Reconfigurable Intelligent Control Architecture for small scale unmanned helicopter

Vishnu Kumar Kaliappan,† Hanmaro Yong,‡ Min Dugki,§ Eunmi Choi,‖ and Agus Budiyono,¶

† Associate Professor, (T) +91-98942-50056 (F) +822-458-1395, vishnudms@gmail.com
‡ Master Student, (T) +822-450-3490 (F) +822-458-1395, maro1076@gmail.com
§ Professor, (T) +822-450-3490 (F) +822-458-1395, dkmin@konkuk.ac.kr
‖ Professor, (T) +822-450-3490 (F) +822-458-1395, emchoi@kookmin.ac.kr
¶ Assistant Professor, (T) +822-450-3817 (F) +822-444-6670, agus@konkuk.ac.kr

† Department of Computer Science and Engineering, Jansons Institute of Technology, Coimbatore, India.
‡ School of Computer Science and Engineering, Konkuk University, Seoul, Korea.
§ Department of Aerospace Information Engineering, Konkuk University, Seoul, Korea.
‖ School of Business IT, Kookmin University, Seoul, Korea.

Key Words: Rotorcraft-based UAV, Architecture, reconfigurable control, Fault detection and identification, Obstacle avoidance, Auto-Landing control, Point Navigation control

ABSTRACT

In the past decades there has been substantial research undergone in design of intelligent architecture for Rotorcraft based Unmanned Aerial Vehicle (RUAV). Designing intelligent architecture is a challenging problem since future RUAV’s are utterly autonomous and their performance is comparable to that of manned vehicles. This paper deals with design and development of a layered architectural framework which addresses the issue arising in autonomous intelligent control system. The architecture consists of two layers: the high-level layer occupied by planning routines. In this level the way-points, mission tasks from command center is executed. The low-level layer function is to stabilize the flight and follow the commanded trajectory from upper layer. These layers integrate the following functionalities: (a) way-points navigation and control

* Corresponding author

Copyright 2012 by the American Society of Civil Engineers

which includes auto-landing (b) obstacle detection and avoidance, (c) fault detection and identification, and (d) system reconfiguration in two levels (high level and low level controllers). The resulting layered architecture is discussed in detail. Moreover, the novel fault detection and identification method is developed to address multiplicative and additive faults. The testing environment for RUAV is developed to validate our architecture. Complete setup is carried out and run under QNX RTOS, based PC104 embedded board. The algorithms are tested and evaluated using HILS (Hardware-in-the-Loop simulation). Simulation result proves that the proposed architecture demonstrates desired efficiency and reliability.

**NOMENCLATURE**

- $P_f$: Future position
- $u_S$: Forward velocity
- $W_p$: Next target way-points
- $W_s$: Start point
- $W_t$: Target point
- $V_d$: Desired side velocity
- $u_d$: Desired forward velocity
- $l_d$: Detection box length
- $(\tilde{\xi}, \tilde{\eta})$: Intersection points
- $l_f$: Detection box length
- $B_f$: Braking force
- $\psi_d$: Desired angle
Future north position

Future east position

Distance between obstacle and helicopter (north, east)

**INTRODUCTION**

Rotorcraft-based Unmanned Aerial Vehicles (RUAV) are essentially useful not only for military application, but also for civil purpose (Ollero and Merino 2004) like Exploring unknown environment, Mapping the terrain (Lacroix and Jung 2002), Cooperative fire detection (Meriono et al. 2006), Autonomous deployment of sensor network for civil monitoring system (Corke et al. 2004) and more. In many applications the usefulness of the aerial vehicles are the best way to get information or to deploy equipment without any interaction of human. The challenge of developing such control system not only deals in the development of control algorithms, but also in the approaches and techniques used to integrate those control algorithms on the actual system. To accomplish such system successful there must be highly efficient intelligent infrastructure for autonomous control systems. Hence our objective is to develop efficient framework which is capable of integrating autonomous functionalities like obstacle detection and avoidance control, fault-tolerant control, system reconfiguration control, way-points navigation including auto-landing control and multiple-UAV coordination control.

Various intelligent architecture methods have been investigated in the past to address the problem arising in integration of control algorithms. Open Control Platform (OCP) is one of the most reliable method proposed by researchers at Georgia Tech. OCP is used to compensate in-flight failure of a low-level flight control system. It is achieved by reconfiguring the software-enabled control (SEC) system autonomously (Toon 2002). The architecture increases the level of autonomy of the system by integrating five functionalities: fault detection and identification, active system restructuring, reconfigurable flight control, reconfigurable path planning and mission adaptation.
The fault detection and identification algorithm identifies the probable fault in the system and issue fault declarations. When fault exceeds the level of flight controller that cannot manage, then the system actively restructures and selects one of multiple reconfigurable flight controllers. The reconfigurable controller uses adaptive neural network strategy. Reconfigurable path planner employs an adaptive model of the vehicle to re-assign the flight path. Finally, a mission adaptation component estimates limitations on the closed loop performance of the aircraft and adjust the aircraft mission accordingly.

