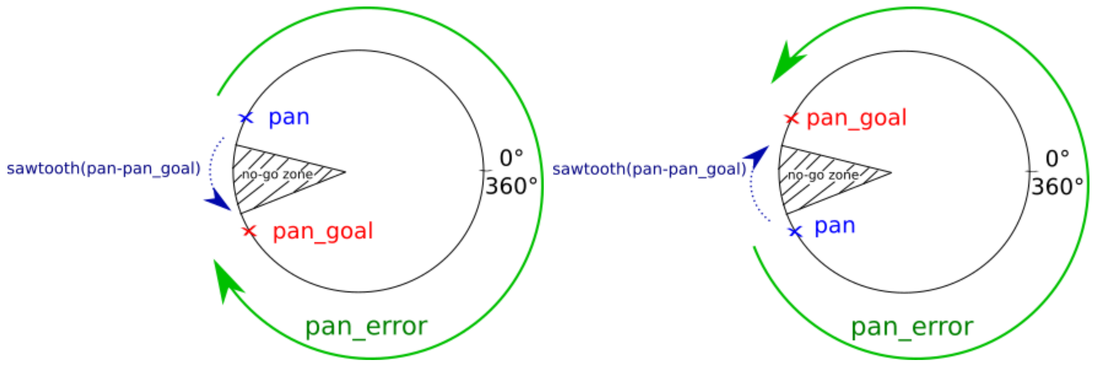


Figure 7: Illustration of the *Pan_error* function

4.4.2 Place saving

Now able to position the camera in a desired configuration thanks to the controller, I set up an association between a particular position and a place. Thus after configuration, using the `Order` topic with `Goto` as key, you can point the camera on the Kitchen, Living room, Bedroom, TV, etc., of the apartment. And save it using the `Order` topic with `SaveAs` as key. The dictionaries thus created are saved in local `.yaml` files to keep in memory these associations.

```
 conf/preset_coords.yaml #Preset point association - Place :
preset_point_number
 conf/preset_position.yaml #PTZ configuration association - Place : (Pan,Tilt
,Zoom)
```

4.4.3 Calibration

The camera has been calibrated using the MonocularCalibration tutorial from the officale ROS wiki (http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration). The result is saved in the file:

```
 conf/ost.yaml
```

4.4.4 Aruco detection

Now that the IP camera is calibrated, it can be used to detect Aruco. Two possibilities could be exploited:

- Use OpenCV which has an ArUco library, do the processing via this library directly in the package and publish the result

- Use the ROS package *aruco_detect* which must first have an already corrected image

Schematically here are the two solutions:

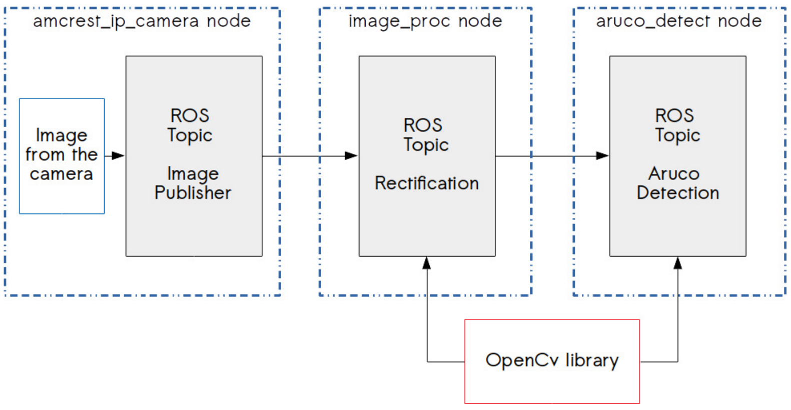


Figure 8: Aruco detection directly using OpenCV library

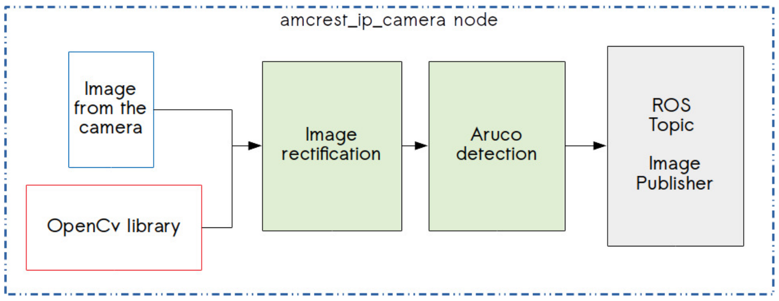


Figure 9: Aruco detection using *aruco_detect* ROS package

ROS consuming many resources, I chose to do the treatment internally in the package and not to use the existing package. The OpenCV function gives the translation and the rotation vector of the Aruco in the camera's reference frame. With a change of base and using the Rodrigues' formula [2], it is possible to locate the Aruco in the apartment's reference frame.

4.4.5 Scan algorithm: scan.py

The camera is now able to:

- Point in a desired direction
- Detect an Aruco and locate it in the camera reference frame

We can now consider an algorithm allowing the camera to scan the apartment in search of an Aruco and providing its location in the reference frame of the apartment.
Here is the simplified algorithm implemented in the form of a flowchart:

