Platform-Based Embedded Software Design and System Integration for Autonomous Vehicles

EE 290-O, UC Berkeley

Presentation by: Darren Liccardo and Mark McKelvin

April 11, 2002

References/Acknowledgements

Platform-Based Embedded Software Design and System Integration for Autonomous Vehicles, Benjamin Horowitz, Judith Liebman, Cedric Ma, T. John Koo, Alberto Sangiovanni-Vincentelli, Shankar Sastry

Part I: Outline

- ✓ Overview
- ✓ A Helicopter Based UAV Example
  - Background for a model helicopter
  - Analyze the current Flight Control System
  - New Generation Flight Control System
- ✓ Platform-Based Design Methods
  - Synchronous control

Overview

What is the problem?

- Automation control systems incorporate legacy code and components designed to operate independently (e.g. lacks re-usability)
- These systems operate under strict timing and safety constraints
- Current design strategies ignore or grossly estimate implementation constraints when designing control laws
- Missed timing constraints and subtle transient errors cause costly re-designs in the system

Overview

What is the proposed solution?

- Develop a methodology based on the concept of "platform-based" design
  - Definition: a layer of abstraction that hides unnecessary information about the layers below
  - Build in modularity
  - Make code re-usable and substitution of new subsystems simple
- Guarantee performance
  - Use a time-based controller
  - Using Giotto software platform versus other approaches

System Characteristics

The proposed methodology works to integrate systems…

- …that contain a sizable amount of real-time embedded software
- …that integrate subsystems originally designed to work independently of one another (i.e., sensors from various vendors).
- …that must operate properly for human safety
- …that often re-use existing code in the form of drivers or controllers
- Example: a helicopter based UAV

A Helicopter Based UAV

Why a helicopter?

- A helicopter is a dynamically complex machine. One needs to combine sensors (GPS and INS), servo actuators, a wireless network, a central computer, and control laws describing the dynamics of the helicopter.
- An autonomous helicopter requires a complicated hybrid controller to control changes in flight modes and to sustain system stability.

Background for a Model Helicopter

Autonomous flight is difficult because:

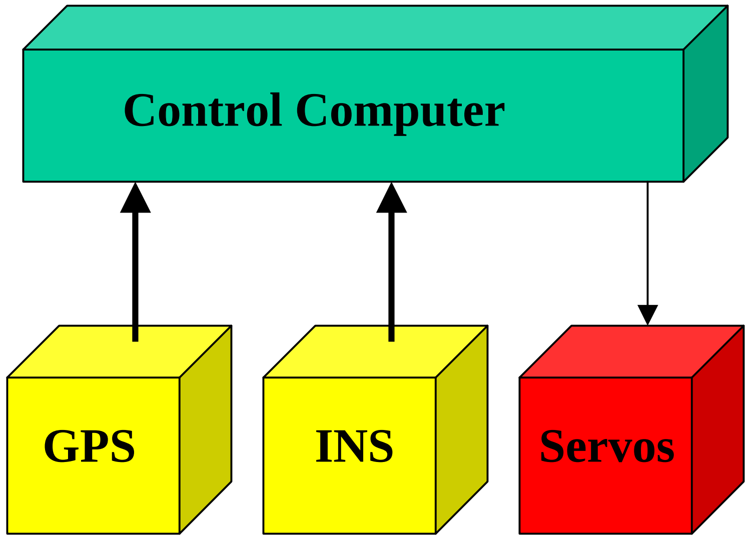
- The helicopter is unstable during hover
- Crashes are dangerous (even at low speeds)
- Electronic and mechanical systems must operate harmoniously under harsh conditions
- Difficulty in obtaining an accurate dynamic model
  - The controls are coupled
  - Behaviors of a helicopter are different in various flight modes

The Berkeley Aerial Robot (BEAR) Fleet

- Kyosho Concept 60
- Yamaha R-50
- Yamaha R-Max

Current Flight Control System (Components)

- Actuators
  - consists of servo-motors to control helicopter dynamics (main rotor and tail rotor pitches)
- Sensors
  - Inertial Navigation System (INS)
  - Global Positioning System (GPS)
- Control computer



Current Flight Control System
(Helicopter Dynamics)

P´(t) = V(t)

V´(t) = (1/m) R(✈(t)) f(u(t))

✈´(t) = ✡(✈(t)) ◆(t)

◆´(t) = I^{-1} [⧫(u(t)) - ◆(t) x ◆(t)]

where the linear position and velocity are given by P(t) and V(t) respectively. Other parameters: m is the body mass; ◆ is the angular velocity; I is the inertial matrix, I ♏ ♑$^{3\,x3}$; Euler angles: ✈ = [♐,❑,⌂]^T ; input vector, u = [❑$_M$ ❑$_T$, B, A]^T (main rotor collective pitch, tail rotor collective pitch, longitudinal cyclic pitch, lateral cyclic pitch); ✡ : ♑^{3} ✹ ♑$^{3\,x3}$ maps the body rotational velocity to Euler angle velocity; and x = [P^T V^T ✈^{T} ◆^{T}]^T is the state vector.

Current Flight Control System
(Helicopter Dynamics)

- For control design, experimental system identification was used to obtain the dynamic model of the helicopter
- A specific set of output tracking controllers were designed
  - Each with static feedback: $u(t) = K_i\,(x(t), r(t))$, where $u(t)$ is associated with an output $y_i(t) = h_i(x(t))$ such that $y_i(t)$ shall track $r_i(t)$ where $y_i, r_i \in \Re^4$, $h_i : \Re^{12} \rightarrow \Re^4$, $k_i : \Re^{12} \times \Re^4 \rightarrow \Re^4$ for each $i \in \{1, \ldots, N\}$ and N is the total number of output tracking controllers
  - Hence, appropriate switching between the controllers allows high level tasks such as way-point navigation and high-altitude are accomplished

R-50 Hovering

GPS Card

GPS Antenna

Current Flight Control System (Sensors Overview)

- Goal: basic autonomous flight
  - Need: UAV with allowable payload
  - Need: combination of GPS and Inertial Navigation System (INS)
- GPS (senses using triangulation)
  - Outputs *accurate* position data
  - Available at *low rate* (5 Hz)
- INS (senses using accelerometer and rotation sensor)
  - Outputs estimated position with *unbounded drift* over time
  - Available at *high rate* (100 Hz)
- Fusion of GPS & INS provides needed high rate and accuracy

INS

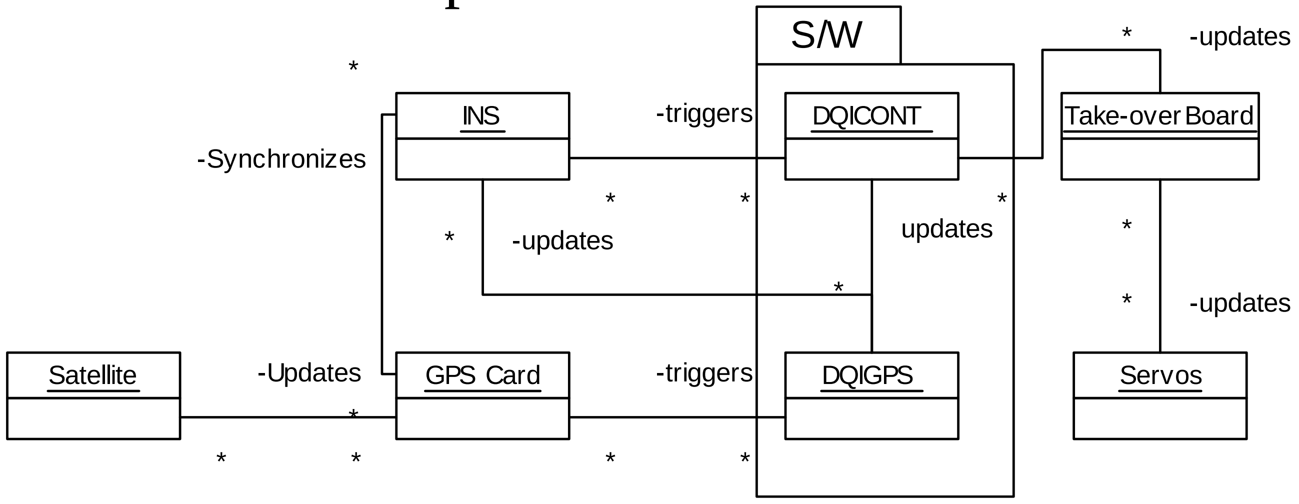
Two Concurrent Processes

DQIGPS

- correct INS drift w/GPS
- Slow (5Hz)
  - 1Hz for INS update

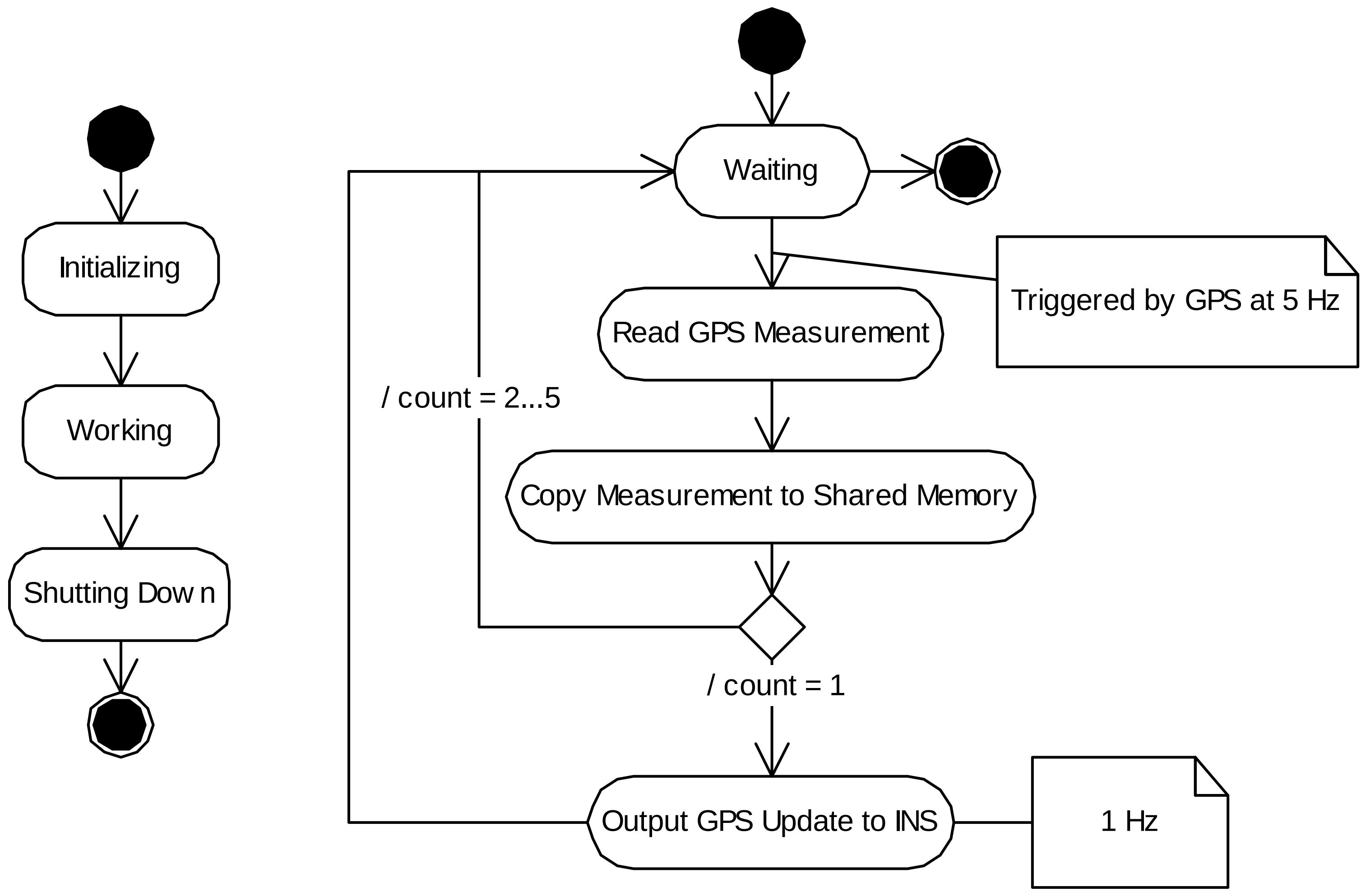
DQICONT

- Main Control Loop
- Fast (100Hz)
  - 50 Hz servo
  - 25 Hz network



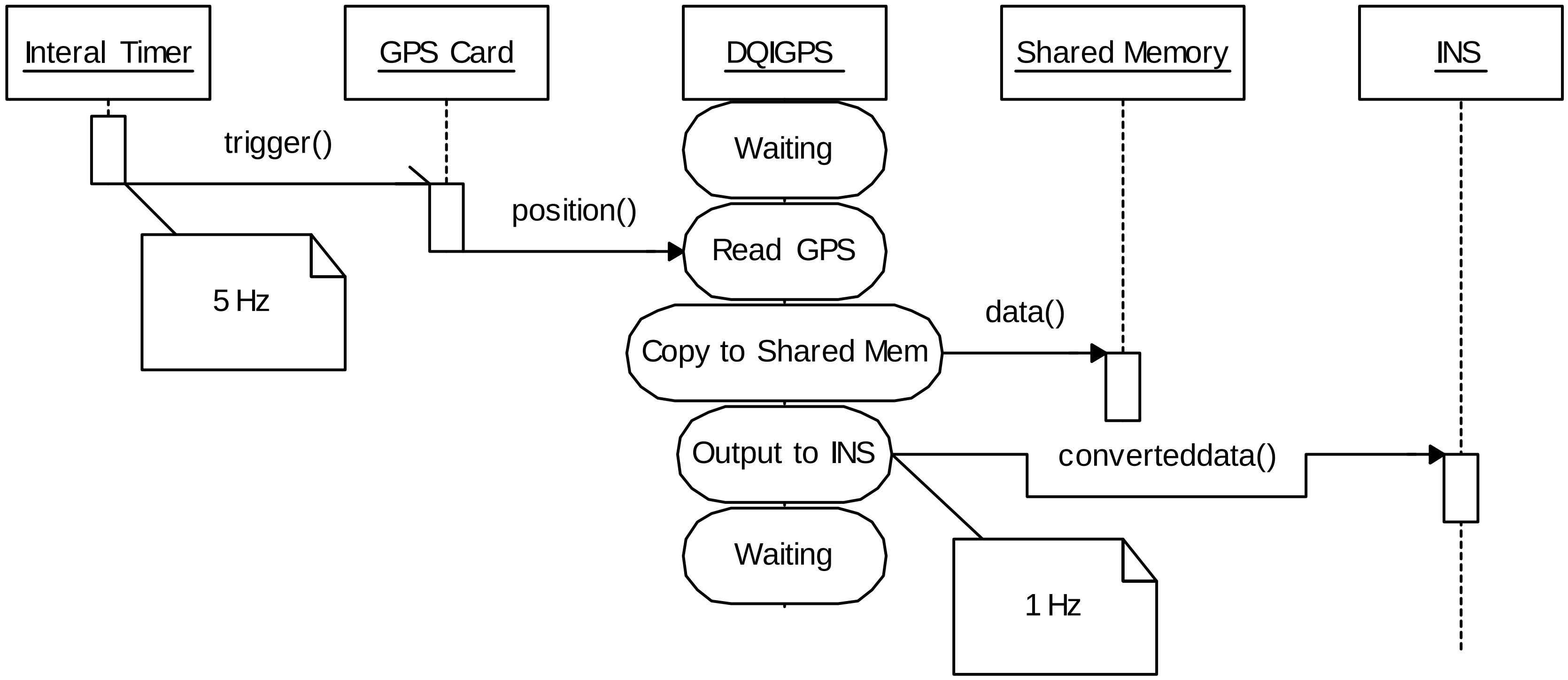
Process DQIGPS

(UML State & Activity Diagram)