Boskovic 2002 has addressed four-layer hierarchical architecture. These layers integrate the following functionalities: Fault-tolerant redundancy management, trajectory generation, path planning and Decision making. Firstly the operation of redundancy management layer is to follow the desired trajectory in the presence of different disturbance, failures and upsets. This layer includes fault identification system and reconfiguration module which response to different faults by reconfiguring the controller. In his work Multiple Models, Switching and Tuning (MMST) method has proposed to identify the fault and to reconfigure the faulty system. Second layer is trajectory generation layer which deals with generating optimal trajectory to the assigned way-points. Path planning layer is the third layer in the architecture, the role is to generate way-points for the overall missions and compute spatial constrains needed for trajectory generation layer. Finally, the decision making layer has the information about the overall mission objectives. The sensory information and situational awareness is used to make the decisions among the missions.

Clements 2003 developed a fault tolerant architecture which has the capability to expand both vertically and horizontally. In his architecture, fault detection and fault isolation occurs at the top most layers. Fault isolation includes restructuring to prevent spreading the fault to working components. The middle layer conducts three tasks to optimize the system which response to fault. Firstly, the redistribution controller restructures the interconnection to optimize the control. Then based on the information from redistribution controller, the set-point controller adjusts the set-points of the low-level controllers. Finally, the low level gain controller adjusts the feedback gains
Napolitano et al. 2000 proposed a neural network based fault control for sensor and actuator faults. His fault-tolerant flight control system integrates sensor and actuator failure detection, identification, and accommodation (SFDIA and AFDIA). Initially the main neural network (MNN) was created to detect the fault in sensor by set of \( n \) decentralized neural network (DNNs). The main theme is to enhance the handling capability of soft failure as well as addressing the issues of integrating the SFDIA and AFDIA schemes without the degradation of performance in terms of false rate and incorrect failure identification. The bank of neural scheme which has been used may degrade the system performance and system efficiency when it is applied to small scale control system.

Friedland and Bernard 1982 developed likelihood failure detection algorithm. Based on the fault free observer design, the sensor data are compared with fault free data. The faults are detected once if the difference of both values exceeds the threshold. This method is applied to detect the faults in gyro and accelerometer vertical axis of an aircraft flight control system. Guillermo Heredia et al. 2009 developed fault detection and identification (FDI) method for multiple-UAV. Both DGPS and inertial sensors are used for sensor FDI in each of UAV. This method uses one additional FDI system which makes use of image from planar scene taken from two different UAVs. By this method the UAV, noise level is reduced in the process of detection position faults. Observer based fault detection and isolation method is proposed by Chen et al. 1997. He uses disturbance principle for fault diagnosis. To detect the fault the model based fault detection method has been utilized. Few approaches in development of model base fault detection are discussed in Isermann et al. 1997.

Fault tolerant distributed system based on recursive algorithm for fault tolerant is proposed by Pham et al. 1991. He discussed how to determine the design policies when the objective is to minimize the average cost given to each processor and cost of the system failure. Agrawal 1985 has developed an algorithm called recursive algorithm for fault tolerance (RAFT). The RAFT system...
basic idea is to schedule the incoming job to a pair of processors. Each processor then generates a result and a signature. The two signatures are compared, if the signatures are matched, then the result is given to the user. If not process schedule the job on a processor and comparing the generated signature continues until a matching pair of signatures has been found.

Bateman et al. 2007 developed a fault tolerant and isolation scheme build with bank of linear quadratic controllers. The proposed scheme is based on a signal processing approach. Once the fault has identified by the fault detection module, each control surfaces is stimulated with a specific signal which represent its signature. The amplitude of the signature disrupts the state variable without increasing the effect of the failure. The signatures are then isolated which represent the faulty control surface. Magrabi et al. 2000 proposed parity based fault detection and diagnosis (FDD) scheme, which is applied to decentralized architecture developed for unmanned aerial vehicle (UAV).

Among these methods, Observer-based and parameter based fault detection techniques are most popularly used. Furthermore, fault tolerant reconfiguration controller is classified into two approaches, namely Multiple-model approach and Adaptive control approach. In Multiple-model approach, bank of dynamic model connected in parallel which is used to control the system while in normal and various faulty conditions. For each of the dynamic model the corresponding model is connected in parallel. In the faulty condition the suitable model are switched using the switching mechanism Maybeck et al. 1989. This gives a robust performance under variety of fault condition.

Adaptive Control Approach utilizes an adaptive controller to ensure the stability and performance of the system under rapid change in system parameters. Generally, this approach is classified into two, indirect adaptive control method and direct adaptive control method. First method employs a parameter isolation process and second method does not require any isolation process. Boskovic 1998 implemented a reconfigurable control for Tailless advanced fighter aircraft (TAFA) using multiple models, switching and tuning (MMST). The system consist collection of
plant dynamic models with a fixed pseudo-inverse controller for each model. In the case of fault the controller selects a model which is closely approximate to the actual model and activates the corresponding controller. The assumption in this method is that at least one model which is implemented in the system should be close enough and the proper switching mechanism is applied.

