The JAviator Quadrotor

An Aerial Software Testbed

Rainer Trummer
Department of Computer Sciences
University of Salzburg
Austria

Computational Systems Group

Introduction

- The JAviator Project
- The JAviator Quadrotor
- Airframe Construction
- Avionics Components
- Computer System
- Quadrotor Dynamics
- Control System Design
- Control System Performance
- Software Architecture
- Conclusions

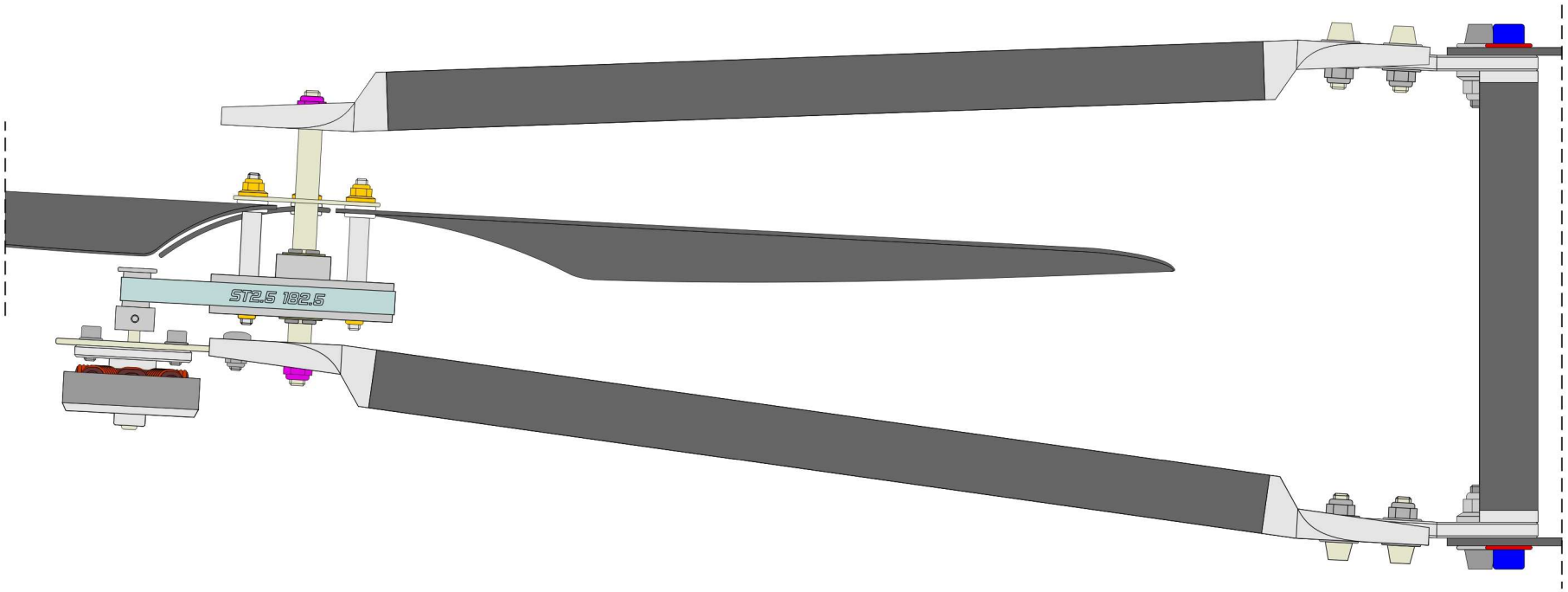
The JAviator Project

- Project goals:
  - Develop high-payload quadrotor model helicopters
  - Develop high-level real-time programming abstractions
  - Verify solutions on JAviator (Java Aviator) helicopters
- Real-time programming in Java:
  - Write-once-run-anywhere also for real time (time portability)
  - Exotasks vs. Java threads (collaboration with IBM Research)
- Real-time programming in C:
  - Time-portable software processes (CPU, I/O, Memory)
  - Real-time operating system Tiptoe: tiptoe.cs.uni-salzburg.at

