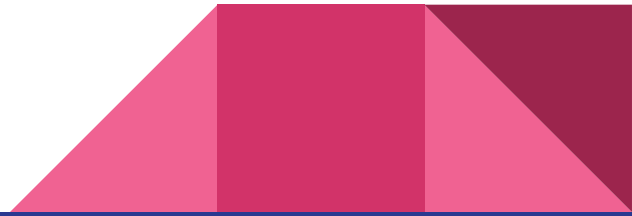
Mixed-Criticality UAV with Quest-V

Zhiyuan Ruan
Craig Einstein
Anam Farrukh
Prof Rich West

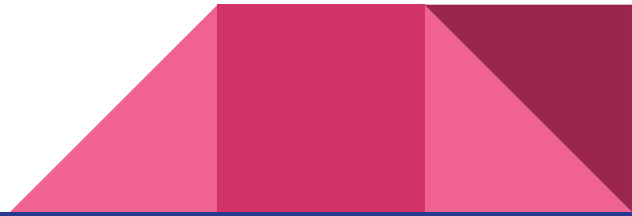
Roadmap

1. Background
2. Project Overview
3. Plan of Work
4. Completed Work
5. Conclusion



Background

- Motivation
  - Drones are becoming increasingly ubiquitous
  - Current paradigm of drone control is not ideal



An ideal paradigm for drone control

- Flexibility
  - Can handle variety of tasks -- from lightweight tasks like sensor queries to more intense tasks like object tracking
  - Easily reconfigurable tasks
- Power Consumption
  - Consume as little power as possible
- Safety
  - Can execute tasks in a timely manner
  - Core functionalities like flight control are hard to compromise

Current Paradigm

- Hardware
  - A flight controller board which connects to a variety sensors and runs autopilot software such as Cleanflight
    - Most commonly used board is built around STM32 -- based on ARM Cortex M-series processors. The SoCs operate below 200 MHz and has RAM at the magnitude of kilobytes.
      - Sufficient to host radio-controlled flight controller.
      - Insufficient for tasks like autonomous flight
  - A companion board is used to complement the basic flight control board
  - Dual board setup example: Intel Ready-to-Fly Drone

Intel Ready-to-Fly Drone

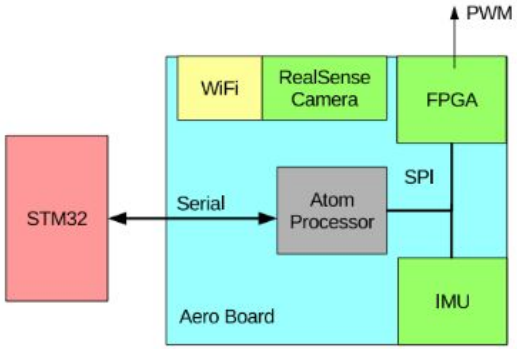


Fig.1 The hardware Architecture of Intel Ready-to-Fly Drone

- Runs PX4 flight controller on an STM32 based control board.
- Runs Linux on Aero board to execute mission computation
- Connect two boards with serial link
- Ease the development process

- Two boards increase the weight of drone and thus need extra force
  - Needs longer rotor blades, bigger drone frame
  - Increases power consumption
- STM32 based board is not designed to work with high-speed buses like USB
  - Serial link is typically configured to operate at 115,200 Kb/s (about 1.4 MB/s)
  - Most recent STM32 can support USB but still not as fast as communication via memory

Current Paradigm

- Software
  - Dual board setup
    - Flight controllers are either implemented as application on RTOS or as bare-metal firmware
    - Issues with dual board discussed
  - Single board setup
    - Execute both flight control and autonomous mission logic on one board
    - This approach typically uses one operating system
      - Linux or an RTOS