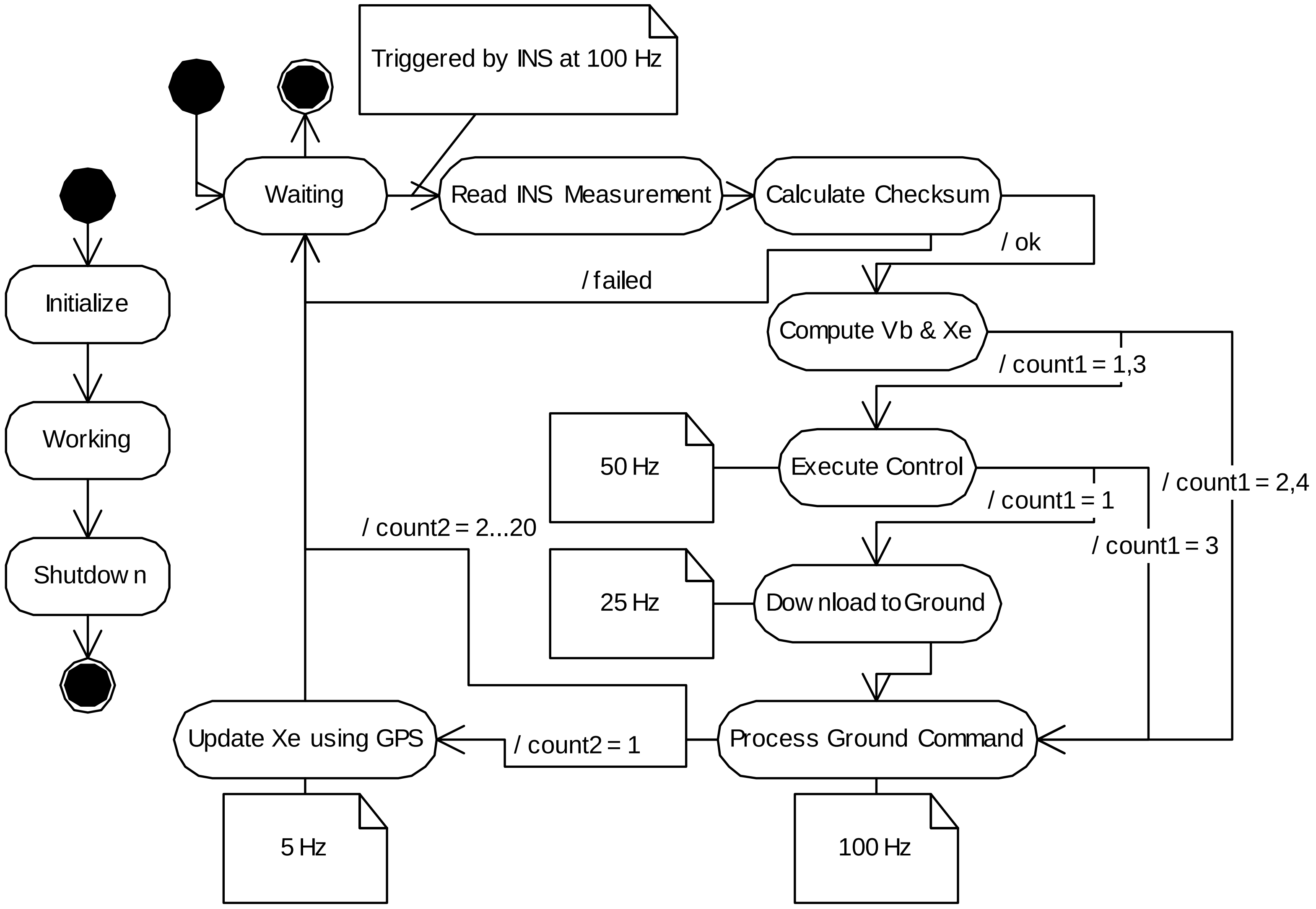
Process DQIGPS

(UML Sequence Diagram)



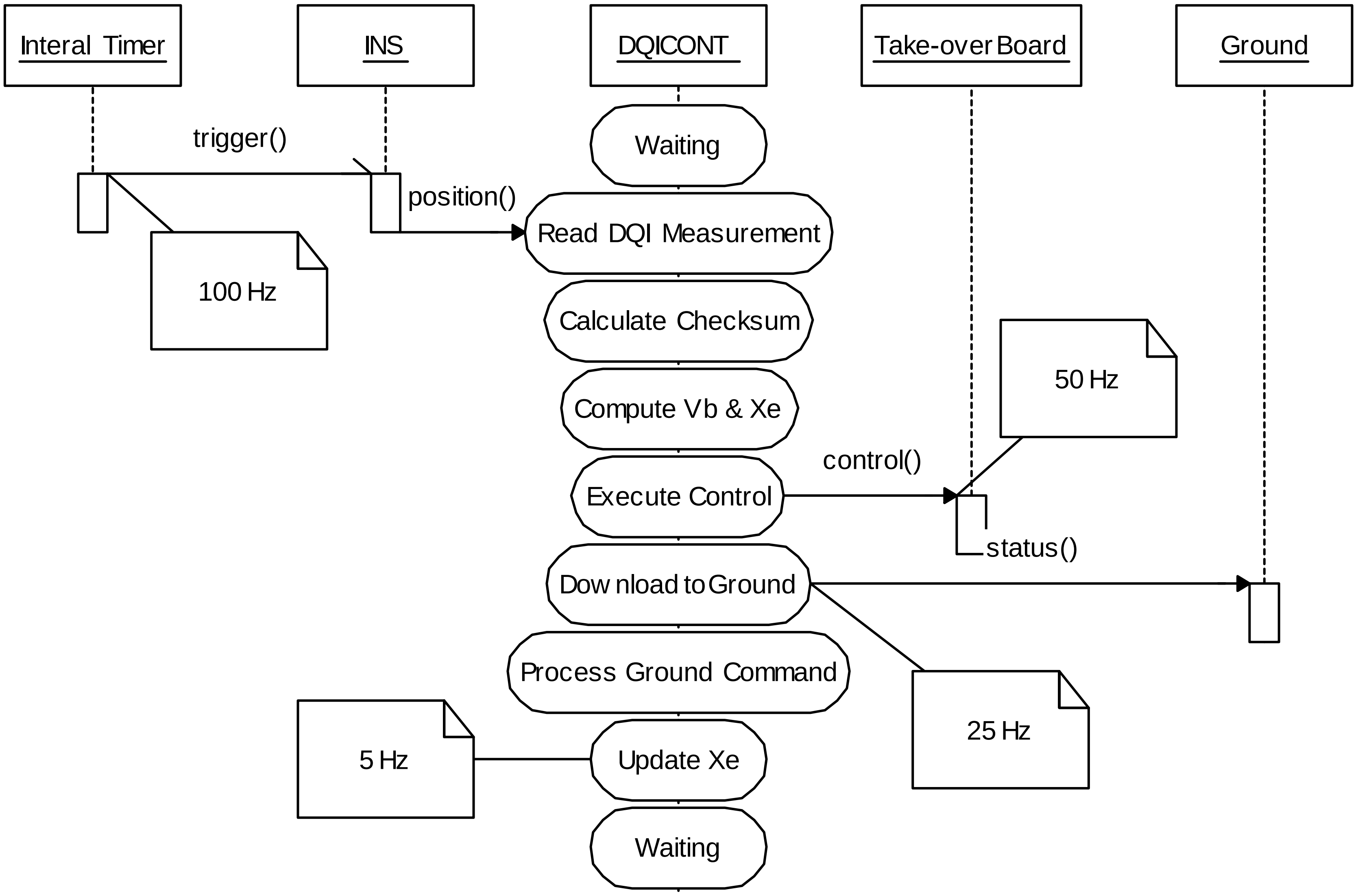
Process DQICONT

(State & Activity Diagram)



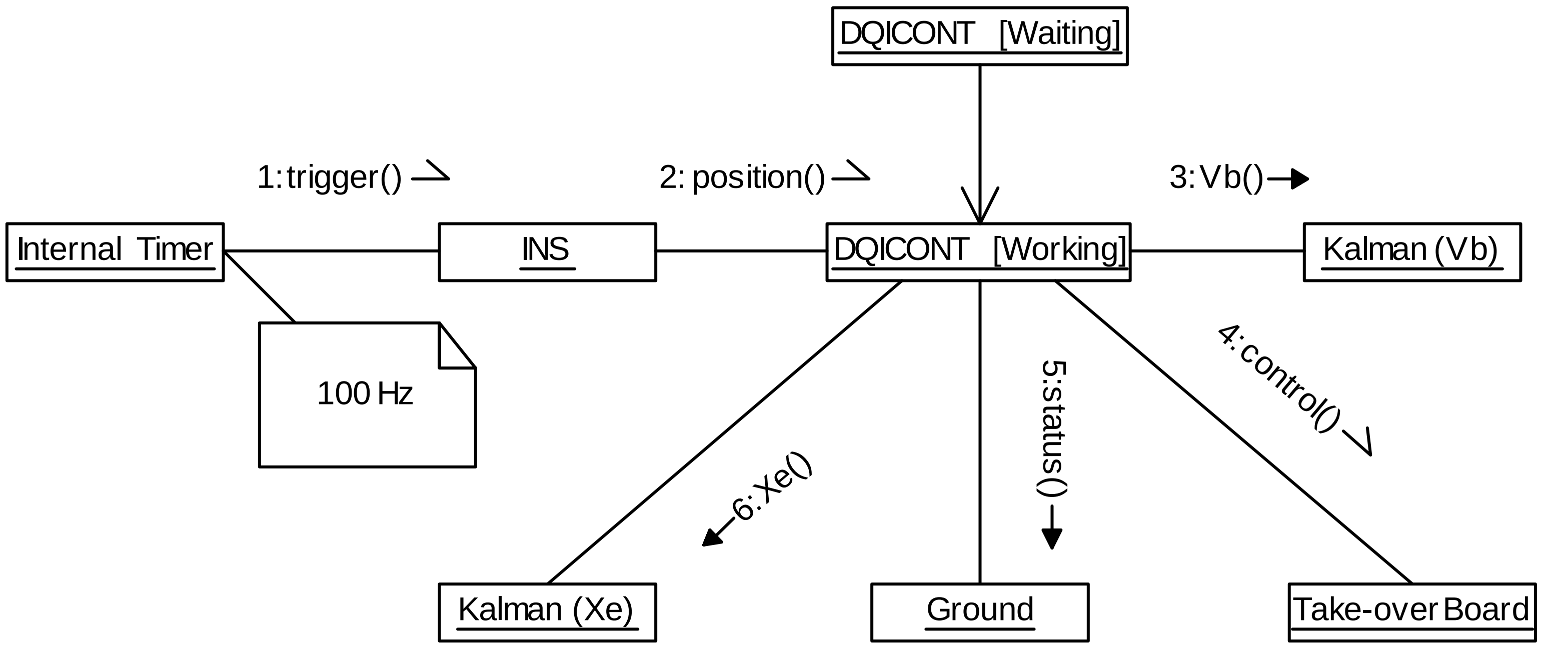
Process DQICONT

(UML Sequence Diagram)



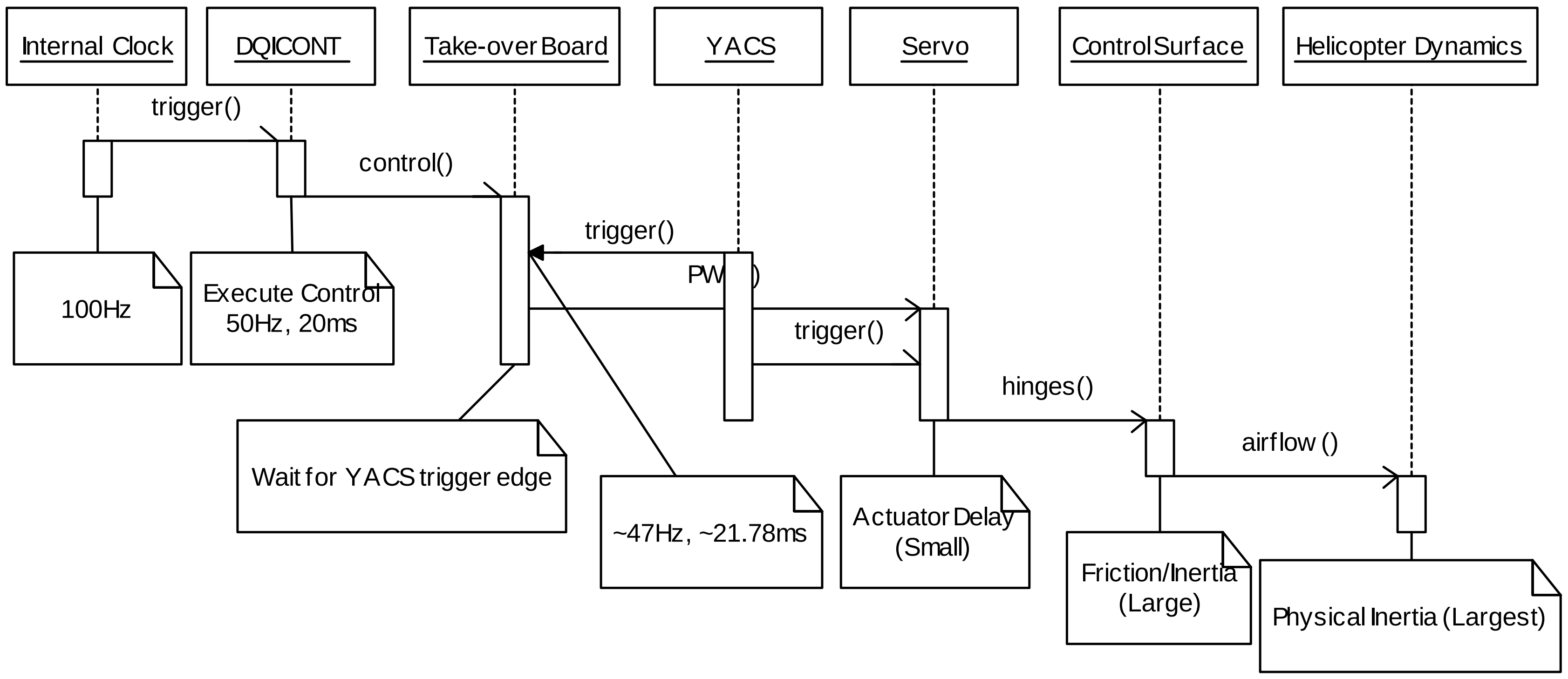
Process DQICONT

(Collaboration Diagram)



Who’s at the Controls?

