Drone Parcel Delivery Service: Casper

Cody Argyle, Corey Buchanan, Vanessa Bentley, Natalie Platil
Computer Engineering, University of Utah

***Abstract*—The field of drones has been growing substantially in the past few years, with many applications ranging from sports photography to military operations. Delivery drones have such a variety of uses, and therefore much potential for growth especially with them being more eco-friendly and cost-efficient than traditional methods. Casper is a custom quad-copter drone designed to pick up small packages and deliver them to a specified location, manually piloted.**

I. Introduction

DRONES are becoming increasingly utilized with applications in the military, shipping, delivery, sports, recreation, photography, and many others. One such multi-disciplinary application is item delivery. Many companies are investing in this technology, including Amazon, UPS, and even Domino's Pizza [1]. Though it's unlikely that drone delivery would be more efficient for heavy payloads than other means, we see potential in the delivery of items between small businesses in the same area, household deliveries between neighbors, and even delivering first aid kits to hikers while they await search-and-rescue.

Delivery drones solve a common problem of delayed delivery and energy waste [2]. They can provide a more energy-efficient small-scale delivery system with pre-programmed routes, obstacle avoidance, and real-time tracking. Drones are more energy efficient by nature of their compact size relative to highway vehicles. Local deliveries can be made more quickly because they don't have to be batched with other items on a cargo van (a technique often used to cut down on vehicle mileage). Several ideas that can be tested with delivery or other payload-capable drones are:

- Automatic food delivery from restaurants to homes
- Delivery of small supplies or medical equipment to people in hazardous situations (e.g. a wounded soldier or a trapped skier or climber)
- Delivery from local auto parts stores to repair shops
- Neighbor to neighbor deliveries (Facebook Marketplace pickups, for instance)
- More efficient and accurate crop dusting (Avoids expensive use of tractors or sourcing a pilot)
- Neighborhood pesticide application
- Small-scale specialty recycling pickups (e.g. cell phones, batteries, light bulbs)

We have built a custom quad-copter drone capable of delivering small payloads, such as letters or bubble mailers. Although our original design may not be able to carry large or heavy items, it demonstrates a solution to reducing the human carbon footprint of a combustion vehicle driving several miles to deliver a coffee or letter. Our design pattern could be easily modified with larger and more powerful drone equipment, if desired.

Part of our motivation in building an intelligent drone is that many components of our code, as well as hardware, apply to many different fields. For example, designing a smartphone app that works directly with other software and hardware. Indeed, this project is a learning experience for us in communication protocols, sensors, image processing, physics, embedded systems, firmware, and cyber-physical systems.

Our drone is a custom-built quad-copter. We've sourced our parts from a variety of places that are listed later in this report. Our drone is programmed to respond to commands via a radio transmitter, always giving the user full control over its flight. The user can load a parcel onto the quad-copter, and using the assistance of GPS and video feed, fly the drone to the destination of their choosing and release the parcel via a servo motor. The drone's video feed, as well as a map showing the drone's current GPS location with pointers towards the destination and pilot, are displayed via a custom app. When the weather was not cooperative we tested indoors and prerecorded demonstrations.

II. Related Works and Future

A key component of task-laden drones is autonomy. Autonomy in a drone means that a pilot isn't required to perform all of the tasks required to operate a drone and can leave some of the work to the hardware and software. Autonomy comes in various levels. One can program a drone to operate with the level of user control they desire. A high level of autonomy requires less user control (and vice-versa) but is more difficult to implement and is subject to more regulatory concerns. An example of a system with a high level of autonomy is a self-driving car. Systems that add low levels of autonomy are easier to implement. Two such examples are cruise control and lane-assist features in automobiles.

Florian Petit, in his article, refers to 6 levels of autonomy, ranging from level 0 to level 5. Level 0 refers to no autonomy (complete driver/pilot control) while level 5 refers to complete autonomy (complete machine control). Fig. 1 describes each level in better detail [3]. In the future we want to implement autonomy in our drone at level 2 or 3, meaning that the drone flies itself towards the destination whilst avoiding obstacles, but the pilot always needs to be attentive and ready to retake control. Ensuring the pilot always has this ability reduces our risk and liability.