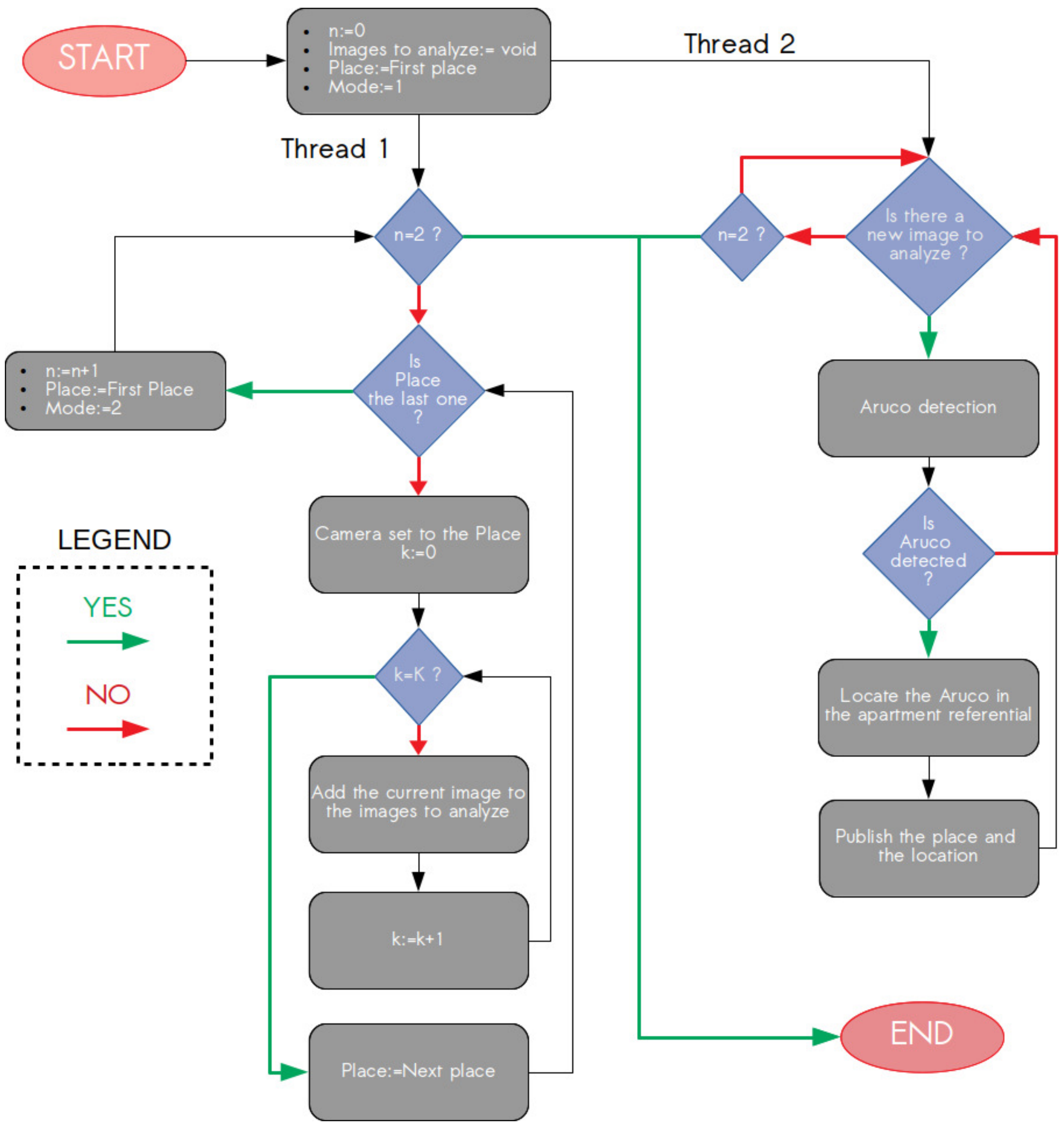


Figure 10: Flowchart of the simplified scan algorithm

It is therefore divided into two main threads:

- The first is to make two turns of the different places defined in arguments. For each place we capture K (integer in argument) photos (from the stream RTSP) to have slightly different images for the same place. These captures are stored in an image queue to be analyzed. During the first turn, the camera configuration is in a first mode in Black and White with a high contrast, which allows to see better the Aruco. During the second turn, the camera is in a color mode. Thus we finally capture 2 * K images including K of each mode.

- The second is to scroll through the queue of images to be analyzed from the first thread. As soon as a new image appears, it is analyzed to detect an Aruco. If this is the case, then one calculates its position in the reference frame of the apartment having the configuration in Pan and Tilt of the camera, knowing its initial position and obtaining the position of the Aruco in the reference frame of the camera. We can then make a change of reference and publish the position of the calculated Aruco and the place in which it was detected.

The use of threads allows you to go faster by doing parallel operations. Thus the analysis of the images can be done while the scan routine continues. For instance, the scan of the apartment in 9 places with K = 8, so 144 images, is in about 2 minutes.

4.5 Package use - [TUTORIAL]

4.5.1 Requirement

- **amcrest_ip_camera** package installed
- **python-amcrest** upgraded package installed
- OpenCV version > 3.2: The scan algorithm use it to detect the Aruco.

4.5.2 Configurations

In the `conf/` repository you can find the several files of configuration:

- Diagnostics Publisher: `conf/amcrest_analyzers.yaml`
- ROS and Camera configuration: `conf/parameters.yaml`
- Preset saved: `conf/preset_coords.yaml` and `conf/preset_position.yaml`
- Camera rectification: `conf/ost.yaml`
- RViz configuration: `conf/ip_camera.rviz`
- Lab's map configuration: `conf/pm_lab/*`

To change the number of analyzed images for each place and to change the time between the move and the snapshot taking, the variables are in `src/scan.py`:

```
    class Scan(object):
        def __init__(self):
            [...]
            self._routine_delay=4#Delay between a motion and a snapshot capture
            self._rtsp_snap_number=3#Number of taken snapshot
            [...]
```

4.5.3 Run the package

To run the **amcrest_ip_camera** you just have to launch it:

```
roslaunch  amcrest_ip_camera   amcrest_ip_camera.launch
```

It will launch:

- amcrest_ip_camera: the main node
- map_server : publication of the lab's map avalaible in the `/conf/pm_lab/` folder
- RViz : display the lab's map loaded and the three TF (center,Camera and Aruco) when an Aruco is detected
- Image viewer : display the RTSP and the analyzed images during the scan routine if enabled
- Robot Monitor: display the `diagnostics` messages from the package

On the Raspberry Pi you have to save resources to use effectively the package:
First disable the diagnostic and alarm topics. In the `conf/parameters.yaml` file:

```
    DIAG_RUNNING : False
    ALARM_RUNNING : False
```

Then you can take only one snapshot from each place and set the delay between the motion and the taking to 5 seconds. In the in `src/scan.py` file:

```
    class Scan(object):
        def __init__(self):
            [...]
            self._routine_delay=5
            self._rtsp_snap_number=1
            [...]
```

You can export the ROS master to have access to the topics on another computer:

```
    $ export ROS_MASTER_URI=http://COMPUTER_IP_ADDRESS:11311
```

To finish, you can directly run it using rosrun:

```
    $ rosrun     amcrest_ip_camera    amcrest_ip_camera.py
```

4.5.4 Known issues

Some trouble might occur using the package.