(Sequence Diagram)

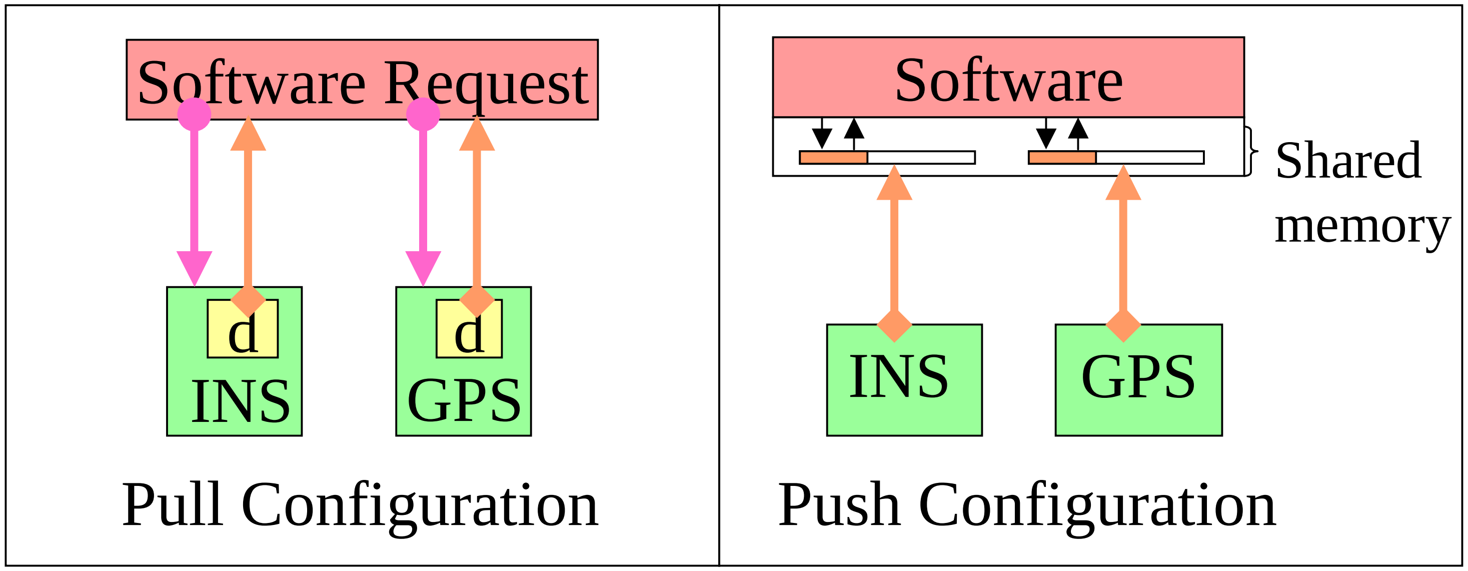


Who’s at the Controls?

- Servo has different sampling period (21.78ms) than INS (20ms)
  - phase difference constantly changing
- Delay from DQICONT control calculation to PWM generation varies
  - jitters by up to 20ms
  - Problem: difficult to analyze

Current Flight Control System
(Sensor Configurations Example)

- Sensors may *differ* in:
  - Data formats, initialization schemes (usually requiring some bit level coding), rates, accuracies, data communication schemes, and even data types
- Differing communication schemes requires the most custom written code per sensor



Current Flight Control System
(Limitations)

- Diverse assortment of devices
  - Each new device communicates differently (asynchronously)
  - Lacks modularity
- Event-based nature
  - Sensors are set to “push” data
  - Incoming data is processed and sends the control output to the actuators immediately
  - Actuation does not occur synchronously, thus the system tolerates a substantial amount of jitter
  - Non-deterministic timing behavior

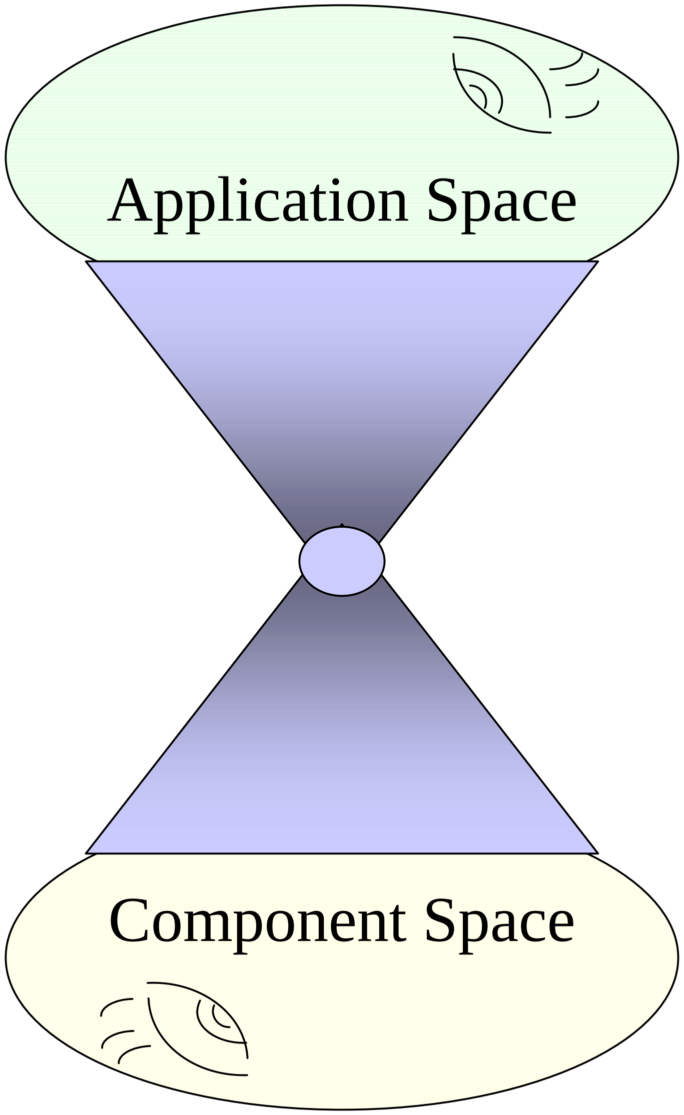
Next Generation Flight Control System

- Time-based design (Giotto)
  - Allow easy analysis of its closed loop behavior
  - Maintain compatibility with existing devices that are not time-based, such as sensors
  - Creates a defined boundary between a system’s synchronous and asynchronous elements
- Modular design (platform-based design)
  - Ability to allow designer to choose between a mixture of devices
  - Must allow a configuration of the same software to run on different helicopters which may have different physical dynamics and devices

Synchronous Control

- Advantages of **time-triggered framework**:
  - Allows for *composability* and *validation*
    - These are important properties for safety critical systems like the UAV controller
  - Timing guarantees ensure *no jitter*
- Disadvantages:
  - *Bounded delay* is introduced
    - Stale data will be used by the controller
  - Implementation and system integration become more difficult
- Platform design allows for time-triggered framework for the UAV controller
  - Use Giotto as a middleware to ease implementation:
    - provides real-time guarantees for control blocks
    - handles all processing resources
    - Handles all I/O procedures

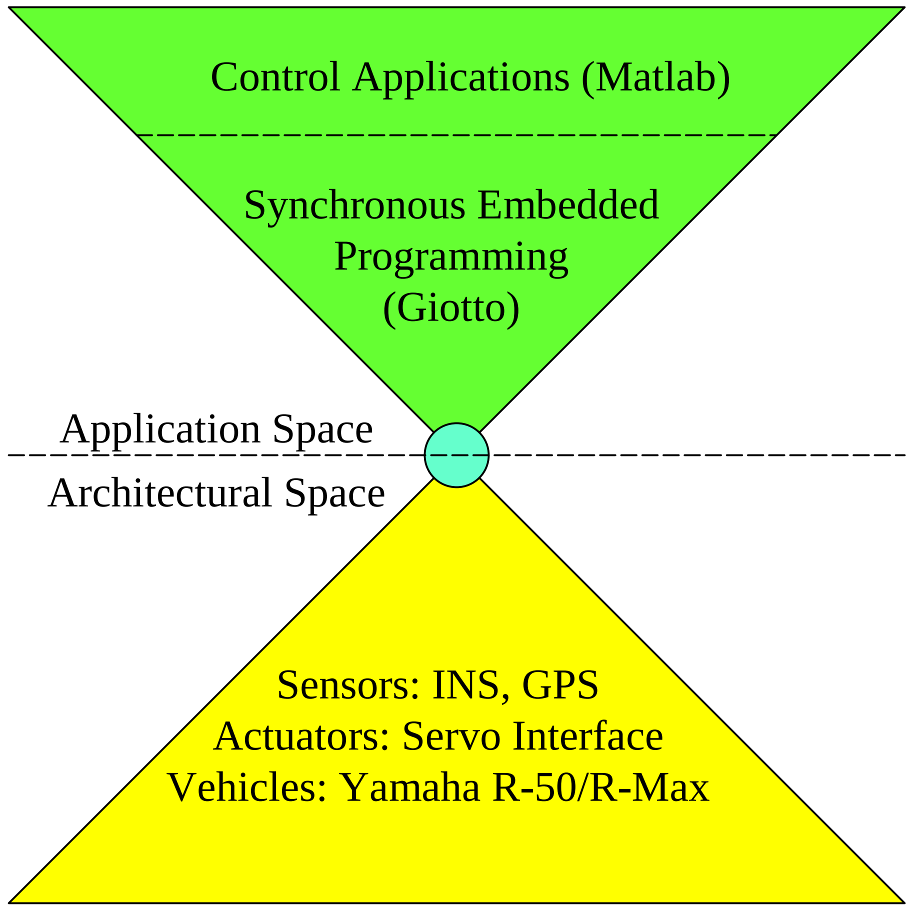
Platform-Based Design Overview



- Universal design strategy
  - Goal is *design reuse*
- Decouple two design views
  - Upper View: Application Space
  - Lower View: Component Space
  - Main motivation of project is this decoupling of the control process from the sensors & devices
- Interact through well-defined interface
  - Platform instance is an implementation of the interface
- Both views help specify the platform making this a *meet-in-the-middle* approach

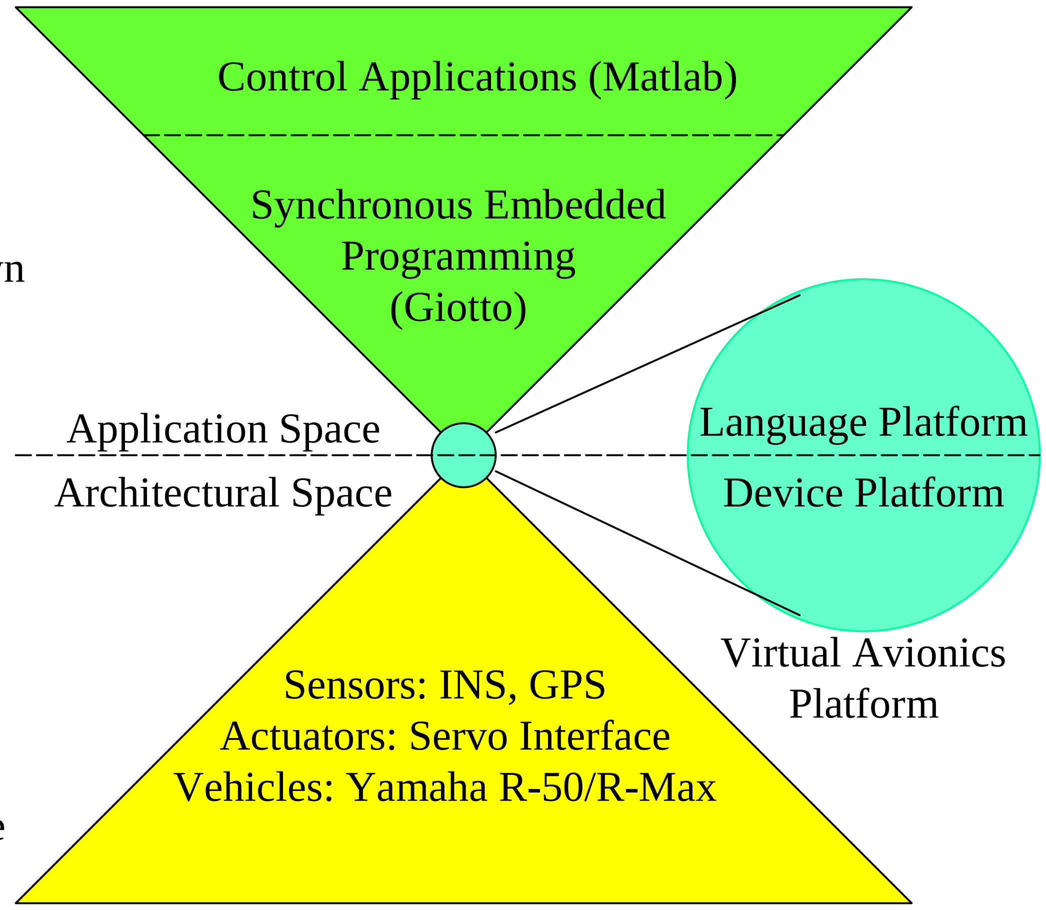
Platform Based Design for UAVs

- Goal
  - Abstract details of sensors, actuators, and vehicle hardware from control applications
- How?
  - Synchronous Embedded Programming Language (i.e. Giotto)
  - Platform


