Autonomous Drone

Joseph Helland, Jesse Whitaker, Patrick Cowan, Scott Glass
Dept. of Electrical and Computer Engineering, University of Utah

Abstract

An autonomous scalable nano-drone would be useful for a variety of purposes including pollinating plants and performing reconnaissance. We propose building a drone that can safely navigate an area while simultaneously searching for a specific target, such as a QR code or a red dot. Rather than building our own drone, we will modify a quadcopter called the CrazyFlie 2.0 by attaching a camera and proximity sensors so that it can avoid obstacles and search for targets. Our project is successful if our drone can navigate a room without crashing and find all of the targets.

I. Introduction and Motivation

For our senior project we propose to modify a quadcopter so that it can fly autonomously and search for targets. We will take an existing quadcopter called the CrazyFile 2.0 and attach proximity sensors for avoiding obstacles and a camera for finding targets. We will demo this by placing targets (such as QR codes or red dots) around a room and then releasing the quadcopter. We will know our project is successful if the quadcopter can find all the targets and report their location back to the central computer without running into walls or other obstacles.

The idea for this project came from reading an article about colony collapse disorder (CCD). CCD is a mysterious phenomenon responsible for the deaths of as much as 30 – 90 % of adult honey bees within a colony. The effects of CCD could be economically disastrous because about 130 agricultural plants in the United States are dependent on honey bee pollination (valued at about $15 billion annually) [1]. We think our autonomous drone could be a great solution for CCD. The central computer could use an algorithm to recognize plants that need pollinating and direct the drone to the targets. CCD is just one example of where an autonomous drone could be useful. Other good examples include finding survivors after a natural disaster, performing reconnaissance in a military situation, or tracking animals for hunters.

II. Background

An autonomous drone is not a particularly new or revolutionary concept. The military has been using drones for the past several decades and some companies even sell drones as toys such as the AR Drone 2.0 by Parrot, the FPV X4 Mini RTF Quadcopter by Hubsan, and the Phantom Aerial UAV Drone Quadcopter by DJI. Even Amazon has been doing research on using drones as a delivery system for packages. Although our project is not completely original, the aspect that sets it apart from all these other drones is scale. To date, no one has implemented a completely autonomous quadcopter as small as the CrazyFlie (which weighs only 27 grams and is as big as the palm of your hand). The closest product we could find to this is called the Zano and weighs 55 grams.

III. Hardware

A. *Quadcopter*

We briefly considered making our own quadcopter for this project but quickly decided against it. Due to time constraints and the complexity of flight, we thought that making both a quadcopter and making it autonomous would be unrealistic. We even found a senior project from 2013 who attempted to create a quadcopter and it proved to be a huge amount of work [2].

Fortunately, we were able to find a quadcopter that was perfect for our purposes: the CrazyFlie 2.0 by Bitcraze (see Fig. 1). This quadcopter uses both open source hardware and software, making it an optimal platform to host our modifications. The CrazyFlie also provides a number of other useful features such as online documentation, a radio module and communication protocol, easy flashing via radio, and some basic flight code. The tiny quadcopter is also surprisingly powerful, with 168MHz Cortex-M4 MCU with 192kb of SRAM and 1Mb of flash memory [3]. The CrazyFlies MCU comes preloaded with some basic flight code. It basically uses stabilization algorithms to point the quadcopter forward and keep it somewhat level. The CrazyFlie also comes with code to interface with the radio and manage power usage.

B. *Proximity Sensors*

Our quadcopter will use a combination of infrared and ultrasonic sensors for collision avoidance. The infrared portion will consist of many infrared LEDs coupled with a receiver in each of the cardinal directions. Infrared LEDs are perfect for this platform because they are extremely light and draw very little power; however, infrareds lose a lot of accuracy depending on the angle the light hits the obstacle.

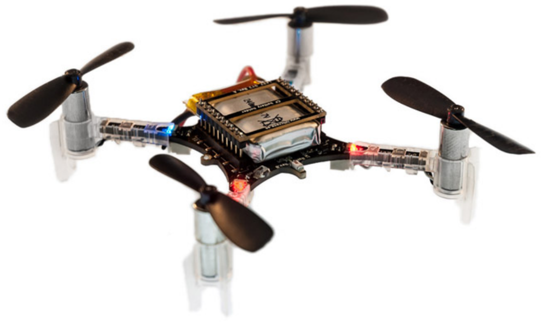

Fig. 1: Picture of the CrazyFlie 2.0 quadcopter [4].

