Autonomous Drone Control with Computer Vision

Banji Oyewole and **Adam Lewis**

Abstract

In recent years, the effects of Moore's law, and improvements in rechargeable batteries have created a new industry around airborne unmanned aerial vehicles informally referred to as drones. The current all-in-one paradigm provides benefits but at significant cost. We propose a novel system for drone control, decoupling the drone from the vision system. Using cheap store-bought drones and a Microsoft Kinect, we demonstrate a successful proof-of-concept for this approach.

Introduction

Quadcopters and other drones use data from various sensors such as on-board cameras in order to track and maintain their position in 3D space. While this approach works well, we recognize the issues that come with it, including price, power consumption, and the tight coupling of the drones with the vision system. We have propose an approach to drone control in a distinctly new way to address these issues and open up the doors for potential new applications of drones as well. We can increase the agility and flight time of similarly sized drones, and we can reduce the cost of these drones all while enabling them to do more by making it easier for them to together. We do this by decoupling the flight control system from the drone, and moving that work to a nearby computer. Using this technique we have been able to achieve sustained autonomous flight with an off-the-shelf image sensor and an almost completely unmodified pair of drones.

Related Work

Our drone research fits into contexts including: 3D imaging and other usages for drone technologies, including: tracking 3D points using stereoscopy, drones for photography, drones for light shows, drones' swarm detection technology, and PID controllers.

3D Imaging with Stereoscopy Similar to human eyes, it is possible to determine an object's distance away by using two adjacent cameras. This is achieved by determining which pixels in the two visual feeds correspond to the same point and then performing advanced trigonometric calculations based on the cameras' angles from the point and the disparity between the cameras' locations from each other (Valente 2011). While the theoretical groundwork has been laid out for a 3D vision system of this kind, there are implementation details that are not trivial to overcome. These hurdles include ensuring both cameras are properly calibrated, compensating for imperfect horizontal alignment between the two cameras, consistently identifying an object in every frame of a video stream, and performing all of these calculations correctly for every frame (Valente 2011, 4). In addition, these calculations must be completed quickly enough to keep up with the desired FPS. While we had originally looked into a stereoscopic vision system for use in this project, the idea was discarded in favor of a less problematic approach.

Drones for Storytellers Freefly Systems is a corporation that specializes in drones for photographic and cinematic applications. Freefly is unconventional in that its specialized drones allow for camera systems to be mounted above the rotors as well as below. Their systems stabilize using GPS, accelerometers, gyroscopes, barometers and magnetometers, as well as having the option to be operated by a dedicated drone pilot. Freefly Systems is one of the players in the professional drone market offering reliable, capable, and intentionally built systems for cinematographers and photographers alike (Freefly Systems 2018).

Drones for Light Shows Intel Corp. is another player in the drone market. They have developed two different classes of drones. One is a drone designed for night time light shows, called "Shooting Star Mini," 1218 of which were coordinated together to create the Olympic rings at the 2018 Winter Olympics in Pyeongchang (Intel) (Gartenberg).

Swarm Intelligence in Autonomous Cars This research speaks to the requirements and design ideas of swarms in automotive applications as well as generally. This paper speaks to specific configurations of swarmed objects as they relate to natural phenomena like flocks of birds, schools of fish, and swarms of insects, as well as a breakdown of the general principles we take away to govern the interactions between members of the swarm (Prakash et al.).

Proportional/Integral/Derivative Controllers PID controllers are a part of the field of control theory. PID control has a wide array of applicability and is a general set of principles that govern how to create controls that arrive at domain specific targets in efficient and consistently effective ways. We initially emulated the effects of PID by taking into account fringe cases that this system has built into how it behaves before implementing it more completely. PID can be split up and taken as PD, or PI control based on the requirements in the domain in which it is being applied (Douglas).