Control Applications (Matlab)
Synchronous Embedded Programming (Giotto)
Application Space
Architectural Space
Sensors: INS, GPS
Actuators: Servo Interface
Vehicles: Yamaha R-50/R-Max

Platform Based Design for UAVs

- Device Platform
  - <u>Isolates</u> details of sensor/actuators from embedded control programs
  - <u>Communicates</u> with each sensor/actuator according to its own data format, context, and timing requirements
  - <u>Presents</u> an API to embedded control programs for accessing sensors/actuators
- Language Platform
  - <u>Provides</u> an environment in which synchronous control programs can be scheduled and run
  - <u>Assumes</u> the use of generic data formats for sensors/actuators made possible by the Device Platform


Control Applications (Matlab)
Synchronous Embedded Programming (Giotto)
Application Space
Architectural Space
Language Platform
Device Platform
Virtual Avionics Platform
Sensors: INS, GPS
Actuators: Servo Interface
Vehicles: Yamaha R-50/R-Max

Recent Developments

- Kyosho Concept 60
  - Hovering ID Model
  - Autonomous Hover
- Yamaha R-50:
  - Hovering ID Model
  - Autonomous Hover
  - Waypoint Navigation
- Yamaha R-Max
  - ?

Outline: part II

- Time-Based Control Platform
  - Modern Control Architectures
  - Platform Based Design with Giotto
- Case Study: BEAR Helicopter
  - Synchronous Control
  - Helicopter Platform

Modern Control Architectures

- Modern control architectures
  - Programmable components
    - μProcessors, DSP
  - Memory
    - FLASH, RAM, ROM
  - Sensors and Actuators
- Control laws implemented in software
- Unique difficulties with this mapping

Difficulties Mapping Software Control to Programmable Architectures

- Real-time
  - Software is slower than hardware
  - True concurrency is lost with single processor
  - Efficient dynamic scheduling algorithms are unverifiable
- Sensor and Actuator Characteristics
  - Must be accounted for in software
  - Must be abstracted for software portability

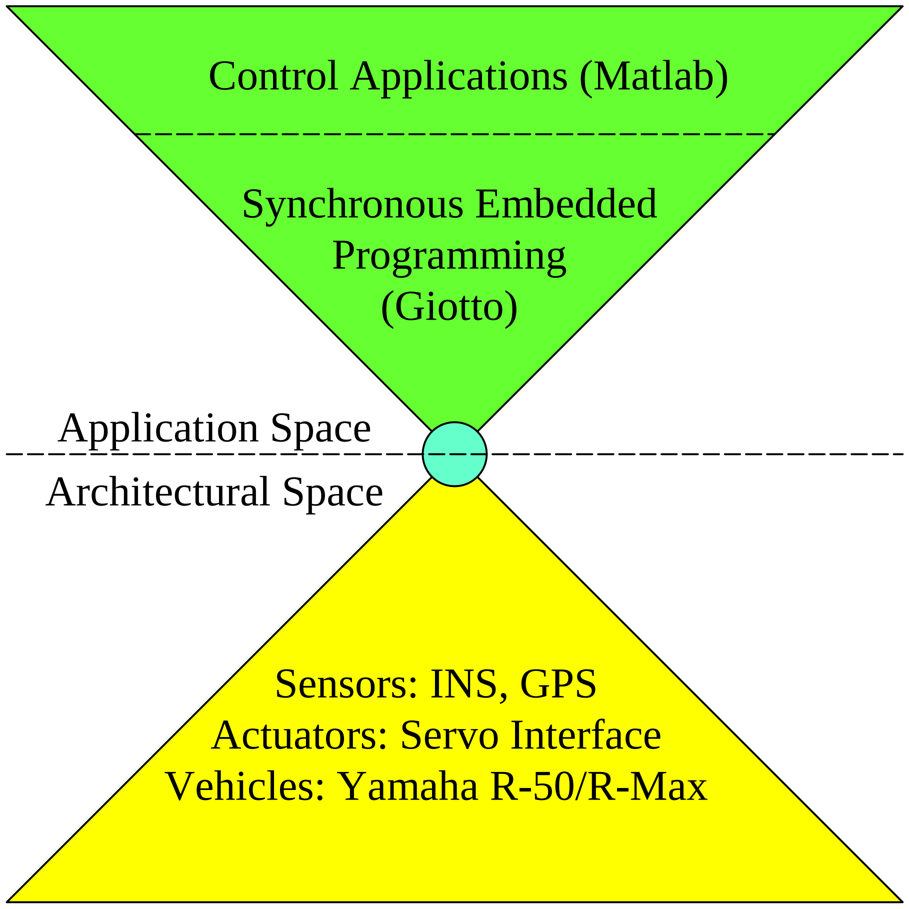
Introduce Abstraction

- Use platform based design
- Enforce static scheduling
  - Restrict design space
  - Verifiable real-time constraints

- Use Giotto!

Platform Based Design with Giotto

- Goal
  - Abstract details of sensors, actuators, and vehicle hardware from control applications
  - Real-time verification
- How?
  - Platform
  - Synchronous Embedded Programming Language (i.e. Giotto)


Control Applications (Matlab)
Synchronous Embedded Programming (Giotto)
Application Space
Architectural Space
Sensors: INS, GPS
Actuators: Servo Interface
Vehicles: Yamaha R-50/R-Max

Introduction To Giotto

- Giotto is an abstract programmer’s model for implementing embedded system software
- Created to model periodic software tasks and mode switches with hard real-time constraints
- Sensor readings and tasks (periodic functional units) are time triggered

More Giotto

- Giotto guarantees model will meet real-time requirements on any platform
  - Separates the platform-independent from the platform-dependent concerns
  - Abstracts away scheduling and platform-dependent issues
    - Designer can concentrate on system model and assume deadlines are met independent of chosen platform

Fitting Software Into Giotto

- Model periodic functional units as *tasks*
  - sensor data reading and control calculations occur periodically and can be modeled as tasks
- A set of concurrent tasks make up one *mode*, for example: Hover Mode
- Modes repeat until a mode switch is requested

Giotto Specifications

- Communication is instantaneous
  - Time is accounted for within the tasks
- Input data cannot be refreshed in the middle of a task
- Outputs from a task cannot be used until the task deadline time
- Tasks are only guaranteed to finish by end time, not to start at beginning time

V. Case Study: Helicopter UAV

Case Study: Helicopter

- Goals:
  - Incorporate asynchronous input devices and real-time controller
  - Modular - to allow replacing subsystems
- How?
  - Use Platform-Based Design
  - Use Time-Based Controller

UAV System:
Sensor Overview

R-50 Hovering

GPS Card

GPS Antenna

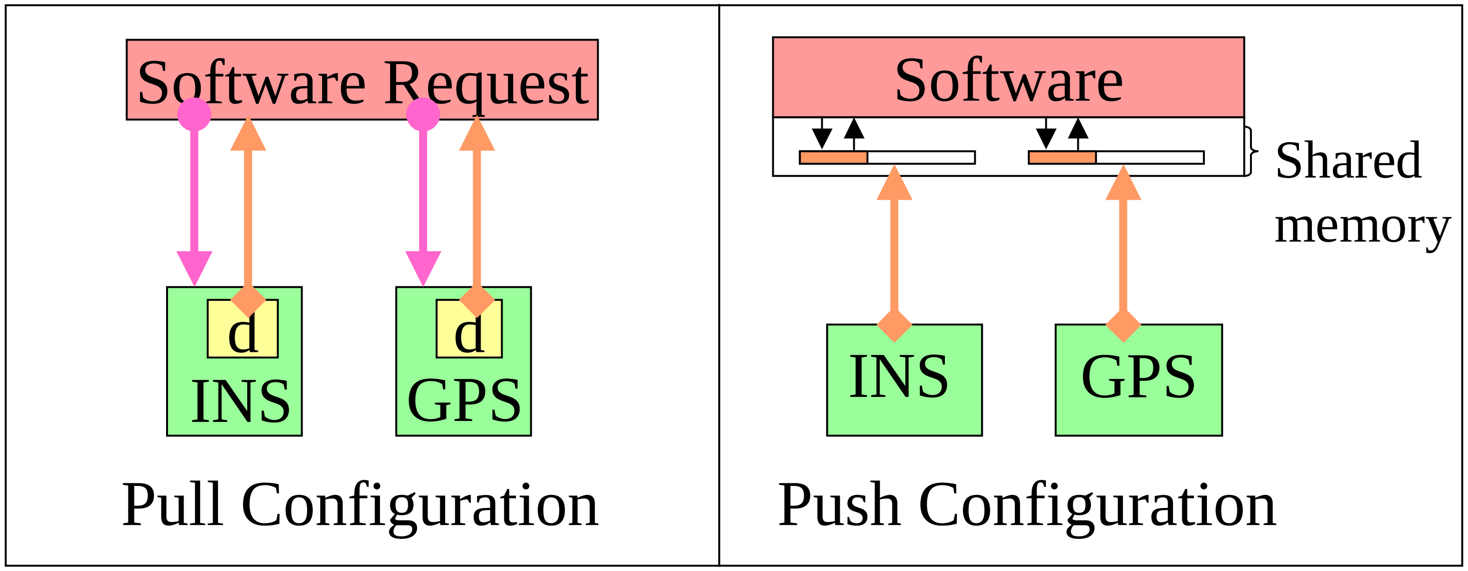
- Goal: basic autonomous flight
  - Need: UAV with allowable payload
  - Need: combination of GPS and Inertial Navigation System (INS)
- GPS (senses using triangulation)
  - Outputs *accurate* position data
  - Available at *low rate* & has jamming
- INS (senses using accelerometers and gyroscopes)
  - Outputs estimated position with *unbounded drift* over time
  - Available at *high rate*
- Fusion of GPS & INS provides needed high rate and accuracy

INS

II. UAV System:

Sensor Configurations

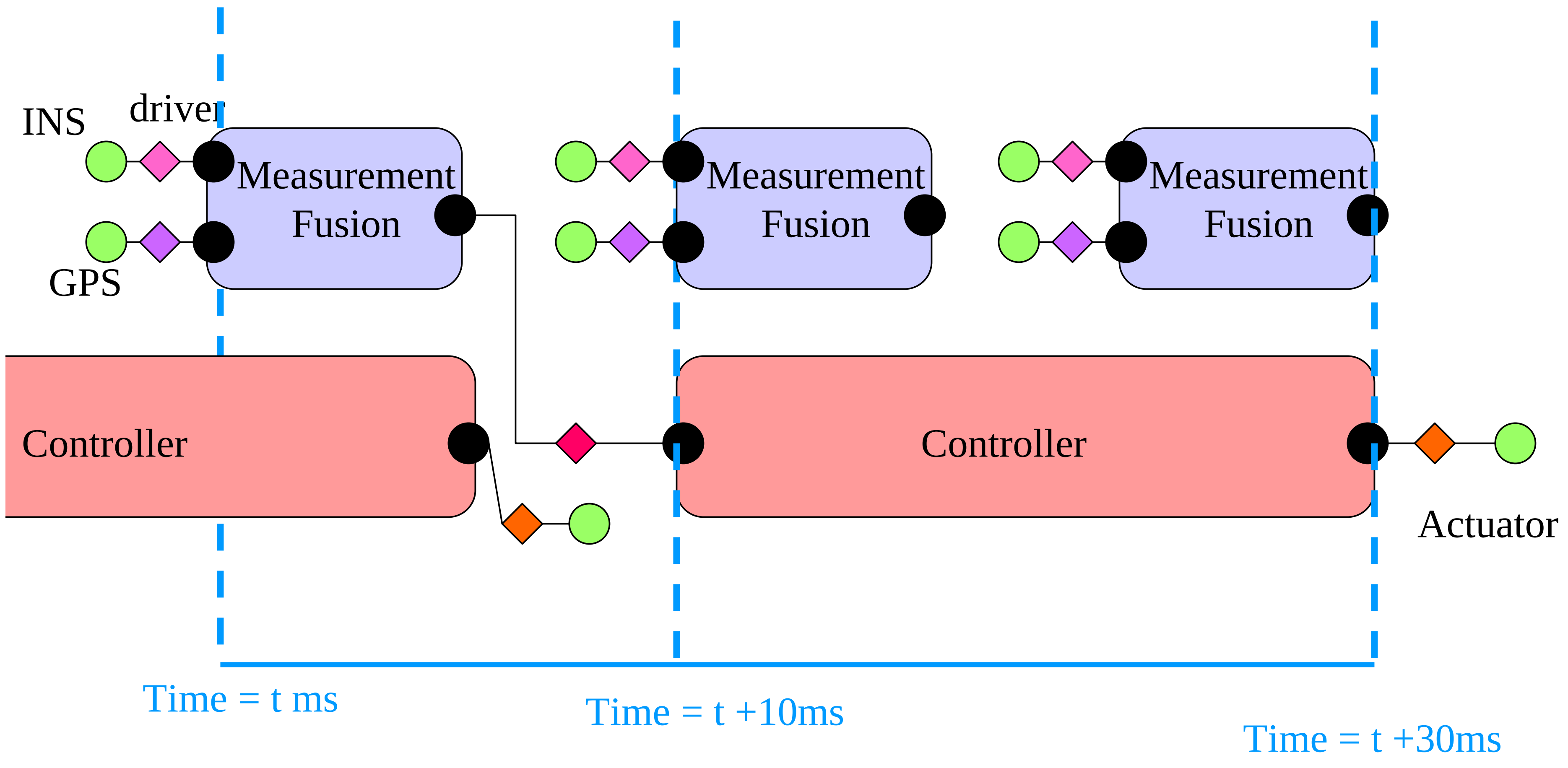
- Sensors may *differ* in:
  - Data formats, initialization schemes (usually requiring some bit level coding), rates, accuracies, data communication schemes, and even data types
- Differing Communication schemes requires proprietary code for each sensor



