New requirements

- ☐ detailed design specifications of the hardware and software to meet the functional requirements and the practical design constraints
- ☐ use circuit diagrams, logic diagrams, block diagrams, flow charts, class diagrams and/or sequence diagrams
- ☐ detailed justification of design choices
- ☐ highlight the novel design aspects
- ☐ evaluate the effect of design choices on your system quality attributes
- ☐ not required to include all of these sub-sections, instead you need to decide how to best communicate your solution's design

0.1 Overview

The overview of the proposed solution is depicted in Fig. 1. The hardware components consist of an ANAFI drone, a Raspberry Pi with an external battery, and a Wi-Fi dongle. The built-in camera of the drone is used to take pictures of the targets and the Wi-Fi dongle is used to communicate with the command and control system, which sends high-level commands whenever the connection is present. An object detection model trained using RoboFlow detects and identifies the targets in the pictures. In addition, a deep reinforcement learning (DRL) agent is trained in a cyber-physical simulator called Sphinx to visit virtual targets having the same mobility pattern as the real ones. This finished model is loaded into the Raspberry Pi that acts as an onboard computer to the ANAFI drone and instructs it where to move to cover the targets in a minimum amount of time.

The following subsections will explain in more detail why this solution is chosen and what the integral components are.

0.2 High level architecture

Fig. 2 shows a high-level architecture of the complete working system, in which a group of connected adapters and devices are combined into a single functional system. The architecture is composed of three sections namely interfacing, controlling, and targets. The interfacing section contains the drone that will handle the onboard computer, its power source, and the connection adapters. In the controlling part, a personal computer will be responsible for contacting the onboard computer to adjust settings, execute scripts, and get live updates and results. Finally, there will be multiple moving targets in the target section. For example, remotely controlled (R/C) cars are controlled manually and moving in a specific mobility pattern with varying directions and destinations. In the next section, hardware and software components will be presented in a more detailed manner.

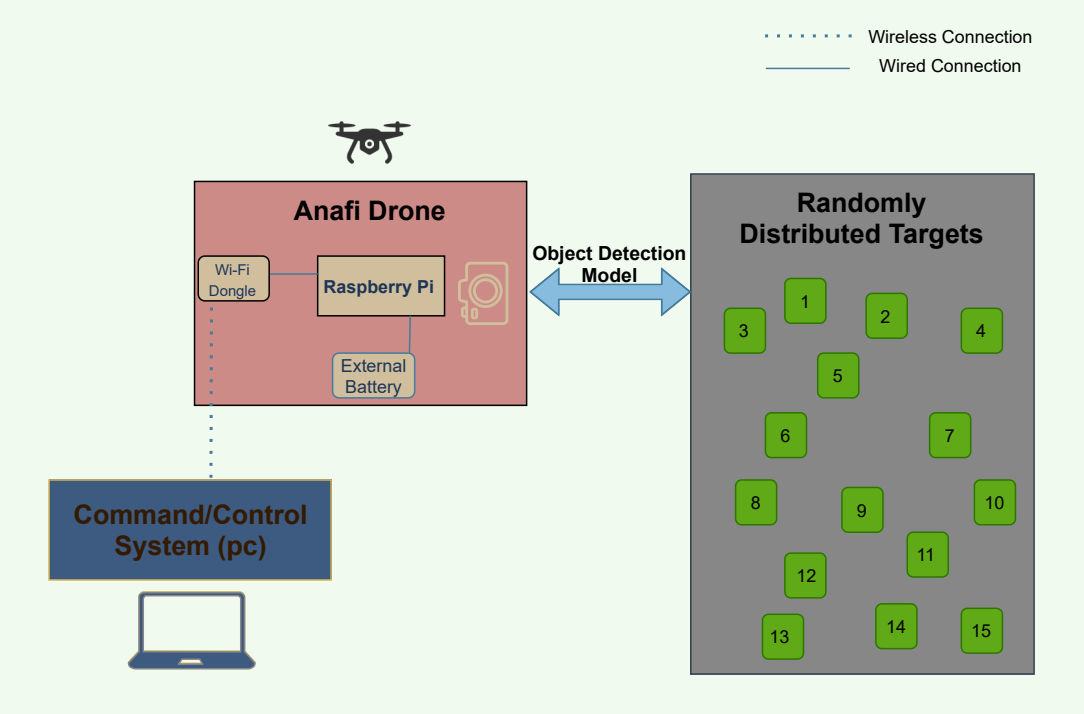


Fig. 1. A general overview of the solution.

