Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

User's Manual

Pizzair

Submitted to
Manish Gupta and Nirav Dagli
Spinnaker Analytics
200 Clarendon St, Boston, MA 02116
(617) 303-1938
manish.gupta@spinnakeranalytics.com and nirav.dagli@spinnakeranalytics.com

by
Team 4

Team Members

Compton Bowman comptonb@bu.edu
Ahmet Caliskan aeclskn@bu.edu
Yafei Chen yafeic@bu.edu
Usman Jalil ujalil@bu.edu
Quentin Clark qtcc@bu.edu

Submitted: 4/19/2024

User Manual

Table of Contents

● Executive Summary

Current challenges in pizza delivery, including delays, cold pizzas, and operational inefficiencies, prompted the development of an innovative solution. Traditional human-driven delivery services face increasing difficulties, exacerbated by factors such as the COVID-19 pandemic. Pizzair aims to transform the pizza delivery industry by introducing an AI-enabled, fully autonomous pizza delivery drone. The final prototype will meet specific criteria including maintaining food temperatures, having a secure harness mechanism, and costing under $3000. The navigation and codebase will support self-navigation, notifications, and prioritization of delivery routes.

Pizzair proposes a technical approach featuring an AI-enabled drone for pizza delivery. The drone ensures timely and safe delivery within a 10-minute radius. Equipped with a specialized harness, Pizzair maintains food temperatures and optimizes routes through autonomous navigation. This system aims to enhance customer experience and operational efficiency for pizza restaurants.

Pizzair employs machine learning algorithms for self-navigation, allowing the drone to adapt to dynamic environments during delivery. The drone features a specialized harness to securely pick up and deliver pizzas while preserving their temperature, ensuring customers receive hot and fresh pizzas. Also, Pizzair's system includes real-time notifications to customers and pizzerias, keeping everyone informed about the delivery process.

1 Introduction

In response to the persistent challenges affecting the pizza delivery industry, we propose an innovative solution aimed at redefining food delivery norms. Many customer quality of service problems lie in the inherent inefficiencies of traditional delivery methods, ranging from delays and compromised food quality to escalating demands on delivery personnel, especially in the context of the recent COVID-19 pandemic.

One potential solution to ameliorate many of these problems is leveraging increasingly powerful autonomous systems to help offload delivery work onto drones. Drones are fast, increasingly reliable, and after the onset cost of purchasing a drone, cheaper both economically and in terms of spent resources (human hours, energy, etc.) than a traditional delivery person in an automobile.

However, current autonomous delivery systems are large, difficult to use, and expensive, being primarily designed for longer-range delivery of larger payloads, like health resources to remote regions. Our project is investigating the viability of a lower-cost alternative, primarily designed for urban and suburban low-range delivery, that would be a viable option for restaurants to individually purchase and use as an alternative delivery mechanism in addition to or in lieu of human delivery.

The purpose of Pizzair is to introduce an AI-enabled, fully autonomous pizza delivery drone. Pizzair aims to alleviate the constraints faced by traditional delivery services, offering cost-effective, efficient, and technologically advanced alternatives that redefine the customer experience. Our team's approach features the development and implementation of an autonomous delivery platform. Pizzair seeks to optimize delivery operations for pizza restaurants, ensuring prompt and high-quality service to customers within a 10-minute radius.

Pizzair addresses the problems with the traditional delivery service listed above. The autonomous drone system guarantees swift and reliable deliveries, overcoming issues of delayed arrival and cold pizzas. Through real-time monitoring and secure harness mechanisms, Pizzair ensures the safety, quality, and efficiency of pizza deliveries.

Special features of the project include the following:

- Pizzair will operate autonomously.
- Pizzair will have a specialized harness mechanism to preserve the temperature of the pizza during flight.
- Pizzair will offer a cost-effective alternative to traditional delivery services.
- Pizzair will inform customers and the pizzeria with real-time notifications throughout the delivery process.

