Prototype Demonstration Paper for Workshop on Cognitive Architectures 2020

Towards a General-Purpose Cognitive Drone

Sam Jijina, Adriana Amyette, Ramyad Hadidi, Hyesoon Kim
Georgia Institute of Technology, Atlanta, GA
{sam.jijina, adriana.amyette, rhadidi, hyesoon.kim}@gatech.edu

Introduction

Drones are everywhere today. Drone help everyone from researchers to videographers to emergency services [1,2]. With this high demand for drones, a new market segment has opened up for high efficiency and high performance drones [3]. However, with the present architectures, drones still lack the ability to carry out on-the-fly transitions into different roles. For example, a drone used by geological researchers cannot be used by a police department. This is because those drones are incapable of transitioning out of their use case while still executing a flight mission.

There are vast and different use cases each individual organization has for a drone. This fact has made it almost imperative that a general purpose drone system architecture and accompanying flight stack be developed. The need for a fully autonomous drone capable of decision making and also having the ability to change its own firmware, mid-flight, is crucial. Additionally, the design should be flexible and powerful enough, so that it can be adapted in several use cases. To this end, we present a prototype of an open source drone which is controlled by a Raspberry-Pi-based flight controller, as shown in Figure 1a. Please see our demo here.

Objectives

The objective of our project is to integrate cognitive functions into a drone controlled by a Raspberry Pi with the backing of the Navio2 [4] hat, a general purpose drone controller shown in Figure 1b, to control flight functionality. Using this setup, we program a drone to run advanced waypoint navigation algorithms and autonomously execute certain actions based on the results of the algorithms. The drone is able to automatically choose an algorithm to execute. The drone has a total cost of $500 and has the ability to carry 200 g payloads or additional equipment.

Analysis of Drone Architecture

The drone architecture is split up into four main layers, as illustrated in Figure 2.

Cognitive Functions

The cognitive functions layer consists of the high level and low level APIs which can be used by a developer to write custom code and custom firmware. Then, the custom firmware is converted to a Linux service and run on the Pi in the background while further coding and development occurs with the help of an IDE. To achieve this, we utilized the DroneKit[5] C++ and Python APIs

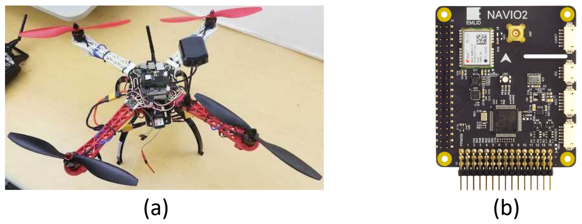

Figure 1: (a) Raspberry-Pi-Based Drone. (b) The Top view of flight controller[4].

which were modified for our use case to allow the drone to be re-configured mid flight.

Modified Linux Kernel

The Linux Kernel is modified to support the RT-Preempt patch which enables the Linux operating system to become suitable for robotics. This also gives the added benefit of being able to completely shut down an instance of a drone mission and spool up a new mission while the drone is executing the current mission. This ability is truly unique and opens up the field of general purpose drones to the mass consumer and industrial markets. The Linux kernel was also modified to support continuous loop-back and server instances so that the drone could be controlled using multiple devices such as through 915 Mhz telemetry or a laptop through SSH.

Flight Controller

The flight controller used for this prototype is the Navio2 open-source controller developed by Emlid. This controller is a HAT for the Raspberry Pi and interfaces with the Pi using the GPIO pins. The Navio2 is preconfigured with GLONASS and GPS abilities and comes equipped with multiple IMUs to support guided navigation. The flight controller is the interface between the Linux kernel and the hardware devices of the drone. The Raspberry Pi sends electrical pulses to the flight controller which is decoded by the controller. The controller then translates the decoded instructions into PWM signals and then outputs signals to the four motors.

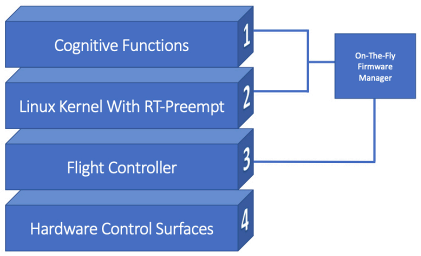
Cognitive Functions
1
Linux Kernel With RT-Preempt
2
Flight Controller
3
Hardware Control Surfaces
4
On-The-Fly Firmware Manager

Figure 2: High Level Drone Architecture

Hardware Control Surfaces

The hardware on the drone mostly consisted of sensors and four motors. The physics of the rotational direction of opposite-pairs of propellers allows the drone to make varied movements in 3D space by only changing the PWM output to the motors. This simple design decreases the processing burden on the Pi and allows us to use the Pi for other activities such as flight control algorithms or high level machine learning algorithms.