Comparing the efficiency and reliability, aforementioned methods addresses only fault tolerant architecture. In this research, we addressed integration approach for intelligent control algorithms and in addition to that multi-RUAV control operation is also investigated. Following that the Hardware-in-the-Loop system are developed to demonstrate the proposed system.

RECONFIGURABLE INTELLIGENT CONTROL ARCHITECTURE

In this section, the proposed RICA (Reconfigurable Intelligent Control Architecture) for rotorcraft based unmanned aerial vehicle is briefed. The architecture block diagram is depicted in Fig. 1. The multilayered architecture consists of two levels (a) High-level: Guidance and navigation Control (GnC) in which planning routine, the way-points, mission tasks from command center are executed. (b) Low-level: Reconfigurable flight control stabilizes the flight and follows the command trajectory from upper layer. Each of the layers is described below.

High-Level: Guidance and Navigation Control (GnC)

The Guidance and navigation component translates a high level mission task to low-level task. The mission can be accomplished as a sequence of action like fly to a desired destination and hover, while flying maintain same velocity, avoid obstacle among the way-points and coordinate with other RUAV’s. Flying to a desired destination is achieved with way-point navigation and control (Kim and Kim 2006), where to avoid the obstacle biologically inspired steering behavior control for obstacle avoidance (Kaliappan et al. 2011) is utilized and to coordinate with other RUAV’s our previously developed novel approach Linear velocity based predictive control has been used.

Unmanned aerial vehicle requires certain mechanism to identify potential thread if it is to make any attempt to avoid it. Sometimes the possible obstacle may be avoidable through onboard
information such as terrain altitude and some known threat. For real time maneuver unknown obstacle may appear, the mechanism should respond to these kinds of obstacle even though it’s trivial. The obstacle detection and avoidance control system is subdivided into two: obstacle detection algorithm and obstacle avoidance algorithm. Obstacle detection algorithm traces the obstacle coordinates which possibly cascade on the RUAV’s path. The navigation module generates the trajectory where the RUAV assigned to fly. The flying location is predefined by the set of way-points from ground control station. During the maneuver time, it is necessary to maintain a local obstacle map to generate the obstacle avoidance trajectory. More detailed description of Obstacle detection and avoidance algorithm is briefed in Kaliappan et al. 2011.

**Behavior based Multiple RUAV coordination control**

Behavior-based decentralized multiple RUAV coordination approach is a part of steering behavior control. It allows an RUAV to carry out its own mission of flying to a specified region while the distances between RUAVs are maintained constantly to avoid collision. The main goal of the proposed controller is to make the RUAV cooperate among each other to achieve the defined task. Unmanned aerial vehicle requires intelligent control mechanism to communicate and control with each other if it to make any attempt to coordinate with other RUAV’s. In order to model such mechanism we utilize biologically inspired Reynold’s flocking model (Craig et al. 1987). In the following how an individual RUAV maneuvers based on the positions and velocities of its nearby RUAV is described. **Separation** is defined as “the tendency of an RUAV to move away from neighboring RUAV in order to avoid collision” and **Cohesion** is defined as “the tendency of an UAV to move towards the center of the flock that is formed by neighboring UAV”.

The algorithm traces the neighborhood RUAV coordinates which possibly cascade to its path. The way-point navigation module generates the trajectory where RUAV assigned to fly. The flying locations are predefined by the set of way-points from ground control station (GCS). While flying in the desired path the group behavior algorithm should find the possible RUAV which procured the path and calibrate the separation or cohesion strategy. More detailed description of
multiple RUAV coordination is briefed in Kaliappan et al. 2011 and Kaliappan et al. 2011.

**Low Level: Flight control**

The purpose of the low level controller is to stabilize the vehicle and force it to follow the commanded trajectory generated by the high level controllers. This is achieved by limiting the control vector with feedback gain. The forward speed hold system is realized by the following commands respectively.

\[
\delta_{lon} = k\theta \left( \theta_{cmd} - \theta \right) + k_\phi \phi
\]

\[
\delta_{lat} = k\phi \left( \phi_{cmd} - \phi \right) + k_\psi \psi
\]

(1) \hspace{1cm} (2)

While, the altitude and heading hold system is achieved by command control in the collective and rudder channels as follow

\[
\delta_{col} = k\dot{h} h_{cmd}
\]

\[
\delta_{rug} = k\psi \psi_{cmd}
\]

(3) \hspace{1cm} (4)

**FAULT DETECTION AND RECONFIGURATION CONTROL**

The Fault detection and reconfiguration control exploits an eccentric process to increase the safety of the vehicle in worse scenario. The system reconfiguration is attained by transforming the control vectors in the unmanned helicopter. Proposed work examines the control system in two prospects. First one is to ensure the possibility of reconfiguration. If the system fault is phenomenal, which cannot be reconfigured then the controller sends the query to switch the helicopter from auto mode to manual mode. The pilot from GCS (Ground Control Station) tries to stabilize the RUAV safely. The second prospect is to generate reconfigurable commands with the help of rule engine. The rule engine has the set of rules to generate for fault. Base on pre-defined rules the engine tries to generate commands.