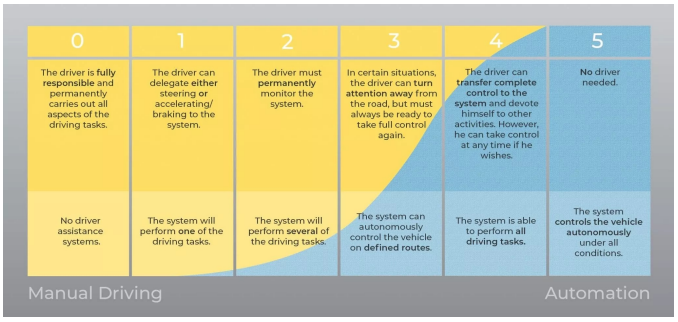

Fig. 1: Autonomy Levels [3]

III. Project Completion and Performance

A. Engineering and Design Decisions

The drone follows a typical quad-copter design with some additional features. A quad-copter typically has the following: a flight controller board, 4 motors, a propeller and an electronic speed controller (ESC) for each motor, an RC receiver, a battery, and a frame consisting of a body and four wings. Flight controllers vary in features, ranging from basic flight maneuverability to a complete autopilot system with peripheral support and a developer API.

TABLE I: Parts List

Item	Quantity	Vendor
Raspberry Pi 4	1	Ebay
Raspberry Pi Acrylic Case	1	Amazon
Raspberry Pi Camera w/ Casing	1	Amazon
Monoprice Weatherproof Case	1	Amazon
Mallofusa Universal Landing Gear	1	Amazon
BrosTrend USB Wifi Antenna	1	Amazon
Adafruit Mini GPS PA1010D	1	Adafruit
SIGP 1300mAh 6S Lipo Battery	1	Amazon
LiPo safe battery bag	1	Already owned
FCONEGY B6 Battery Charger	1	Amazon
iFlight Brushless Motor 1300KV	4	Banggood
HQ Prop Bi-Blade 7 inch Prop	8	Race Day Quads
M5 Low Profile Motor Nut	5	Race Day Quads
FPVDrone 7-inch Drone Frame	1	Amazon
1:18 Servo Motor	1	Already owned
Cargo Hold	1	3D Print
XT60 Plug Connectors	1	Amazon
Zip-ties, Screws, Other Hardware		Various
BETAFPV LiteRadio 2 SE Radio Transmitter - Frsky	1	GetFPV
Radiomaster R81 8CH Nano Receiver	1	GetFPV
MAMBA DJI F722 MK2 55A-128k 6S Flight Controller Stack	1	Diatone

See Table I for the parts we used for our project. For the electrical and mechanical power needed for the drone, we selected a high torque 1300kV motor to drive a 7-inch propeller. Larger propellers generate more lift (which helps us carry parcels) but typically require higher amounts of torque and are generally slower, which is why we went with a 1300kV motor instead of the 2000kV-3000kV motors used in racing drones. The kV value refers to rotations per minute (RPMs) per volt, so a lower kV motor can still achieve high speeds with higher voltage batteries. We selected a 6s (6 cell) battery with a capacity of 1300mAh and a nominal voltage of 22.2 volts. The battery capacity was lower than desired for a production delivery drone of its size, but it lasted long enough for our tests.

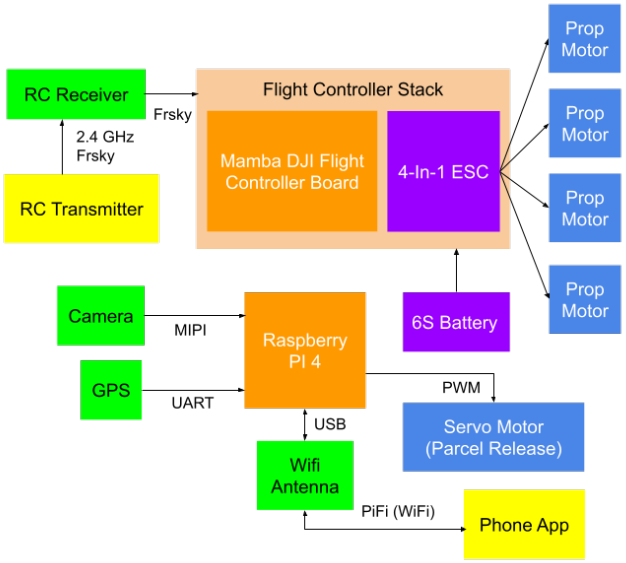

Fig. 2: Drone Block Diagram

Although the original plan called for 3D printing the drone frame, after further research, we decided to order one made from carbon fiber. Drone operation can take a toll on the frame, and 3D-printed plastics don't hold up to as much stress as we would need to put it through. During the drone construction, we found that we needed a protective case for the raspberry pi and landing gear to achieve smoother landings and reduce damage from crashing the drone.