2 System Overview and Installation

2.1 Overview block diagram

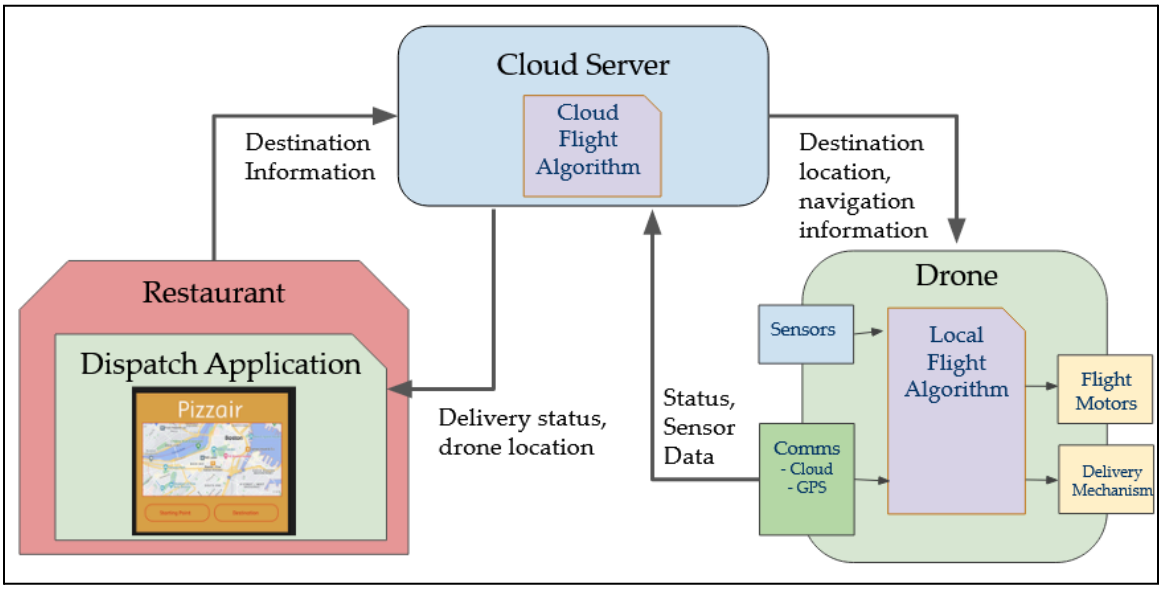

Figure 1: Autonomous Drone Architecture

Our software system is split into three components: software that runs on the drone hardware, software that runs on a cloud-hosted web server, and software that runs on a client application for communicating with the drone.

2.2 User interface

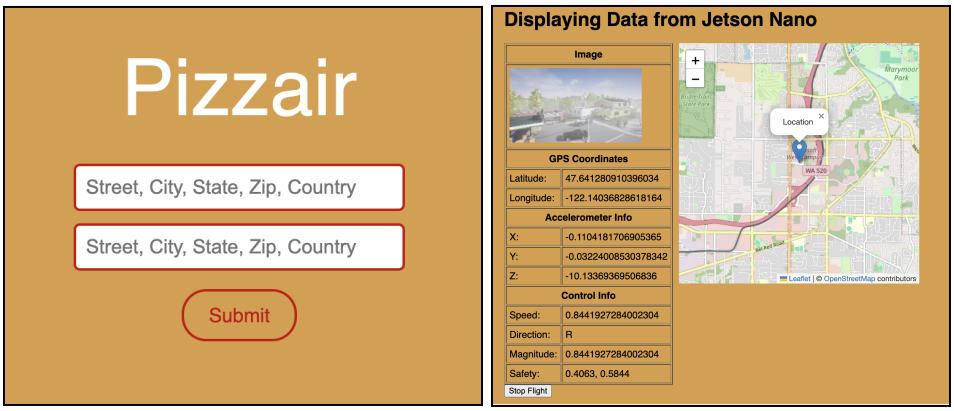

Figure 2: UI

We have developed a simple web application enabling users to input starting and destination addresses in ***Street, City, State, Zip, and Country*** format. Upon submission, these addresses are converted into longitude and latitude coordinates and stored in a DynamoDB table. The Jetson Nano onboard the drone continuously retrieves data from

this table. During the flight, the Jetson sends updates back to the web application to keep the user informed. Below, you'll find a visual representation of this process.