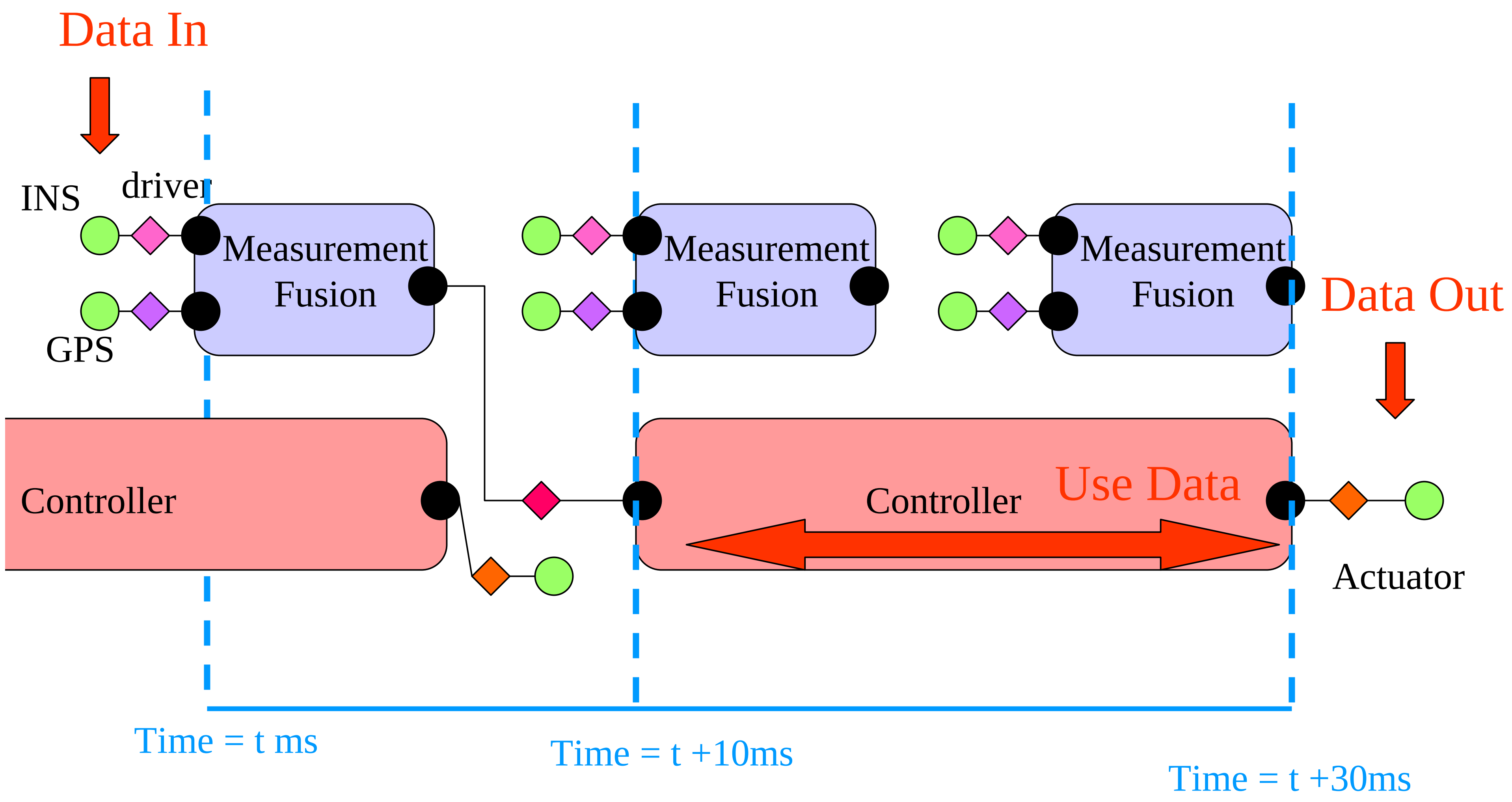
Control Tasks

- Two tasks:
  - – Sensor Fusion
    - Inputs from sensors
    - Kalman filter for GPS and INS
  - – Control
    - Computes control law
    - Output to actuators

Synchronous Control: Giotto



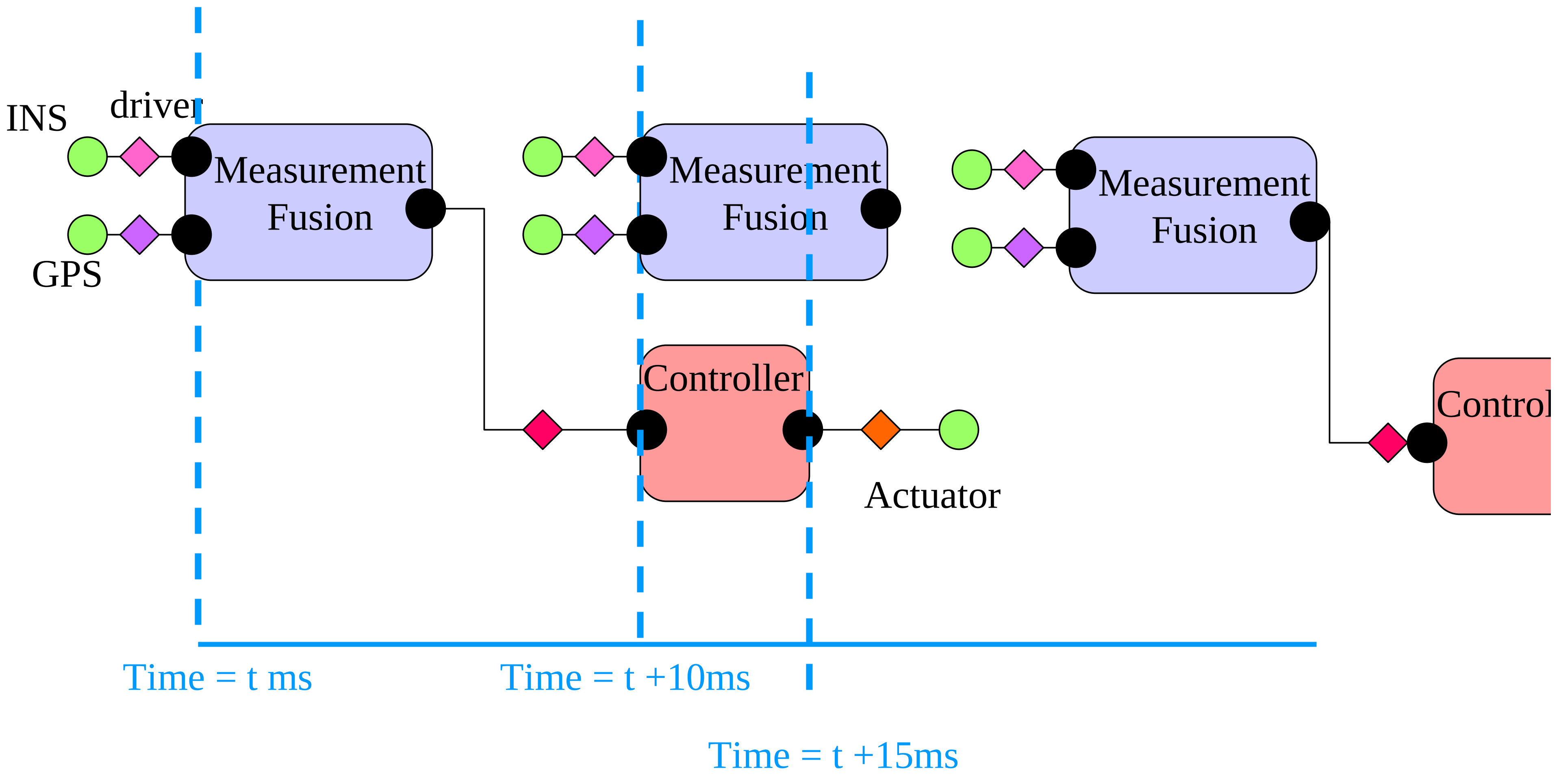
Delay Analysis



III. Synchronous Control

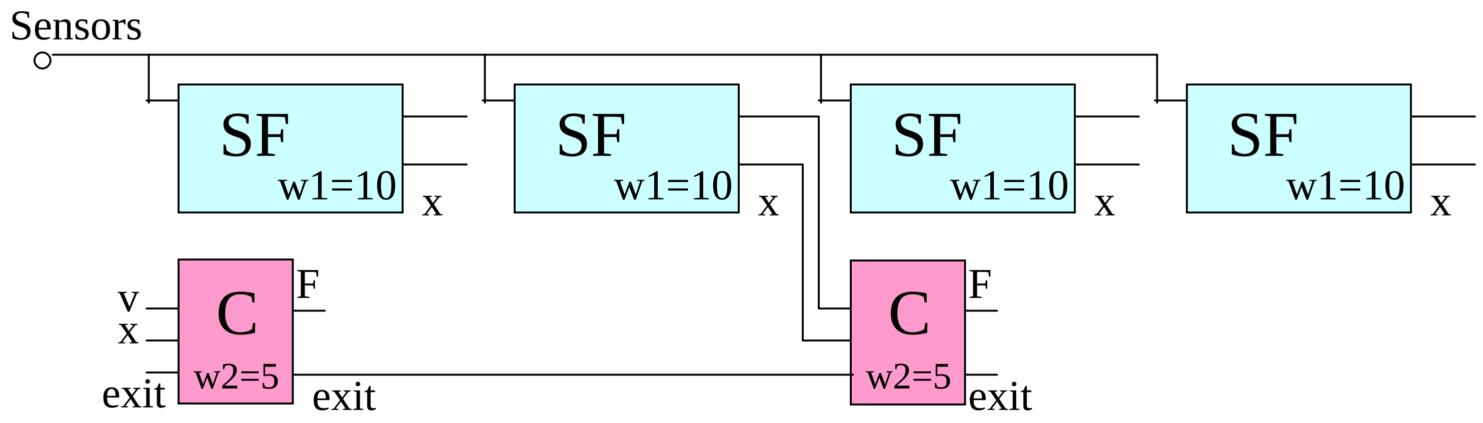
- Advantages of **time-triggered framework**:
  - Allows for *composability* and *validation*
    - These are important properties for safety critical systems like the UAV controller
  - Timing guarantees ensure *no jitter*
- Disadvantages:
  - *Bounded delay* is introduced
    - Stale data will be used by the controller
  - Implementation and system integration become more difficult
- Platform design allows for time-triggered framework for the UAV controller
  - Use Giotto as a middleware to ease implementation:
    - provides real-time guarantees for control blocks
    - handles all processing resources
    - Handles all I/O procedures

Reduced Delay Model



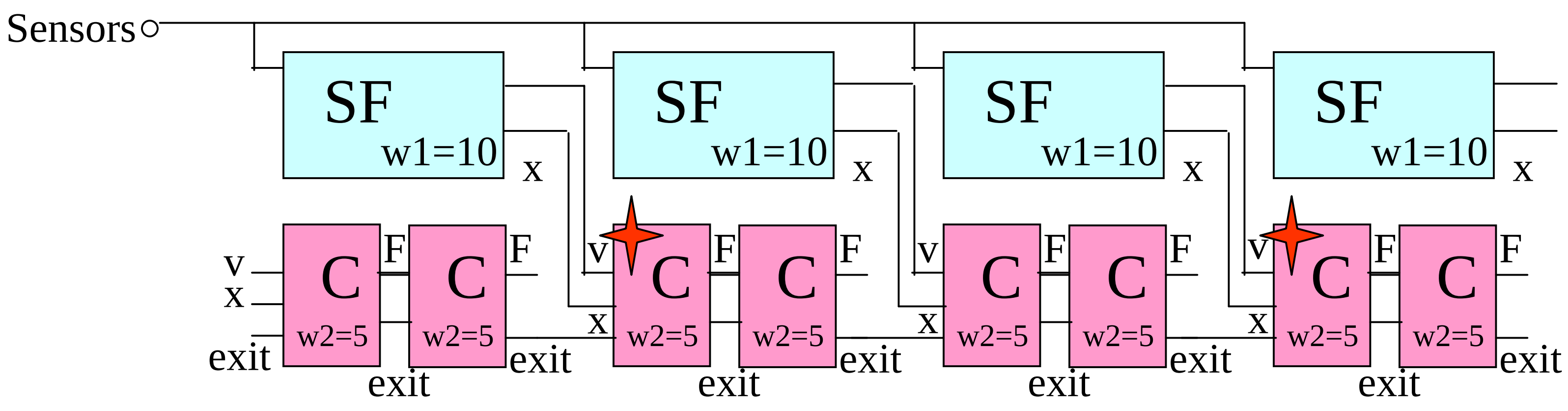
Ideal Giotto System

- Would like to guarantee that control block finishes as the system requires (say 5ms), but still only runs every 20ms
  - would like to model as follows:



