University of Pisa
School of Engineering

Internet of things

JUNO: IoT telemetry system for asteroid exploration

Martina Burgisi

February 14, 2025

Contents

Chapter 1

Introduction

Objective

This project aims to create an IoT system for autonomous navigation of a rover intended for asteroid exploration, characterized by low gravity and irregular surfaces. An adaptive navigation system that uses real-time data to stabilize the rover and allow it to safely explore complex, low-gravity surfaces.

Such a rover would have to take several factors into account. Assuming, for example, that the mission objective is to visit Vesta, among the largest bodies in the asteroid belt of the inner solar system, the rover would face the following challenges:

- **reduced gravity** Vesta, with a diameter of about 525 km, has a surface gravity of about 2.5% of Earth's gravity. A conventional rover would risk "bouncing" or having difficulty maintaining contact with the ground. To overcome this problem, it could be equipped with: robotic legs similar to those of the Philae lander from the Rosetta mission (landed on comet 67P/Churyumov-Gerasimenko, with spikes to cling to the ground; gas thrusters to maintain stability.

- **Erratic terrain** Vesta's surface is full of craters and rocks, which would make locomotion by a classic wheeled rover difficult. An alternative locomotion system could include: a hopping rover, such as those proposed for smaller asteroids (E.g., the MINERVA-II of the Hayabusa2 mission); a rover with legs, capable of insect-like movement.

- **Absence of atmosphere** Vesta has no significant atmosphere, so parachutes cannot be used for landing.

- **Thermal excursion** The rover must be designed to withstand large thermal excursions between day and night.

- **limited solar energy** Vesta's average distance from the Sun is about 2.36 AU, so sunlight is weaker than on Earth. The rover should have larger, more efficient

solar panels; long-life batteries or RTGs (radioisotope thermoelectric generators) to provide continuous power.

- **Communication delay** Given the distance from Earth, the radio signal takes between 5 and 20 minutes to travel to and from, depending on orbital position. Direct manual guidance is not feasible. A possible alternate solution is therefore teleoperation with delay: the operator sends a command and waits for feedback (as with rovers on Mars).

For educational purposes, I have therefore chosen to deal with the case where the rover, which has already successfully landed on a large asteroid such as Vesta or on the dwarf planet Ceres, consists of robotic legs with harpoons and has the mission of moving over the ground to map the surface. Thus focusing exclusively on possible motion management, the goal is then to take care of part of the network of sensors and actuators that would be part of the rover's electronic components to help it move by managing obstacles. I have named this fictional rover JUNO, the corresponding Roman goddess of the Greek Hera, who gives her name to one of the last missions to asteroids launched by ESA in October 2024.

Chapter 2

IoT Network

2.1 Sensors

The project includes two simulated sensors, both handled using:

```
MQTT
    motion-hub
        motion-hub.c
        motion-hub.h
        project_conf.h
        Makefile
```

Motion-hub takes care of subscribing to Mosquitto MQTT broker, receiving commands from the application controller and publish messages generated from sensors.

2.1.1 LiDAR

A LiDAR (Light Detection and Ranging) is a sensor that uses light to detect the environment and objects. A laser light is sent from a source, the transmitter, and reflected from an object in the scene. The time of flight is used to develop a distance map of the objects in the scene. For the JUNO project, a LiDAR is simulated to map the asteroid's surface and detect obstacles.

LiDAR Simulation A real LiDAR sensor has a typical scan range of shapes circular or semi-circular. In the project, the LiDAR contribution is simulated generating randomly distances in 3 directions: front, right and left, as a simplification of a semi-circular-range device.

Project C files

```
sensing
    lidar
```

```
lidar.c
lidar.h
project_conf.h
```

2.1.2 Gyroscope

A gyroscope is a rotating device that uses the law of angular momentum conservation to maintain the rotation axis in a fixed direction. If it's made in a way that it can orient freely in the 3 space dimensions, the gyroscope can stay in the same direction even if the support varies in orientation. This sensor can be used to provide stability or maintain a reference direction in navigation systems or automatic pilots.