After receiving the parts for the drone and beginning to work on putting everything together we had to alter a few things from the original design to be able to achieve our goals. We had mistakenly believed that the Mamba DJI F722 had included WiFi capability. It did not. Looking back, this poor assumption was likely because several other similar controller boards have WiFi capability. This meant that Raspberry Pi had to handle any data transmission to and from the phone app, including the camera feed. See Fig 2 for an overview of the hardware interfacing on completion of our drone.

We also made some adjustments to our software approach. We found that the best-supported firmware (out of the box) for our flight controller board is actually Betaflight, for which there is a "configurator" application that allows one to plug the flight controller into the computer and load new settings. This is simpler than writing custom routines, which would have required either firmware modification or interfacing with the controller externally via MAVLink over UART, although neither of those was an unrealistic option.

The beginning of the implementation was mostly focused on constructing the drone. The carbon fiber frame was easy to construct, but not very modifiable because it is very difficult to drill custom mounting holes through. Mounting the Raspberry Pi and the battery in a balanced manner was a bit trickier.

We elected to go with zip ties for mounting components and securing wires so that we could make adjustments as needed.

After the initial construction, we paired the transmitter and receiver and configured the flight controller using the Betaflight software. We first used the software to test the motors, and then set up an arming switch for the controller (to prevent accidental powering of the motors) and tested the motors with the RC transmitter. Not all of the components were secured yet, so we waited to fly.

The next phase of the project was configuring the Raspberry Pi and creating the phone app. We found that by creating a startup script, we could launch multiple programs on startup and keep our design modular. The servers for the camera feed, GPS, and servo control each ran from their own Python script and used TCP sockets over different ports. We used port 8080 for the video stream, 8081 for the GPS, and 8001 for the servo control.

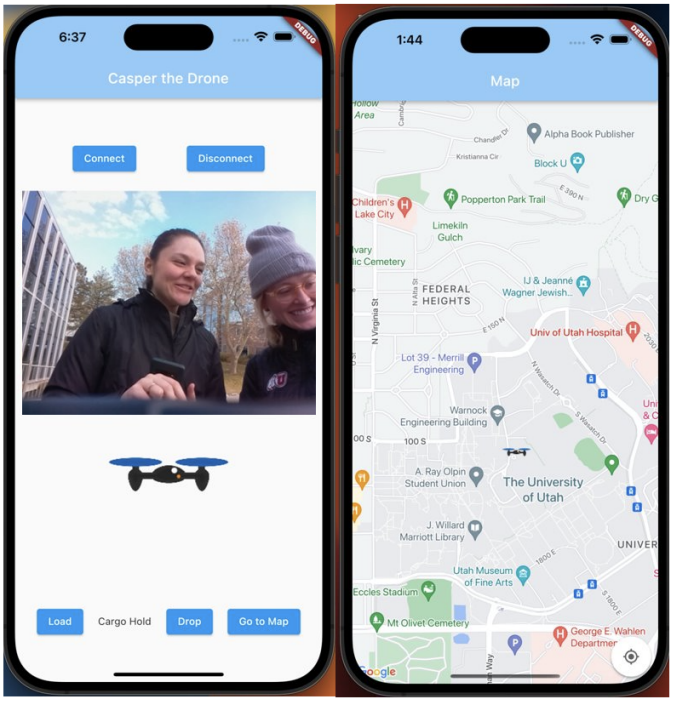

Fig. 3: App

The app (see Fig. 3) was created using Flutter, which is an open-source framework by Google for building multi-platform applications from a single code base. In our case, we created the app to mainly be used on iOS devices.

On the left screen, we have our ”connect” and ”disconnect” buttons for managing the live video stream. At the bottom of that screen, we have ”load” and ”drop” buttons for the cargo hold. We also have a button that navigates to our GPS tracking map. A custom pin is used in order to show the live coordinates of our drone.