System Description

System Overview Our system was implemented in Processing, an IDE/set of Java libraries used for visual applications. Processing works by calling an initialize function and then continuously calling a "draw" function in a loop. This means that we designed our system to operate in a loop. The source of our primary data is our vision system, which looks at the drone and its proximity to the target. We send commands to update the drone's fans, and our vision system sees the drone's new position and feeds that data through the system again. The way we 'see' the drone is via our vision system, the Microsoft Kinect for Xbox 360. We call it a "vision system" because it is more than just a singular camera. The Kinect has a color camera and an infrared camera. The Kinect sends out a specific pattern in infrared that is invisible to the human eye, but is picked up by the infrared camera. Internally the Kinect knows what this pattern should be on a flat surface, and it processes the differences between what it expects and what it sees in the room in order to give us information about how far away objects are from the Kinect in a room. This is the same technique that Apple uses in the iPhone X's FaceID feature. After Kinect's processing we do a little more to match up objects we see in the color image with the depth data from the Kinect. Our processing involves finding the color that we're looking for, and looking up those pixels in the corresponding depth image to get our 3D position we use for our calculations. Our processing of this data happens with a Java program we wrote, Series12. Series12 has been built on top of Processing 3, a library we used for building the UI that allows us to control environment variables that affect the drone's autonomous flight at the highest level. Series12 uses the openCV library to do contour detection' which detects the biggest group of color based on hue. This data can be used to keep track of multiple objects of the same color, as well as be recomputed for a different hue to keep track of objects with another hue. We combine data about the drone's position, the target location, and the drone's velocity in order to generate commands. These commands are sent to the drone in an attempt to get it closer to the target. The commands that we send emulate joystick input, but at a level of abstraction that allows us to use integer numbers to represent physical joystick positions. This information is sent to a single board computer (a Raspberry Pi) that governs these digital joysticks via a string format we have devised called the Mamba format. This means that we can control the drone from any device as long as that device sends out properly formatted Mamba strings. A Mamba format string is of the form
{'Y':*y*, 'XZ':*yaw*, 'R':*roll*, 'P':*pitch*, 'T':*time*}
where 'Y' is the Y-axis, 'XZ' is the yaw, 'R' is the roll, 'P' is the pitch, and 'T' is the time, measured in seconds. A sample string would look like
{'Y':113, 'XZ':96, 'R':112, 'P':91, 'T':14.050}

These integer values (excluding time for obvious reasons) represent joystick states.

We built two programs that are able to send Mamba strings to our drones, Panda_Interface.java which is written in Java, and Panda_Interface.py, which is written in Python. By creating a standardized way to communicate with the drones, we were able to control the drone from a Sony Playstation 4 controller, which was useful during the intermediate phases of our research. The Playstation 4 controller interface has been useful in allowing us to test the drone physically while separated from our autonomous control system.

We send Mamba format strings over UDP/IP (the Internet) to the drone's controller, and then the drone's controller decides whether or not to apply the values to the drone based on the the amount of time it took for the data to arrive. This is possible because the Raspberry Pi and the computers we send our commands from reference the same global clock. The updated values are sent to the drone using the 2.4 GHz communication protocol that existed between the drone and the remote before our modifications. When this data arrives at the drone, it processes the commands and operates the drone's rotors accordingly. We then generate a new Mamba format string based on its new position and distance from the target, and we send that new command. Unlike other applications of control theory, when we reach our target in 3D space we don't end the execution of our program. It continues to run on an infinite loop to keep the drone in the air at the target until another target is set, or until we land the drone.

Series12 Interface Series12 is a control panel that allows us to set high level system parameters for autonomous flight. Series12 allows us to specify the target's 3D coordinate, visually verify the quality of the object detection system, and see the Mamba format strings that are being sent.

Series12 also handles drone detection, and generates commands (commands are generated based on the user-selected target, the weighted average of the most recent average positions of the drone, the current velocity of the drone, and the integral component of the PID control system). Series12 then sends these commands over UDP/IP in the Mamba format via an asynchronous request to the PandaInterface message handler.

Series12 has a live view of the RGB image from the Kinect sensor, as well as a toggle-able colorized depth image.

To operate a drone autonomously, we need to specify the IP address of the Mamba receiver via the console, then select the target in 3D space (which we do by selecting depth via a slider to set the z coordinate and then clicking on a location to set the x and y coordinates). From there we can

easily toggle autonomous flight with a dedicated button, and ground the drone just as easily with the space bar.