Gyroscope Simulation Similarly to the real Gyroscope device, the sensor in the project is represented with 3 axes. The inclination is represented by:

- pitch angle: inclination with respect to x-axis;
- roll angle: inclination with respect to y-axis;
- yaw angle: inclination with respect to z-axis.

All these angles are randomly generated simulating the extremely irregular surface of an asteroid.

Project C files

```
sensing
    gyroscope
        gyroscope.c
        gyroscope.h
        project_conf.h
```

2.2 Actuators

The actuators are handled with the CoAP protocol: a CoAP client has been developed in the Java Application, with the aim of communicating commands and decisions.

2.2.1 Legs Servo Motors

The actuators associated with the LiDAR sensors are the legs servo motors. In order to move a leg and go forward, it should be lifted, extended and then moved. In a typical four-legged space robot, this is accomplished with two servo motors: one is rotated to raise off the ground and the other is rotated to extend the leg. Again, the servo motor 2 is activated to place the leg on the ground. We also need a third motor to make the rover move in a straight position. All the operations that the motors execute in synergy on one leg have to be repeated for every leg.

Servo Motors Simulation To simulate the usage of the legs with all the complex movements involved, the project simply considers the activation of the motors in one direction or in another, based on the decision algorithm. The movement of the motors also will update the position of the rover while it walks on the surface.

Project C files

```
CoAP
    legs-servo-motors
        legs-servo-motors.c
        legs-servo-motors.h
        project_conf.h
        Makefile
    resources
        movement.c
```

2.2.2 Harpoons

Harpoons are associated with the gyroscope values of slope. The irregular surface of the asteroid, mixed with very low gravity, can cause a dangerous inclination of the rover with the risk of flipping over and, as a consequence, damaging its components. To avoid this problem, every leg is equipped with harpoons that anchor to the rock from which an asteroid is made.

Harpoons Simulation Based on the slope detected by the gyroscope, the harpoons can be activated or deactivated to allow better anchoring in different conditions and the continuation of the automatic walk.

Project C files

```
CoAP
    harpoons
        harpoons.c
        harpoons.h
        project_conf.h
        Makefile
    resources
        anchoring.c
```

Chapter 3

Application

As follows, the Application has been developed with the main class Controller and 3 packages: MQTT, CoAP and Persistence.

3.1 Controller

The JUNO app aims to control the walking session and to visualize the latest data collected. In particular, it allows:

- The start and stop of a walking session, in the laboratory (with LEDs and button usage) or over the asteroid's surface, after a successful landing.
- The retrieving and visualization of the last distances sensed by the LiDAR and slopes by the gyroscope.
- The retrieving and visualization of the last decision made for the new direction, commands for the legs' servo motors and for harpoons.

3.2 MQTT

MQTT package includes the MQTTCollector class used to, as the name says, collect the messages from the sensors.

- First, the class takes care to subscribe to the topics defined by sensors ("lidar" and "gyroscope");
- Secondly, it sends a command (like "start" or "test-session") to trigger sensors activity. It also implements the stop and pause of the data generation.
- It handles the message arrived, parsing and saving the data into the database.

3.2.1 LiDAR and legs servo motors

When a message from the LiDAR arrives, an algorithm has been implemented to distinguish between a clear vision and obstacles, and to define the next direction and command for the actuators, the legs servo motors:

Algorithm 1 Servo Motors New Direction Algorithm

```
1: front ← newLidarRecord.getFront()
2: right ← newLidarRecord.getRight()
3: left ← newLidarRecord.getLeft()
4: if front < 20 and right < 20 and left < 20 then
5:     direction ← BACKWARD
6:     step ← max(front, right, left) / 2
7: else if front = 100 then
8:     direction ← FORWARD
9:     step ← front / 2
10: else if right > left then
11:     direction ← RIGHT
12:     step ← right / 2
13: else
14:     direction ← LEFT
15:     step ← left / 2
16: end if
17: if direction = LEFT or direction = BACKWARD then
18:     step ← -step
19: end if
```