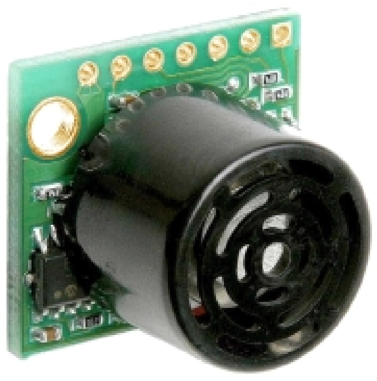

Fig. 2: Picture of the LV-MaxSonar-EZ2 ultrasonic sensor [5].

For the ultrasonic portion we will be using the LV-MaxSonar-EZ2 ultrasonic range finder (see Fig. 2). The following characteristics make this sensor ideal for our project: 1) It only weighs 5 grams; 2) It supports input voltages from 2.5 to 5.5 volts; and 3) It can detect objects from 0 to 254 inches (6.75 meters) [5]. The LV-MaxSonar-EZ series offers options in a variety of beam widths, but we decided that the EZ2 (medium width) was best because any beam wider may be disturbed by the spinning of the rotors. If we had a bigger quadcopter we would probably use these sensors exclusively because they work well regardless of the material and angle of the obstacle.

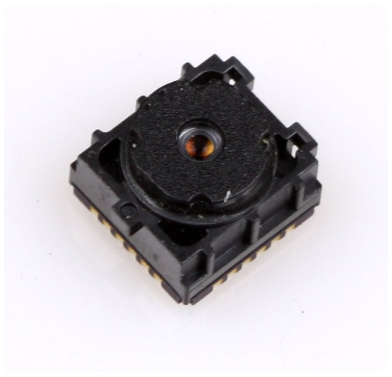

Fig. 3: Picture of the OV3640 image sensor [7].

C. Camera

We will be using the OV3640 image sensor for our camera (see Fig. 3). The main advantages of this camera is it operates between 2.5 to 3.0 volts and is only a few centimeters across [6]. The biggest disadvantage of this camera is how many IO pins it takes to run it (12 out of the 13 available). We will solve this problem by using an additional micro controller unit (MCU) to interface with the camera.

IV. Interfaces

All of the proximity sensors will communicate with the MCU via analog to digital converters (ADC). We can configure these ADCs so that they trigger an interrupt whenever they detect a nearby obstacle.

The camera needs a total of 12 IO pins to operate. Four of them control the camera itself, doing functions such as telling it when to take a picture. The other 8 IO pins are for transferring data. Due to how many IO pins this camera requires, we will have the camera interface with an MCU solely dedicated to control it on its own printed circuit board (PCB). The camera MCU will periodically transfer camera data to the main MCU through a universal asynchronous receiver/transmitter (UART) interface. The main MCU will then transfer this data via the radio to the central computer.

V. Project Tasks

This section describes all the tasks involved with this project. A testing and development strategy is also described.

1) Collision Detection - This involves using proximity sensors to determine if the drone is about to fly into something. If the proximity sensors detect something, the software should update the flight path accordingly. [Testing and Development Strategy] : The first step is to create a simple breadboard circuit to display an output on an oscilloscope when something is nearby. The next step will be to interface this simple circuit to the quadcopter (which will not be flying at this point). The final step will be to use a custom designed PCB in place of the breadboard circuit and to test the quadcopter in flight.
2) PCB Design - Both the proximity sensors and the camera will need a PCB. For the final application these will probably be on the same PCB so as to weigh less, but initially we will have them separate. The rationale for doing this is we want to be able to start writing and testing the collision detection code as soon as possible. By separating into two PCBs at first, this allows us to better work in parallel; one of us writes application code, while someone else can design the final PCB. [Testing and Development Strategy] : Hopefully by the time the final PCB is manufactured we will already have a good amount of code written. Testing then becomes simple because we can simply run the collision detection code already written.
3) Image Processing - This involves interfacing with the camera module and designing algorithms to recognize basic objects. [Testing and Development Strategy] : The first step will be to design a way to interface the camera with an MCU. The next step will be to design a protocol to allow the second MCU to communicate with the primary flight MCU to allow image transmission back to the central application. The final step will be to write the image processing algorithms that will run on the central computer.
4) Central Application - This is the main program that will be used to fly the drone. It will be written in Python and contain all the logic to track progress and update the flight path. This program will be built off of the existing CrazyFlies flight program to save development time. [Testing and Development Strategy] : The most simple test will be to get a drone to take off and land. When this has passed, we will try to get the drone to fly autonomously around a room. The final test will be to integrate the image processing and have the drone search for targets.
5) Documentation - All documentation will be done on our project website. We will keep weekly logs of what was accomplished that week, what difficulties we encountered, and what we plan on doing the following week.