- **Network bandwidth trouble**: when there are too much devices connected to the local network, the HTTP request used by the package to communicate by the Camera are long to treat. The request queue increases faster than it decreases. In *rqt_robot_monitor* appears trouble with `IDLE` state and the RTSP stream freezes. It occurs when the RTSP stream is running. First you can disable it using the implemented `Order`. Furthermore, it is possible to decrease the bit rate of the stream (the better is to set it below 2048 kb/s) using the camera server software `IP_CAMERA_ADDRESS:80`. If you are using the package on the Raspberry and the scan routine, make sure that you have set the parameters as described above and increasing the delay can probably help.

- **Aruco location**: when the Place is defined by a PTZ configuration in which the Zoom is more than 1, the Aruco location cannot be provided. Only the place is available in the *ip_camera_scan/result/place* topic. This is because the real Zoom, for a particular measured level, can be any one between two consecutive levels. A second trouble can occurs with the Aruco location. Sometimes the rotation matrix of the founded Aruco appears as wrong. In reality it is a singularity problem with the OpenCV function. You can have more information about it online.

4.6 OpenHAB integration

The camera can be managed from the online HMI provided by the manufacturer and now from ROS. It seemed interesting to me to control it from OpenHAB2, the initial idea. So I created a new Gitlab project: **ip_camera_openhab**.
This project consists in generating the necessary scripts to integrate the camera, used by OpenHAB2 from two configuration files:

- The first being the necessary information for its use ie: IP address, user, password, port.

- The second is the dictionary of Places and preset points that can be generated and modified via ROS (see above)

The principle is then to generate switches for each recorded place making it possible to point the camera according to the corresponding place. But also to have a visual via MJPG stream that can be enabled or disabled (to save the bandwidth of the network). It is by directly using the HTTP GET requests of the manufacturer's API that it is possible to integrate these elements to OH2.

Thus the generator program resulting from this project is based on Parser techniques in order to generate codes. I used Python3.6 and the Jinja2 module for their ease of use to generate scripts based on templates. In addition, this program allows you to add several different cameras using different configuration files, that can be useful if the laboratory decides to buy new IP cameras.

The generated scripts are:

- The interrupter - `.items` - associated with the camera: Multi-choice switch for the places and the switch for activating or deactivating the MJPG stream.
- An entity - `.sitemaps` regrouping these switches, as well as the video stream.

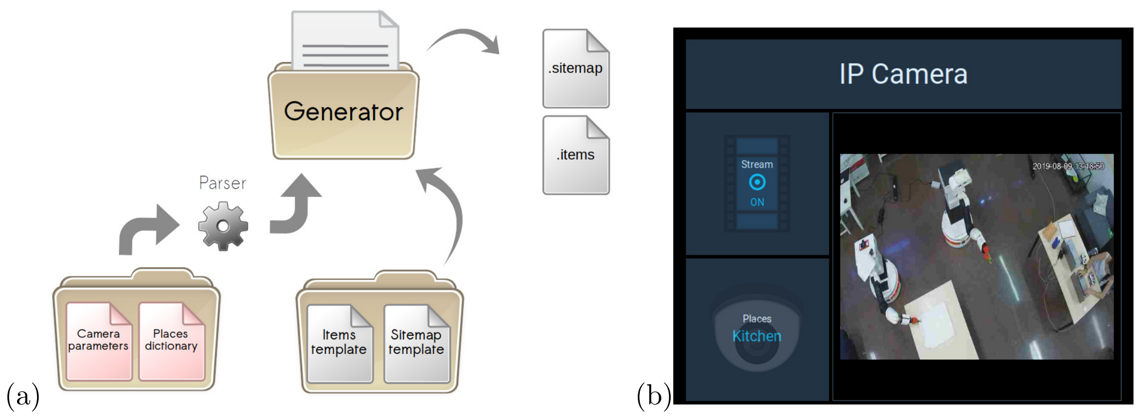


Figure 11: The script generator process (a) and its result in HABPanel (b)

After comparison, this method allows a much faster camera management via OpenHAB2 than when using the developing binding initially considered.

To use it you have to run:

```
python3  generator
```

The configuration files are in the `conf/` repository and the result files in the `results/` repository. You have to copy/paste the scripts in the corresponding repository in the Raspberry : `/etc/openhab2/`.

5 Implementation of a mission

After having implemented the scan algorithm in the ROS package, we are now able to use it with TIAGo in a mission of Aruco detection. Here is, under a flowchart form, an algorithm tested to success this kind of mission.