3.2.2 Gyroscope and harpoons

When a gyroscope message arrives, the following simple algorithm has been implemented to decide the new harpoons state and continue to walk:

Algorithm 2 Harpoons Activation Command Algorithm

```
1: x ← gyroData.getX()
2: y ← gyroData.getY()
3: z ← gyroData.getZ()
4: if |x| > 45 or |y| > 45 or |z| > 45 then
5:     command ← ACTIVATE
6: else
7:     command ← DEACTIVATE
8: end if
9: return command
```

3.3 CoAP

CoAP package includes the CoAPClient class used to interact with CoAP servers that expose the actuating resources. The class:

- retrieves the actuators URI from a static configuration file;
- handles the CoAP request to communicate commands and decisions.

3.4 Persistence

The Persistence package is used to handle the integration of the database, implementing entities class, connection and operations.

3.4.1 JUNO database: rover_control

A simple MySQL database has been designed and implemented. The architecture comprehends:

Table 3.1: `lidar_readings`

Field	Type	Description
id	INT (AUTO_INCREMENT)	Primary Key
timestamp	TIMESTAMP	Default: CURRENT_TIMESTAMP
distance_front	INT	NOT NULL
distance_left	INT	NOT NULL
distance_right	INT	NOT NULL

Table 3.2: `gyro_readings`

Field	Type	Description
id	INT (AUTO_INCREMENT)	Primary Key
timestamp	TIMESTAMP	Default: CURRENT_TIMESTAMP
angle_x	INT	NOT NULL
angle_y	INT	NOT NULL
angle_z	INT	NOT NULL

Table 3.3: `legs_commands`

Field	Type	Description
id	INT (AUTO_INCREMENT)	Primary Key
timestamp	TIMESTAMP	Default: CURRENT_TIMESTAMP

Field	Type	Description
new_direction	INT	NOT NULL
step_size	INT	NOT NULL

Table 3.4: `harpoons_commands`

Field	Type	Description
id	INT (AUTO_INCREMENT)	Primary Key
timestamp	TIMESTAMP	Default: CURRENT_TIMESTAMP
fire	INT	NOT NULL

Table 3.5: `rover_position`

Field	Type	Description
id	INT (AUTO_INCREMENT)	Primary Key
timestamp	TIMESTAMP	Default: CURRENT_TIMESTAMP
x	INT	NOT NULL
y	INT	NOT NULL

3.4.2 DataManager

The main class is DataManager, developed as a singleton class and instantiated at the beginning of the execution of the app. This class connects with the JUNO database and operates INSERT and SELECT operations.

3.4.3 Entities

Entities subpackage includes:

- **LidarReading, GyroscopeReading, MotorsCommand, HarpoonsCommand**: used to represent the records stored in the correspondents tables in the db;
- **Position**: a singleton class of which the instance is instantiated at the beginning of the execution. Used to quickly update and retrieve the position of the rover on the surface, and to monitor the movement while on laboratory test, in which it's initialized as (0,0).

3.5 Data Encoding

The data collected by the sensors and commands to be sent to the actuators are encoded in JSON format. This format allows interoperability across different applications and is supported on many platforms. The JSON format has been chosen because it is a lightweight method of encoding data. It limits the overhead in terms of processing and data exchange while being extremely popular and supported.

Chapter 4

Demo

4.1 Firmwares

The project has been tested by building a network of 3 IoT devices: NRF52840 dongles. Considering that the available devices are only 4, the final firmware was uploaded in:

- **Mote 1**: border-router
- **Mote 2**: motion-hub, that handles LiDAR and gyroscope
- **Mote 3**: legs servo motors
- **Mote 4**: legs harpoons

4.2 Laboratory test mode

Selecting `labTest` on the JUNO App main menu allows the user to start a rover walking session in "laboratory test mode". This modality is designed to support technicians and engineers during a test phase in which the rover walks using the IoT networks deployed. To allow the user to clearly see the sensors status in terms of obstacles or dangerous inclination detected. The motion-hub device shows:

- **Blue LED**: when the LiDAR sense the maximum distance possible (100 m), so there's no obstacles, or the gyroscope sense a low inclination;
- **Green LED**: when the LiDAR sense a distance shorter than the maximum, similarly the gyroscope sense a "medium" inclination;
- **Red LED**: when the LiDAR see an obstacle (distance less than 20 m), or gyroscope detect a dangerous inclination. In these cases the actuators will be activated.

