A Plug and Play GNC Architecture Using FPGA Components

K. KrishnaKumar, J. Kaneshige
NASA Ames Research Center, Moffett Field, CA, 94035

R. Waterman
NASA Kennedy Space Center, Cape Canaveral, FL

C. Pires, C. Ippoloito
NASA Ames Research Center, Moffett Field, CA, 94035

The goal of Plug and Play, or PnP, is to allow hardware and software components to work together automatically, without requiring manual setup procedures. As a result, new or replacement hardware can be plugged into a system and automatically configured with the appropriate resource assignments. However, in many cases it may not be practical or even feasible to physically replace hardware components. One method for handling these types of situations is through the incorporation of reconfigurable hardware such as Field Programmable Gate Arrays, or FPGAs. This paper describes a phased approach to developing a Guidance, Navigation, and Control (GNC) architecture that expands on the traditional concepts of PnP, in order to accommodate hardware reconfiguration without requiring detailed knowledge of the hardware. This is achieved by establishing a functional based interface that defines how the hardware will operate, and allow the hardware to reconfigure itself. The resulting system combines the flexibility of manipulating software components with the speed and efficiency of hardware.

I. Introduction

A scalable plug-and-play software and hardware architecture that can be adapted for GNC of small autonomous robotic surface vehicles as well as for large autonomous and human piloted transport vehicles is the goal of the NASA plug and play avionics research. As a result of PnP, new or replacement hardware can be plugged into a system and automatically configured with the appropriate resource assignments. An intelligent plug-and-play avionics (*iPapa*) system can provide critical capabilities for different space exploration missions. Many exploration missions involve the guidance, navigation and control (GN&C) of different systems and their sub-systems. Designing individual avionics solutions for each of these systems substantially increases the overall program costs. A plug and play system that can support the different GN&C requirements of many of these systems in their different flight/ground phases provides an attractive affordable solution to this problem. Besides working across different systems, the *iPapa* system can have on-board intelligence with verifiable-adaptation of the controller for unforeseen system failures and changes in mission. This provides an important analytical fault-tolerant capability that can be augmented with hardware fault-tolerance by using devices such as field programmable gate arrays (FPGAs).

A plug-and-play architecture for real-time intelligent avionics is a critical capability for the crew exploration vehicle (CEV) as well as for other human tended non-terrestrial surface elements defined as part of the human and robotic exploration missions. Modular reconfiguration of a system is well understood, from a mechanical perspective; however, often the avionics and software design is much less modular and reconfigurable. This design artifact leads to larger operations costs and inflated software development, and sustaining engineering to update and reverify, validate and certify the software changes due to reconfiguring the hardware.

Modular FPGA-based avionics with flexible control software can also be used in various robotic applications. They can serve as single-axis motion controllers with sensory feedback and fault tolerant computing capabilities. They can be located close to each actuator to reduce complex cabling costs. They can also serve as controllers for robotic arms coordinating several degrees-of-freedom. All these components require the acquisition of various sensory information, the fusion and processing of that knowledge, and the generation of behaviors that result in either estimated knowledge or system motion. However, to operate reliably in various environments and under different system configurations, these controllers must be able to learn and adapt to changes in their environment. These changes can range from large variations in the load experienced by manipulator joints, variations in terrain roughness for mobility platforms, variations in lighting condition for vision applications, or variations in the quality of the measurements from different fidelity sensors.

In the sections that follow we present the concepts of *iPapa*, the use of FPGAs, and the application to intelligent GNC.

II. Plug and Play Concept

Plug and Play as a computer jargon was introduced as a functionality of the Windows 95 operating system. Ever since then, PnP has stood for components that can be hot-swapped between systems, via an automated service discovery system, without manual setup by the user.

The *iPapa* system proposed here involves two elements, a plug-n-play component and an intelligent plug-n-play architecture that supports the component. The plug-n-play architecture must be able to recognize the components and establish communication, requiring a level of standardization be imposed on the architecture that specifies the communication protocol, data bus, power requirements, physical interface requirements, etc. Protocols must also be established to allow the necessary modifications to the avionics system as well as the plug-and-play modules to accommodate a previously unknown module configuration in a wide spectrum of possible platforms such as rovers and spacecraft. For instance, sensor information and actuator configurations will be often radically different on each platform. Standardization requirements definitions will establish the communication protocols to allow this information to be passed between the system and modules, and must define the tradeoff between flexibility and complexity of the architecture to ensure usability without curtailing functionality.

Figure 1 presents a decentralized *iPapa* architecture along with the GN&C components. The component, once "plugged-in", publishes itself as a *iPapa* GN&C component and subscribes to the required input variables. It also publishes the output variables for use by the system. The required GN&C components are downloaded form memory into the FPGA chip to enable reconfiguration and reprogramming capability.