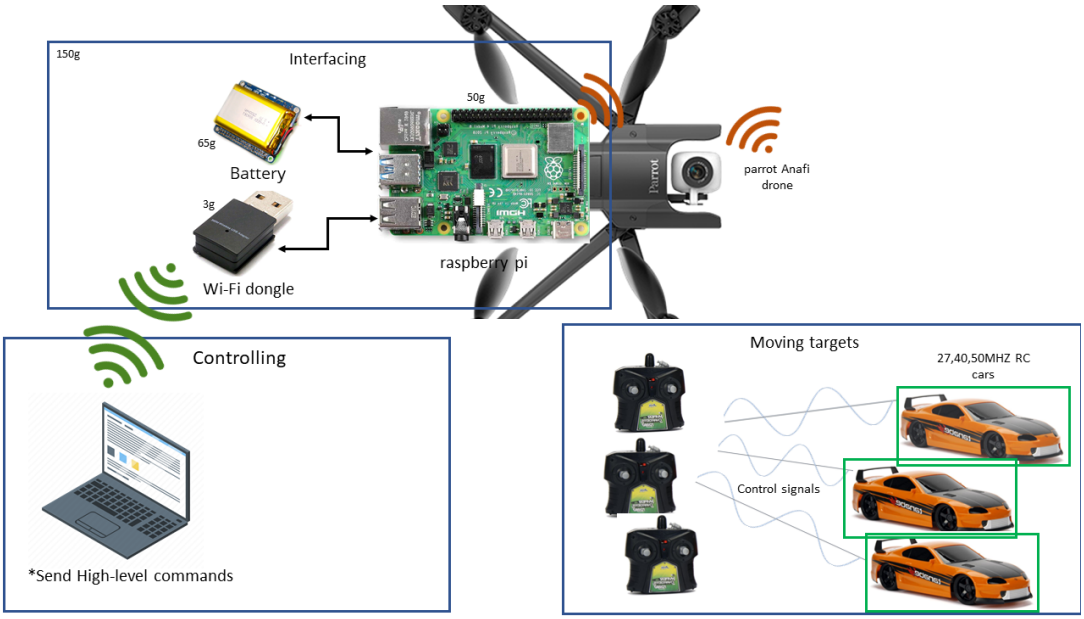


Fig. 2. The high-level architecture of the overall system.

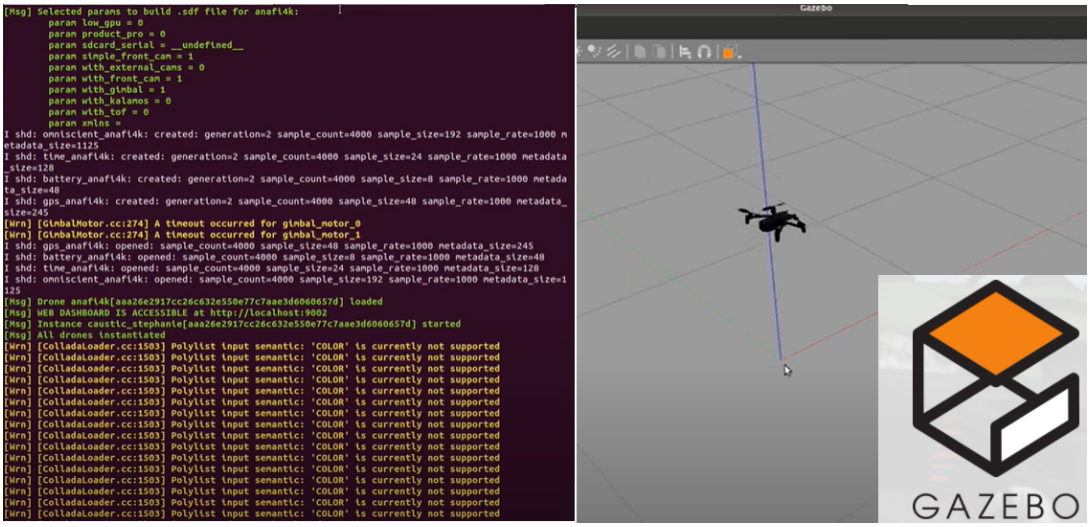


Fig. 3. The Sphinx program that runs on top of Gazebo.