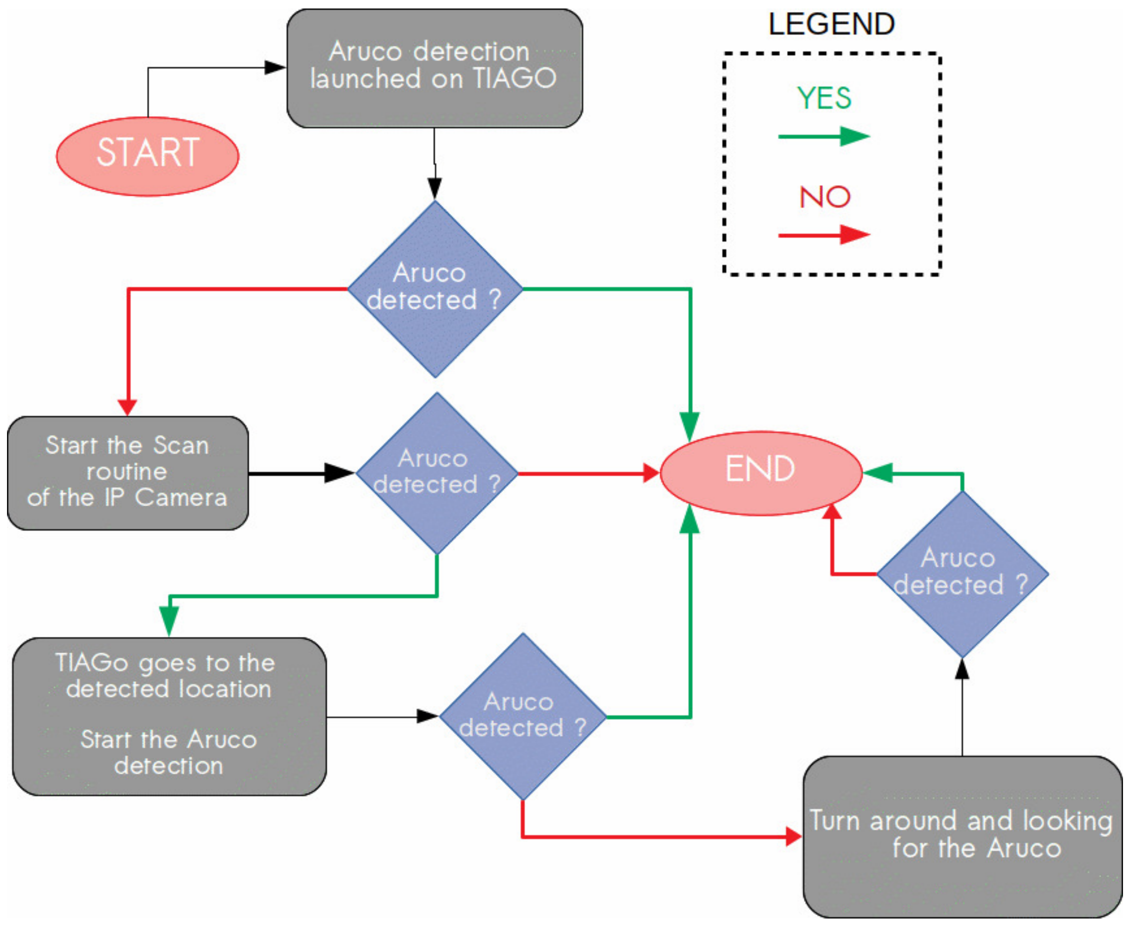

START
Aruco detection launched on TIAGO
Aruco detected ?
Start the Scan routine of the IP Camera
Aruco detected ?
END
TIAGo goes to the detected location
Start the Aruco detection
Aruco detected ?
Turn around and looking for the Aruco
Aruco detected ?
LEGEND
YES
NO

Figure 12: Flowchart of the algorithm used in the mission of Aruco detection

To make it possible I used the Lab's API on a computer making the bridge between the Raspberry Pi and TIAGo and providing a way to monitor the mission.

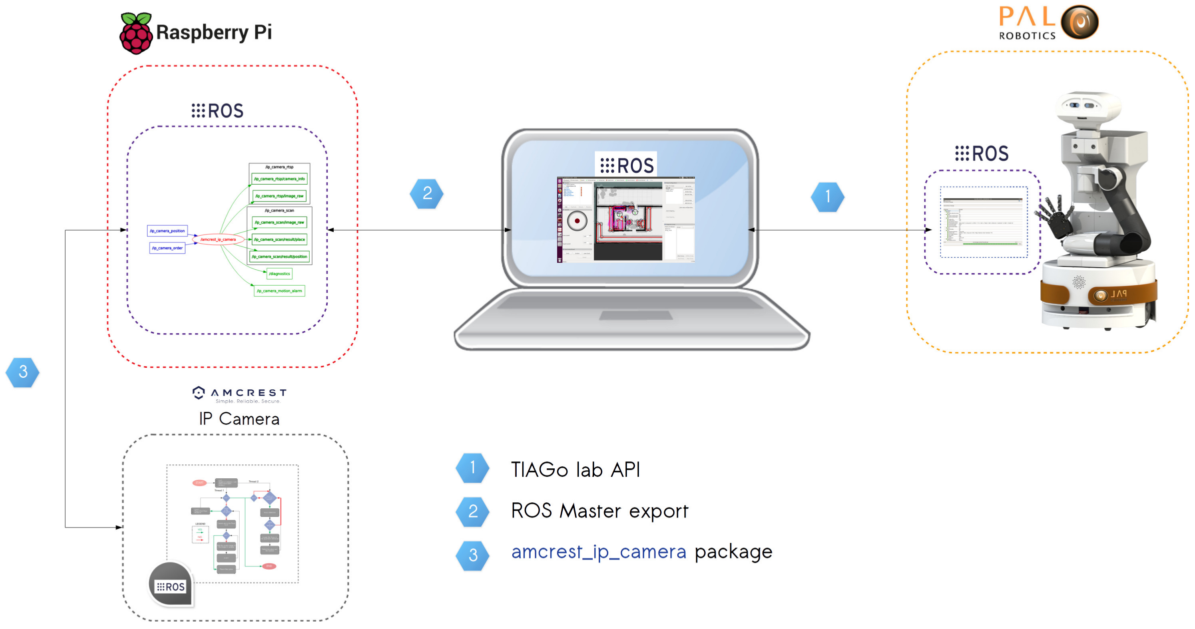


Figure 13: Architecture used in the mission of Aruco detection

Here is the process used in the performed test:

- An Aruco number 582 is placed on the table in the living room near to the sofa. The TIAGo is the bedroom.