The user can also use the device button to test the rover in case one or both sensors are out of order, situation that can likely happen during a space mission. In a real scenario, this function can help to evaluate the rover efficiency and behavior in a numerous IoT network, with other sensors and actuators. In the JUNO project, the user can:

- **Quickly press the button**: this action will disable LiDAR sensing. The same, repeated, will restore it.
- **Press the button for more than 2 seconds**: this action will disable the gyroscope sensing. The same, repeated, will restore it.

Selecting, instead, `explore` from the main menu, makes the rover starts to walk, so the sensors begin to collect data and the actuators to be activated consequently. Only in this case, the data are stored in the database and LEDs and buttons are deactivated, with the aim to simulate a real situation where the rover moves in a human-less environment.

4.3 Grafana

A Grafana dashboard is provided with the JSON export code. Through this tool, it's possible to visualize graphics about:

- Distance to obstacles in the 3 directions available sensed by LiDAR;
- Decision about the next direction that will be performed by legs' servo motors;
- Inclination detected in the 3 axes by the gyroscope;
- State in time of harpoons (activated or not activated);
- Position of the rover.

In the case of `explore` mode selected, these data can be shown while updating in real-time.

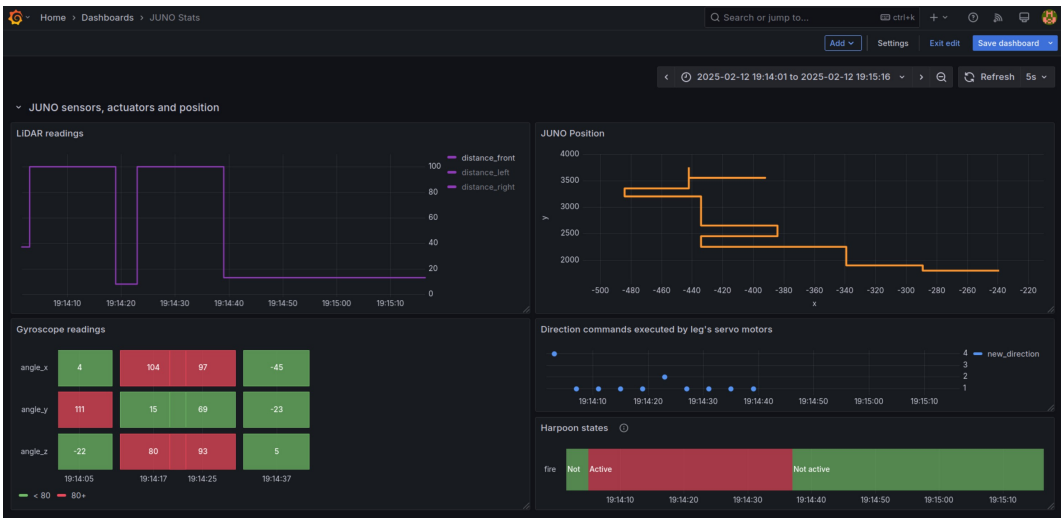

Figure 4.1: Screenshot of the Grafana dashboard during a walking session running with explore mode.

Chapter 5

Conclusions

JUNO is a project designed for a hypothetical scenario in the field of space exploration, specifically made for a very hostile environment like the ones belonging to an asteroid. This is a topic in constant evolution but also involves extremely complex systems. Sensors, actuators and science tools are chosen considering a big number of factors, such as the mission goal, the dimension of electronic components, weight and - no less important - cost. Every element must be optimized and every physical part must work in harmony with the others. Accepting a relevant risk of failure, whether of individual components or the entire mission, the JUNO rover as a whole would be much more complex and complete equipping all the necessary devices. This project aims to represent only a part of these components, where one of the most thrilling aspects of space exploration takes place: the discovery of a new world.