The database of the GN&C functional elements will consist of needed guidance algorithms, navigation algorithms, and control algorithms that will be used by the reconfigurable plug and play component as functional blocks [1]. The *iPapa* architectural manager will use these functional blocks for constructing the GN&C system on the reconfigurable plug and play component knowing the system and its operational phase. Figure 2 illustrates the relationships between the *iPapa* architectural manager, the guidance, the navigation and the control component of an avionics system and their interaction with the physical system. The *iPapa* architectural manager identifies the physical system as well as the flight/ground operational phase of the system. It then uses this information to reconfigure the individual GN&C components for that particular system and its phase of operation.

The database of the GN&C functional elements will consist of all the different guidance algorithms, the navigation algorithms, and control algorithms that will be used by the reconfigurable plug-n-play component as functional blocks. The *iPapa* architectural manager will use these functional blocks for constructing the GN&C system on the reconfigurable plug-n-play component knowing the system and its operational phase.

A. Phased Approach to *iPapa*

The *iPapa* architecture is planned to be developed over three years in three different phases. In Phase A (See Figure 3A), multiple control algorithms will reside in a FPGA-based hardware component that can be reprogrammed on the fly. In Phase B (Figure 3B), Multiple (complex) control algorithms residing in a processor that can be uploaded to an FPGA on-the-fly, for reconfiguration will be developed. In Phase C (Figure 3C), reconfigurable FPGA capability in a decentralized framework will be developed.

III. Plug and Play using FPGAs

Field Programmable Gate Arrays (FPGAs) are programmable devices that have reconfiguration capability that can be used for fault recovery or performing multiple tasks. The big benefit of FPGAs is the ability to do "on-the-fly" reconfigurability. Another benefit of FPGAs is the time needed to fabricate special purpose chips (typically 3 months) is eliminated. Since the popularity of FPGAs is phenomenal, many off-the shelf tools are available for rapid prototyping. For example, tools are readily available for FPGA design using Matlab and C languages. This enables GNC specialists to prototype hardware solutions in a rapid manner.

In addition to the above benefits, today's FPGA gate densities and increased clock speeds reduce the performance gap between FPGAs and fixed silicon processors. The configurability and flexibility of FPGAs allows for more parallel processing in hardware vs. the traditional serial processing paradigms used by single fixed silicon chips.

Another advantage posed by reconfigurability is in fault-tolerance of FPGAs [2,3]. Figure 4 depicts a typical failure scenario that the FPGA can encounter. After being partly damaged by an external or internal source, the FPGA can be reprogrammed using its undamaged gates. This capability can be used to correct latent design errors as well as on-chip and off-chip failures. Such autonomous repair capability provides an alternative to device redundancy that offers potential weight savings in NASA missions. At the same time the characteristics of all the different failures need not be diagnosed in order to initiate repair.

IV. Plug and Play GNC

In an accompanying paper at this conference [1], we have presented a survey of GNC algorithms for space missions to low-earth orbit as well as missions to moon and back. Surveying the GN&C problems in various flight phases bring out the requirements for the GN&C database needed for designing the *iPapa* component. This database can be classified in terms of different block-sets.

A. Guidance Block-set

The guidance algorithms in the different flight phases reviewed consist of

- ❖ algorithms supporting thrust vector control where the required velocity for the maneuver is computed. The required velocity vector and the estimated vehicle velocity vector provide the velocity to be gained vector. The guidance algorithm then commands an angular velocity for aligning the thrust vector with the velocity to be gained vector.
- ❖ earth entry phase algorithm where a reference lift/drag profile is chosen and the guidance algorithm provide open-loop and closed loop angle of attack and bank angle commands for maintaining this drag profile. This algorithm is general enough to be applicable for any atmospheric planetary entry.
- ❖ earth boost phase algorithm, when a table-lookup based logic is used for providing the commanded values of angle of attack and side slip as a function of the vehicle velocity
- ❖ steering algorithm that smoothes the commands given by the guidance algorithm and provide desired commands to the control architecture.

B. Navigation Block-set

The navigation algorithms in the different flight phases reviewed consist of:

- ❖ state estimation algorithms that take inputs from different sensors to provide the state of the vehicle. These consist of the extended Kalman filter like algorithms that take inputs from IMUs and rate gyros. Vehicle attitude is also given by sensors such as the star trackers, sun and horizon sensors.
- ❖ sensor failure detection and identification algorithms that compare the inputs of redundant sensors and provide the appropriate sensor input to the estimation algorithms.
- ❖ vision processing algorithms that can compute relative range, range rate and attitude of the target spacecraft during AR&C flight phase
- ❖ general digital signal processing algorithms that act as static filtering algorithms for various sensors.