Faults are event that take place in different parts of the helicopter. In general the faults are classified
according to the location of occurrence in the system, they are actuator faults, sensor faults and components fault. Actuator faults represent the partial or total loss of the control action. Sensor faults denote the incorrect readings from the sensor or failure of the sensor itself. Finally component faults occur if there is any change in internal component. Further, the faults can be modeled in two ways; they are additive faults and multiplicative faults. The additive faults can be modeled for component faults in the system, while the multiplicative faults are suitable for sensor and actuators. Fig. 2 depicts multiplicative and additive faults. The reconfiguration process for both additive fault and multiplicative faults are briefed in detail.

**Additive fault detection**

Additive fault detection and isolation method coupled with fault tolerant control architecture is developed in order to deal with component faults for a rotorcraft-based unmanned aerial vehicle (RUAV). The failure considered is malfunction with internal components of the helicopter which occurs during the maneuvers: rotor angular rate variations, etc. These faults lead from trivial to catastrophic damage of the system. The proposed fault detection and reconfiguration control is based on a parameter estimation approach which drives a reconfigurable control system (RCS) build with the Pseudo-inverse method.

For instance, the linear parameter-varying (LPV) system can be formulated by the following state space representation

\[
\begin{align*}
\dot{x}(t) &= Ax(t) + Bu(t) \\
y(t) &= Cx(t)
\end{align*}
\]

(5)

where A, B, C are matrices of appropriate dimension and \( \mathbf{u} \) is the inputs variable, \( y \) is the output measurable variables. If the changes \( f_p \) of the states \( \dot{x}(t) \) and \( f_m \) of the output \( y(t) \) then, equation (5) can be written with state-space model \( S_A \) with additive faults as

\[
S_A \begin{cases}
\dot{x}(t) = Ax(t) + Bu(t) + f_p(t) \\
y(t) = Cx(t) + f_m(t)
\end{cases}
\]

(6)
where $f_p, f_m$ are the fault vectors. $f_p \in \mathbb{R}^n$ is a signal disturbing the faults. Change in the each matrix of the state space representation of the system due to the fact they may all depend on the same change in the physical parameters.

**Fault Detection, Isolation and Reconfiguration**

The component failure detection can be achieved by detecting variation in the angular rates. The flow diagram of fault detection reconfiguration is exemplified in Fig. 3. The fault detection method uses the sliding window data from the flight dynamics. It detects the faults by measuring the rate of change of data with respect to time. Assume the LPV (linear parameter varying) system faults are detected with the following

$$S_p(t) = \max \left( \left| \frac{dy_p(t)}{dt} \right| \right)$$

$$D(r) = \begin{cases} 
0 \text{ if } |S_p(t)| < T \\
1 \text{ if } |S_p(t)| > T 
\end{cases}$$

where $dy_p(t)$ is the output produced by the dynamics as a function of time $t$. Diagnostic signal $D(r)$ takes the value of one if the threshold value $T$ has been exceeded. Fault isolation is to make decision about the appearance of a specific fault with simple threshold logic.

$$FI_i > T_i \rightarrow r_{pi}(t) \neq 0 \text{ where } i \in \{1, 2, \ldots, L\}$$

where $L$ is the total number of faults ($f_i$) to be isolated and $T_i (i = 1, 2, \ldots, L)$ are the threshold corresponding to residuals $r_{pi}(i = 1, 2, \ldots, L)$. Reconfiguration is achieved with PIM (pseudo-inverse method) Antsaklis et al. 1989. To provide graceful degradation in the case of failure, the state feedback gain $K$ can be reassigned. Normal closed loop system can be defined as

$$\dot{x}(t) = (A x(t) + B u(t) K)$$

$$y(t) = C x(t)$$

Assume the closed-loop dynamic model, in which failure occurs is given as
where $K$ is the state feedback gain, the new close-loop system can be in the form of

$$
\dot{x}(t) = (Ax(t) + Bu(t))K + f_a(t)
$$

$$
y(t) = Cx(t) + E\mu(t)
$$

(9)

where $K$ is the new feedback gain. The approximate solution of the system for $K$ is given by

$$
K_h = B_f u(t)^+ (Ax(t) - A_f x(t) + Bu(t)K)
$$

(11)

where $B_f u(t)^+$ is pseudo-inverse of $B_f u(t)$. $K_h$ is calculated for many probable failures and stored in the flight control computer (FCC). Similar way other components faults are calculated as described in table 1.