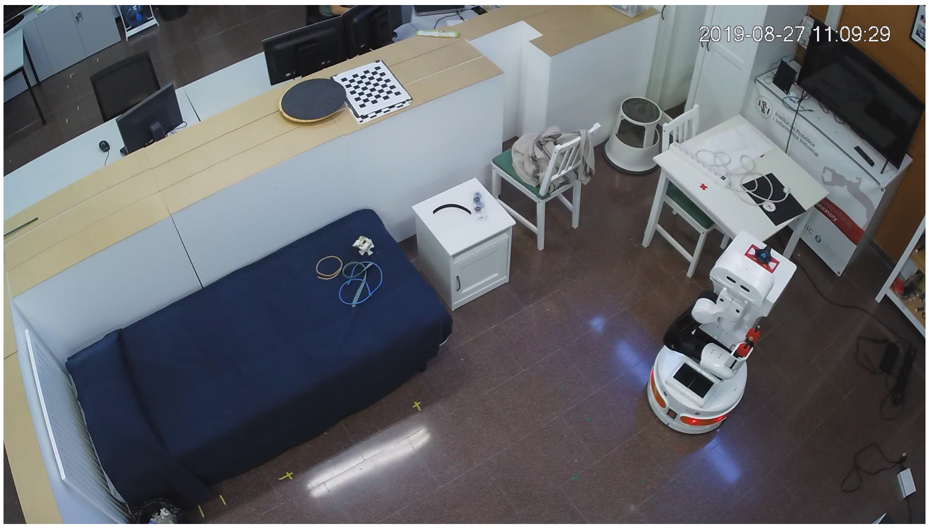


Figure 14: Initial position of TIAGo

- A kitchen point and a living room point are saved in the TIAGo's map.

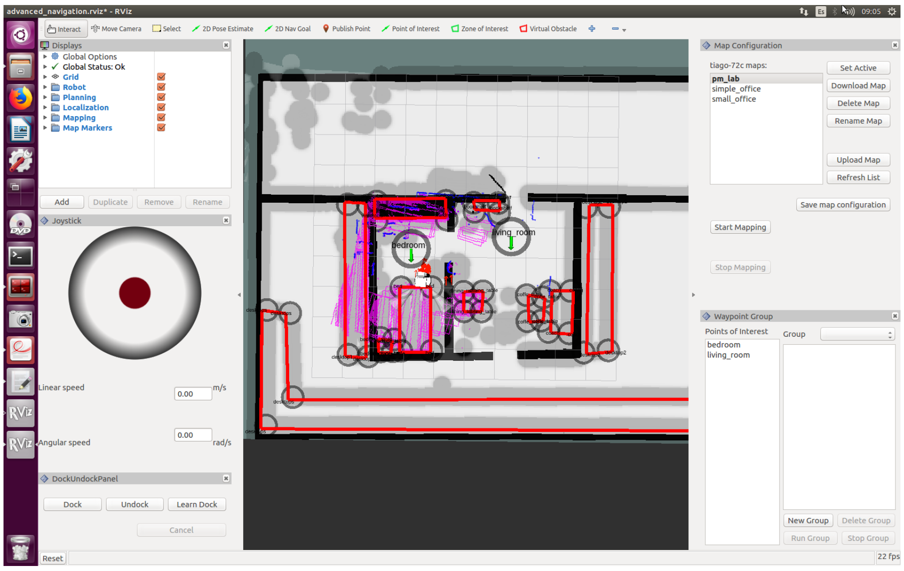
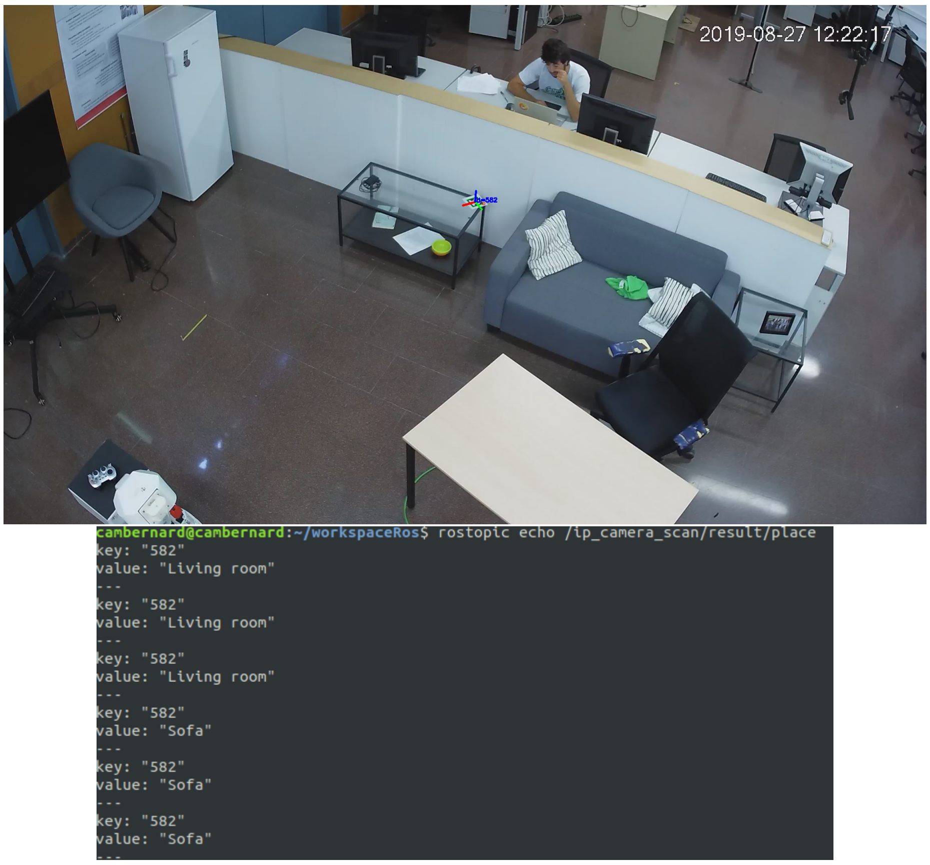