C. Control Block-set

The control algorithms in the different flight phases reviewed consist of

- ❖ the PID control algorithm that is implemented in the shuttle earth boost phase for commanding the SRB deflections and in the shuttle entry phase for commanding the aerodynamic surfaces.
- ❖ phase plane logic that is used to fire the reaction jets in the on-orbit phase, in lunar descent and ascent phases as well as in the earth entry flight phase.
- ❖ thrust vector control laws that consist of digital compensation filters as used in the on-orbit maneuvers and in the lunar descent phase.

- manual control laws such as the rate command-attitude hold algorithm implemented for the LEM manual attitude control design.
- control allocation and blending logic

D. Coordinate Block-set

One of the most important functional forms in implementing any of the GN&C architectures is coordinate transformation. The space shuttle and the Apollo missions use several different coordinate systems such as the body axes system, earth-fixed system, the inertial system, the LVLH system, the lunar-fixed system, the lunar approach guidance coordinate system and so on. The GN&C database therefore needs to include a coordinate block-set that provides support for all its algorithms in various flight phases.

V. Plug and Play Adaptive Control

One of the required characteristics of *iPapa* is the ability of the component to be fault-tolerant. In addition to this, it is also required for the component to compatible to a wide variation in system characteristics. This could be achieved via a system identification process or by using robust/adaptive capability in the GNC components. The approach taken in the *iPapa* component is to use direct adaptive control to achieve both fault-tolerance and robustness to system variations. The basic architecture that provides the robust/adaptive charctaritic is based on an intelligent flight controller developed at NASA Ames Research Center [4,5].

In Figures 6A, 6B and 6C, we present the adaptive control application to three different realistic test platforms: These include: (1) Exploratory Air Vehicle; (2) Docking mechanism; (3) Robotic rover. The architecture highlighted as *iPapa* in the figures is explained next.

P + I Error Controller: Errors in system responses can be caused by inaccuracies in system variations and model inversion. Unidentified damage or failures can also introduce additional errors. In order to achieve desired performance, a proportional-integral (PI) error controller is used to correct for errors detected from sensory feedback.

Learning Neural Network: The on-line learning neural networks work in conjunction with the error controller. By recognizing patterns in the behavior of the error, the neural networks can learn to remove biases through control augmentation commands. These commands prevent the integrators from having to windup to remove error biases. By allowing integrators to operate at nominal levels, the neural networks enable the controller to provide consistent performance. The learning neural networks not only helps control the nominal system, but also provides an additional potential for adapting to changes in system dynamics due to control failures or damage.

Dynamic Inversion Generation/Static NN: The dynamic inversion element converts the summed response commands into virtual control surface commands. Dynamic inversion is based upon feedback linearization theory. No gain-scheduling is required, since gains are functions of system characteristics and sensor feedback. Several methods are available to accomplish approximate model definition: simple linear model methods, nonlinear tables or using pre-trained neural networks (non-changing) to provide estimates of system characteristics. The model is then inverted to solve for the necessary control commands.

VI. Challenges Faced

Many challenges exist to winning acceptance of this intelligent reconfigurable PnP system design paradigm over the more traditional static avionics system. One of the most difficult challenges will be finding an approach that can verify/validate that control capability can be maintained despite the dynamic addition and removal of support services. This approach must be built on an abstraction technique that can effectively hide hardware details from one system to another and an interface that is well tested. This approach however will not be enough to tackle the criticality and certification issues. In order to guarantee that a support service provided by one module does not impede a critical service on another module, multiple data buses supporting a hierarchy of service levels will be required. In this way the dynamic behavior can be isolated and verified in more manageable sizes. The hierarchy of services envisioned may be tiered to support the following critical functions: Control, Life Support, and Mission. A fourth tier may be needed to address non-critical operations. The service classes associated with each of the tiers are as follows:

- ➢ Control: Maximum failure response time is typically on the order of seconds. This would apply primarily to GN&C, Power, Propulsion, and Command & Control.
- ➢ Life Support: Maximum failure response time is generally on the order of minutes. This would apply primarily to Environment Control, Life Support, and Communications.
- ➢ Mission: Maximum failure response time is usually on the order of hours. This would apply primarily to Science Instruments, Data Record & Retrieval, and Human Support Equipment (Galley, Waste Management).
- ➢ Non-Critical: No maximum failure response time. This would apply mostly to Crew Information Network (email, electronic documentation, multi-media).