There were several options and protocols for sending the camera feed, but there are few libraries for retrieving the stream in Flutter. Some methods, such as the VLC plugin, added a few seconds of latency. The best way we found after trying several was via WebSockets with help from Mitrajeet Golsangi’s tutorial [4]. We encoded the frames using OpenCV, send the raw data with WebSockets, and decode the frames in the app. We were able to reduce the latency to less than a half second when the drone was close by. In later tests, we found that latency increased and frame rate decreased with distance over the WiFi signal.

The GPS onboard is connected through UART to our Raspberry Pi. The GPS can be powered with either 3.3V or 5V. We selected 3.3V as we ran out of 5V pins. A guide written by Daniel Hertz [5] was helpful with integrating the GPS. They provided some instructions for configuring the UART interface. Adafruit (manufacturer of the GPS board) provides a python library and an example script [6] for retrieving the latitude and longitude coordinates from the GPS. We wrote a server script that grabs the latest coordinates from the GPS and forwards them to the client (the app) when they connect. Unfortunately, the GPS does not work everywhere (it had problems in buildings and on campus), so we send a set of default coordinates to the app in case we do not receive a ”fix” from the GPS.

The servo motor (our cargo hold/release) is controlled by a PWM signal. We could move the servo arm to the desired location by changing the PWM duty cycle. The servo control server waits for the ”Load
n” or ”Drop
n” command and then moves the servo arm accordingly.

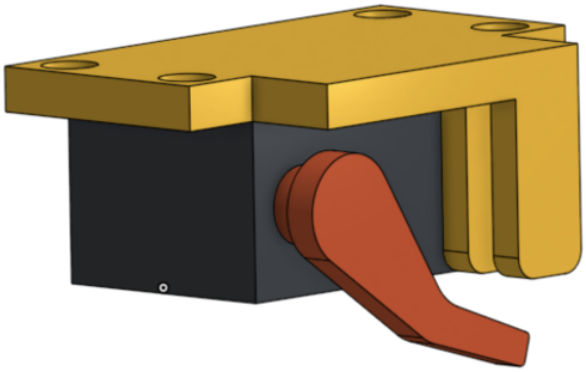

Fig. 4: Cargo Hold

For the cargo hold, we designed a servo arm and a slot for it to rotate into to lock the package in place (see Fig. 4). The package can be held to the arm via a key ring, a carabiner, or another type of loop. We modeled both pieces using Onshape (an online 3D CAD software) and 3D printed them in PLA. After mounting our cargo hold, Raspberry Pi, GPS, and landing gear, our drone weighed 2.2 lbs (according to a team member’s luggage scale).

B. Roadblocks and Mitigation

When we began connecting the wires, we noticed that the pin holes were too small for the pin headers and jumper cables we wanted to use, so we had to solder several of them directly. Our solder joints often came undone, but we mitigated this

with heat-shrink tubing, tie-downs for the wires, and getting better at soldering.

As we got closer to flight testing the drone, we weren't confident in the WiFi range of the Raspberry Pi, and re-configuring the Pi to be a hotspot is tricky to undo (in the case we want the Pi to connect to another WiFi during development). We purchased a WiFi antenna and configured it as a hotspot. We kept the internal WiFi interface free to connect to other networks. Our video feed worked up to 200 feet and the servo script worked up to 300 feet (requires less bandwidth than the video). We call this WiFi hotspot the PiFi™.

Another issue that we ran into was the battery. After one minute of flying on our first flight test, the motors were smoking, the paint was melting, and the battery had died. We assumed that we had a bad battery, so we stuck it in our Li-Po safe bag and charged our other one instead. We charged it to 25.2 volts (a full charge) and prepared for a flight test the next day. The next day came and nothing turned on when connecting the battery. When checking the voltage, we found that our battery had discharged to about 4 volts in total, which was far less than the minimum voltage per cell. The voltage was so low that our battery charger refused to recharge it. At this point, we ordered another battery, but while waiting for the battery, we found that the battery problems were our own fault. We connected the first battery, this time with both the power lead and the balance lead, and ran a discharge cycle, then recharged it. Balancing the cells solved our issues with the length of battery life and the motor temperatures.