Figure 15: TIAGo's map

- TIAGo does not find the Aruco using its own camera and the ROS package *aruco_detect*. The scan routine from the **amcrest_ip_camera** package on the Raspberry Pi is started with a ROS export on the monitoring computer. When the scan routine finds an Aruco, it publishes its location on the *ip_camera_scan/result/place* topic.



```
cambernard@cambernard:~/workspaceRos$ rostopic echo /ip_camera_scan/result/place
key: "582"
value: "Living room"
---
key: "582"
value: "Living room"
---
key: "582"
value: "Living room"
---
key: "582"
value: "Sofa"
---
key: "582"
value: "Sofa"
---
key: "582"
value: "Sofa"
---
```

Figure 16: Result of the scan routine algorithm

- Through the Lab's API, TIAGo is ordered to go to the corresponding place.

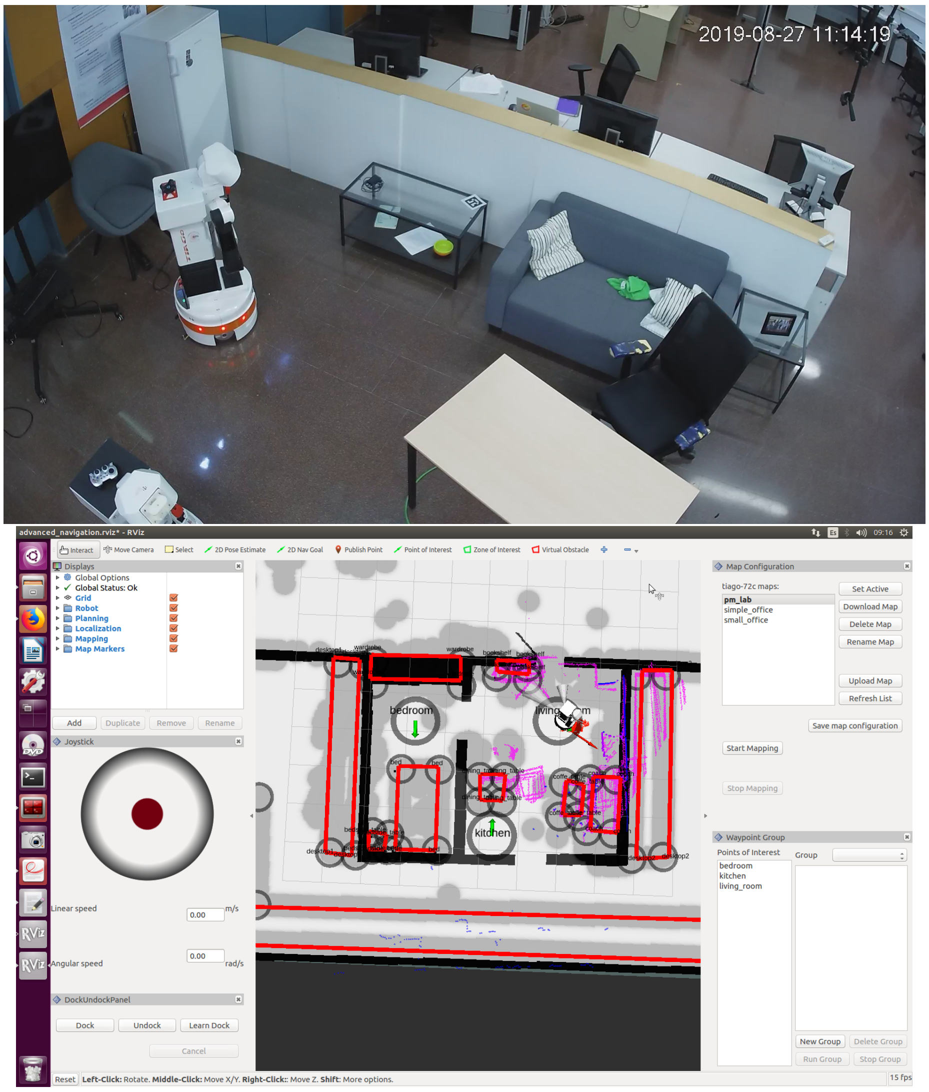


Figure 17: TIAGo's positioning according to the scan routine algorithm

- TIAGo finds the Aruco by itself using *aruco_detect* package.