Using this tiered approach each service class would perform the same service discovery process to determine the capabilities offered by each peer within its class. From a certification standpoint an exhaustive verification would not be required to prove that the communication system does not impact the propulsion system. Likewise it would not have to be proven that the Waste Management System does not affect the communications system.

Another challenge will be differentiating between when a system has failed and when it has been powered down or removed. This differentiation is an important consideration when forming an appropriate redundancy management strategy and determining a set of corrective actions that can recover a failed function. An approach to solving this challenge can be found in several emerging health management algorithms that can learn the behavior of the hardware over time. This ability to learn generally requires a large data set from which prognostic and diagnostic information can be derived.

VII. CONCLUSIONS

There are several difficult challenges that must be met before an intelligent reconfigurable PnP avionics system can be developed. Automated Service Discovery technologies, which have significantly improved the ability to dynamically reconfigure a personal computer, must be evolved to meet the stringent requirements of a critical space-rated avionics architecture. Each challenge must be satisfied by a solution that can be verified, validated and certified. Developing a tiered architecture will provide a way to isolate service criticalities as will strong adherence to abstraction and interface requirements. Although each challenge must be addressed prior to implementing a dynamic reconfigurable avionics system, success in these efforts will be required to enable an affordable, sustainable and safe transportation system that can deliver crew and cargo from the surface of the Earth to exploration destinations beyond

VIII. References

1. Kulkarni,, N., KrishnaKumar, K., Spacecraft guidance, navigation, and control requirements for an intelligent plug-n-play avionics (PAPA) architecture, AIAA Infotech@Aerospace Conference, Washington DC< Sep 26-29, 2005.

2. Lohn, J.D., G. Larchev, R. DeMara, "A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGAs," *Intl. Conf. on Evolvable Systems*, March, 2003.

3. Lohn, J.D., "A Coevolutionary Genetic Algorithm for Autonomous Fault-Handling in FPGAs," *Intl. Conf. on Military and Aerospace Programmable Logic Devices*, Laurel, MD, September 10-12, 2002.Kaneshige, John and Gundy-Burlet, Karen, Integrated Neural Flight and Propulsion Control System, AIAA-2001-4386, August 2001.

4. KrishnaKumar, K., Limes, G., Gundy-Burlet, K., Bryant, D., *An Adaptive Critic Approach to Reference Model Adaptation.* AIAA 2003-5790, August, 2003.

5. Kaneshige, John, John Bull, and Joseph J. Totah, *Generic Neural Flight Control and Autopilot System*, AIAA 2000-4281, August 2000.