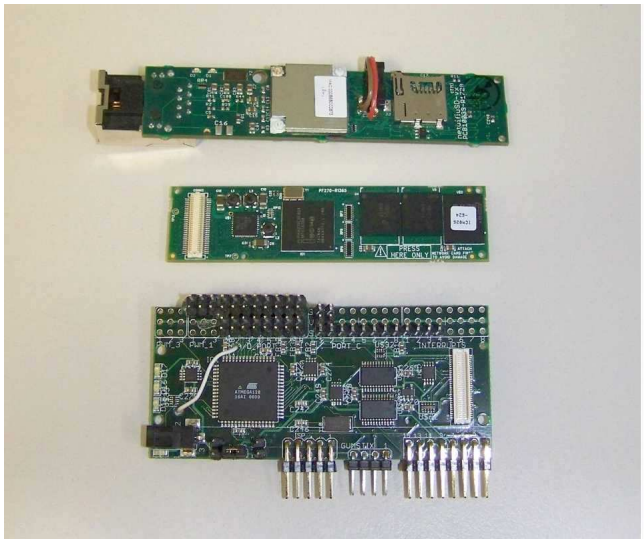
The JAviator Quadrotor

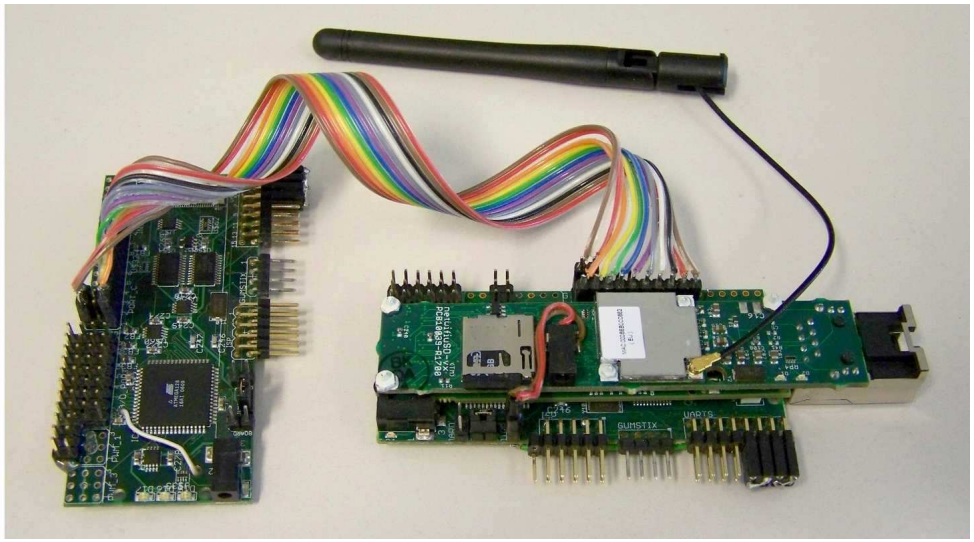
- Since February 2007: **JAviator V2**
  - CNC-fabricated, flow-jet-, and laser-cut components
  - Total diameter (over spinning rotors): 1.3 m
  - Curb weight (including all electronics): 2.2 kg