Position-Lock We implement Position-Lock using a PID controller (see Douglas videos "PID Control - A Brief Introduction" and "Simple Examples of PID Control"). PID stands for "Proportional-Integral-Derivative". A PID controller allows us to compensate for errors in the drone's flight. In this context, error is defined as the difference between the drone's current position and the target position. The Proportional path looks at the drone's position and compensates proportionally based on the error. In other words, the farther away the drone is from the target, the greater the compensative force. The Integral path compensates for steady-state errors by keeping track of the accumulation of error over time. As the drone continues to not be at the target, the accumulated error increases, and the drone moves towards the target based on this value. The Derivative path compensates for errors based on the velocity of the drone. This is mostly used to restrain the drone from overshooting the target when our system detects that it is approaching too fast. Each path is added together, and this sum is the PID controller's error compensation. We calculate the PID error compensation for the x, y, and z dimensions, which translate to the roll, y, and pitch commands respectively. We format these commands into a MAMBA string and send it to the Raspberry Pi. The above process occurs in a Processing loop which repeats multiple times per second. Here is a high level overview of our position-lock algorithm:

Algorithm 1 Position-Lock

```
1:  procedure POSITION-LOCK
2:      current ← drone's current 3D position
3:      target ← 3D position of target
4:      Update current position
5:      xCom ← sum(xPComp, xIComp, xDComp)
6:      yCom ← sum(yPComp, yIComp, yDComp)
7:      zCom ← sum(zPComp, zIComp, zDComp)
8:      commandVector ← (xcom, yCom, zCom)
9:      MambaCommand ← format(commandVector)
10:     send(MambaCommand)
```

It is worth mentioning that this is only the Position-Lock algorithm itself. Our implementation of the system also includes special failsafes, overriding the recommendations by the PID system to handle cases where the drone ascends too quickly, attempts to exit the field of view, actually leaves the field of view, or otherwise needs to be suppressed.

Wireless_Map.py Wireless Map is a python script which runs on the Raspberry Pi that controls the drone.

The Raspberry Pi has been wired into the circuit board of the drone remote. We replaced the underlying circuit component of the joystick with a version that we can interface with digitally instead of physically. We replace the analog potentiometers under the joysticks with digital potentiometers that we can control with the Raspberry Pi.

Potentiometers are essentially water faucets for electricity. The drone's remote interprets the 'flow' to control the drone's fan speeds. Wireless Map takes in Mamba format strings and updates these digital potentiometers with the correct flow rates for how we want the drone to fly. Wireless Map also handles receiving Mamba format commands on the Raspberry Pi. It can be configured to accept commands when typed in by the user, or when sent over the network via UDP/IP. UDP, which stands for User Datagram Protocol, is a low level protocol that allows us to send data without concerning ourselves with its guaranteeing its delivery. The most common protocol for sending data over the internet is TCP/IP which sends data and awaits ACKs or acknowledgements of data having been received by the remote machine. This process ensures data completeness and integrity, but at the cost of speed. By exempting itself from these ACKs, UDP/IP finds itself particularly useful when sending information that is only relevant for a moment, like when streaming video or streaming data like the Mamba format string.

Electrical Engineering The way we send data to the digital potentiometers is over something called the Serial Peripheral Interface(SPI). To do this we used a circuit prototyping board to wire the serial pins on the Raspberry Pi to our digital potentiometers. At the highest level, we provide 5v power to the digital potentiometers, and get out some fraction of that based on what we told it via Wireless Map. The way we talk to it is by sending values between 0 and 128 that correspond to resistance values. Lower values correspond to higher resistance, or a tighter faucet. To overextend this metaphor just a little, the water pressure we were controlling meant that the relevant range for us was between 60 and 128, which is why most values sent over the Mamba format hover around the high 90s when they are at a rest state. Our circuit has four such potentiometers, each of which corresponds to a key in the Mamba format string. Via the specifications of the Serial Peripheral Interface, we send the same data to all four potentiometers, but we tell three of the four to ignore that information via a designated "selection" pin. This configuration is important because sending values can only be done over two pins on the Raspberry Pi whereas addressing the potentiometers can be done using almost any of the 38 other pins available to us. By sending the data to all of the potentiometers using only one of these two specialized pins we are in theory able to control up to two drones with one Raspberry Pi.