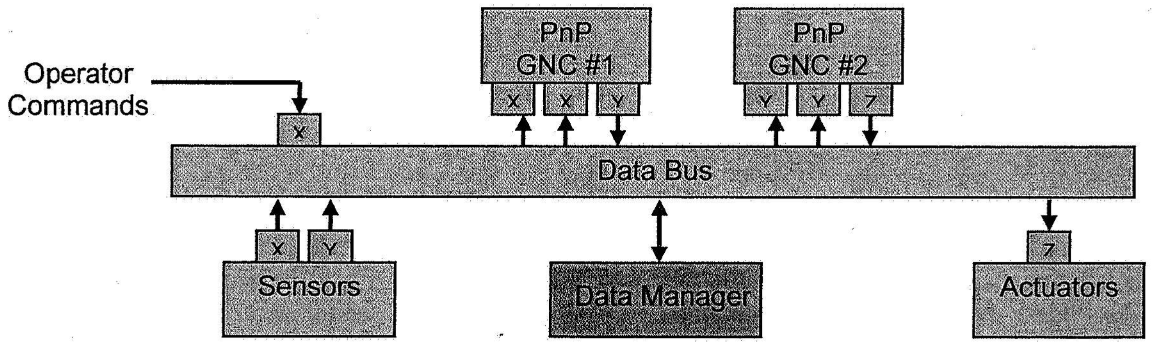


Figure 1. A Decentralized *iPapa* Architecture

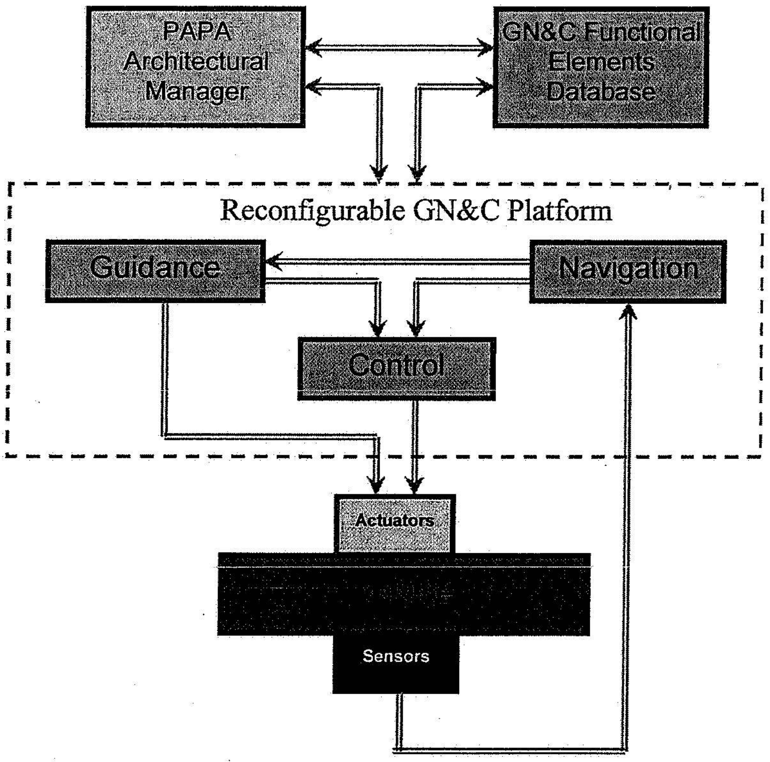

PAPA Architectural Manager
GN&C Functional Elements Database
Reconfigurable GN&C Platform
Guidance
Navigation
Control
Actuators
Sensors