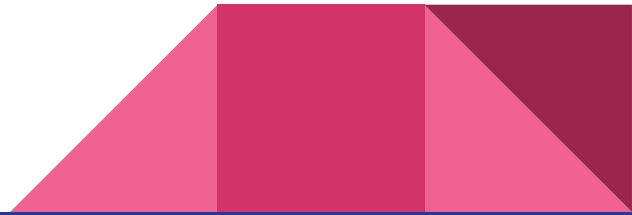
Current Paradigm

Software Issues

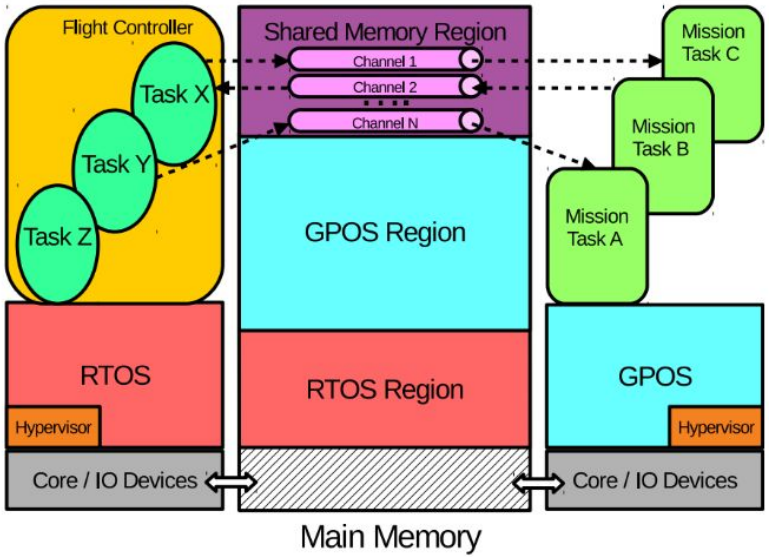
- With Linux
  - Real time issues with heavy I/O intensive tasks
  - Known to be insecure => faulty application can compromise kernel and terminate the flight controller which runs as a user process
- With Real time operating system (RTOS)
  - Lack of existing libraries

Project Overview

UAVs require the computation to be performed under precise time constraints. However, it is hard to develop desirable functionality on a real time system due to a lack of existing libraries. At the same time, Linux is not ideal either. Hence we propose to integrate RTOS and Linux so that the UAV platform will be easy to develop for and will provide real time guarantees.



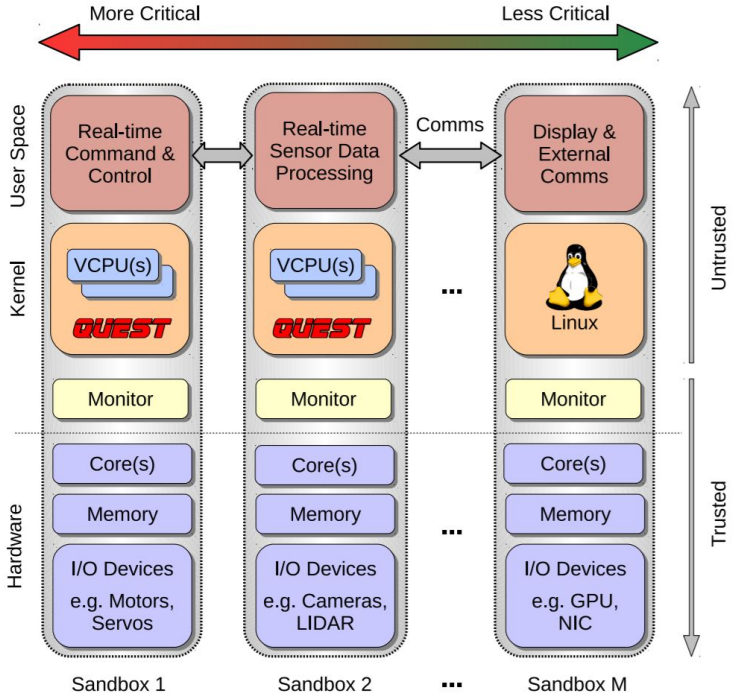
Project Overview



FlyOS Architecture Diagram

- Host Quest and Linux on Aero Board
  - Using Quest-v partitioning Hypervisor
- Mount Aero Board on UAV
- Run the Cleanflight flight controller in Quest
- Run object tracking application in Linux
- Aim: Autonomously track objects

