Abstract
This project combines design philosophies from three different areas: embedded systems design, synchronous embedded control, and unmanned aerial vehicle design. The main objective of this project is the application of the principles of Platform-Based Design to the specific area of UAV system design in order to yield a greatly enhanced overall UAV control system. We demonstrate this application by implementing the final design and testing it using a hardware-in-the-loop simulation.

1. Introduction
Implementing a control system on a UAV is a tremendous undertaking. One needs to combine complicated sensors such as GPS and INS, servo actuators, a wireless network, a central computer, and control laws derived from an approximate dynamic model of the vehicle. Current UAV design implementations have usually been developed by focusing on proving the feasibility of concept rather than elegance in design. This oversight often results in an overall system that is unverifiable and inflexible. For example, in many implementations, the embedded software is written exclusively for a particular mix of sensors and actuators. The substitution of a different GPS sensor in place of the original one would mean a major overhaul in the embedded software followed by an extensive verification process. We believe that by incorporating mechanisms for isolating the programmatic details of a particular sensor and actuator from the core control program, these types of changes would be much easier to handle. This modularity allows a UAV to take advantage of the latest developments in sensor technologies with minimum hassle. Therefore, in this project we strive to achieve a new design for an embedded UAV control system that maximizes modularity and enables verification.

2. Design Foundations
In this section, we first present the foundations regarding the Platform-Based Design methodology, UAV components and design, as well as synchronous control methodologies, before explaining the details of our combination of the three.

2.1 Platform Based Design Approach
The “Platform-Based Design” approach to embedded systems design has been developed to address several key issues in the IC development domain[1] [2]. This type of design strategy, however, is universal and can be described in a general way that will subsequently lend itself to its deployment in the area of UAV control. Figure 1 illustrates the idea of a platform. A platform can be defined as a layer of abstraction with two views. The upper view is from the application space. Here the platform allows a designer, perhaps a control engineer, to develop control applications without having to deal directly with the lower levels. The lower view is from the components and tools available. Here the platform provides specifications that the components and tools need to provide. The main benefit is that the upper view is decoupled from the lower view, and the two interact through a well-defined interface. This decoupling allows for either the applications or the components to be altered, provided the two can still meet in the middle.

A platform instance is the particular component and tool design chosen to implement the platform. It is developed by mapping the functionality needed by the upper layers onto specific components below. The choice of this platform instance may affect the final specifications of the platform. That is, there is a feedback loop connecting the platform specifications and the platform instance. Hence,
the application space as well as the component space affects the platform specification, and the overall design methodology is a ‘meet-in-the-middle’ approach.

The Application Programming Interface (API) is the front end of the platform that the control designer sees. Therefore, the API is the abstraction of all of the components and tools below and is the interface layer to the application space. The final main idea for general platform-based design is that this design methodology is hierarchical, or fractal in nature. That is, a platform may be comprised of several layers or sub-platforms. Similarly, a platform and its application may be used as a platform instance for a higher level platform.

2.2 Helicopter, Sensors, Actuators

The helicopter group has many different models of helicopters. The Yamaha R-50 and R-Max helicopters have been decidedly easier to equip and fly due to their superior payload carrying abilities. Recently, autonomous flight has been demonstrated on one of the R-50’s and the other R-50 and R-Max’s are being prepared for similar flight experiments. We have used a reasonably accurate dynamic model of the R-50 and the control algorithm developed for the R-50.

Since the model helicopters have a relatively fast physical response time, precise and detailed sensors are needed to make autonomous flight possible. While a variety of sensors can be used for landing, takeoff, and intricate maneuvers, a setup consisting of a combination of two sensors is best suited for basic autonomous flight. These two sensors are the Inertial Navigation System (INS) and the Global Positioning System (GPS). The INS uses a gyrosopic measurement system and linear accelerometers to provide acceleration and rotation data at a fairly high rate (at 100 Hz). This data can be used to recover the position of the vehicle. However, if uncorrected, the estimated position data drifts unbounded over time and is therefore insufficient if used alone. The GPS uses a triangulation scheme involving multiple satellites to recover position data that is highly accurate with bounded error at all times. However, this data is available at a lower rate (at approximately 4 Hz) than required by the controller, is occasionally unavailable, and is also subject to jamming and radio interference. Therefore, the combination of the two instruments remains the best setup.

There are many choices of INS and GPS units in the market. Each may have different data formats, initialization schemes (usually requiring some bit level coding), operation rates, accuracies, data communication schemes, and even data types (velocity vs. acceleration). Needless to say, incorporating a new sensor in the control system requires significant programming. In fact, the differing communication schemes provide the most challenging part of switching to a new sensor. For example, if the sensor used to be a ‘data push’ sensor that would interrupt the computer when it had data available, the control program would have to undergo a major rewrite in order to use a ‘data pull’ sensor that would store data until the computer asked for it.