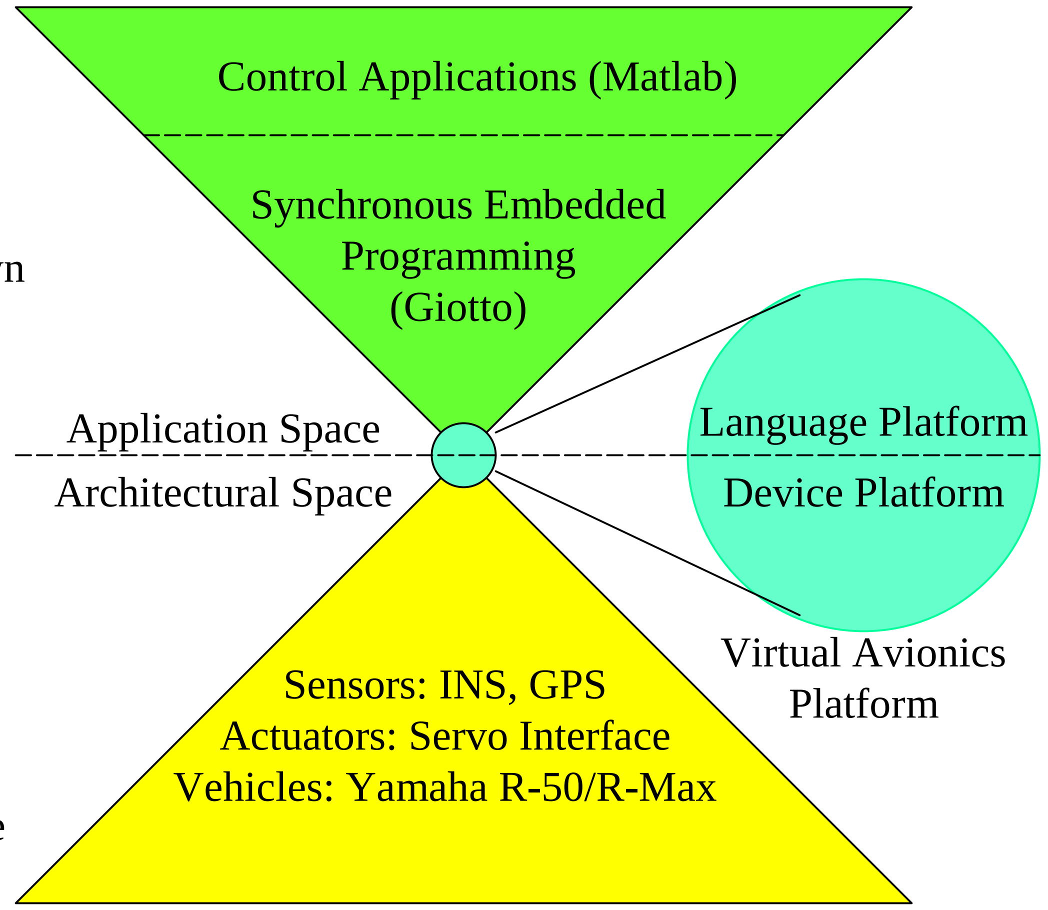
Reduced Delay Giotto System

- Cannot model such a system in Giotto
  - Giotto mandates that finish time of a task be based on its periodicity
  - Instead create a new model
    - call the control block with *frequency = max delay tolerated* but only actually compute control when needed



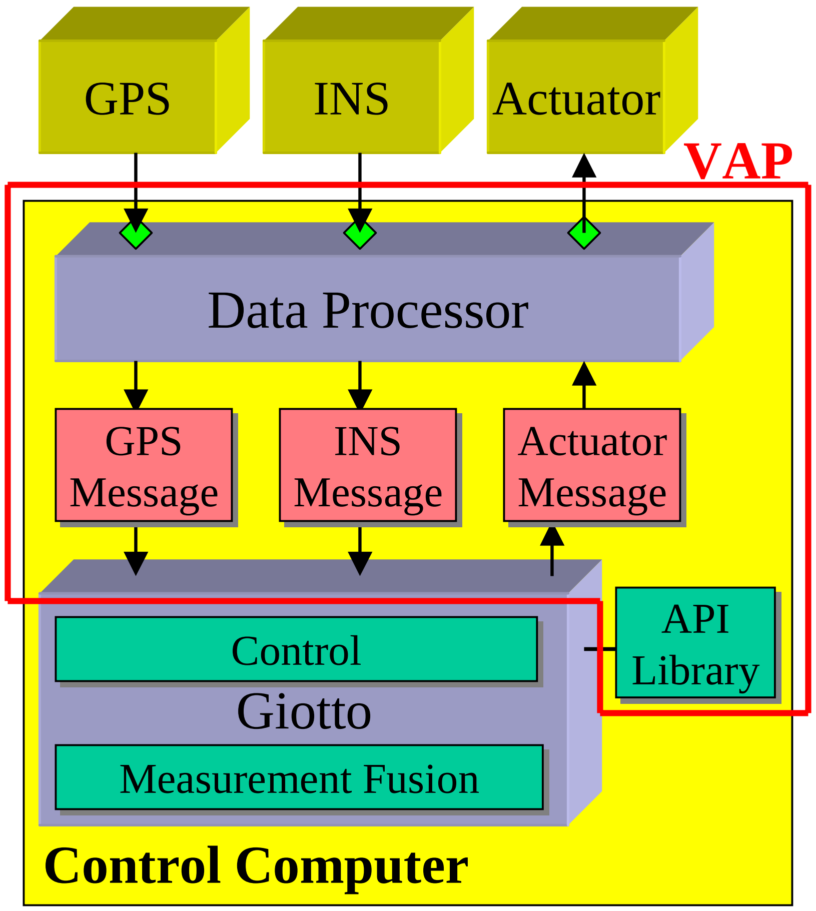
Platform Based Design for UAVs

- Device Platform
  - Isolates details of sensor/actuators from embedded control programs
  - Communicates with each sensor/actuator according to its own data format, context, and timing requirements
  - Presents an API to embedded control programs for accessing sensors/actuators
- Language Platform
  - Provides an environment in which synchronous control programs can be scheduled and run
  - Assumes the use of generic data formats for sensors/actuators made possible by the Device Platform


Control Applications (Matlab)
Synchronous Embedded Programming (Giotto)
Application Space
Architectural Space
Language Platform
Device Platform
Virtual Avionics Platform
Sensors: INS, GPS
Actuators: Servo Interface
Vehicles: Yamaha R-50/R-Max

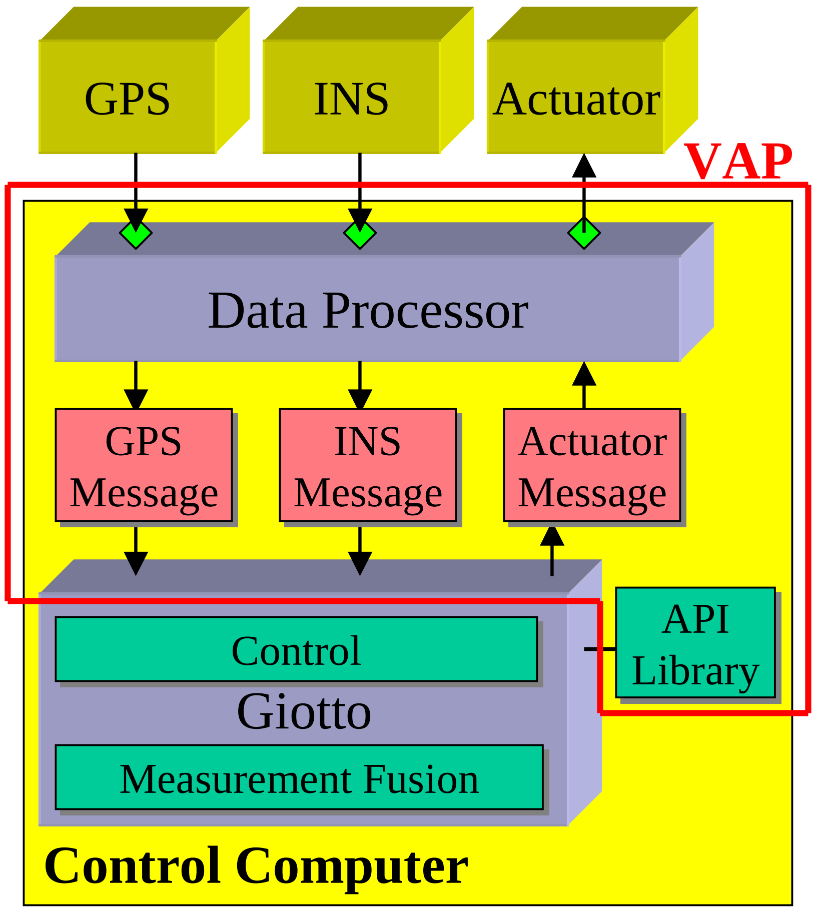
The Control Computer

- Goal: Control UAV via sensors/actuators
- Data Processor
  - Handles the timing/interrupt of sensors and actuators
  - Moves sensor/actuator data
    - No format conversion
    - Saves time for Giotto tasks
- Shared Memory
  - Serves as bridge between synchronous and asynchronous parts of system
  - Circular buffer: allow simultaneous read/write

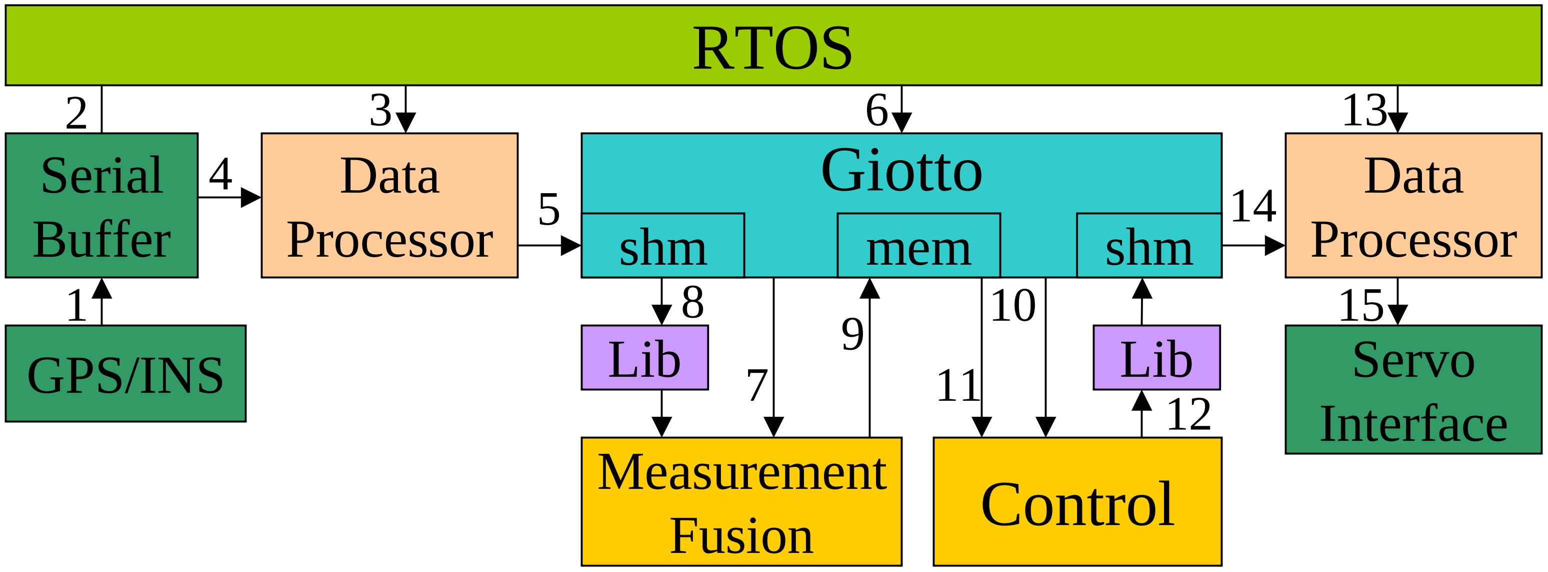


The Control Computer

- Giotto Program
  - Where control algorithms (Control) and Kalman filter (Measurement Fusion) reside as Giotto *tasks*
- API Library
  - Allows control programs to interpret sensor data and send data to actuator as generic, device independent format
  - Implemented as C routines



Example – From Computer's Point of View



1. GPS/INS sends *sensor data* via serial
2. RTOS generates interrupt
3. RTOS fires Data Processor as ISR
4. Data Processor gets *sensor data*
5. D/P saves *sensor data* to shared memory
6. RTOS fires Giotto *process*
7. Giotto fires Measurement Fusion *task*
8. M/F interprets sensor data via library
9. M/F computes *combined measurement* and stores it to memory
10. Giotto fires Control *task*
11. Control uses *combined measurement*
12. Control *task* generates *control* and saves it to shared memory
13. RTOS fires Data Processor
14. Data Processor gets *control*
15. D/P sends *control* to Servo Interface

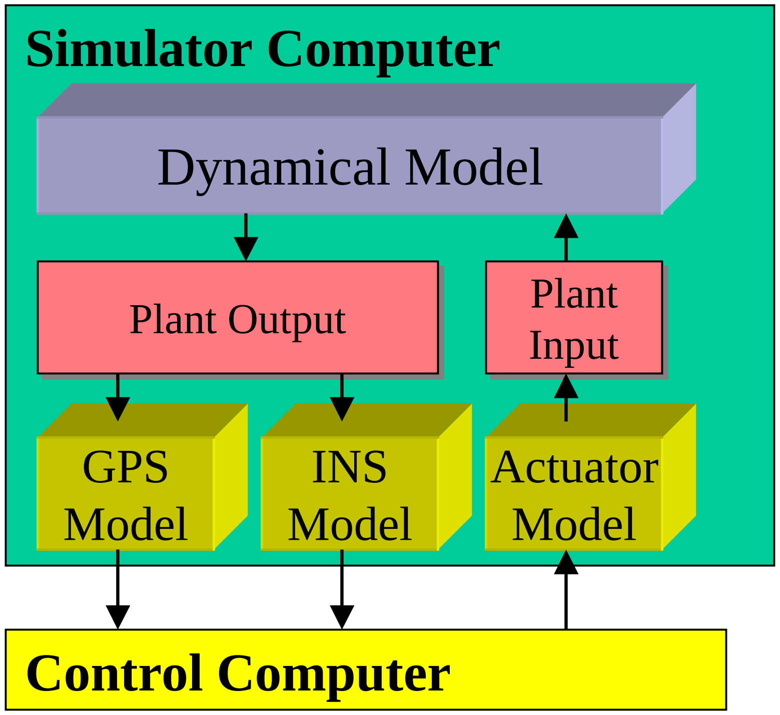
Example – From Controller's Point of View

```
Sensor
c_input sensor_input uses
   c_get_sensor_inputs;
.
.
.
task control('inputs')('outputs') {
   schedule c_control_task('inputs')
}
.
.
.
taskfreq 1 do
   control(control_driver);
```