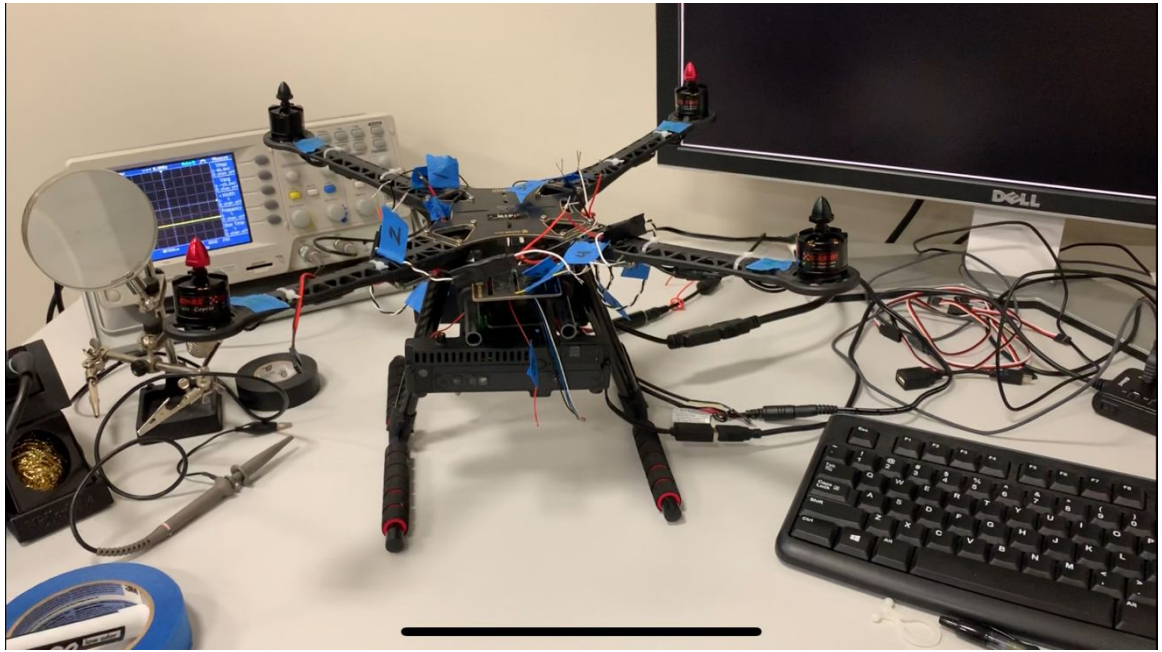
Quest-V



- Quest-V partition hardware to each sandbox at boot time
  - I/O and memory partition
  - Each guest OS has direct hardware access
- Interrupt is delivered directly to each guest OS
- Guest OSes can communicate via memory channel with real time guarantee

Fig. 1 Example Quest-V Architecture Overview. Quest and Linux will be partitioned into their respective sandbox with dedicated hardware resources.

Project Overview

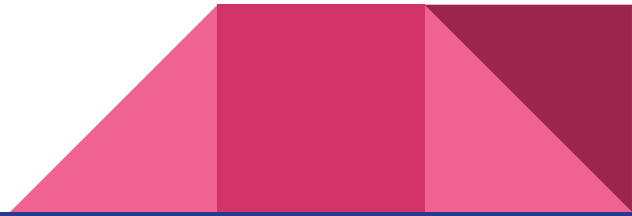


Picture of drone. Aero board is mounted on the drone. A camera will be mounted on the drone to perform object tracking.

Plan of Work

1. Compare our platform’s performance with firmware-based Cleanflight in terms of latency with human in the loop (control with a radio controller)

2. Add autonomous object tracking to the platform



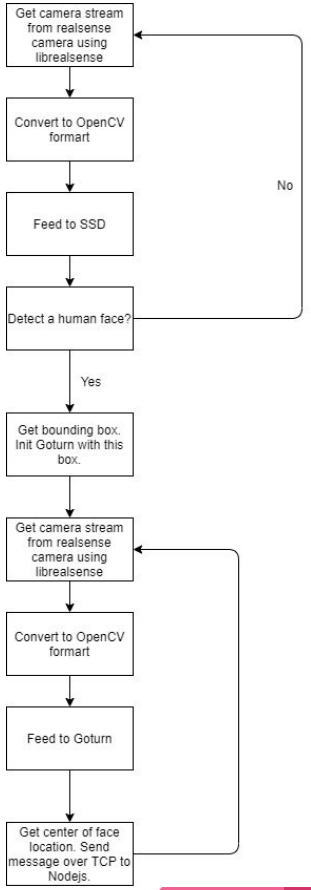
Completed Work

- Object Tracking
  - Implemented with OpenCV
  - Neural Network: Mobilenet with Single Shot Detection, Goturn
- Test of latency on message pass from Linux to Quest
- Setup Cleanflight for autonomous mission

Object Tracking

- Goturn: Achieve 165 fps on Titan X and 100 fps on GTX 680
- Single Shot Detection: 58 fps on Titan X GPU
- Mobilenet

Version	iPhone 7	iPhone X	iPad Pro 10.5
MobileNet V1	118	162	204
MobileNet V2	145	233	220