**Multiplicative fault reconfiguration**

Multiplicative fault detection and isolation method coupled with fault tolerant control architecture is developed in order to deal with multiplicative fault for a rotorcraft based unmanned aerial vehicle (RUAV). The failure considered is malfunction with different sensors of the helicopter which occurs during the maneuvers: failure of IMU sensor, GPS sensor, etc. The fault detection and reconfiguration control is based on parameter estimation approach which drives a reconfigurable control system (RCS). The diagnostic reasoning approach has been utilized to isolate the faulty sensors. The multiplicative sensor fault can be modeled as follows

$$
\dot{x}(t) = Ax(t) + Bu(t)
$$

$$
y(t) = Cx(t)
$$

(12)

$x(t) \in \mathbb{R}^n$ denotes the state of the system at time instance. $y(t)$ is the output measurable variable, $x(t)$ is the vehicle state vector and $u(t)$ is control input vector. It can be represent as
Multiplicative sensor faults is modeled as

\[ y(t) = y(t) + \Sigma_S \left( \tilde{y}(t) - y(t) \right) \]  

(14)

where \( \tilde{y} \in \mathbb{R}^p \) is an offset vector and

\[ \Sigma_S = \text{diag} \left[ \sigma_1, \ldots, \sigma_p \right] \]  

(15)

\( \sigma_p = 0 \) represents a total fault of \( i^{th} \) sensor and \( \sigma_p = 1 \) represents a normal operation of \( i^{th} \) sensor. The state space representation of multiplicative fault model is

\[
\begin{bmatrix}
x(t) \\
y(t)
\end{bmatrix} =
\begin{bmatrix}
A & B \\
0 & C
\end{bmatrix}
\begin{bmatrix}
x(t) \\
y(t)
\end{bmatrix} +
\begin{bmatrix}
\xi(t) \\
\Phi(t)
\end{bmatrix}
\]  

(16)

Fault Detection, Isolation and Reconfiguration

The fault detection method uses the sliding window data from the flight dynamics. The example illustration of sliding data is depicted in Fig. 4. It detects the faults by measuring the rate of change of data with respect to time. Assume the LPV (Linear Parameter Varying) systems faults are detected with the following

\[
\phi_{\sigma_p}(t) = \left[ (n_1-1), (n_2-2), \ldots, (n_{\sigma} - j) \right]
\]

\[
S_p(t) = \left[ \phi_{\sigma_p}(2(t) - \phi_{\sigma_p}(1(t))), \ldots, \phi_{\sigma_p}(n_\sigma - j(t)) - \phi_{\sigma_p}(n_\sigma - j(t) - 1(t)) \right]
\]

\[
D_p(t) = \begin{cases} 
0 & \text{if } |S_p(t)| < T \\
1 & \text{if } |S_p(t)| > T 
\end{cases}
\]
where $dy_P r(t)$ is the output produced by the dynamics as a function of time $t$, diagnostic signal $D_P r(t)$ takes the value of one if the threshold value $T$ has been exceeded.

Diagnostic reasoning approach is utilized for fault isolation process. It is performed by means of a rule based single logical operations state as IF-THEN-ELSE rules like

\[
\begin{align*}
&\text{IF } <D_P r1(t) \text{ AND } \geq D_P r2(t) \text{ THEN } F1_i \\
&\text{IF } <D_P r3(t) \text{ OR } \geq D_P r4(t) \text{ THEN } F1_2 \\
&\text{IF } <D_P r2(t) \text{ AND } \geq D_P r4(t) \text{ THEN } F1_3
\end{align*}
\]

The fault that has been detected from the fault detection module is considered as symptoms. These faults are formulated in verbal manner. Based on the observations these conditions fed as rules to the system. For each fault ID $F1_i(t)$ the reconfiguration module reconfigures the system to maintain the stability of the vehicle. Table 2 illustrates the fault flag, symptoms and reconfiguration

**SIMULATION ARCHITECTURE**

Most of the unmanned aerial vehicle control system cannot be validated in real time due to the cost effectiveness. There are numerous approach used to validate control system, among that HILS (Hardware-In-the-Loop-Simulation) is well known method. One of the HILS system presented in (Budiyono et al. 2010) used pc104 based xPC target as a controller unit.

Development of simulation environment based on real-time operating system QNX neutrino is investigated in this section. In addition, external pilot command input and virtual flight simulation environment is created for virtual flight test. The real hardware for autopilot is placed inside the simulation. The approach with generic components that can be tailored to establish the coupling between software and hardware components is introduced. Two stages of testbed are developed in order to test and simulate the proposed algorithms. Stage1: Single RUAV-HILS Testbed and Stage2: Multi RUAV-HILS Testbed. In Stage1 HILS setup is used to test obstacle avoidance control, point navigation control, Fault detection, isolation and reconfiguration and auto-landing
control. Stage 2 HILS setup is used to test multiple RUAV algorithms like group behavior, dynamic obstacle avoidance. In the following each modules are discussed in brief.