Figure 2: *iPapa* Architectural Flow

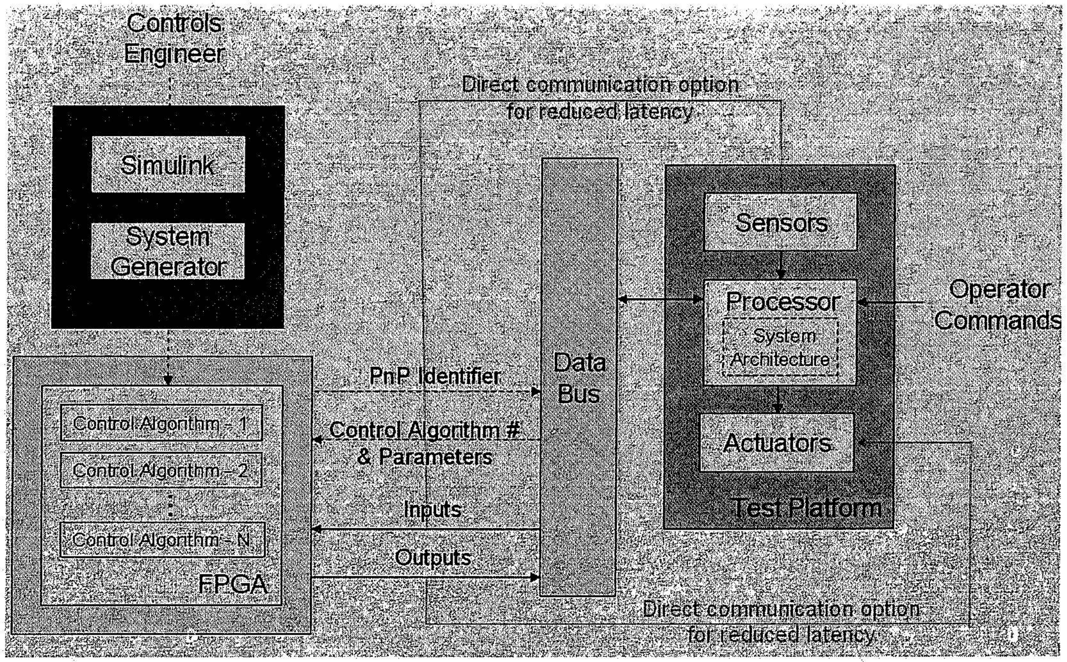


Figure 3A. Phase A of the *iPapa* Architecture

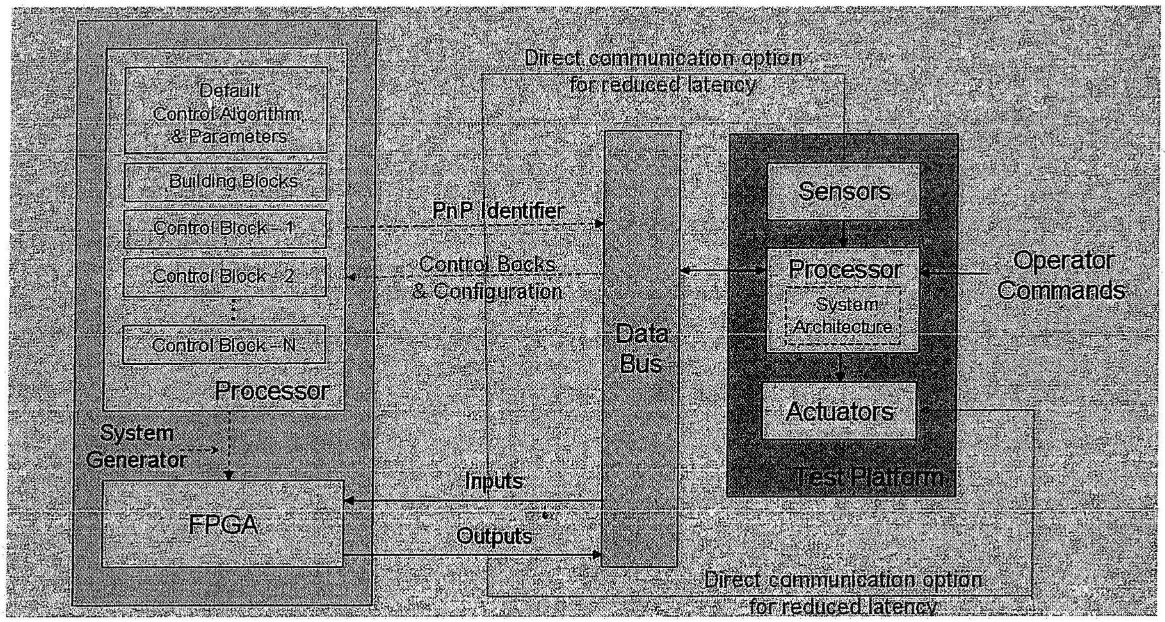