0.3 Hardware/software used

New requirements

- ☐ Describe each hardware component and how it works and the **scientific principles of its inner working**.

0.3.1 Software

There are three primary categories of software depending on the usage: simulation, training, and application, and they are listed in Table 1. The first part will focus on simulating the environment, testing the models, and flight control. Before discussing the software to be used, we have selected Ubuntu 18.04 (Bionic Beaver) as an operating system for several reasons. One key reason is that in addition to still being supported, it is compatible with the Parrot's Olympe and Sphinx programs, which are only supported on limited distributions and operating systems. Another reason is that it is a lite OS and can be installed on the onboard computer that will be attached to the drone. For the simulation part, using Sphinx and Gazebo software is very helpful to visualize the environment, control the drone, and apply the DRL model.

Sphinx is a simulation tool built on top of Gazebo to run the Parrot's drone firmware on personal computers, which comes with helpful features for simulation such as visualizing flight data at runtime, running the unmanned aerial vehicle (UAV) remotely, and executing scripts with the command line. Gazebo is a robot GUI simulation which simulates the visual and physical surrounding of drones and custom 3D objects. Fig. 3 shows how the Sphinx program looks like.

The training part is divided into reinforcement learning (RL), which is used to teach the ANAFI drone to complete the task, and object detection and identification, which is used to detect and count the targets required in determining the reward during the training. The RL agent is based on the DRL model developed by **Ged21**. The action space is composed of 9 movements in the forward, backward, left, right, forward-left, forward-right, backward-left and backward-right

Table 1. Software used in the project.

Software	*Logo*	*Justification*
Parrot Olympe	Parrot For Developers	A controller for the Parrot ANAFI drone. It makes controlling the drone possible using a Python script
Parrot Sphinx	Parrot For Developers	A simulator for the Parrot ANAFI drone. It loads the Parrot's drone firmware in the simulation environment. It is largely based on Gazebo.
Roboflow		A framework for Computer Vision development that can be used online. It facilitates labelling of images, splitting and merging datasets, applying image transformations and filters, and generating a link for the augmented datasets which will be used in the training notebooks.
Jupyter Notebook		A web application for writing, testing and sharing of code. It is free and does not require internet access like the Google Colab. It is used heavily in this project to experiment with new ideas in the Sphinx simulation.
Google Colab		A Jupyter notebook environment running in the cloud. It makes it easy to write, run, and share the code. Most importantly, it gives an option to use remote processing resources in addition to the local ones.
Python Flask		A backend framework written in python, it does not require any libraries or tools which qualifies it as a microframework.
Gunicorn		A Python Web Server Gateway Interface HTTP server. It is very commonly used for web app deployment as it implements PEP 3333 Python WSGI interface.

Fig. 4. An example of a user interface that will be built using sensor data coming from the ANAFI drone.

directions as well as a hover. The state space is given as

$$s = \{t, cell, [I_1, I_2, I_3, \ldots, I_m]\} \tag{1}$$

where t is the current time step, $cell$ is the ID of the cell above which the drone is, I_k is the binary variable that indicates if the target with an ID of k has been visited, m is the total number of targets, and the vector of length m is the container for the I indicators. For example, if only targets with ID's 3 and 7 have been visited in the current and previous time steps since the beginning of the episode, then the vector will have elements of one for I_3 and I_7 while the other elements are zeros. It follows that all the targets must be uniquely identifiable. In addition, the reward is how many new targets are captured in the current cell, and if the drone flies past the boundary, then a big negative reward will be incurred.

For the object detection, we used simulation tools to generate some training datasets. Firstly, we placed random objects and captured the images using the simulated drone camera. We have used a website called Roboflow which helped us labelling the objects and generate new datasets from the existing ones with different types of augmentation such as rotation and scaling. For the object detection model, Google Colab notebook was a sufficient tool to start training using convolutional neural network (CNN) YOLOv5 in addition to the Jupyter notebook which was very helpful in code experimentation.