**Hardware-in-the-Loop Simulation**

The architecture of Hardware-in-the-Loop for autonomous system, as shown in Fig. 5 composed of four independent systems namely Flight visualization computer (FlightGear), Flight control computer, GCS (Ground Control System), HIL Bridge and Helicopter model. The bridging application uses a simulation library to communicate with FlightGear and connects to the embedded PC104 board (Fig 6-a), emulating the inputs and outputs from the helicopter model. FCC outputs serial data in packed binary identical to the packet produce by the Flightgear and helicopter model. It also accepts command outputs from GCS. This provides a realistic test of remaining components of the system. The FlightGear forwards a raw data to FCC. The given data is in engineering unit. Therefore, the HIL Bridge application must perform reverse conversion on the data before sending it to the PC104 embedded board. Similarly the control surface position received from the embedded board is in raw format the HIL Bridge converts normalized positions to helicopter plant model. In this architecture the work presented in (Kim et al. 2010) is used as plant model.

FlightGear is an open-source flight simulator developed and maintained by Curt Olson. FlightGear objective is to create a professional flight simulator framework to use in both academic and research environments. The screen shot of FlightGear is illustrated in Fig. 6-b. In this work the FlightGear is investigated to develop cost effective environment for testing our control algorithms which is capable of real time testing. More complete platform setup is briefed in (Kaliappan et al. 2011). The extension of stage one is developed to provide a testing environment for both static and dynamic obstacle with multiple RUAV’s. The setup runs with three identical HILS system in parallel, where the local coordinates of each RUAV are shared. It is achieved by communicating with UDP socket. Fig.7 provides complete hardware architecture of multiple HILS system.

**EXPERIMENTAL RESULT AND ANALYSIS**
In this section, the experimental test results for intelligent architecture are discussed. Each of the modules is tested and validated separately. All the experiment is done using the HIL simulation using PC104 based target PC running under QNX real time operating system.

**Way-point Navigation and Control**

The way-point navigation control is validated by flying the RUAV autonomously. The set of way-points are assigned from the ground control station where the RUAV should attain all the way-points and reach the assigned destination point. Initially, the flight should reach the altitude of 25mts above the sea level then fly towards the following way-points (30, 25), (30,-25), (-10,-30), (-10,-10) once it reaches the destination it should execute the landing sequence. From Fig. 8 it clearly proves that RUAV passed the assigned way-points and reaches the destination successfully. Fig. 9 illustrates the landing trajectory of unmanned helicopter using our proposed method. The solid line represents the real flight trajectory generated by RUAV. From the achieved results it has been proved that the tracking and auto landing works smoothly and accurately.

**Obstacle Avoidance Control**

The performance of the algorithm is verified with two different phases. First phase is to check with single RUAV to multiple static obstacles and second phase is to check with multiple RUAV’s to multiple static and dynamic obstacles. First obstacle avoidance experiment is attained by flying the RUAV in counter clockwise direction to achieve the given set of way-points and return to its target. For simulation purpose in the middle of flying zone the obstacles are placed in different size. The algorithm should detect the obstacle and calculate the trajectory to avoid and reach the desired target. In this experiment three obstacles are placed in the local coordinates. First one is at (30, 10) with four meters of radius. Second one is at (0, -30) with 3.5 meters of radius and finally at (0, 40) with six meters of radius. From the Fig. 10 it clearly proves that our algorithm avoid obstacle with different obstacle radius.
Second experiment is to validate the algorithm for both static and dynamic obstacles. In this experiment two RUAV’s has been used, first RUAV fly’s in counter clockwise direction where another fly’s in clockwise direction with the same set of way-points. For both of the RUAV’s same static obstacles are assigned as shown in first phase. Fig. 11 clearly illustrates the avoiding trajectory generate by two RUAV’s.

**Fault Detection and Identification**

**Addictive faults:** To test our designed controller the fault in main rotor angular rate is considered. When there is any change in the rotor angular rate, the RUAV’s heading and altitude hold system affects instantaneously. These faults occur due to the wear and tear of bearing in rotational equipment, friction due to lubricant deterioration, tooth breakage and crack in gear of a gearbox system, etc. Aforementioned faults may have trivial to catastrophic damage to the system.

Thus, it is extremely crucial to diagnose these faults at early stages in order to avoid catastrophic damages to the system. The main aim is to detect smaller variation in the system and make the RUAV to stabilize. After stabilizing the system, RUAV checks for the landing zone and executes the landing sequence. When fault \( f_p \) is injection to the dynamic model, the RUAV oscillate its assigned trajectory (Fig. 12). The time based software fault injection method is utilized to add faults to the dynamic engine. Based on the fault, controller should respond and reassign its state feedback gain to stabilize the controller. These gains are calculated with pseudo inverse method and few with trial and error method.