VI. Group Management and Communication Plan

During the school semesters our group will be meeting once a week for at least an hour to discuss problems and assign tasks. Over the summer we will use a combination of email, texting, and Skype to keep in touch and stay synchronized.

VII. Schedule

This project consists of three main milestones: collision detection, video streaming, and a functional central application. We hope to finish the first by July 3, the second by August 28, and the third by October 15. The project will basically be done by the third milestone; all that would remain at that point is testing and performance enhancements. Although this might be somewhat optimistic, we hope to complete the bulk of the work over the summer and finish the project a couple of months before demo day. What follows is our planned schedule to achieve our milestones:

Task	Assigned to	Deadline
Begin initial experimentation with proximity sensors.	Jesse	March 27
Connect a proximity sensor to the MCU.	Jesse and Joey	April 17
Order all IR sensors.	Jesse	May 15
Begin testing with the camera module.	Scott and Patrick	May 15
Connect all IR sensors to the MCU.	Jesse and Joey	May 29
Send IR PCB to fab.	Patrick	June 5
Construct IR PCB.	Patrick	June 19
Completely functional collision avoidance.	Joey	July 3
Basic stabilization code complete.	Joey	July 3
Sent Camera PCB to fab.	Patrick	July 3
Construct camera PCB.	Patrick	July 17
Interface camera PCB with CrazyFlie.	Scott and Patrick	July 31
Start work on the central application.	All of us	August 15
Stream camera data back to central application.	All of us	August 28
Update the application to process images.	All of us	September 18
Update the application to direct the CrazyFlie to targets.	All of us	October 15

VIII. Risk Assessment

Due to the nature of quadcopters and the complexity of flight, there are unfortunately a huge number of risk factors involved with this project. In this section we list several possible risks and describe a contingency plan should the problem occur. Following the name of each risk is a color indicating how severe we believe the risk is with RED being the most severe, YELLOW moderately severe, and GREEN being either unlikely to happen or easy to work around.

1) Total weight [RED]: The most obvious problem is weight of the sensors. If the combined weight of all the sensors and circuits to make them work weigh more than the carrying capacity of the quadcopter, then it will not get off the ground or have a very short flight time. [Possible solution]: We could use all the same software and electrical components used in the CrazyFlie 2.0, but just replace the motors and battery with bigger options. The problem with this is the time and effort required to source the new components and assemble the board.
2) Breaking the quadcopter [YELLOW]: We will try to be as careful as we can, but there is a very good possibility that we will break the quadcopter at some point during our testing. [Possible solution]: Thankfully, the designers of the CrazyFlie were smart enough to design the quadcopter such that the motors are the most likely thing to break in a crash, which are cheap and easy to replace. If the quadcopter broke completely (not just the motors), then we would have no choice but to just buy another one. Because of the liklihood of this problem, it is imperative that we complete the riskiest task (i.e. collision detection) early enough so that we would have enough time to order a new quadcopter if necessary.
3) Bandwidth [GREEN]: If we need to stream image data off the quadcopter it will require a ton of bandwidth. [Possible solution]: We don't think this problem is very likely to occur because the CrazyFlie's radio runs at 2.4 GHz. However, if the problem were to occur there are several possible solutions. One solution is to send out the image data at regular intervals, rather than streaming it live. Another is to scale down the image quality. If all else fails, we could just do the image processing onboard.
4) Balance [YELLOW]: If the weight we add makes the quadcopter unbalanced, it could cause the quadcopter to drift and potentially ruin our stabilzation code. [Possible solution]: We will try to minimize this risk as much as possible by evenly distributing weight around the quadcopter. If it proves impossible to balance the quadcopter this way, then we could use the information from the infrared sensors to detect drift and adjust the motors accordingly.