Airframe Construction

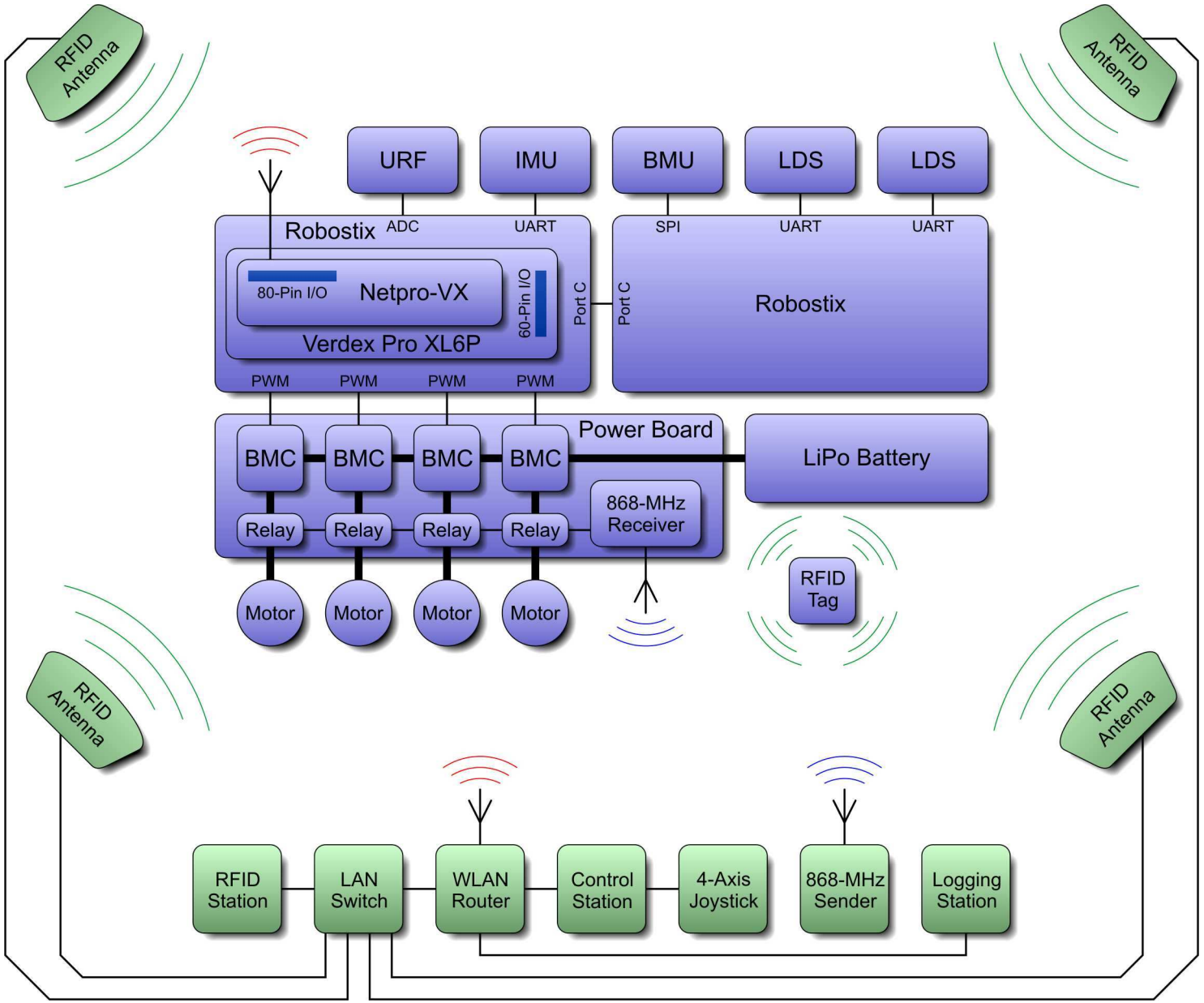


Avionics Components

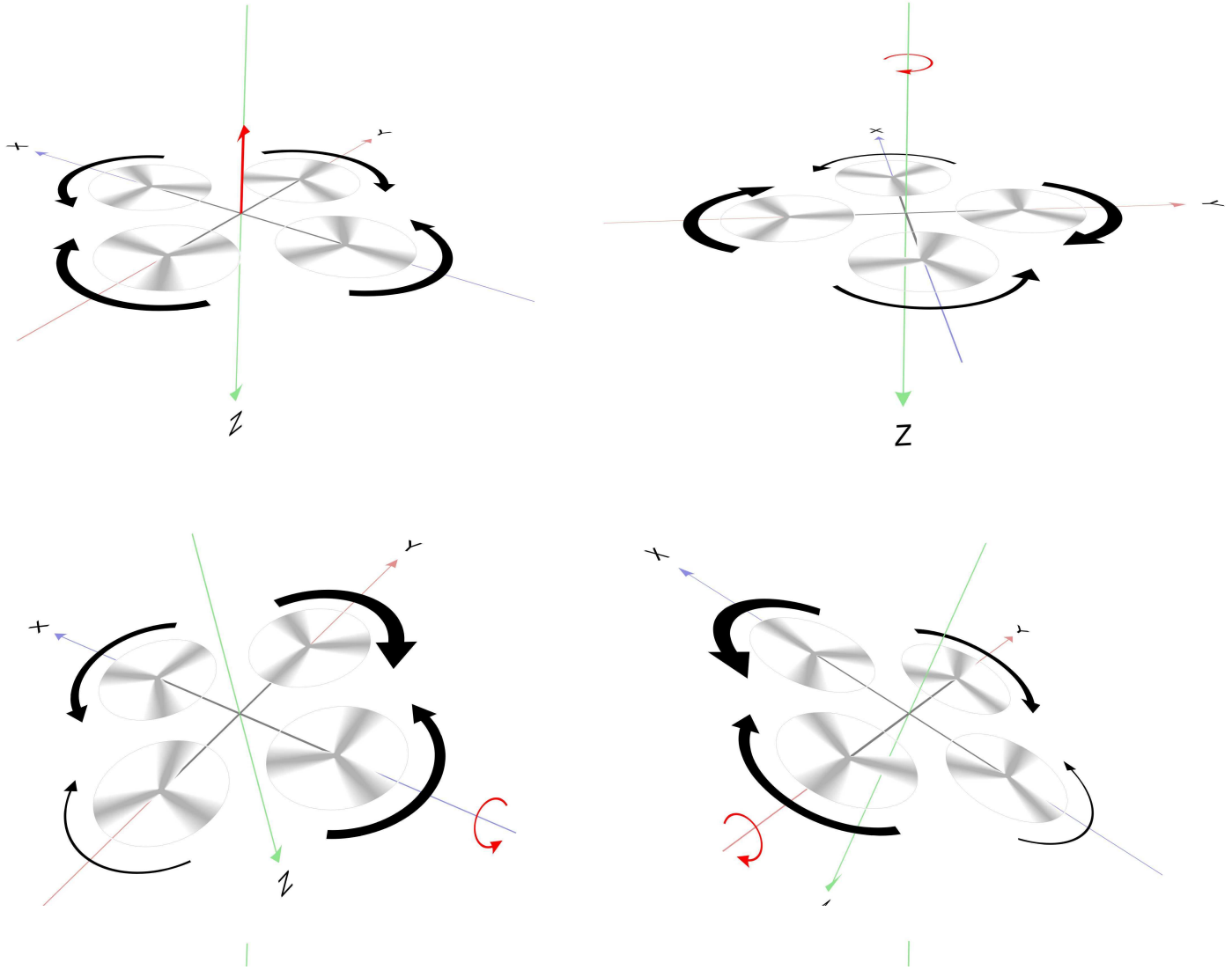




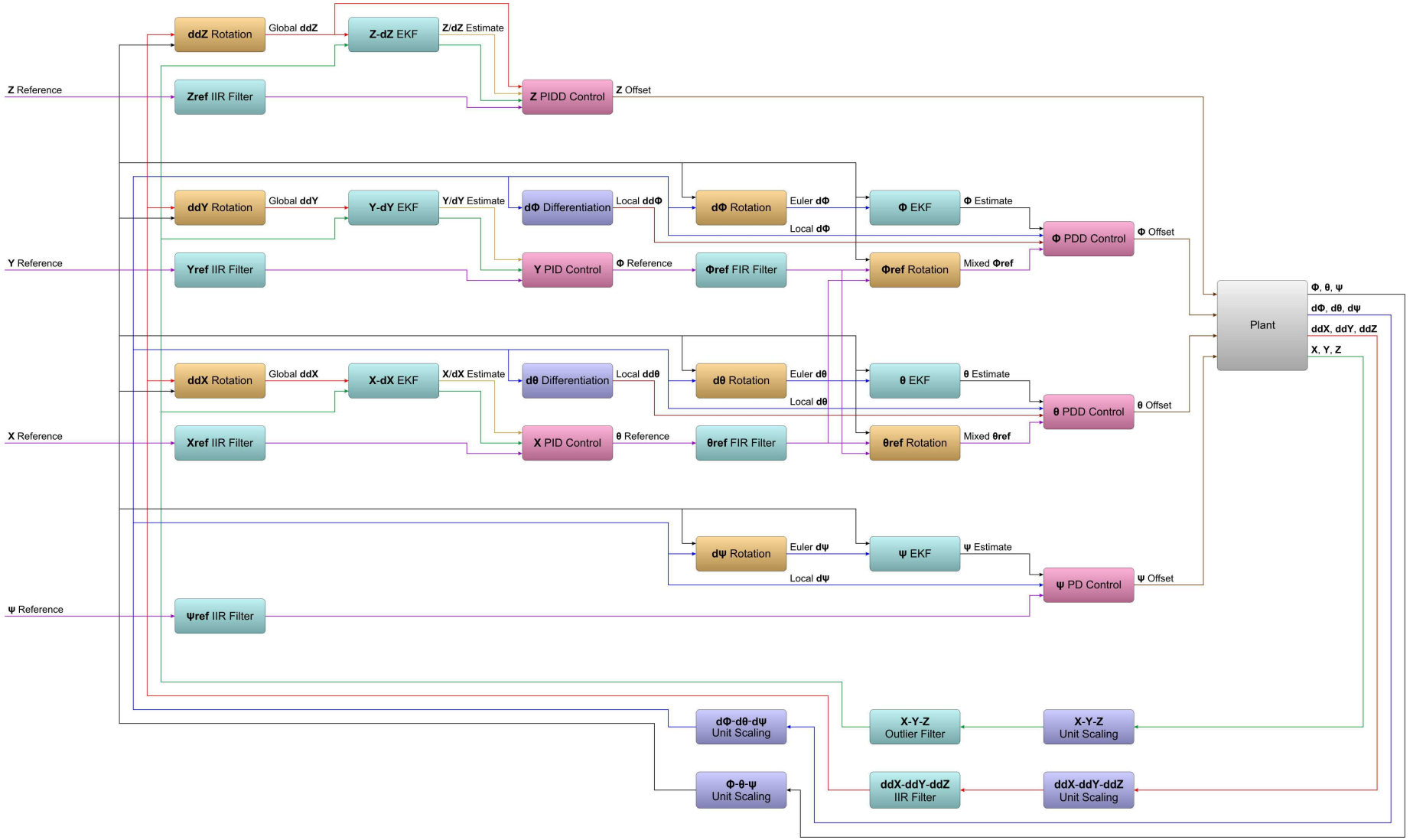
Computer System



Quadrotor Dynamics



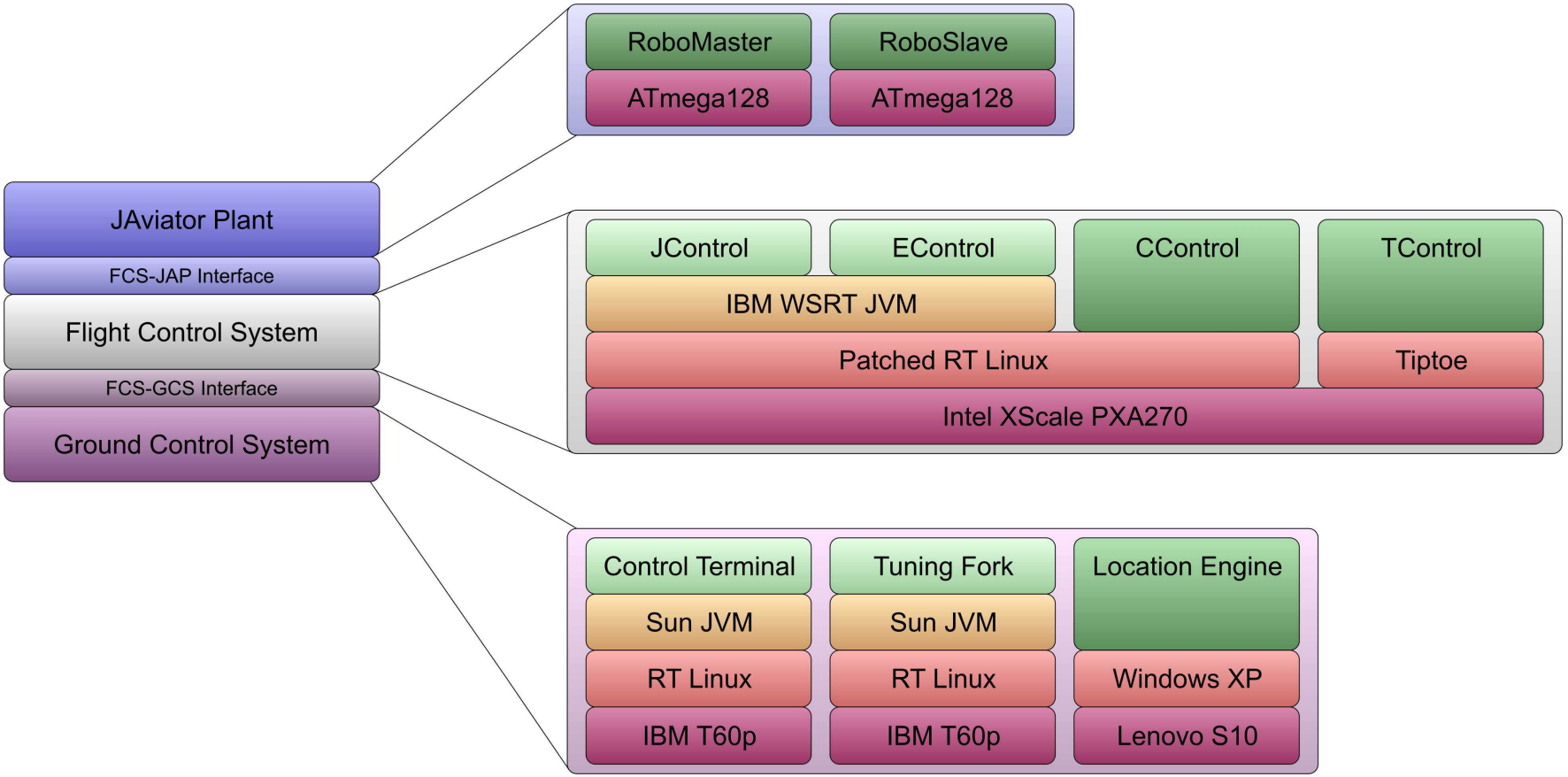
Control System Design



Control System Performance

- Initial Status
  - Many problems with automatic altitude control
  - Very unsatisfying attitude stability and response
- Current Status
  - Excellent stability with extended Kalman filters
  - Perfectly tuned and working control system
- Position Control
  - RFID accuracy varies from 20 cm to > 50 cm
  - Advanced Kalman filters to improve position hold
- Robustness
  - Very fault tolerant in regard to timing issues
  - Highly sensitive to lost or discarded sensor data

Software Architecture



Conclusions

- Hardware
  - Helicopter development was least time-consuming
  - Custom-built hardware increased production costs
  - Unique platform with high demonstrative impact
- Software
  - No way around embedded programming and writing individual low-level driver software
  - Great amount of time was spent solving pure control engineering problems
  - Complexity increased rapidly and raised interesting computer science challenges

Thank You!

Questions?