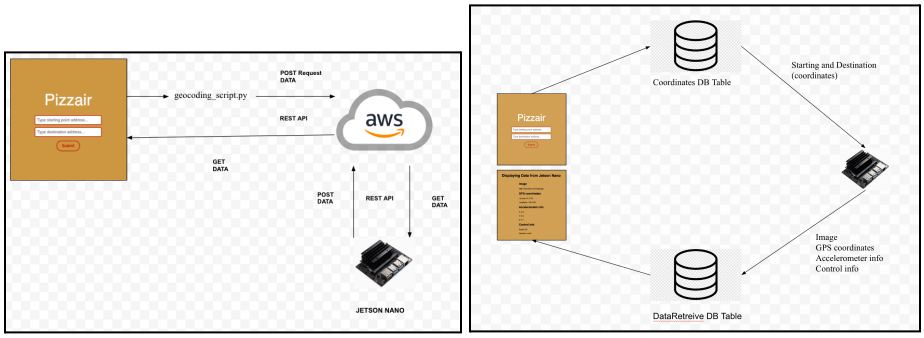

Figure 3: Software Architecture Diagram

2.3 Physical description

Figure 3 shows the main physical components of Pizzair without installed propellers. Figure 4 left shows a 3D print of our harness for the prototype drone. The bottom disk slips into the top disk, which snaps into the drone, so that payloads can be attached to the drone with ease. Figure 4 right is a CAD section view of our harness. During flight, the interlocking teeth are pushed together by the weight of the payload, allowing for secure travel.

Figure 4: Our fully constructed drone.

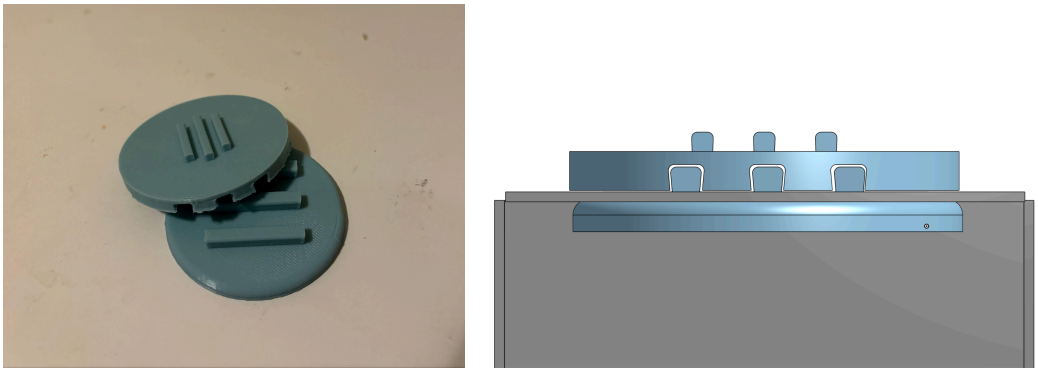

Figure 5: harness

2.4 Installation, setup, and support

Our system is primarily based in Python. Instructions for installing Python can be found here. We recommend using an environment manager like Anaconda to manage your environment. The required packages are described in our "requirements.txt" file in our repository. Users can either install these manually or through the environment manager of their choice. See here for examples of Miniconda.

We use an NVIDIA Jetson Nano for the compute platform on our prototype. We installed the default recommended operating system from NVIDIA, which is a distribution of Ubuntu 18.04. Information on installing the operating system on the Nano can be found here. The Nano is based on an Arm64 CPU and so cannot run full x86 applications. We used Archiconda for our package manager and recommend users do the same if installing on the same computing platform. More powerful platforms may have other opportunities available to use.