The final issue we ran into was the weather this year. By the time we had enough features together (such as the camera feed to the app and the cargo hold) to perform meaningful test flights, Utah's snowiest November in a long time hit us, ensuring that the ground would never be dry. We decided to test in a hallway indoors, which proved to be tricky. Nevertheless, we had a few successful demo runs. For the final test before demonstration day, we installed some landing gear so we could fly over concrete. We crashed the drone into a small concrete enclosure and thought that our drone was completely destroyed. To our surprise, it held together, only needing a few small repairs.

C. Delivery of Requirements

Our baseline deliverables were to have a working drone (see Fig 5) with manual control, a video stream and GPS data sent to the user, an ultrasonic sensor to warn of upcoming collisions, and the ability to load and drop a small parcel for delivery. We achieved our baseline deliverables with the exception of utilizing the ultrasonic sensor. We found that with one sensor, we'd only be able to warn of collisions in one direction, and that to test it, we'd have to fly the drone closer to obstacles than we were comfortable with (if one didn't see the obstacle and the sensor had to warn them, it would be too late to prevent the collision). Our drone can be piloted fully via the RC transmitter. The video feed and GPS coordinates are sent via the drone's onboard WiFi to the pilot's phone app. The app has controls for loading and dropping packages via the drone's cargo-hold. We've also proven that our drone holds together in the case of a moderate crash, only requiring modest repairs.

Fig. 5: Finished Product

IV. Conclusion

Though crude and not easy (still fun, nonetheless) to pilot, we were able to demonstrate the drone's ability to deliver a small package from a loading point to a drop-off point. In the future, this drone could be improved to add some degree of autonomy or some features designed to make it easier to pilot. One issue with the package drop-off is that the sudden drop in weight when the package is released causes the drone to launch upwards until the pilot corrects. If we used the Raspberry Pi to communicate the timing of the drop to the flight controller, that correction could happen with less human intervention. Possibly then, we'd be comfortable testing packages closer in weight to the theoretical 22.8 lb weight limit of our motor/battery/prop combo. The drone has proven to be durable, holding together in moderate crashes and requiring only minor repairs.

V. Resources and Acknowledgements

- Dusty Argyle - A brother to one of the team members. Gave some critical advice during the research and design phase of the project. He is a former University of Utah computer engineering student.
- Nathan Jennings - Guide on realpython.com [7] helped us to handle multiple concurrent socket connections for the servo script.
- Daniel Hertz - Tutorial [5] helped us configure the Raspberry Pi UART for the GPS.
- Adafruit - Provided python library and sample script [6] for interfacing with their GPS breakout board.
- Mitrajeet Golsangi - Their tutorial [4] helped us get a camera feed from the Raspberry Pi to the app.
- RaspAP - This software allowed us to configure our wireless antenna as a WiFi hotspot and keep the internal WiFi interface configured as a WiFi client.
- Betaflight Configurator - This software provided us with several out-of-the-box configuration options for our Mamba DJI F722 flight controller board.

REFERENCES

[1] Dane Bamburry. Drones: Designed for product delivery. *Design Management Review*, 26(1):40–48, 2015.

[2] Joshuah K Stolaroff, Constantine Samaras, Emma R O'Neill, Alia Lubers, Alexandra S Mitchell, and Daniel Ceperley. Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery. *Nature communications*, 9(1):1–13, 2018.

[3] F. Petit. What are the 5 levels of autonomous driving? https://www.blickfeld.com/blog/levels-of-autonomous-driving/, Mar 2022.

[4] Mitrajeet Golsangi. Creating a live video streaming application in flutter. https://medium.com/dscvitpune/creating-a-live-video-streaming-application-in-flutter-43e261e3a5cc, Nov 2022.

[5] Daniel Hertz. How to use a gps receiver with raspberry pi 4. https://maker.pro/raspberry-pi/tutorial/how-to-use-a-gps-receiver-with-raspberry-pi-4, Feb 2020.

[6] ladyada. gps_simpletest.py. https://github.com/adafruit/Adafruit_CircuitPython_GPS/blob/main/examples/gps_simpletest.py.

[7] Nathan Jennings. Socket programming in python (guide). https://realpython.com/python-sockets/#handling-multiple-connections, Feb 2022.