Figure 3B. Phase B of the *iPapa* Architecture

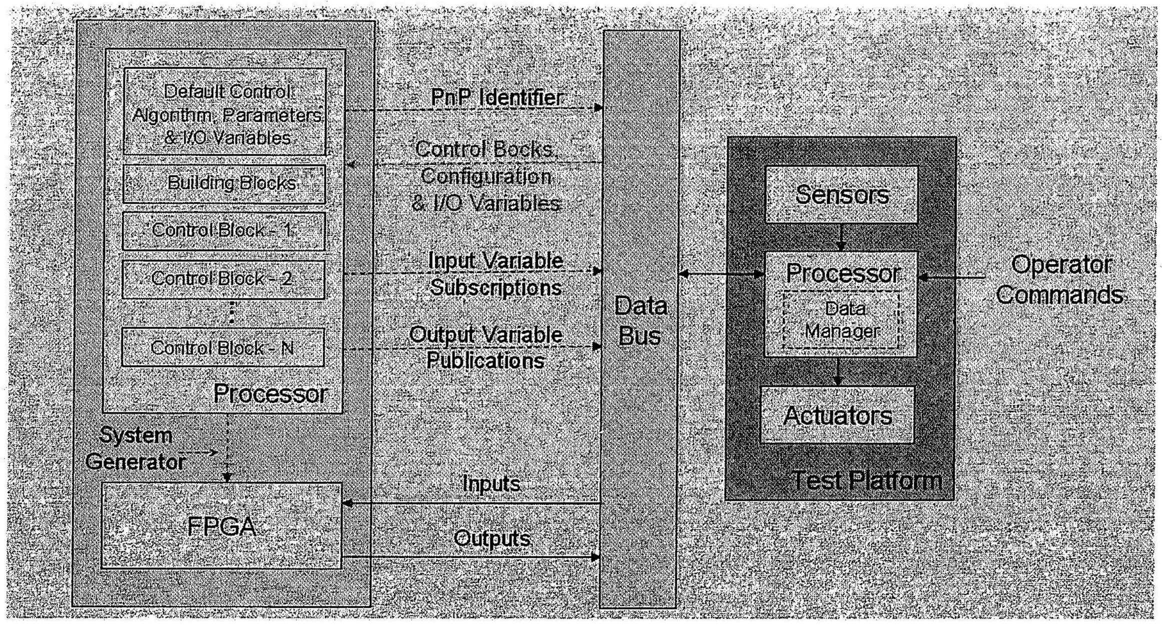
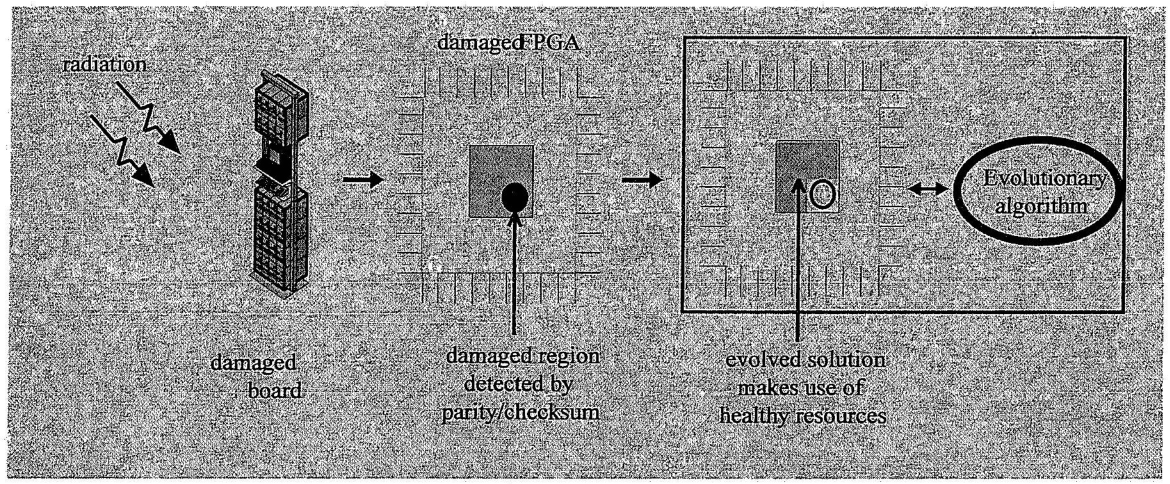