2.3 Synchronous Control

It has been effectively argued that time triggered interfaces allow for composability and validation, both of which are essential for building complicated embedded systems[4]. It has further been proposed that using an entirely time-triggered framework for the control software of embedded systems will allow for ease of composability and validation for not just the interfaces, but the entire control system[4]. The validation guarantees of this layer are highly valued since correct execution of the low level control is a safety critical task. Furthermore, in an event-driven system, jitter is possible because no timing guarantees can be made. Since jitter negatively affects the controller’s ability to compute the correct results, the timing guarantees that come along with a time-triggered architecture are highly desired. The tradeoff of using a time-triggered architecture is that a decidable bounded delay is introduced causing a certain amount of stale data usage.

![Figure 2: Synchronous Embedded Programming with Giotto](image)

In this project, we opt for the guarantees of a synchronous control system. We utilize a middleware that allows the control designer to assume the correct execution of timing and communication between tasks. This middle layer of software, named Giotto, provides real-time guarantees for control blocks, and handles all processing resources and I/O procedures. Giotto is being developed here at UC Berkeley for use in embedded systems[5]. Figure 2 displays our embedded software design under Giotto of the
Measurement Fusion and the Controller tasks. The delay in data usage is shortened by setting an early deadline for the control task[4]. This synchronous system has a predictable behavior and can therefore guarantee safety critical control performance.

2.4 Hardware-in-the-Loop Simulation Development

In this project we are not only proposing a new design methodology for UAV software development. We are also using this new design methodology to create a Control Computer that will eventually be used to fly the R-Max Helicopter. Instead of trying out these new ideas in embedded software immediately on the real helicopter, we have decided to start by developing a hardware-in-the-loop simulation tool.

The hardware-in-the-loop simulation framework is useful for many reasons. First of all, it allows for exhaustive yet safe and relatively inexpensive testing of embedded software because it replaces the real and possibly dangerous environment with a simulator. Second of all, contrary to the actual environment, the simulator allows for repeatable testing. It also allows for testing pieces of a system independently. These two conditions are critical when developing a new design methodology for the embedded software.

The VAP is a classical platform in Platform Based Design terminology: it is represented as a “node” in Figure 4. Above the node is the Application Space, which includes the synchronous embedded programs and the control applications. The Application Space is separated into two layers because control applications are typically written in mathematical form such as Matlab. To make such an algorithm implementable in a synchronous manner, it must be translated into a synchronous programming language such as Giotto. Finally, below the node is the architectural space, which includes the flight hardware such as sensors and actuators.

Figure 3: Hardware-in-the-loop Framework

Our setup consists of two computers: the Control Computer and the Simulation Computer. The Simulation Computer uses the dynamic model of our helicopter and numerical integration to produce the same outputs that the real helicopter would produce. The Control Computer is the actual hardware that will fly on the helicopter. This computer takes the input from the Simulation Computer or the input from sensors on the helicopter during flight, and produces the control outputs. The Control Computer houses the proposed platform and API as well as the control algorithm. In the next section we present the platform design and exact implementation of the platform instance. Finally, a 3-D visualization tool is connected to the Simulation Computer so that the performance of the controller can be easily visualized.

3. Platform Based Design of UAV

Here we present a conceptual view of a platform for UAV control system. Figure 4 illustrates the use of a platform, which we call “Virtual Avionics Platform,” or VAP, to achieve the goal of isolating the details of particular sensors, actuators, and other UAV hardware from the control application. The VAP serves as a “bridge” between the top layer, the control application, from the bottom layer, the flight hardware.

The VAP is a classical platform in Platform Based Design terminology: it is represented as a “node” in Figure 4. Above the node is the Application Space, which includes the synchronous embedded programs and the control applications. The Application Space is separated into two layers because control applications are typically written in mathematical form such as Matlab. To make such an algorithm implementable in a synchronous manner, it must be translated into a synchronous programming language such as Giotto. Finally, below the node is the architectural space, which includes the flight hardware such as sensors and actuators.
allowing the use of synchronous embedded programming languages above.

The Device Platform does not merely consist of device drivers. Recall that our goal here is to isolate the details of particular sensors and actuators from the control programs. To do this, the Device Platform will utilize specific information regarding each sensor and actuator to communicate with each device according to its own format, context, and timing requirements. To the embedded control programs above, the details of particular sensors and actuators are absent because an API is presented as the front end of the VAP to the embedded programs. For sensing, the Device Platform would convert the sensor information to a generic format independent of the particular sensor chosen, such as x, y, z coordinates for position, so that the synchronous embedded program can use it readily. For actuation, the Device Platform allows the use of generic control input such as angles and throttle from the embedded control programs and presents it to the servo interface as the signal it desires. The API for the Virtual Avionics Platform thus allows the embedded control programs to be relatively simple.

The Language Platform assumes the use of generic data formats made possible by the Device Platform and provides an API for the embedded control programs to communicate with the devices as well as an environment in which synchronous programs can be scheduled and run.