Evaluation/Results & Analysis

Our implementation of the overall system was a partial success; we successfully built a system that allowed for the fully autonomous operation of a drone for several minutes at a time. The Position-Lock algorithm itself consistently works, but there are challenges because our specific implementation does not allow the system to work in all conditions including direct sunlight. The vision system experiences problems with consistent tracking due to the low-resolution of the Kinect, the ad-hoc nature of our color sensor, and the varying light-conditions based on location. In ideal conditions, Series12 works well enough for us to consider this a successful proof of concept, but since it cannot compensate for these factors on its own, we do not consider our imple-

mentation of Series12 to be a total success. The limitations on Series12's viability originate in some of the design decisions we made. One significant problem in our system is the unreliability of our tracking data. We tracked a drone's position by attaching a mass of colored cotton balls onto the top of our drone and then using OpenCV to track the cotton balls' x and y coordinates. This method is not entirely accurate, and the Kinect's low resolution caused OpenCV's color tracking to sometimes falter by jumping to a completely different point in the field of view instead of staying on the cotton balls. The technique we used to track the drones does not scale well. A more robust way to track the drone would be to build an image model of the specific drone we are using. By building a model for each drone, we would be able to easily detect them in a way similar to how OpenCV tracks faces, and the drones would not have to be physically modified to work within the system. Another viable technique would be using the infrared spectrum instead of the color spectrum, and looking for trackers in an image by shape instead of by color. This has the added benefit of giving us more information about the orientation of the drone when we use more trackers on them. This technique would provide significantly better accuracy than our current implementation as well, but would require us to change the way we access the drone's distance from the camera. This is because the Kinect uses infrared to get an object's distance from the camera. While the Kinect is easier to implement, using a vision system that calculates depth differently would allow us to work with these improvements. Due to the limitations in the quality of our drone tracking because of our system, we had to look for working environments that met our size requirements as well as met our lighting requirements. Specifically, an experimentation room had to be large enough to provide enough space for the drone to fly around, be available for us to work in, and because of the hue of the tracker we used during testing had warm enough lighting to not confuse OpenCV's color tracker. The room which most consistently fit all of these requirements was the Murray Boardroom, so we performed most of our experiments in there. Whenever we made a change, we set up a screen recording to capture the Series12 UI as we performed each test. The UI contained information including the coordinates of the drone, the coordinates of the target, the Mamba string being sent to the drone, the drone's velocity, and the drone's accumulated error. When the drone would act in a way we would not expect, we would review the recording to investigate what could be behind the behavior. Since our experimental tests number in the hundreds, we have decided to include a select few of them to include here to illustrate our process.

Selected videos can be found at www.banj.io/projects/capstone/video_tests

In video 1 we see the drone shoot up to the ceiling. In order to determine why it behaved this way, we reviewed the Mamba commands being sent and discovered that we needed to implement a failsafe "anti-Icarus" code to restrain the drone from ascending too fast.

In video 2 we observed the drone travelling back and forth through the target. Our research into PID controllers gave us

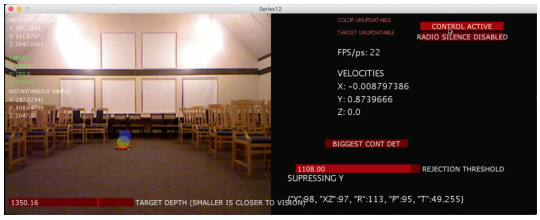

Figure 1: Screenshot of an early form of our UI

reason to believe that this behavior could be compensated for through the introduction of a Derivative controller.

In video 3 we see the drone immediately shoot itself out of frame once the target is reset. On inspection of the video, we discovered this was because the Integral error was not being reset upon new target selection.

Conclusion

The idea of decoupling drones from their control systems is a novel architecture. This project does the work of showing at a very high level that it can work. In our future work we go on to describe ways in which our system can be made more robust and work in more varied conditions, as well as detail new applications of drones that are made possible by having a system that has absolute and relative information about one or multiple drones in a space. From our results we have learned that there are a lot of components to our system, and we learned that combining data in meaningful ways opens up a lot of possibilities, but that it also comes with its own set of challenges specific to the domain. We have also learned that while drones are fickle they are also immensely capable, and we have a renewed excitement for the future that they will bring.