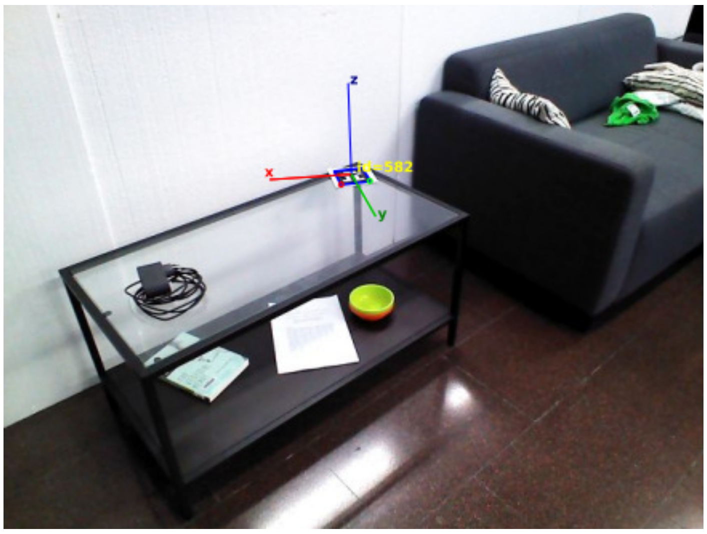


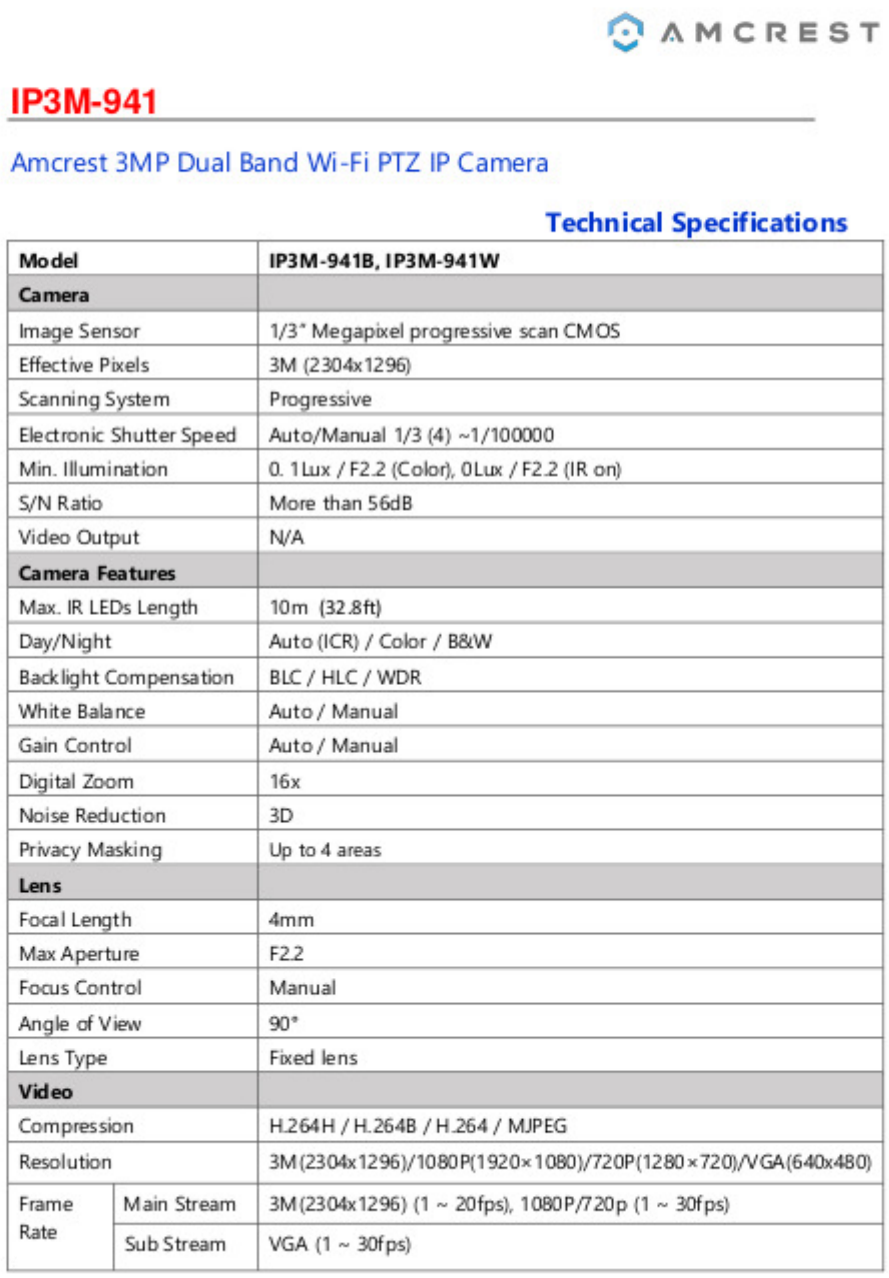
Figure 18: Aruco detection from TIAGo

This example of implementation shows that it is useful to use IoT devices in a mission for assistive automation. We can now imagine more complex algorithms and missions where IoT devices are part of the robot's environment.

6 Conclusions

This project has shown that it was currently possible to integrate the many devices of the world of IoT within a service robot using the ROS middleware. Such an implementation not only enriches the robot's sensors but also increases its range of actions. The use of IoT in robotics thus makes it possible to increase the speed with which a robot can take information by providing it with new tools. As we do it, the robot can - according to the mission which has been given to him - use or not this information. I was able to highlight this phenomenon through the mission of detection and location of an Aruco in the apartment. However, using several IoT devices such IP cameras can consume more resources than a Raspberry Pi 3B+ is able to provide. It would be interesting to study other hardware solutions and compare it.