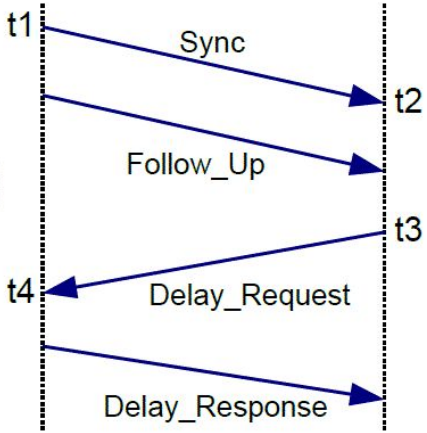


Test of latency setup (Linux to Quest)

- A radio receiver application on Linux side send motor commands to Cleanflight
- Cleanflight on Quest receive commands and send to ESC
- Measure time between the moment Linux receive the radio message, to the moment message is sent to ESC
- Precision Time Protocol is used to synchronize time

IEEE Precision Time Protocol

Master clock



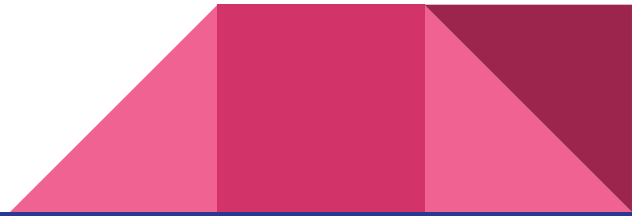
Slave clock

There is a constant time offset O between master and slave. Denote transmission delay as D.

- T2 - T1 = D + O
- T4 - T3 = D - O
- O = (T2-T1+T3-T4)/2

Results

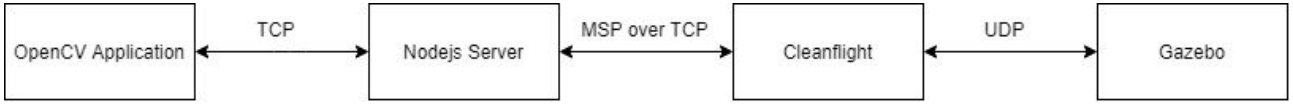
- Mean: 94155.31579 ns
- Standard Deviation: 10279.56297 ns
- Firmware based Cleanflight end-to-end latency: 2x10^6 ns
- Clock frequency on Aero board is in Ghz while STM32 is in Mhz