Figure 3C. Phase C of the *iPapa* Architecture



radiation
damaged board
damaged FPGA
damaged region detected by parity/checksum
evolved solution makes use of healthy resources
Evolutionary algorithm

Figure 4. Fault-Tolerant FPGA

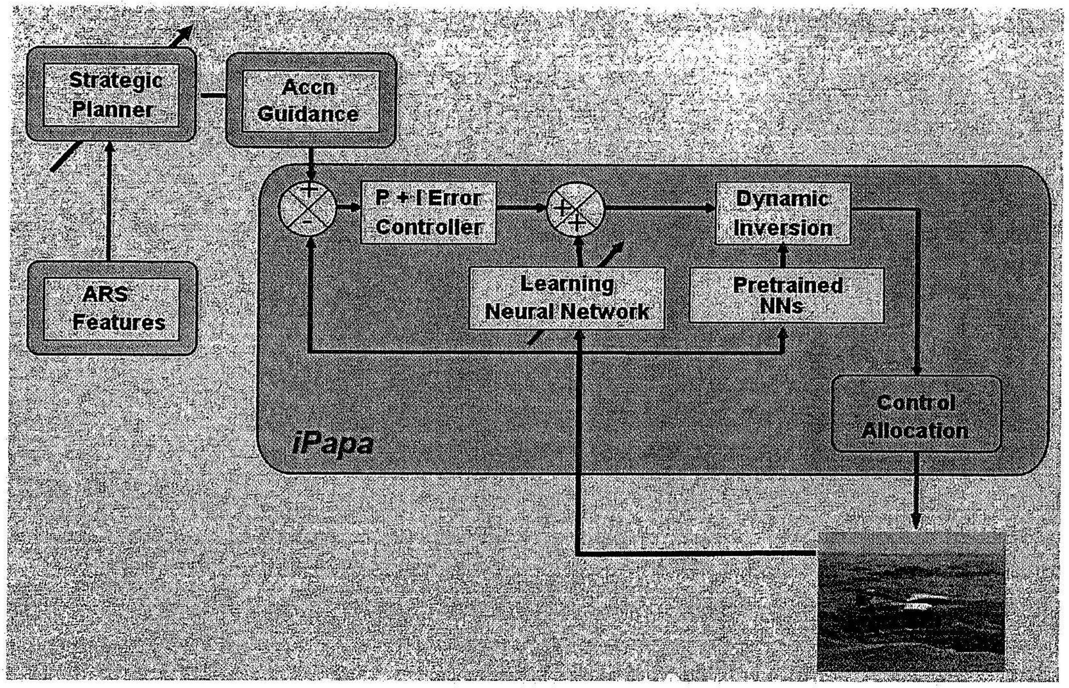


Figure 5A. Adaptive Control for an Exploratory Air Vehicle

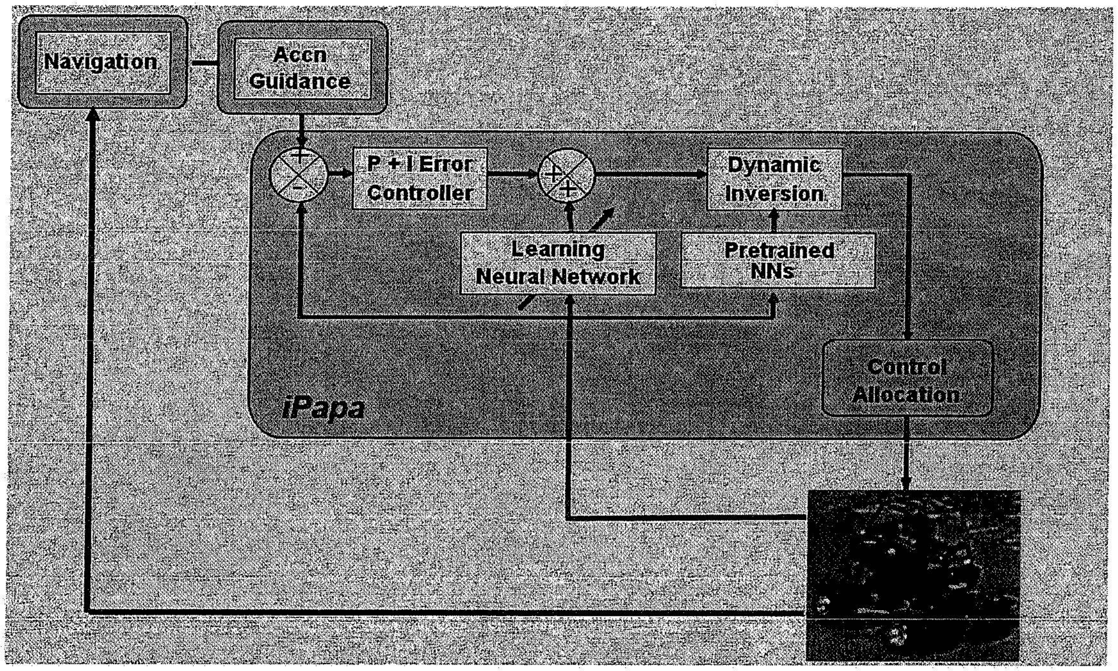


Figure 5B. Adaptive Control for a Rover

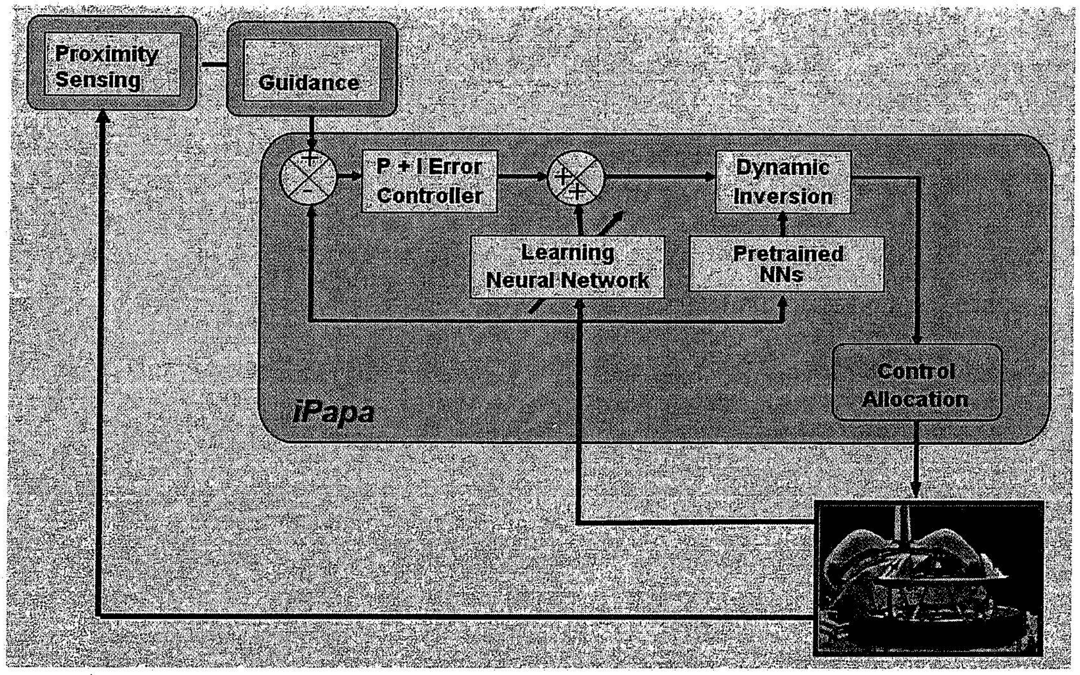


Figure 5C. Adaptive Control for a Docking Mechanism