For the application software, Parrot Olympe was used to send commands to the physical as well as the simulated drone and control the flight trip and how the drone moves. Parrot Olympe uses Python controller programming interface for Parrot drones which makes controlling simple and easy using a Python script. Moreover, Olympe allows us to read sensor data, such as the GPS fixes and camera feed, from the ANAFI drone. This will make it possible to build a user interface on the control and command station shown in Fig. 4 to monitor the progress of the drone when it is autonomously executing the task visitation mission.

0.3.2 Hardware

The main core of the hardware part is the drone, which will be the Parrot ANAFI. Table 2 lists the hardware components used in this project and their justifications. The second important device is the Raspberry Pi. It acts as an onboard computer and is used in this project mainly to

facilitate the low-level control of the drone by sending frequent control commands to the drone in a certain direction or keep hovering. The choice of action is determined by the DRL model that will be installed in it. Hence, the tasks of the Raspberry Pi are:

1. connecting to the drone's access point using a Wi-Fi interface,
2. controlling the drone by executing Olympe to send low-level control signals and to receive sensory data,
3. applying the DRL model supported by the command and control system, and
4. receiving high-level commands and sending data to the command and control system.

The Parrot ANAFI drone is connected to the Raspberry Pi, which is the onboard computer, using both devices' internal 2.4 GHz Wi-Fi interfaces. Firstly, we add the ANAFI drone's access point to the saved devices list in the Raspberry Pi. Once the Raspberry Pi boots up, it automatically keeps searching for the access point and connects to it once it is available.

For the connection between the command and control system and the Raspberry Pi, the Raspberry Pi will use a 300 MBps Wi-Fi adapter dongle connected to its USB port. This will allow it to create an access point to which the command and control device will connect and by which the Raspberry Pi can be controlled. This control of the onboard computer is done through the Secure Shell (SSH) protocol or the Virtual Network Computing (VNC) if the graphical interface is needed.

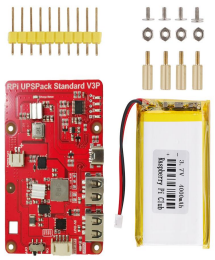
Regarding the power source for the Raspberry Pi, we thought of taking power directly from the drone's battery, but, after some research, we found that the ANAFI drone's socket is somehow different. It is also challenging and the drone battery is susceptible to shutting down immediately if the voltage reaches less than 3.0 Volt. So, we did not want to take the risk and used a lithium battery with a power board called UPSPack Standard Power Supply attached to the main Raspberry Pi board. It includes a 4000 mA h lithium battery, which provides enough power and time for our application.

0.4 Hardware design

New requirements

- ☐ For major hardware subsystems in your design, you must present the theory behind the different technological approaches, the tradeoffs associated with each approach, and your justification for selecting a particular approach. For instance, selection of a particular microcontroller for your design; what were the main factors that contributed to this specific choice. For example, you may choose between Arduino, MC68HCxxx, BasicStamp, and PIC18Fxx and you have decided to use PIC18F47 due to:
  - ☐ its programming compatibility with C
  - ☐ its low cost
  - ☐ its development platform availability

Table 2. Hardware used in the project.

Hardware	*Picture*	*Justification*
Parrot ANAFI Drone		Available in the university, can be controlled easily with simple Python script, 4K-high resolution camera.
Raspberry Pi 4		Specifications are enough for our application, support Wi-Fi and its small size and weight is an advantage.
RPI UPSPack V3 with 4000 mA h lithium Battery		Support up to 4 hours which is more than enough , the board got an LED indicator for charging level also the weight and shape is an advantage.
Wireless N Nano USB Adapter		Cheap and do its job, good coverage range.
Laptop		Any laptop with good WiFi interface card will be enough for our case.

- ☐ ability to input analog signals

☐ To document the design of the hardware components you can use as many of the following as possible:

- ☐ Circuit Diagram (Showing actual circuit diagram with all the components used)
- ☐ Connectivity diagram for modular hardware.
- ☐ Logic Diagrams (Showing logic flow with respect to signals/numbers)
- ☐ Functional Diagrams (showing subsets of the circuit as functional blocks)
- ☐ State Diagrams (to explain the sequential logic flow)
- ☐ Any other drawings to communicate your design

☐ Organize the content of this section using appropriate subsections. It is recommended to have a sub-section of each of the recommended artifacts listed above.