Capabilities

The drone is capable of full manual and autonomous flight. We pre-configured the drone with the following flight modes. **Stabilize:** In this configuration, the drone automatically carries out wind correction. However, altitude and heading is controlled manually through the RC transmitter. **Loiter:** In this configuration, the drone is able to maintain its relative position (x,y,z) in 3D space and carries out wind correction. **Guided:** This configuration is the one used for autonomous flying. The drone uses a server running on the localhost loop-back UDP port where we can send commands to the drone flight control stack. The drone, in this mode, monitors all parameters and controls every action.

Our current setup is basic, yet powerful. We are able to configure the drone for waypoint navigation and switch to automatic GPS navigation mid flight without any loss of flight control. This also enables us to switch out the firmware completely, shut down the firmware in case of unexpected errors, and resume manual control all without any loss of altitude. The following steps are executed to perform a firmware switch.
(1) If the compiled firmware is not present, the firmware files are compiled using the waf build file and gcc compiler. **(2)** The compiled firmware is then loaded into the swap memory of the Pi. **(3)** The already-executing firmware is given the command to load a custom boot file into memory and load a mission-hold file. **(4)** The Linux daemon is configured to read the new boot file on reboot. **(5)** The executing firmware is instructed to exit and write log files. **(6)** As soon as the firmware quits, the daemon reloads and restarts. **(7)** While the daemon is restarting, the drone is executing the mission-hold file to maintain heading and altitude. **(8)** As soon as the daemon restarts, the new boot file spools up the new firmware with the aforementioned log files, and a new mission can start executing.

In any stage mentioned above, if any step fails to correctly execute, the drone flies back to a pre-configured location. However, if the flight-control stack faces an error, the entire service and firmware is preempted and manual flying control is handed to the pilot in command. As demonstrated, the drone has multiple redundant systems to ensure safe flight.

Design choices and reasoning

The main reasons for using the aforementioned setup revolve around the basic objectives that our drone is trying to accomplish which is a drone that can be used for multiple purposes and be configured for a different purpose mid flight. The use of Linux in robotics and flight control was for a long time highly debated as Linux did not have an ability to be fast enough in preempting a service on demand and context switching with low overhead as in robotics applications a constant output is needed to ensure stability and correctness of the application. However, with the RT-Preempt patch, we are able to utilize the full functionality of the Linux kernel while also being able to utilize it for controlling the drone's functions and parameters in real time. Our choice of flight controller was due to the fact that the Navio2 is open-source and easily configurable for different applications. The Navio2 also let's the developer directly manipulate any parameter which is accessible by the Pi therefore granting complete access to all control surfaces and systems.

Performance

To analyze performance, we used the Linux perf command and htop tool [6]. We logged how many threads the firmware had instantiated and how much CPU time the processes got. We used this information to further optimize the firmware so that we could have more processing power available for algorithms such as Dijkstra's and Menger's Dual. To achieve this, several other features of the Linux kernel were deactivated and the window manager system was removed to have a command-line interface only. The additional power consumption of the Navio2 HAT on the Pi was found to be negligible and this meant that both the Pi and the Navio2 could be powered from a 3S battery which also powered the drone motors. The performance analysis is still an ongoing part of the project and and additional data to gather before any conclusions become solidified.

Future Work

We aim to continue more research into optimizing the drone flight-stack and Linux kernel to enable more high performance computing with our end goal being to test the drone in a real-life scenario with deep learning workloads [7,8]. We are building a baseline model for a general purpose drone capable of switching between firmware versions and changing missions mid-flight.

References

[1] L. P. Koh and S. A. Wich, "Dawn of drone ecology: low-cost autonomous aerial vehicles for conservation," *Tropical Conservation Science*, 2012.

[2] S. J. Kim and et al, "A survey of drone use for entertainment and avr," in *Augmented Reality and Virtual Reality*, Springer, 2018.

[3] F. Giones and A. Brem, "From toys to tools: The co-evolution of technological and entrepreneurial developments in drone industry," *Business Horizons*, 2017.

[4] Emlid, "Emlid navio2 hat for raspberry pi." emlid.com/navio.

[5] Open-Source, "Dronekit api." dronekit.io.

[6] S. Jijina, A. Amyette, R. Hadidi, and H. Kim, "Understanding the software and hardware stacks of a general-purpose cognitive drone," 2020.

[7] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Towards collaborative inferencing of deep neural networks on internet of things devices," *IEEE Internet of Things Journal*, 2020.

[8] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Distributed perception by collaborative robots," *IEEE Robotics and Automation Letters*, 2018.