While we do not believe our work has any immediate impact on the development of drone technologies, we do believe our approach can be looked to as an proof of concept for the validity of autonomous drone control with decoupled vision or control systems.

Future Work

This implementation of decoupled controls was a proof of concept. We were able to tangibly demonstrate the ability to fly a drone autonomously without the use of data from onboard sensors, which effectively reduces the load on the system to maintain its own flight. We wrote our control system in a way that allows for the ability to control more than one drone at the same time, but did not work to overcome swarm issues like proximity that come up when the drones are close to each other.

Using a more powerful vision system would be another avenue of work we would pursue. One of the largest challenges during testing was our limited field of vision. Expanding the visual field by performing sensor fusion of data from multiple Kinects, or some other 3D world visualization system would be a meaningful expansion of this project and a logical next step once support for multiple drones is achieved.

Another extension of this work would be to switch to using infrared instead of visible light to detect the drone in 3D space. We would also want to use more than one tracker per drone so we can have access to orientation information in addition to the 3D coordinate we can access now. Having access to the drone's orientation information means that we gain the ability to rotate the drone.

We were able to set targets for the drone in 3D space, and we escaped some of our timeline challenges by pipelining features. Of our extensions, the pre-programmed movement patterns are still of interest to us. The idea of having dynamic targets that are set by Series12 and the coordination of multiple drones in that space is an approach to Intel's night time drone shows that we find interesting.

Bearing in mind that this system is designed to put less pressure on the resources of the drone, another area of exploration would be to have specialized drones completing tasks. Example tasks include window washing or coordinated object handling by multiple worker drones governed by a master drone outfitted with the vision system. This drone mounted vision system still serves our initial goals of minimizing cost, in that the vision hardware isn't duplicated by all worker drones. It also meets our goal of modularity. Low-cost worker drones can be swapped out for others when damaged without having to make any changes to the higher cost governor drone.

Achieving Intel Corp. style night shows in our decentralized control configuration would involve every part of the future work described as well as needing to make use of multiple governor drones to overcome occlusion of drones by other drones. Bringing together visual data from more than one air based vision system would allow us to do 3D reconstruction, which would increase our effectiveness at being able to distinguish between high volumes of close proximity drones.

Another necessary component would be to develop an easy way to manage the airborne governor drones and the worker drones. Our low cost approach meant making minimal changes to the lowest level drones in the system, but designing a multi-tiered system would mean that the way we communicate with these drones may need to change and the 2.4GHz radio component of our control stack may need to be removed as we scale.

Acknowledgements

Many thanks to Dr. America Chambers, Dr. Anthony Mullen, and Ethan Russell for their assistance and support.

References

Douglas, Brian B. *PID Control - A Brief Introduction.* YouTube, YouTube, 13 Dec. 2012, www.youtube.com/watch?v=UR0hOmjaHp0.

Douglas, Brian B. *Simple Examples of PID Control.* YouTube, YouTube, 21 Dec. 2012, www.youtube.com/watch?v=XfAt6hNV8XM.

ALTA 8 Drone - Intelligent, Heavy Lifting Multirotor by Freefly. Freefly Systems, freeflysystems.com/alta-8/intelligent.

Gartenberg, Chaim. Ïntel's Winter Olympics Light Show Featured a Record-Breaking 1,218 Drones.T̈he Verge, The Verge, 9 Feb. 2018, www.theverge.com/2018/2/9/16994638/winter-olympics-2018-intel-drone-show-world-record.

D̈rone Light Shows Powered by Intel.Ïntel, www.intel.com/content/www/us/en/technology-innovation/aerial-technology-light-show.html.

Prakash, Yadhu, et al. "Incorporation of Swarm Intelligence in Autonomous Cars." International Journal of Computer Science and Information Technologies, vol. 5, no. 5, Oct. 2014, pp. 13.

Valente, Rosrio D. V. 3D Points Recover from Stereo Video Sequences Based on OpenCV 2.1 Libraries. MS Thesis. AGH University of Science and Technology of Krakow, 2011. Web. May, 2018.