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.

0.5 Software design

New requirements

☐ You should document the structural and the behavioral aspects of your software components. For projects that only use embedded code (e.g., programming microprocessor), a flow-chart describing your software design could be sufficient. Projects with user interfaces may elaborate on the functioning of the interface using some of the models mentioned below.

☐ The software components could be documented using class diagram for the whole system. While Class may be too specific to JAVA or C++ environments, you may use equivalent components applicable to your design. For instance, VI for LabVIEW designs, MDL files in MATLAB, etc... If the model is too big, partition the diagram using some reasonable criteria. For example, you may provide the entity classes and the controller classes as separate diagrams.

☐ Specific emphasis should be given to the elaboration of the software components that are responsible for interfacing with the hardware components of your design. For example, if a protocol-like procedure was developed in connecting to a hardware

module then it should be explained in detail. Such components may include packetization/depaketization procedures, etc…

- ☐ Wherever applicable, all associations between classes/software-modules should be identified through defining the association name and the multiplicities on both ends. Aggregation and inheritance relationships should be identified. A brief explanation should accompany each diagram.
- ☐ Overall software logic may also be described in order to know the appropriate logic flow. In case of LabVIEW or SIMULINK programs, this may not be needed. For other programming environments, the state diagram or extended flow chart may be sufficient.
- ☐ In case of graphical codes (such as LabVIEW programs) the software structure may be described by various sub-Vis of the system. Similarly for SIMULINK diagrams, the models should be explained individually. The complete program (graphical/text) should be included in appendix and should not be listed completely in this section.

0.5.1 Simulation

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.

0.5.2 User Interface

We created a simple graphical user interface that allows an authorized user to connect to the drone, run different scripts on it, and even watch the live stream from the drone's camera. As illustrated in Fig. 5, the solution is mainly a simple website that was built using html, css, and javascript for the frontend. As for the backend, we relied on Python Flask as the web framework due to its simplicity and practicality. Moreover, using Flask was beneficial for us because it is a lightweight framework with a built-in development server and a fast debugger provided. We relied on a web server gateway interface (WSGI) HTTP server called Gunicorn which can forward requests to other web applications and frameworks that are written in Python. Furthermore, Nginx is used as a reverse proxy server to ensure the smooth flow of traffic in the network.

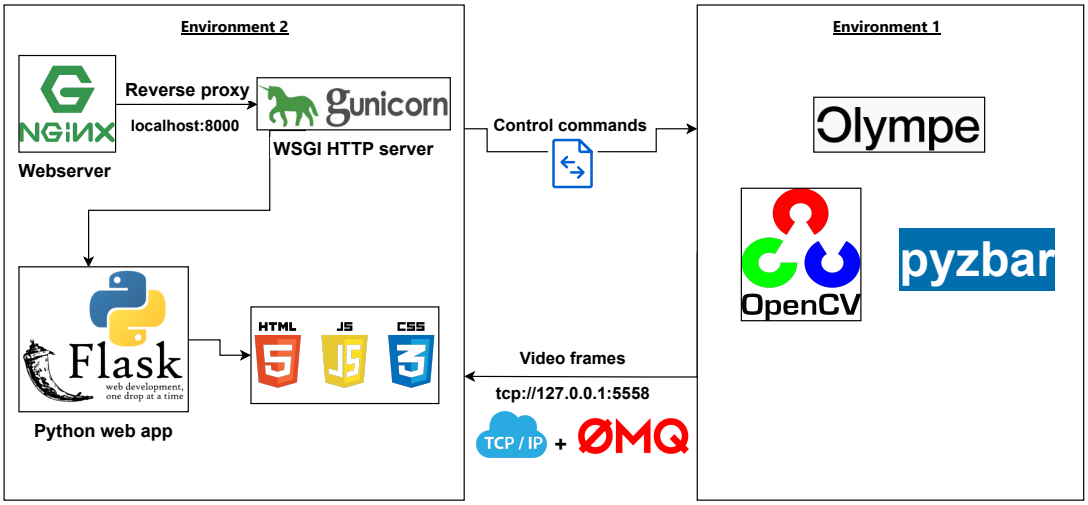


Fig. 5. shows the interactions between software components that were included to build the web-based GUI.