3.1 Virtual Avionics Platform Implementation

The implementation of the VAP resides in the Control Computer half of the hardware-in-the-loop framework as previously mentioned. The Virtual Avionics Platform instance is made up of an API library, shared memory, and a Data Processor. The Data Processor is an independent process in the OS, and in the sensing case, it responds to new sensor data sent over the serial port by the devices. The Data Processor reads this sensor data off the serial buffer and saves it to a shared memory space with a format that depends on the sensor. The Data Processor is a necessary part of the Virtual Avionics Platform and independent of Giotto because it interacts with the asynchronous part of the system by handling interrupts generated by the sensor devices. When a control task in Giotto is executed it will retrieve the sensor reading from the shared memory and use the function provided by the API library to convert it to the generic format (or sensor measurement) for use. In the actuation case, the control task computes actuation data in a generic format and then uses a function from the API library to save actuation data to the shared memory in a format depending on the actuator. The Data Processor then sends the actuation data to the actuator interface. The reasoning behind this implementation, in which we provide a library instead of translating the data inside of the Data Processor, is that we want to make the most processor time available to Giotto tasks, in keeping with the Giotto framework, while minimizing the other processes outside of Giotto. We chose to implement the shared memory discussed above as a circular buffer because it enables smooth communication between the Data Processor, which could be asynchronous depending on the sensors, and the Control task in Giotto, which is synchronous. It allows simultaneous writing of data to and reading of data from the shared memory without causing problems.

![Diagram of Virtual Avionics Platform Implementation](image)

**Figure 5: Hardware-in-the-Loop Simulation**

We chose to allow the embedded control programmer to decide on the details regarding the Kalman Filter that is needed to blend the INS and GPS data. For this reason the Kalman Filter is not included in the VAP implementation. Therefore, the control programmer should plan to include at least the following tasks: Control, which takes the sensor measurement and computes a control according to the control law, and Measurement Fusion, which takes the sensor measurement from the GPS and INS and computes the single position measurement.

In the implementation of the Simulator, the plant model is represented as an independent process. The input and
output of the plant model are implemented as shared memory, again using circular buffer. The GPS, INS, and actuator are each implemented as an independent process that writes to or reads from the plant input and output, and communicates independently through its serial port to the Control Computer. This implementation structure mirrors the actual behavior of the sensors and actuators.

In the view of Platform Based Design, we mapped the function and behavior of the Virtual Avionics Platform, consisting of the Language Platform and the Device Platform, into an architecture of the VAP itself, consisting of software components such as shared memory and the Data Processor. Our particular implementation, or mapping, is a platform instance and just one way of realizing the platform in actual hardware and software.

3.2 Design in Use – an Example
We here present a simple example to show exactly how the API is used by the control process in Giotto and how the platform implementation accomplishes the necessary tasks. We start with the controller’s point of view. The Giotto program links C functions to the Giotto structures: sensor ports, drivers, tasks, and actuator ports, and then runs the linked internal structures. The control process first assumes that the correct measurements from the GPS and the INS are waiting in shared memory. We have written C functions to access the shared memory. In the Giotto program we link these functions to the sensor ports. If a more detailed sensor model is incorporated in the future, the measurements will need to be transformed. In this case the C library functions would be called. With these two installments of the API: the measurements located within easy access and the available functions to transform the measurements, the control process in Giotto was simple to implement.

In our implementation of the VAP we used a very simple example sensor format instead of all the details of the sensor and actuator models. As a result, there is currently no need for the Measurement Fusion task to sit on top of the platform or for library routines for data format translation that are described above; however, we have implemented the framework for adding these provisions.

4. Conclusion
In this project the platform-based design methodology has been extended to the domain of autonomous helicopter system development. We have presented both the new design methodology and an implemented example using a hardware-in-the-loop simulation. The final implementation consists of the simulation process running in C on the Simulation Computer, the Controller task running on top of Giotto on the Controller Computer, the Data Processor running in C on the Control Computer, all of the shared memory blocks that are part of the platform, and the framework for the library that is also part of the platform. The result of this work does exhibit the desired effects of the new design method. For example, with this design if one wanted to move to a new hardware implementation, such as a new sensor, no changes in the control program are required. In the case of using a new helicopter, little change in the control program beyond what is needed to accommodate the new dynamics is needed. The platform instance would have to be adapted in each case by changing the data collecting process to expect the new data types. The library functions would also need to be updated in order to handle the varied data. With these changes in place in the platform, the controller could immediately be switched onto the new underlying system. The controller process could be exchanged without any changes assuming that all controllers adhere to the API of the Virtual Avionics Platform. These new features of the

![Figure 6: Virtual Avionics Platform from a dataflow point of view](image)
embedded software systems are invaluable since the changes described above are common and are much more troublesome without such a platform.

Figure 7: Visualization of the VAP and Hardware-in-the-loop Simulator system at work

Figure 8: Nested Platforms

5. Future Work

The eventual goal of the hardware-in-the-loop simulator is that the Control Computer would fly a physical UAV. In order to make this transition we would need to add more details to our first-run design example in order to carefully mimic the exact behavior of each of the sensors. We would then need to implement the Measurement Fusion Giotto task that we have described. We would also need to add other flight details such as communication with a ground station, and incorporation of other sensors that are only used during maneuvers.

Another interesting application of platform-based design in the area of UAVs is to consider the low level controller part of the architecture and consider a high level controller to be sitting on top of the platform. This idea utilizes the hierarchical nature of the platform-based design methodology. In this scenario, a new type of platform would have to be defined to bridge the low and high level UAV controllers.

References