During the course of flight the additive faults are injected to the internal component of the RUAV. Due to the fault, RUAV’s angular rate reduces from 129.8021 rad/sec to 110.8021 rad/sec. Fig. 13 shows the change in engine speed at time 1400 milliseconds. Due to the change in engine speed the Heading and altitude of the RUAV distracted which is exemplified from Fig.13-b-g. Based on the angular fault the other control angular rate is changed accordingly. From Fig. 12 it is clear that
when the fault is injected, the RUAV misalign to its trajectory

The Fault detection module detects the engine speed fault and flags the fault ID. Based on the fault flag the reconfiguration manager module reassigns the feedback gain to the new predefined gain. At time 2100 millisecond the RUAV executes normal trajectory (Fig. 12) even the RPM of the helicopter is at 110.8021 rad/sec. For understanding the fault clearly, the delay of 700 milliseconds is added after fault detection process.

**Multiplicative faults:** Fault detection in navigation system of RUAV is carried out. The simulation is based on navigation data from sensor model. The bank of estimator checks for errors at navigation data, if the error is detected, then the system flags a fault ID to the fault recovery reconfigurator. Based on the error residual the reconfiguration module generates reconfigurable commands to switch redundant sensor. When fault injection to the sensor model, the RUAV oscillate its assigned trajectory. Based on the fault, controller should respond and reassign redundant sensor to stabilize the system.

During the course of flight the multiplicative fault is injected to the sensor model of the RUAV. Due to the fault, RUAV is unstable. Fig. 14-a shows the change in heading at time 420 seconds. Due to the change in heading the pitch and climb rate of the RUAV distracted. The Fault detection module detects the faulty sensor and initiates the fault flag. Based on the fault flag (Fig. 14-b) the reconfiguration manager module reassigns the redundant sensor. At time 800 seconds the RUAV stabilized once the system reconfigured to redundant sensor (Fig. 14-f). The threshold values are manifested based on the maximum and minimum value that the sensor that can cope with. Based on the sensor hardware specs the threshold values are calculated.

**Multiple RUAV Coordination Control**

RUAV coordination algorithm is validates using two cases. To test the separation and coherence strategies three RUAV’s are considered, and each RUAV should fly through way-points in the
The unit position of RUAV is measured in meters. In our case the RUAV1 communicates with RUAV2 and RUAV3, RUAV2 with RUAV1 and RUAV3 and so on. When the RUAV reaches the $i^{th}$ way-point the $(i+1)$ pre-specified way-point is considered as next target way-point. Case1 is to make the RUAV to fly in single direction to check whether the RUAV executing the separation strategy. At the point of $(20, 0)$ the RUAV separate among each other. Fig. 15 shows the Case1 RUAV’s trajectories. Finally all three RUAV reaches their target with five meter distance between them.

Case2 utilize the following set of patterns.

$$P_0 = \{[30,-30],[-30,30]\}, P_1 = \{[30,-30],[-30,30]\}, P_2 = \{[30,30],[-30,30]\}$$

During the execution of their patterns, at point $(30, 10)$ RUAV2 detects the RUAV1’s trajectory. Once the RUAV1 approaches inside the detection box of RUAV2, then the separation strategy is executed and tries to move away by changing its trajectory. The trajectory obtained in the course of flight is exemplified in Fig. 16. Meanwhile RUAV3 detects RUAV2 at position $(30, 20)$ and RUAV1 at position $(30, 10)$ it tries to generate greater trajectory to avoid both the RUAV’s. Later RUAV3 tries to reach the neighborhood RUAV’s by switching from separation behavior to cohesion behavior.

CONCLUSION
The autonomous helicopter need highly efficient autonomous intelligent infrastructure for autonomous control systems. The intelligent control system should focus more about safety condition to avoid potential accidents. Fault detection and isolation plays an important role in this context. This research has presented a design of multilayer architecture and integration process of several basic control algorithms. The challenge faced by a RUAV in responding to potential obstacles which recline in the path of assigned trajectory is also addressed. A low cost algorithm based on biologically inspired steering behavior method for group coordination has been proposed. Along with this fault-tolerant control is developed which extends the state of the art in fault-tolerant control of RUAV’s. Some example scenarios have been discussed, involving stationary and moving obstacles, group RUAV coordination. Planning for RUAV in real world environments with uncertainty and disturbance creates much more difficulty, and the HILS single and multi-RUAV testbed has helped in testing and validating the proposed architecture.

ACKNOWLEDGEMENTS

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency)” (NIPA-2010-C1090-1031-0003).

This paper was supported by Konkuk University in 2012.

