Paper: ASAT-16-070-CT

16th International Conference on
AEROSPACE SCIENCES & AVIATION TECHNOLOGY,
ASAT - 16 – May 26 - 28, 2015, E-Mail: asat@mtc.edu.eg
Military Technical College, Kobry Elkobbah, Cairo, Egypt
Tel : +(202) 24025292 – 24036138, Fax: +(202) 22621908

PID Control of a Lab Scale Single-Rotor Helicopter System using a Multicore Microcontroller

M. Moness*, M. Khaled†, M.bakr‡, A.Omar§

Abstract: In this paper, a lab-scale single-rotor helicopter system is modeled and controlled. The introduced system is first modeled mathematically and then modeled with an online identification method using MATLAB. The model is then used to design a PID controller for the system. The designed controller is then implemented using a multicore microcontroller. The control task is one of the tasks in the introduced control software where the role of the microcontroller is to execute it along with many different tasks like interfacing sensors and actuators, tuning control parameters, data filtering and logging. The implemented control system uses a multicore microcontroller to execute all tasks simultaneously and hence improves performance and functionality. It is also demonstrated that using multicore microcontrollers can reduce design-time, implementation-time and cost while keeping higher performance rates. This contribution shows that it is possible to design and implement complex real-time embedded control systems that employ advanced control algorithms using multicore microcontrollers.

Keywords: Lab-scale, Single-rotor helicopter, Modeling, Identification, Linear controller, PID, Fuzzy logic, Kalman filter, Embedded system, Multicore, Implementation

Nomenclature

$l_{(\mathrm{m})}$	Length of link	$f_{x(\mathrm{N})}$	Force along x_b WRT B-frame
$m_{Mot(\mathrm{kg})}$	Mass of BLDC Motor	τ_m	Motor torque
$m_{l(\mathrm{kg})}$	Mass of link	τ_p	Propeller torque
$l_{1(\mathrm{m})}$	Length of half link	τ_{pm}	Propeller torque on motor axis
$g_{(\mathrm{m}s^{-2})}$	Gravity of earth	θ	Angular position around y_1 WRT E-frame
$\dot{\theta}$	velocity around y1 WRT E-frame	$\ddot{\theta}$	angular acceleration around y_1 WRT E-frame
P_e	Electric motor power	P_m	Mechanic motor power
P_p	Mechanic Propeller Power	COG	Center of Gravity
BLDC	Brushless DC Motor	DOF	Degree of Freedom
PWM	Pulse Width Modulation	ESC	Electrical Speed Controller
MCUs	Microcontrollers	J	Moment of Inertia
RPM	revelation per minute	DMP	Digital motion processor
IMU	Inertial measurement unit	FLT	Fuzzy logic tuning

* Professor, Computers and Systems Engineering Dept., Faculty of Engineering, Minia University. Email: m.moness@mu.edu.eg.

† Teaching assistant, Computers and Systems Engineering Dept., Faculty of Engineering, Minia University. Email: mkhaled@mu.edu.eg.

‡ Teaching assistant, Computers and Systems Engineering Dept., Faculty of Engineering, Minia University. Email: muhammad.bakr@mu.edu.eg

§ Teaching assistant, Computers and Systems Engineering Dept., Faculty of Engineering, Minia University. Email: aly.omar@mu.edu.eg

Introduction

Microcontrollers are single-chip computers which can be used to control real-time systems. Such controllers are also referred to as embedded real-time computers. MCUs are well suited to control applications, especially with widely changing requirements. These devices are cost-efficient, single-chip, power-limited, easy to reprogram, and fast to deploy. Many control applications are computer-based, where a digital computer or an MCU is used as a main or supplementary digital controller. Early MCUs were very limited in resources and computation power and required many interfacing circuits while today's MCUs are computationally powerful and they are equipped with interface modules to facilitate the connection to controlled processes.

Latest study [1] for embedded systems market revealed that industrial control and automation applications are the most common project types for embedded systems development. Almost all of these applications combine smaller embedded software tasks such as digital control algorithms, parameter tuning procedures, fuzzy-logic or ANN models and controllers, PWM generation, interfacing, logging, or fault-tolerance tasks. Therefore, the use of multiprocessing systems for such application would be of a great benefit where each task is executed by an independent part of the system [2]. Moreover, real-time embedded control systems have tight time windows to gather data, process that data, and update the system while conducting additional tasks like tuning of control parameters or executing a fault- tolerance algorithm. If this time window is missed, the stability of the system is degraded. This reduced control can be catastrophic to some applications, such as power conversion and advanced motor control [3].