A Amcrest IP3M-941W IP camera features



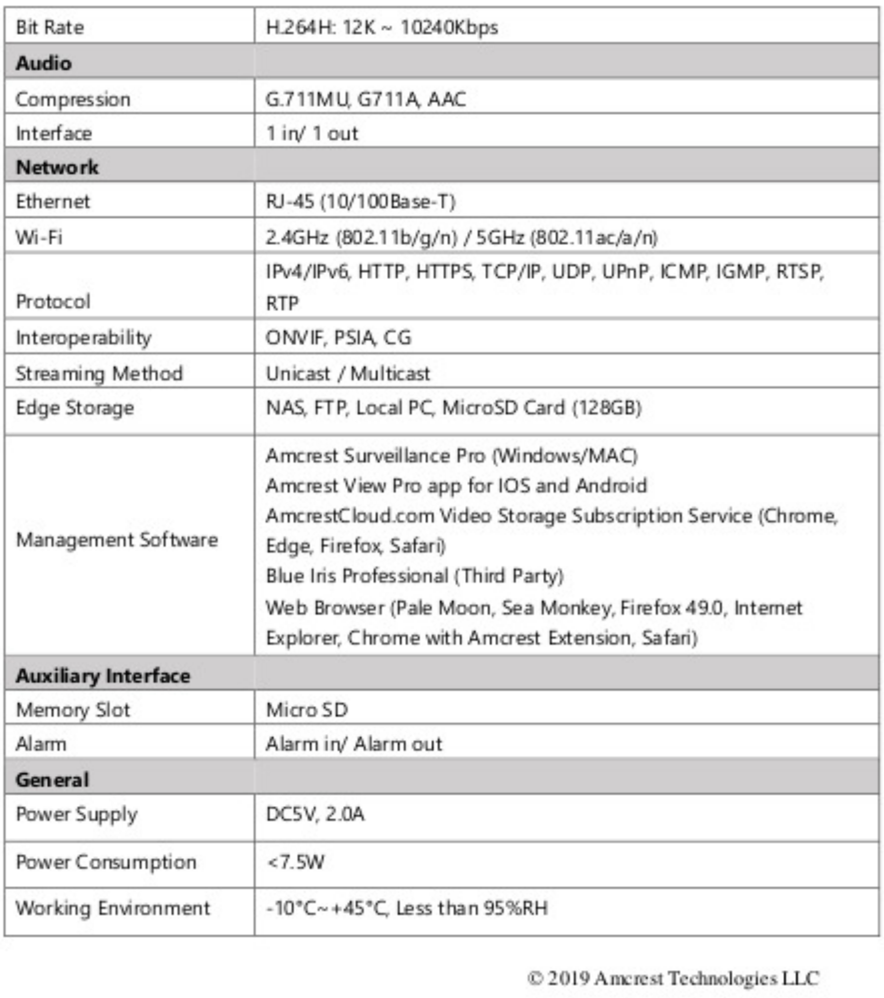
AMCREST

IP3M-941

Amcrest 3MP Dual Band Wi-Fi PTZ IP Camera

Technical Specifications

Model		**IP3M-941B, IP3M-941W**
Camera		
Image Sensor		1/3" Megapixel progressive scan CMOS
Effective Pixels		3M (2304x1296)
Scanning System		Progressive
Electronic Shutter Speed		Auto/Manual 1/3 (4) ~1/100000
Min. Illumination		0. 1Lux / F2.2 (Color), 0Lux / F2.2 (IR on)
S/N Ratio		More than 56dB
Video Output		N/A
Camera Features		
Max. IR LEDs Length		10m (32.8ft)
Day/Night		Auto (ICR) / Color / B&W
Backlight Compensation		BLC / HLC / WDR
White Balance		Auto / Manual
Gain Control		Auto / Manual
Digital Zoom		16x
Noise Reduction		3D
Privacy Masking		Up to 4 areas
Lens		
Focal Length		4mm
Max Aperture		F2.2
Focus Control		Manual
Angle of View		90°
Lens Type		Fixed lens
Video		
Compression		H.264H / H.264B / H.264 / MJPEG
Resolution		3M(2304x1296)/1080P(1920×1080)/720P(1280×720)/VGA(640x480)
Frame Rate	Main Stream	3M(2304x1296) (1 ~ 20fps), 1080P/720p (1 ~ 30fps)
	Sub Stream	VGA (1 ~ 30fps)



Bit Rate	H.264H: 12K ~ 10240Kbps
Audio	
Compression	G.711MU, G711A, AAC
Interface	1 in/ 1 out
Network	
Ethernet	RJ-45 (10/100Base-T)
Wi-Fi	2.4GHz (802.11b/g/n) / 5GHz (802.11ac/a/n)
Protocol	IPv4/IPv6, HTTP, HTTPS, TCP/IP, UDP, UPnP, ICMP, IGMP, RTSP, RTP
Interoperability	ONVIF, PSIA, CG
Streaming Method	Unicast / Multicast
Edge Storage	NAS, FTP, Local PC, MicroSD Card (128GB)
Management Software	Amcrest Surveillance Pro (Windows/MAC) Amcrest View Pro app for IOS and Android AmcrestCloud.com Video Storage Subscription Service (Chrome, Edge, Firefox, Safari) Blue Iris Professional (Third Party) Web Browser (Pale Moon, Sea Monkey, Firefox 49.0, Internet Explorer, Chrome with Amcrest Extension, Safari)
Auxiliary Interface	
Memory Slot	Micro SD
Alarm	Alarm in/ Alarm out
General	
Power Supply	DC5V, 2.0A
Power Consumption	<7.5W
Working Environment	-10°C~+45°C, Less than 95%RH

B Accounts and configurations

- Raspberry access:
  - IP address: `172.16.0.3`
  - User: `ubuntu`
  - Password: `ubuntu`
- Raspberry Wifi hotspot:
  - SSID: `ubiquityrobotXXXX` where XXXX is part of the MAC address
  - Password: `robotseverywhere`
- OpenHAB access:
  - Local access: `172.16.0.3:8080`
- myopenhab.org account:
  - User: `perceptionmanipulationlab@gmail.com`
  - Password: `5uX7QNw9Go9u`
- Amazon Alexa account:
  - User: `perceptionmanipulationlab@gmail.com`
  - Password: `pmlab1234`
- OpenHAB MySQL database:
  - Database name: `OpenHAB`
  - Root access: User: `root` - Password: `OpenHAB2MySQL`
  - Other access: User: `openhab` - Password: `MySQLOH2`
- IP Camera:
  - IP address: `172.16.0.9`
  - Admin access: User: `admin` - Password: `5uX7QNw9Go9u+`
  - OpenHAB access: User: `openhab` - Password: `IpCameraOH2`
  - ROS access: User: `ros` - Password: `IpCameraROS8+`

References

[1] "IRI - Robots and IoT devices for assistive automation." [Online]. Available: https://www.iri.upc.edu/pfc/show/179

[2] J. S. Dai, "Euler–rodrigues formula variations, quaternion conjugation and intrinsic connections," *Mechanism and Machine Theory*, vol. 92, pp. 144–152, 2015.

Acknowledgements

Thanks to Guillem Alenyà Ribas and Sergi Foix Salmerón for their welcome, support and trust throughout this project.
The research leading to these results has received funding from the BIoTRoS project (CSIC-201850I098).

IRI reports