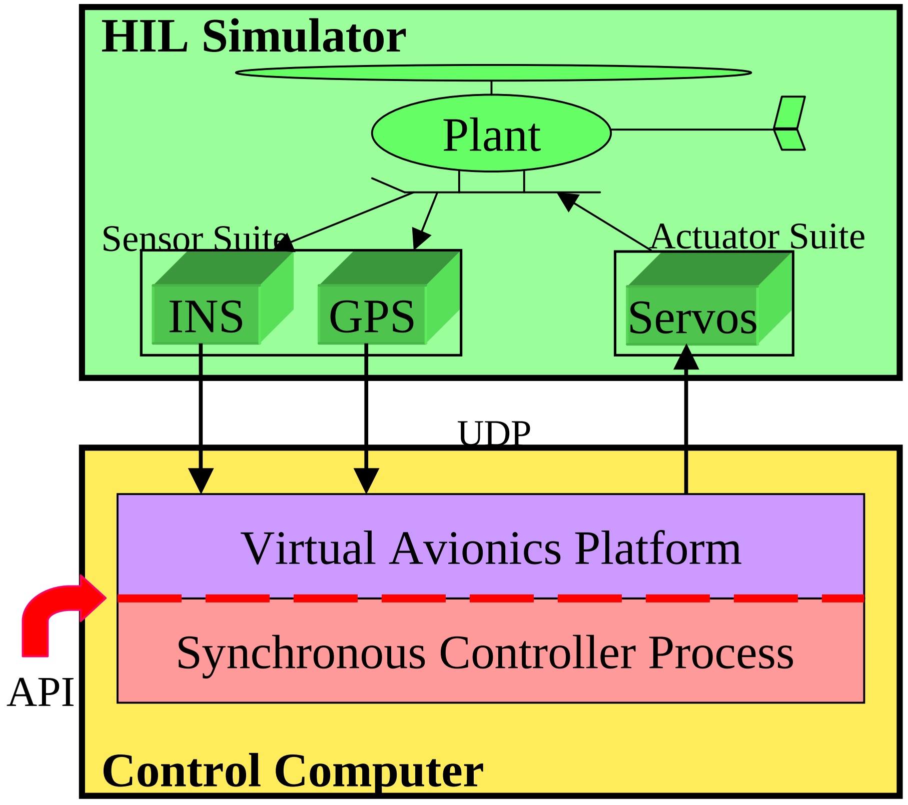
- Refers to C function
  - Sets shared memory for Giotto's internal use
  - Assumes shared memory will be filled with measurements
- Refers to C controller function
  - Assumes measurements are waiting in buffer that shared memory points to
  - May call library functions to convert measurements
- Giotto runs the control task
  - Shared memory passed into control task by control_driver

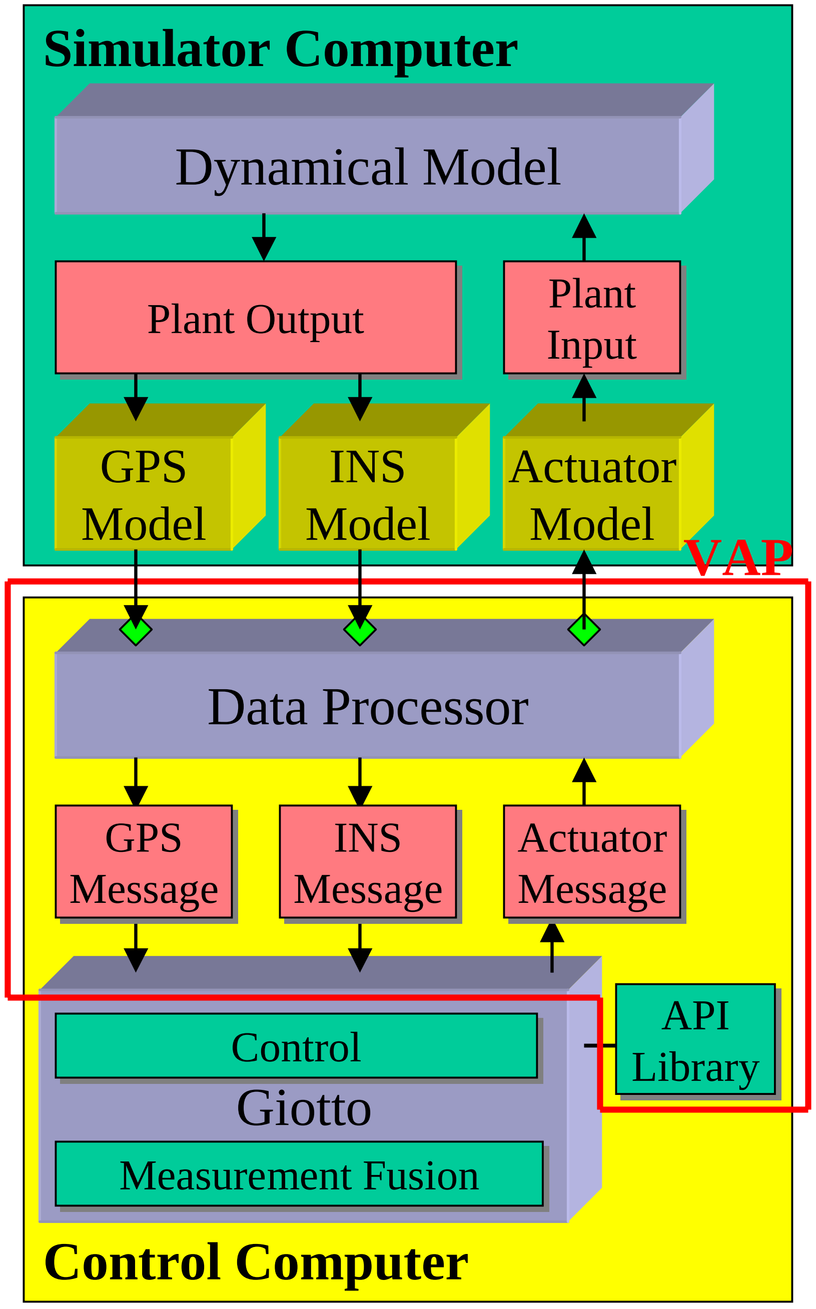
The Simulator Computer

- Models the environment surrounding Control Computer
- Dynamical Model
  - Helicopter Dynamics
- Sensor/Actuator Models
  - Simulate sensor/actuator timing behavior and data format
- Shared Memory
  - State and control information
  - Circular Buffer

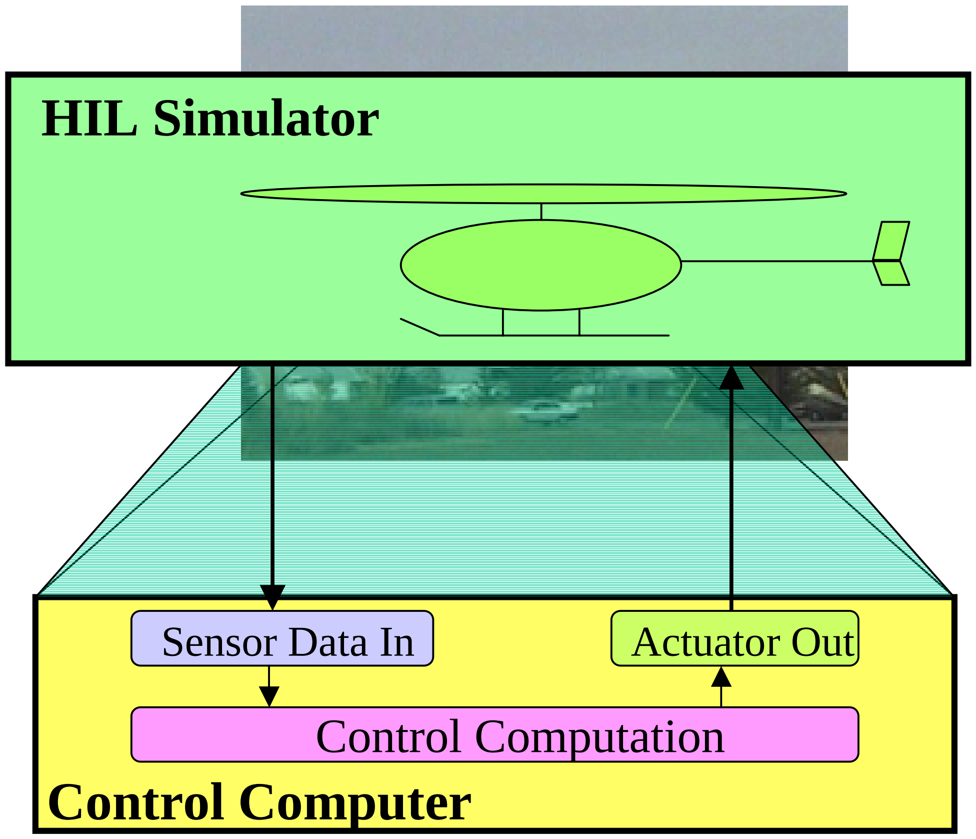


Combined System



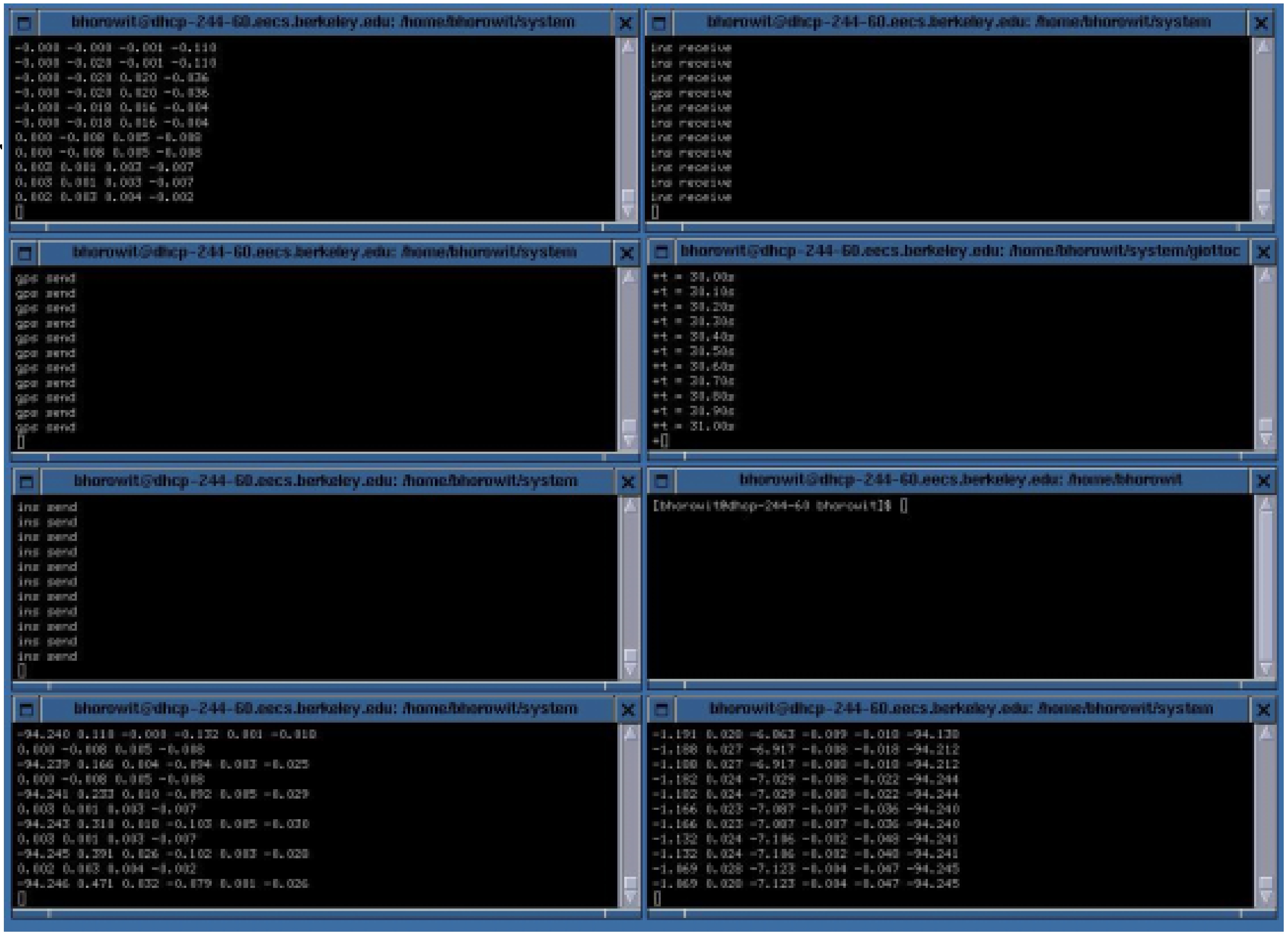


Hardware-in-the-Loop Framework



- Project uses platform based design & synchronous control to implement a controller computer that will:
  - (eventually) fly on a UAV!
  - (now) 'flies' a UAV simulator
- Hardware-in-the-loop has several advantages:
  - Safe & inexpensive testing
  - Repeatable tests
  - Partial simulations
    - All have been useful for implementing the new design methodology

In Action – Process Windows Running



Plant's Receiver

```
bhorowit@dhcp-244-60.eecs.berkeley.edu: /home/bhorowit/system
-0.008 -0.008 -0.001 -0.110
-0.008 -0.028 -0.001 -0.110
-0.008 -0.028 0.020 -0.036
-0.008 -0.028 0.020 -0.036
-0.008 -0.018 0.016 -0.004
-0.008 -0.018 0.016 -0.004
0.000 -0.008 0.085 -0.088
0.000 -0.008 0.085 -0.088
0.003 0.001 0.003 -0.007
0.003 0.001 0.003 -0.007
0.002 0.003 0.004 -0.002
```

Data Processor

```
bhorowit@dhcp-244-60.eecs.berkeley.edu: /home/bhorowit/system
ins receive
ins receive
ins receive
gps receive
ins receive
ins receive
ins receive
ins receive
ins receive
ins receive
ins receive
```

GPS *sensor model*

```
bhorowit@dhcp-244-60.eecs.berkeley.edu: /home/bhorowit/system
gps send
gps send
gps send
gps send
gps send
gps send
gps send
gps send
gps send
gps send
gps send
```

Giotto Process

```
bhorowit@dhcp-244-60.eecs.berkeley.edu: /home/bhorowit/system/giottoc
+t = 30.00s
+t = 30.10s
+t = 30.20s
+t = 30.30s
+t = 30.40s
+t = 30.50s
+t = 30.60s
+t = 30.70s
+t = 30.80s
+t = 30.90s
+t = 31.00s
+
```