IX. Bill of Materials

Part	Part Number	Vendor	Cost
Quadcopter	CrazyFlie 2.0	BitCraze	\$240
Ultrasonic Sensor	MB1020	Maxbotix	\$25
Camera	OV3640	Digikey	\$10
PCB (appox. 4 in.^{2})	N/A	Local Fab	\$20
Camera MCU	STM32F407	Digikey	\$15
Infrared LEDs (4)	COM-09349	SparkFun	\$4
Infrared Sensors (4)	TSOP38238	SparkFun	\$8
Total Cost: \$322			

X. Deliverables, Demonstration, Stretch Goals

Our baseline goal is to demonstrate collision detection on the drone during flight. Once we get this to work, we will turn all our efforts towards integrating the camera module and writing the central application. We will consider our project 100% successful if we can get these three aspects working.

There are several things we could do to demonstrate our working project. A good test of the collision detection would be to push our hand towards the quadcopter. If it is working, then the quadcopter should fly away so that the hand never touches it. We could extend this test and have two people "push" the quadcopter back and forth to each other as though they were playing catch. A good demonstration of the image processing and central application would be to display a QR code somewhere in a room and release the quadcopter. The quadcopter should search the room until it finds the QR code. Once it finds it, it could send a message to the central application telling its location.

We don't exactly have a lot of stretch goals for this project; right now we're just concentrating on completing our baseline goals. If we were to finish early then the next step would probably be to scale our application up to handling two quadcopters at a time. We could even implement some kind of protocol so that different quadcopters could forward messages between each other.

XI. Conclusion

At this time we have successfully connected an ultrasonic sensor to a CrazyFlie. We were even able to program the CrazyFlie such that when the sensor detects an impending crash, the quadcopter adjusts its flight path to try to go in the opposite direction. The program doesn't work perfectly (it fails about half the time), but we have learned some important lessons:

1) The CrazyFlie flies extremely erratically. Even the slightest movement on the joystick causes the quadcopter to veer wildly.
2) The CrazyFlie's source code is poorly documented. It took us several hours of digging through code to make even the smallest changes.
3) The CrazyFlie is surprisingly durable. We crashed it several times (some of them pretty severe) and they didn't seem to make a dent in it.

The first item is why we want to get the proximity sensors up and running as quickly as possible. Once we have sensor data in all four directions, we can use this to improve the quadcopter's stabilization algorithms by detecting drift and compensating for it.

Although this project will be difficult, we are pretty excited about it. This is basically the ideal computer engineering project because it involves a lot of hardware design, low level programming, and high level programming. If we can get it working properly then it should make for a pretty neat demo.

References

[1] Chelsea Gifford, "Colony Collapse Disorder, The Vanishing Honeybee (Apis Mellifera)," Ph.D. dissertation, University of Colorado at Boulder, 2011.

[2] Leif Andersen, Daniel Blakemore, Jon Parker, "Project Levitate," Dec. 2012, University of Utah Senior Project Report.

[3] *Crazyflie 2.0 hardware specification*, Bitcraze, 2014, http://wiki.bitcraze.se/projects:crazyflie2:hardware:specification.

[4] *CrazyFlie 2.0 Product Description*, Seeed, 2014, http://www.seeedstudio.com/depot/Crazyflie-20-p-2103.html.

[5] *LV MaxSonar EZ Datasheet*, MatBotix Inc., 2015, http://maxbotix.com/documents/LV-MaxSonar-EZ_Datasheet.pdf.

[6] *OV3640 Product Specification*, OmniVision, 2009, http://www.microelectronicos.com/datasheets/OV3640.pdf.

[7] *OV3640 Product Specification*, Arducam, 2015, http://www.arducam.com/camera-modules/3mp-ov364.

[8] P. Cowan. (2015, May) University of utah project deathray. [Online]. Available: http://deathray.patcowan.com