To install our software, simply clone our GitHub repo located here. This includes all of the software for the drone, web server, and web client. It also includes software we used for creating and testing our ML-enabled control algorithm (for instance, Python scripts we wrote to assist in annotating video game video, and our AirSim scripts for testing in simulation). These are not necessary to use the Pizzair platform, but we included them for completeness and will describe how to use them.

To launch the web application, first navigate to the flask_webapp folder. Then, execute the command: *python3 app.py*. This will initialize a local Flask server. To access the website, simply copy and paste the following URL into your web browser: *http://127.0.0.1:5000* from the terminal.

To launch the drone properly, navigate to the drone_control/jetson_scripts directory and execute *python jetson_inference.py*. This will start the policy on the Jetson. Note that the Jetson must have a camera and an Internet connection, and it may take around half a minute for the control loop to start properly. To exit this process, click on the open video feed and press the q key. To enter "hover" mode while the control process is running, press the h key.

To annotate video clips for imitation learning, go to the drone_control folder and run this command: *python annotate_clip.py -f [video name]* to annotate for direction, and *python annotate_clip.py -f [video name] -s* to annotate for direction. Note that this requires the video files to be in a specific directory in relation to the current Git repository - see the code for examples and the command line command to select a different repository than our case. Then, for direction annotation, place your hands on the number keys 1-4 and 6-9 (so the 5 key is in the middle), and your thumbs on the spacebars. Use 1-4 for "left" directions of varying degrees of magnitude, space bar for middle/center, and 6-9 for "right" directions. Lastly, for safety annotation, use the s key for "safe" and the d key for "dangerous".

We tested our algorithm in AirSim for simulator closed-loop evaluation. Details on how to install it can be found here. Once installed, you should be able to run the scripts in the airsim_scripts file directly from your cloned directory (the setup_path.py file takes care of everything!). Note that you are required to import your own environment. We used a free cartoon town from the Epic Games store, but you can use an environment of your choice.

The scripts are outlined as follows:

- get_drone_coordinates.py: Helps for getting coordinates in the virtual environment. To do this, navigate the drone to your desired location using a XInput compatible controller (for instance, an XBox One controller) and run this script. It will print the coordinates to the command line, and then restart.
- pizzanet_test.py: The main operational script. When it first launches, it will connect to the DynaDB server to determine if it should launch. You can override this by pressing 's'. When active, it will go from its starting location to the delivery destination given by the DynaDB server, then return home, in a loop, landing at the delivery destination and starting location. To have the drone hover press 'h' and press 'q' to quit and shut down the simulation.
- Other scripts are testing or configuration scripts that do not need to be used.

Note: to function with DynaDB properly, you need to have a file with the appropriate keys in the flask_webapp folder. To not use the DynaDB functionality, turn the variable "use_dyna" in the pizzanet_test.py file to False.

3 Operation of the Project

3.1 Operating Mode 1: Normal Operation

The main operation of Pizzair will work as follows. Note that not every component of this operation is currently fully complete:

1] The user ensures the Webserver service is active, and connects to it from the web application on their client of choice (desktop, phone, etc).

2] The user activates the drone, including giving it a charged battery and turning the drone on. The user places it outside in a safe location.

3] The user specifies a destination location and tells the drone to begin delivery through the application.

4] To be compliant with current FAA regulations, the user should follow the drone continuously during operation. Should the drone begin to enter an unsafe situation, the user should use the option in the web application to tell the drone to hover or land as appropriate.

5] When the drone reaches its destination, it will complete delivery and then return to the starting address. From there the user should replace the drone's battery and then has the option to begin the delivery process anew.

3.2 Operating Mode 2: Abnormal Operation

Abnormal operations include the Pixhawk flight controller not lighting up. Most likely it is running out of battery or the power module is not connected firmly. Simply detaching the battery from the drone and charging it until its LED turns green or checking all the plugs are tight will solve the problem.