REFERENCES


Double-spaced list of figure captions

Fig.1. Reconfigurable Intelligent Control Architecture

Fig.2. Representation of faults in the control system

Fig.3. Flow diagram of fault injection, detection and reconfiguration process

Fig.4. Example illustration of sliding mode data acquisition

Fig.5. Software Architecture for Hardware-in-the-Loop simulation

Fig.6. (a) PC104 based flight unit (b) Screen shot of FlightGear simulator

Fig.7. Functional Multiple RUAV Hardware-In-the-Loop Simulation system

Fig.8. Trajectory generated by RUAV during execution of way-points navigation and control

Fig.9. Landing trajectory generated from Auto-Landing control

Fig.10. Obstacle avoidance algorithm with multiple static threads

Fig.11. Obstacle avoidance algorithm with multiple static and dynamic threads

Fig.12. Trajectory generated by RUAV during the execution of way-points

Fig.13. Response of control signals in the occurrence of fault and reconfiguration

Fig.14. Triggering the fault flag after the IMU fault occurs and reconfigured to stabilize the vehicle

Fig.15. Three-dimensional trajectory with Case1

Fig.16. Three-dimensional trajectory with Case2
Accepted Manuscript

Not Copyedited
Figure 3 flow diagram of fault injection
Figure 4 sliding mode data

\[ 1 \quad 2 \quad 3 \quad \ldots \quad j \quad n_1 \quad n_2 \quad n_3 \quad \ldots \quad n_j \]
Figure 5 software architecture

**Ground control station**
Executes on GCS laptop, display aircraft state, Get command from operator

**Flight Control Computer**
Executes on embedded board, Aircraft control algorithms

**HIL Bridge**
Manages communication, UDP data, serial data, emulates sensor board

**FlightGear**
Flight Visualization, generates flight environment and sensor data

**Helicopter Model**
C code for 6 DOF helicopter nonlinear model

Aircraft Commands (packed binary)
Aircraft State

Raw sensor data (UDP)
Control Data yaw, pitch, Roll (Packed binary)

Aircraft State (packed binary)
Desired State

Copyright 2012 by the American Society of Civil Engineers

Figure 6b flight gear
Accepted Manuscript
Not Copyedited
Figure 8: Trajectory of UAV
Figure 9 landing
Figure 10 obstacle avoidance
Figure 12 trajectory generated
Figure 13 control signal of reconfiguration
Figure 14 fault imu
Figure 16 3d trajectory c2
### Table 1 Feedback gain with respect to component faults

<table>
<thead>
<tr>
<th>Number</th>
<th>Fault injection</th>
<th>Main rotor angular rate</th>
<th>Stability feedback gain</th>
<th>Reconfiguration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Δω_com Δω_k Δω_q Δω_i Δω_{θ} Δω_{φ} Δω_{ρ}</td>
<td>K_h K_dh K_q K_i K_{θp} K_{θi} K_q K_{φ} K_{ρi} K_{φi} K_{ρ}</td>
<td>Altitude hold</td>
</tr>
<tr>
<td>1</td>
<td>129.8021</td>
<td>0.01 -0.01 0.45 0.02 0.05 0.001 0.3 -0.02 0.1 0.002 0.3 -0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>119.8021</td>
<td>0.01 -0.01 0.45 0.02 0.05 0.001 0.3 -0.02 0.1 0.002 0.3 -0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>115.8021</td>
<td>0.01 -0.01 0.50 0.03 0.05 0.001 0.3 -0.02 0.1 0.002 0.3 -0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>110.8021</td>
<td>0.025 -0.025 0.75 0.0351 0.05 0.001 0.3 -0.02 0.1 0.002 0.3 -0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>104.8021</td>
<td>0.034 -0.034 0.95 0.040 0.05 0.001 0.3 -0.02 0.1 0.002 0.3 -0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>100.8021</td>
<td>0.050 -0.050 1.80 0.040 0.05 0.001 0.3 -0.02 0.1 0.002 0.3 -0.01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 2 Fault Flags and reconfiguration with Diagnostic reasoning approach

<table>
<thead>
<tr>
<th>Flags/ symptoms</th>
<th>Event</th>
<th>Fault</th>
<th>Reconfiguration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 1</td>
<td>Probable fault engine components</td>
<td>Rotor angular rate</td>
<td>Change Feed-back gain factor</td>
</tr>
<tr>
<td>1 0 0 0 0 0 0 0 0</td>
<td>Probable fault in IMU sensor (x)</td>
<td>IMU sensor</td>
<td>Switch Redundant IMU sensor</td>
</tr>
<tr>
<td>0 1 0 0 0 0 0 0 0</td>
<td>Probable fault in IMU sensor (y)</td>
<td>IMU sensor</td>
<td>Switch Redundant IMU sensor</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 0 0</td>
<td>Probable fault in IMU sensor (z)</td>
<td>IMU sensor</td>
<td>Switch Redundant IMU sensor</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0 0 0</td>
<td>Probable fault in altimeter sensor</td>
<td>altimeter sensor</td>
<td>Switch Redundant Alt-sensor</td>
</tr>
</tbody>
</table>