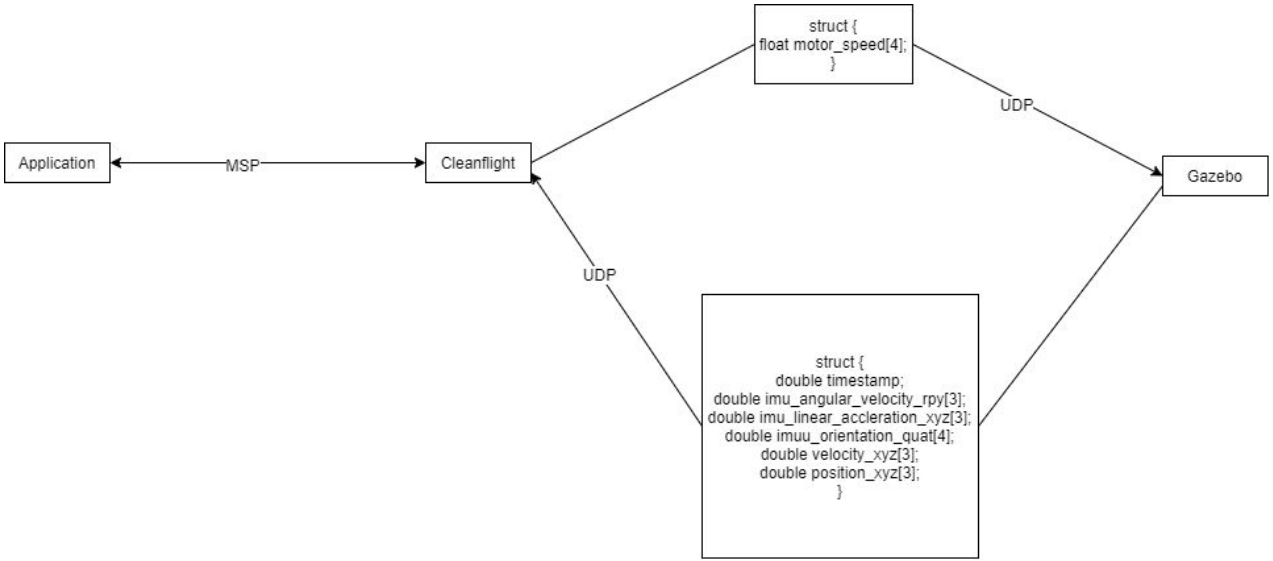


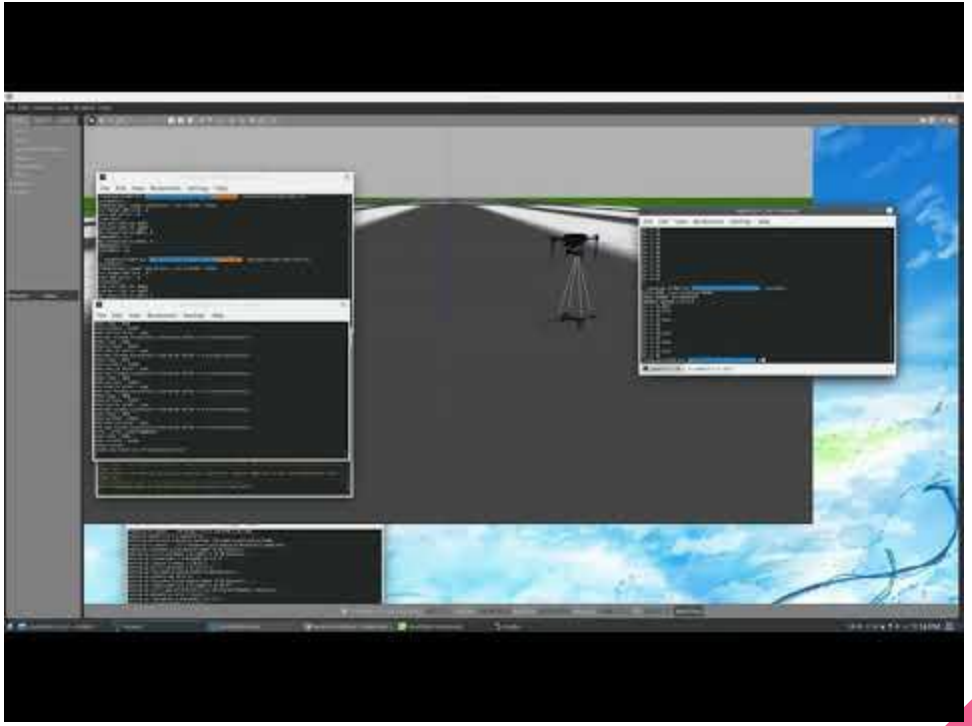
Cleanflight autonomous mission setup

- Vanilla version Cleanflight receive commands sent by a human via radio signal with SBUS protocol (one-way)
- To realize autonomous mission, need to setup two-way channel
  - MSP (Multiwii Serial Protocol)
    - Can send request, commands or receive messages from flight controller
- To test if setup work, run simulation in Gazebo on Linux

Setup

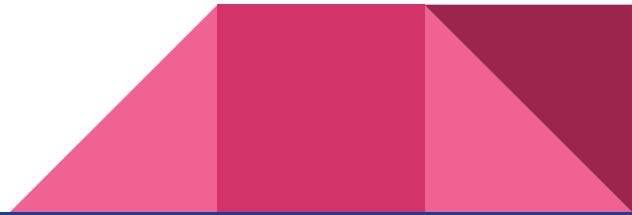

OpenCV Application
TCP
Nodejs Server
MSP over TCP
Cleanflight
UDP
Gazebo


struct {
float motor_speed[4];
}
UDP
Application
MSP
Cleanflight
Gazebo
UDP
struct {
double timestamp;
double imu_angular_velocity_rpy[3];
double imu_linear_accleration_xyz[3];
double imuu_orientation_quat[4];
double velocity_xyz[3];
double position_xyz[3];
}



Conclusion

FlyOS partition hardware resources of the Intel Aero board to host both a real-time system and a general-purpose system. Those two systems are spatially isolated while can communicate through shared memory channel. Through our experiments, it is shown that FlyOS can achieve the same or even better performance than that of an STM32 based flight controller in terms of latency while providing capabilities for autonomous mission tasks like object tracking.



Reference

Richard West, Ye Li, Eric Missimer and Matthew Danish, "A Virtualized Separation Kernel for Mixed Criticality Systems", in ACM Transactions on Computer Systems, Volume 34, Issue 3, Article 8, June 2016 (DOI: 10.1145/2935748)

Zhuoqun Cheng, Richard West and Craig Einstein, "End-to-end Analysis and Design of a Drone Flight Controller", in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 37 Issue 11, pp. 1-12, November 2018, DOI: 10.1109/TCAD.2018.2857399

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”

https://github.com/davheld/GOTURN#evaluate-the-tracker

https://github.com/szaza/android-yolo-v2