…

3.3 Safety Issues

Our product uses a physical drone, which comes with the risks inherent to automated robotics systems, especially drones. While we attempted to create a platform that is as robust as possible, there is still a great inherent danger to using our product. We recommend not using our drone in public areas, around pedestrians or animals, around objects sensitive to physical damage (cars, buildings with fragile walls and windows, etc.), or where the drone could move into a public area.

Our drone has rotors that can cause slashing damage to users. Do not attempt to grab the rotors while in operation. When using a remote controller, strictly follow instructions shown on the receiver and consider limiting access to children.

When the signal is lost between the drone and the base station, RTL mode (Return to Launch mode) will navigate the drone back to the home position.

Additionally, there are safety considerations when the drone is not operating that should be considered. Lithium-ion batteries have the potential to catch fire or explode and cause damage either directly or through malfunctioning fire-control systems. Caution and care should be taken when selecting the battery and charger type to ensure they are appropriate for each other and the drone.

4 Technical Background

○ Control Overview

Pizzair is powered by a combination of classical control algorithms for drone flight with an imitation-learning-based algorithm for obstacle detection and avoidance. It is closest to the Learning to Fly by Driving paper (from now on, the "DroNet Paper").

○ Network Description

Pizzair's obstacle avoidance is based on a deep convolutional neural network (CNN) trained using imitation learning. It takes in a monocular camera image from the drone as input and outputs a safety label, steering direction, and steering magnitude.. Our network architecture uses a relatively small ResNet-18 backbone with multilayer perceptron (MLP) layers following. We did not use any kind of pre-training learning regime.

Our policy incorporates this information from the CNN with a control policy that gives high-level yaw (steering) and forward velocity commands to a low-level drone controller, which translates these into rotor motor control signals. In short, it attempts to steer towards the goal when unsafe, and follows the directions from the CNN when unsafe. We found that this, at a consistent sampling ratio, without modulating speed as a function of safety, works reasonably well.

For some motivation for our approach, imitation learning combines the upsides of machine-learning-based approaches compared to a modular pipeline (lack of required engineer fine-tuning, generalizability, and the ability to connect perception to control via end-to-end learning) but without the downsides of a reinforcement learning approach (high sample complexity, training instability).

○ Training Description

Pizzair is primarily trained with data from high visual-fidelity open-world videogames. Video games offer some advantages and disadvantages compared to collecting data from the real world or in dedicated robot learning simulators (for instance, CARLA or AirSim). The advantages are near-photorealism and stellar environment diversity. For example, Watch Dogs 2 looks nearly photorealistic and contains a scaled-down version of Northern California, including San Francisco and its many districts, suburbs in the Bay Area, and forested park environments. In comparison, CARLA has only a limited selection of small, homogenous maps and poor fidelity. The downsides are that video games are less flexible for researchers and have less realistic physics simulations.

Specifically, we collected approximately 50,000 image samples by flying around various environments in Watch Dogs 2, Forza Horizon 4, and a small number of samples from the real world collected in Brookline, Massachusetts. We flew around in these video games, then manually annotated the samples with safety and steering information.

During training, we incorporated two forms of image augmentation. We randomly added uniform per-pixel (so called specular) noise to the image as well as uniform across-image brightening or darkening to the image, and added random image flips with the steering command correspondingly flipped as well. We found that this helps the model generalize across environments better. We trained with the AdamW optimizer for 150 epochs.

○ Flying The Physical Drone

The communication between the Jetson NANO and the Pixhawk drone controller is handled by MAVLINK. We are using Dronekit Python API which is a well-known source used in drone control and is integrated with MAVLINK. We have Dronekit python scripts ready to fly our drone autonomously, tasks resonating a pizza delivery, for example, going from point A to B and ascending and descending on drop-off points.