INS *sensor model*

```
bhorowit@dhcp-244-60.eecs.berkeley.edu: /home/bhorowit/system
ins send
ins send
ins send
ins send
ins send
ins send
ins send
ins send
ins send
ins send
ins send
```

```
bhorowit@dhcp-244-60.eecs.berkeley.edu: /home/bhorowit
[bhorowit@dhcp-244-60 bhorowit]$
```

Plant

```
bhorowit@dhcp-244-60.eecs.berkeley.edu: /home/bhorowit/system
-94.240 0.118 -0.008 -0.132 0.001 -0.010
0.000 -0.008 0.085 -0.088
-94.239 0.166 0.004 -0.094 0.083 -0.025
0.000 -0.008 0.085 -0.088
-94.241 0.233 0.010 -0.092 0.085 -0.029
0.003 0.001 0.003 -0.007
-94.243 0.318 0.018 -0.103 0.085 -0.030
0.003 0.001 0.003 -0.007
-94.245 0.391 0.026 -0.102 0.083 -0.028
0.002 0.003 0.004 -0.002
-94.246 0.471 0.032 -0.079 0.081 -0.026
```

Display Outputs

```
bhorowit@dhcp-244-60.eecs.berkeley.edu: /home/bhorowit/system
-1.191 0.028 -6.862 -0.089 -0.010 -94.138
-1.188 0.027 -6.917 -0.088 -0.018 -94.212
-1.188 0.027 -6.917 -0.088 -0.018 -94.212
-1.182 0.024 -7.029 -0.088 -0.022 -94.244
-1.182 0.024 -7.029 -0.088 -0.022 -94.244
-1.166 0.023 -7.087 -0.087 -0.036 -94.240
-1.166 0.023 -7.087 -0.087 -0.036 -94.240
-1.132 0.024 -7.186 -0.082 -0.048 -94.241
-1.132 0.024 -7.186 -0.082 -0.048 -94.241
-1.069 0.028 -7.123 -0.084 -0.047 -94.245
-1.069 0.028 -7.123 -0.084 -0.047 -94.245
```

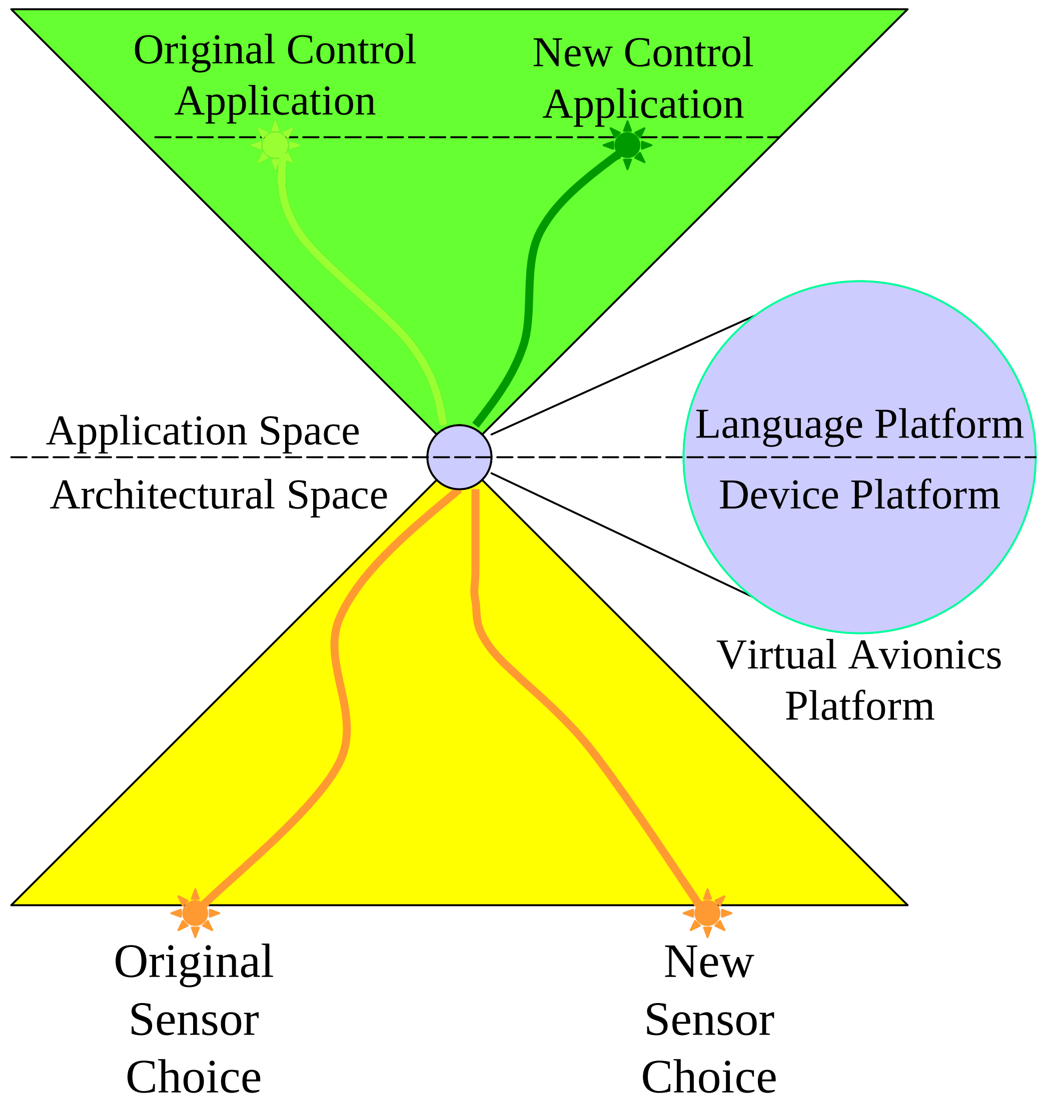
In Action – OpenGL Visualization

Unregistered HyperCam

Conclusions

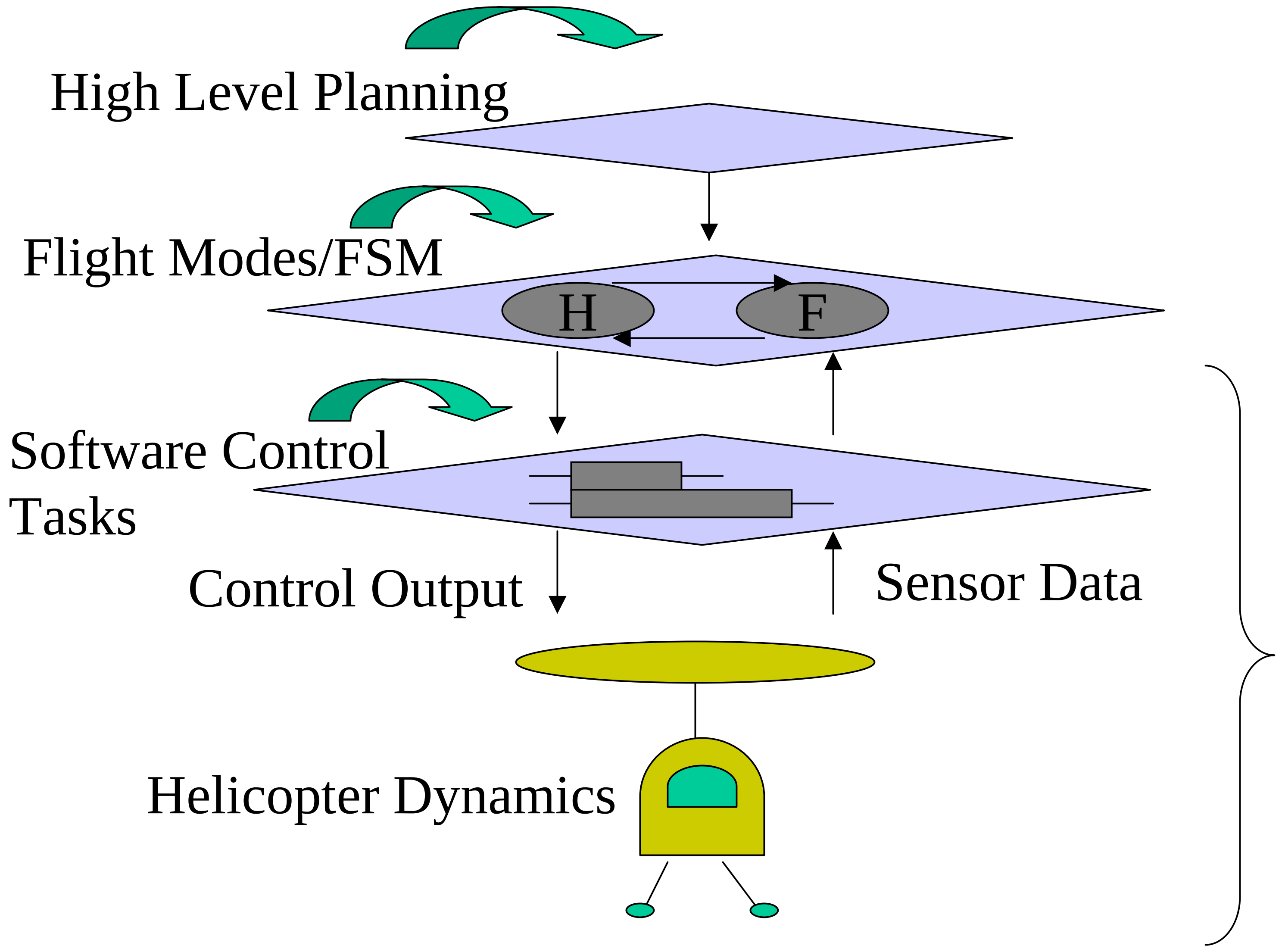
- Developed Methodology
  - Platform-based design
    - Provides appropriate layers of abstraction
    - Eases Software Reuse
    - Eases Hardware Modifications
  - Time-based control
    - Verifiable real-time constraints
    - Eases controller modifications

Conclusions

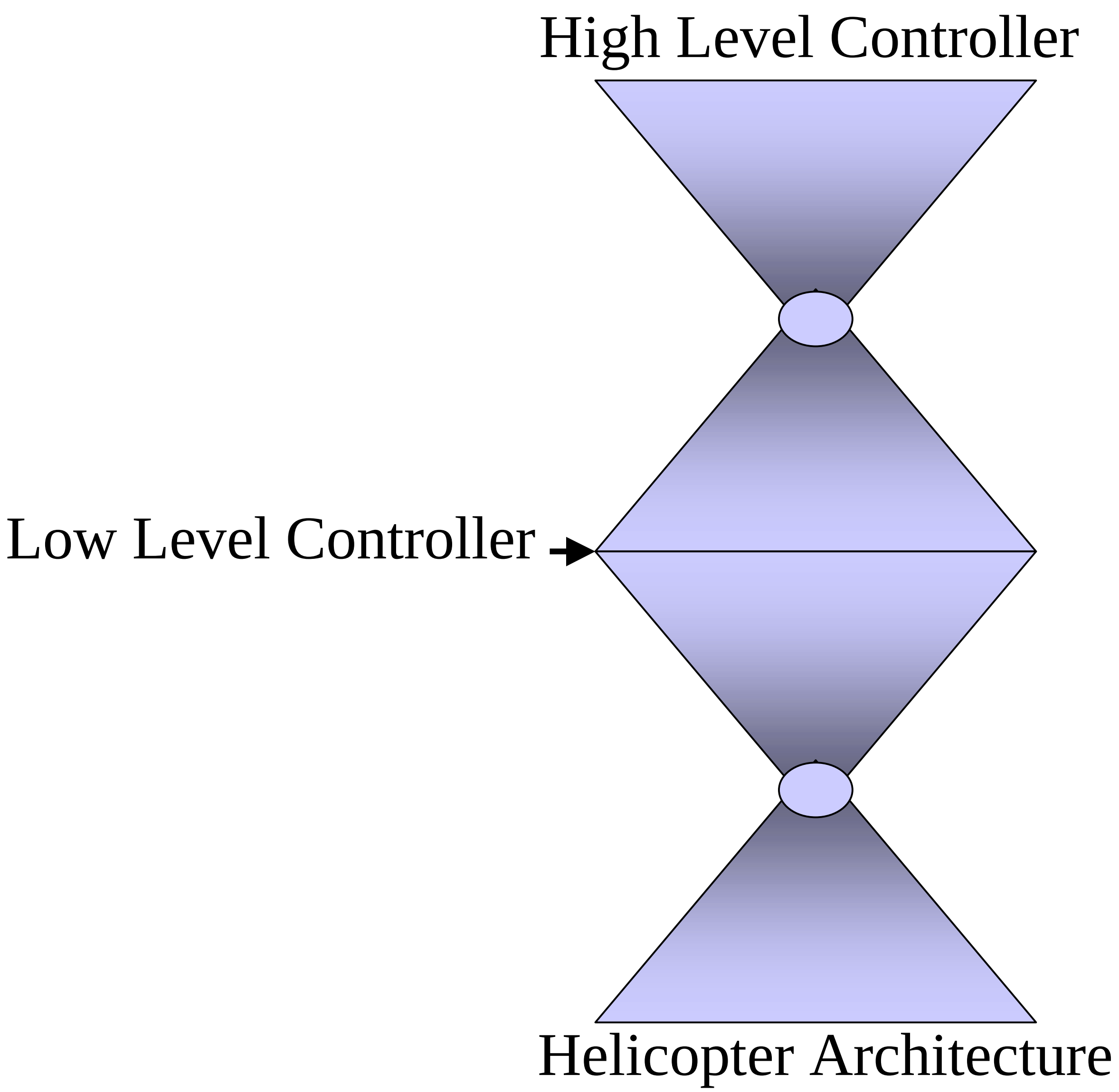


- Exchanging sensors
  - Controller remains same!
  - Platform adapts
    - Handles new data types and formats
- Exchanging controller
  - New controller use the same API
  - Giotto maintains timing requirements

Future Work – Nested Platforms



Future Work – Nested Platforms



YAMAHA

YAMAHA
Rmax
YAMAHA

References/Acknowledgements

Platform-Based Embedded Software Design and System Integration for Autonomous Vehicles, Benjamin Horowitz, Judith Liebman, Cedric Ma, T. John Koo, Alberto Sangiovanni-Vincentelli, Shankar Sastry