Dissertation Title

Luis Yallico Ylquimiche

July 1, 2024

Abstract

This is the abstract of your dissertation.

Contents

Chapter 1

Introduction

1.1 Background

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Chapter 2

Literature Review

2.1 Previous Studies

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Chapter 3

Implementation

3.1 OTA

Integrating Over-the-Air (OTA) updates into robotic swarms significantly enhances the efficiency of deploying and managing software updates, especially since these swarms often operate in hard-to-reach environments. OTA updates provide a scalable solution for software management, allowing us to view our swarm as a remotely manageable fleet of devices. This ensures that the entire fleet consistently operates on the latest software version. Such a strategy is exceptionally valuable in hazardous scenarios, including disaster response, or in challenging locations like space exploration, where direct access to hardware is limited.

Enabling swarms for commercial use require flexibility in the services that they can provide, mirroring successful implementations seen in the automotive industry, OTA updates can be used to achieve this goal in swarm robotics. For example, Tesla has revolutionized the car market with the model known as "software-enabled feature activation" by remotely updating vehicle software to enhance or unlock features. This operating model is not only convenient but also ensures fast deployment of critical updates across a fleet and it allows manufacturers to streamline hardware production. A truly flexible swarm should be robust enough to perform different tasks, and OTA updates might be one way to achieve this.

In swarm literature we come across the single source of failure paradigm with decentralization at its core. With regards to OTA, some risks include, reliance on a central server to host the OTA update, cybersecurity breaches and the failure of the update itself are some valid points of consideration. Our implementation addresses some of these points by firstly using cloud services (AWS S3) to host the OTA updates with 99.99% availability, and by designing the ESP32 to have a data partition which can safely rollback to the previous working version of the application in case the OTA update fails (i.e. battery in robot runs out as the OTA was in-progress). Though, the proposed implementation does not attempt to address this as its main goal, in the future it would be interesting to explore the possibility for the swarm to create and design their own OTA updates that can propagate locally. This might perhaps be possible with the use of decentralised genetic algorithms or machine learning approaches.

Using cloud services over local servers presents distinct advantages, including scalability and reduced capital expenditure costs. For example Apple's iOS updates, showcases the effectiveness of using cloud infrastructure to distribute software updates to millions of devices globally. Such scalability is key for managing large swarms of robots, ensuring consistent updates without the overhead of maintaining extensive on-premise infrastructure. This is particularly the case for swarm systems that might require multiple servers across the world due to data privacy regulations, where some data is ring-fenced to the region it was generated.

Continuous Integration and Continuous Deployment (CI/CD) practices are pivotal in supporting the dynamic nature of robotic software development. These methodologies ensure that new features and fixes are regularly integrated and tested, significantly reducing the time from development to deployment. Notably, space exploration initiatives, such as updates to Mars or lunar mission drones, rely on CI/CD to ensure that every software iteration is robust and capable of operating in extreme conditions.

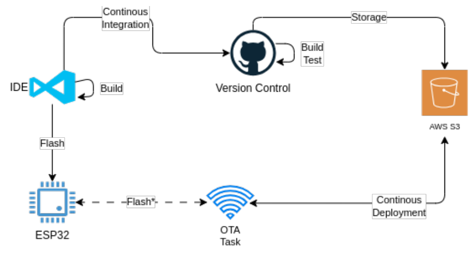

Figure 3.1: System Architecture

By adopting OTA, CI/CD, and cloud-based updates, robotic swarms gain the agility required to adapt to new challenges rapidly, reflect real-time improvements, and achieve true scalability. Figure 3.1 shows the implementation of our system to enable over the air updates (OTA) and continuous integration & continuous deployment (CI/CD) framework over the swam.

- **Local Development Environment**: ESP32 application development takes place locally using VSCode, the IDE environment uses version 5.2.2 of ESP-IDF and Python 3.11 to build and flash the code in-situ to the ESP32 module. This self contained development environment allows for testing new features and updates without affecting previous versions of the application running on the swarm.

- **Version Control System & CI/CD**: The codebase is stored in a public repository on GitHub: `https://github.com/yallico/robotics_dissertation`, this allows for version control and automates the build and test process upon every commit. The ESP32 project is compiled and it generates the .bin binary file used for OTA. This process ensures that the codebase is always in a working state and ready for deployment.

- **Cloud Storage**: Once the OTA binary file is generated, it is uploaded to an AWS S3 public bucket. S3 serves as a reliable and low cost storage solution for the OTA updates. We decided to leave encryption and access control out of scope from the OTA implementation, yet we acknowledge that encryption is a non-trivial task that swarms should consider when deploying OTA updates in terms of computational resources required and security implications in industry.

- **OTA Update Process**: The ESP32, runs a task that is triggered upon initialisation which compares its current application version against the latest version available in S3. If the version in AWS S3 diverges from the current version running in the ESP32. It then downloads the .bin file and performs the update following OTA best practice (see Section X).

3.2 Experimental Setup

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

3.3 Notes

Why use ESP-IDF over Arduino IDE in this project? [1][2]

As we are using a ESP32 microcontroller to build our swarm, which left us with two options to program it: Arduino IDE or ESP-IDF. The reasons we chose the latter are because, first, it is the official development framework for the

ESP32 microcontrollers, this means that ESP-IDF is native to ESP32 whereas Arduino is an API wrap around ESP-IDF. Making ESP-IDF more stable and enabling more advanced features, specially for communication data links such as Bluetooth, Wifi and LORA. Secondly, it is more powerful and flexible than Arduino IDE, because it allows the use of FreeRTOS which allows multi core development support (our M5 Stack has two cores) and is a pre-requisite for running microROS in the ESP32 microcontroller (at the time of writing this microROS does not support Arduino), hence making it more suitable for complex projects like this one. Thirdly, it is more efficient in terms of memory and speed (as it enables parallel processing) which is important for a project that requires real-time communication between multiple devices in a swarm. Finally, it is more professional and an industry standard, it allows dependency tracking, Over the Air (OTA) updates, unit testing, enhanced debugging and comprehensive documentation around it, which means it is more likely to be supported in the future and software is less likely to become deprecated over time.

Bibliography

[1] ESP-BOARDS. *ESP-IDF (IoT Development Framework) vs Arduino Core in 2023*. URL: `https://www.espboards.dev/blog/esp-idf-vs-arduino-core/` (visited on 06/03/2024).

[2] Expressif Expressif. *FreeRTOS Overview - ESP32 - — ESP-IDF Programming Guide latest documentation*. URL: `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html` (visited on 03/04/2024).