5 Relevant Engineering Standards

The most relevant government regulations governing our project are the FAA's regulations around autonomous drone flight. In order to operate commercially in the United States, every company has to get clearance from the Federal Aviation Administration (FAA). In order for a drone to operate beyond visual line of sight (BVLOS) it has to have detect and avoid systems, dedicated modules that help drones avoid obstacles. Specifically, for drones of our size, drones are governed by FAA PART 107—SMALL UNMANNED AIRCRAFT SYSTEMS, and would require waivers at minimum for the following sections of 107 to allow for autonomy: 107.31 (Visual Line of Sight Aircraft Operation), and 107.39 (Operation over human beings). More of the specific clauses might be relevant, depending on the exact nature of the delivery drone. Our prototype would also require waivers for 107.29(a)(2) and 107.29(b) (Operation at night, although adding a lamp to the front of the drone would fix this).

Currently, our project is not compliant because it is not operated by a company with an FAA license for autonomous flight. There is no standard procedure by which drones are certified to operate beyond visual line of sight

Commercial uses of totally autonomous drones are very rare. For more context, CEO of Wing Adam Woodwarth, in an interview with CNBC, states that they invested almost all of their resources into determining the BVLOS because truly autonomous drones cannot function effectively without it. In order to get FAA clearance, we will also implement our own detect and avoid systems. These solutions will likely require complex engineering; Zipline reports that it took them 7 years to fully engineer a system that could fly under any weather conditions, even in thunderstorms. Some companies also take into account the privacy of the customers, or opt for quiet propellers to prevent noise pollution.

There are no construction, software design, or operational environment requirements specifically for drone authorization in the United States (for instance, there is no requirement for formal method analysis of our control algorithm). Approval of our drone for commercial use would likely require close cooperation in testing with the FAA itself.

While internet protocols are used, we are using existing commercial solutions for communication through our cellular data module, and are not using wireless communication beyond that which could require compliance with FTC regulations.

6 Cost Breakdown

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
1	1	Logitech C922x Pro Stream Webcam – Full 1080p HD Camera, Black	$79.99	
2	1	SanDisk 128GB Extreme microSDXC UHS-I Memory Card with Adapter	$16.95	
3	1	HAWK'S WORK F450 Drone Kit to build	$380.00	
4	1	NVIDIA Jetson Nano	$149.99	
5	1	SIM7600G-H 4G / 3G / 2G / GNSS Module for Jetson Nano	$76.99	
			Beta Version-Total Cost	$703.92

A major part of the cost is the Hawk's work F450 Drone Kit. It includes a frame, Pixhawk flight controller, GPS and power module, motors, propellers, battery, remote control transmitter and receiver, as well as everything else we need to build a drone from scratch. The circuit board is already pre-soldered, so the assembly process is much easier and faster. We also use parts from brands for stable and reliable quality. If Pizzair goes into mass production, components will be bought separately and in bulk to further lower cost.

7 Appendices

7.1 Appendix A - Specifications

Requirements	Value, range, tolerance, units
Wheelbase dimension	450 millimeters
Weight	6 lbs.
Loading weight	<= 500 grams
Flight time	10-20 minutes
Battery power	4200 mAh

7.2 Appendix B – Team Information

Team 4, *Pizzair*, consists of Quentin Clark, Usman Jalil, Yafei Chen, Compton Bowman, and Ahemt Caliskan. We all chose to do a multi-disciplinary project to combine knowledge from our respective fields together to address a technological challenge.

Usman Jalil is a graduating computer engineer with a minor in mathematics from Boston University. Upon graduation, he will work as a software engineer at Travelers Insurance in the AI circle.

Quentin Clark is a graduating senior in Philosophy. After graduation, they will be attending the University of Toronto as a PhD student in Computer Science.

Yafei Chen is a graduating senior in Electrical Engineering. She will pursue a Master's degree at Georgia Institute of Technology after graduation.

Compton Bowman is a graduating senior in Mechanical Engineering.

Ahemt Caliskan is a graduating senior in Computer Engineering.