For embedded control applications, multiprocessing systems can employ independent or cooperative control tasks where each task is running simultaneously. This improves throughput of multi-axis controllers, reduces the execution time of real-time controllers or leaves extra time for more tasks in almost every embedded controller. Although parallel implementations are generally known to improve performance, the use of embedded multiprocessing systems for control applications is still in its starting steps [2] [4]. However, many signs show that classical single-processor implementations are migrating towards newer multiprocessing ones. Recently, manufacturers of famous embedded real-time MCUs started announcing new multiprocessor families. The Concerto family of from TI Inc. [5] and the latest members of the SPC56 family from ST Inc. [6] are examples of multiprocessing systems for embedded real-time applications. Moreover, many researchers started to realize that using these multiprocessing techniques and systems would be beneficial even by using multitasking within a single processor [7] [8], by using multiple independent MCUs [9] [10], by using multicore platforms [4] [11] [12] or by customizing their own multiprocessing systems.

Currently, traditional MCU technologies are migrating to multiprocessing systems as multicore MCUs are more and more becoming available. Unlike traditional MCUs, they consist of two or more independent processing units that are connected with an interconnection scheme along with memory modules and interface ports all within a single-chip. Multicore MCUs guarantee truly parallel multitasking, shorter RT frames, low latency with much faster responses, and completely enhanced performance. The Parallax Propeller [13] and the XMOS XCore [14] are also examples for recent commercially available multicore microcontrollers.

In this study, a lab-scale single-rotor helicopter system is modeled and controlled. The introduced system is modeled and then the model is used to design a PID controller for the system. The designed controller is then implemented using a multicore microcontroller. The control task is one of the tasks in the introduced control software where the role of the

microcontroller is to execute it along with many different tasks like interfacing sensors and actuators, tuning control parameters, data filtering and logging. The implemented control system uses a multicore microcontroller to execute all tasks simultaneously and hence improves performance and functionality.
This paper is organized as follows. Section II presents the description and modeling of the single-rotor system. Section III describes the design and implementation phases of the control system. Section IV presents experimental results obtained from the system. Finally, section V shows the conclusions.

System Description and Modeling

The single-rotor helicopter system consists of a wooden bar connected to a wall via a one degree of freedom hinge. A brushless (BLDC) motor with its propeller is mounted at the end- far from the hinge- of the wooden joint. Fig.1. describes this system. As the motor rotates, it generates a force that lifts the joint causing a changing inclination angle from 0 to 90 degrees. The system resembles a helicopter airplane with the horizontal axis fixed. Studying the model, it is noticed that at the far end of the joint, two forces act at this point in opposite directions: the force generated by the rotating propeller and the weight of the motor, also at the middle of the joint the weight of the joint act at this point. The resultant of these previously stated forces causes a torque at the hinge that connects the joint to the wall which causes the joint to rotate around the hinge. A mathematical model is designed using Newton's laws of motion and calculating the torque around the hinge.

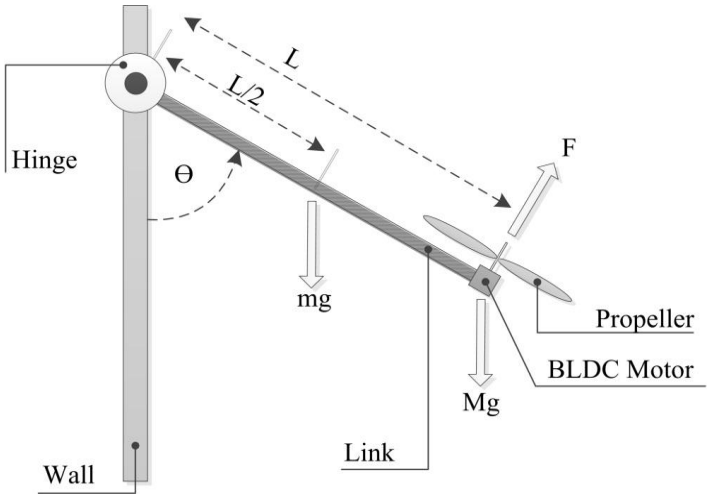


Fig.1. The singe-rotor helicopter system.

$$\tau = (f_x - m_{Mot}.g.Sin(\theta))l - (m_l.g.Sin(\theta))l_1 \quad (1)$$

$$J\ddot{\theta} = f_x l - m_{Mot} g.l.Sin(\theta) - m_l g Sin(\theta)\, l_1 \quad (2)$$

$$J\ddot{\theta} + Sin(\theta)(m_{mot}.g.l + m_l.g.l_1) = f_x l \quad (3)$$

$$\ddot{\theta} = \frac{l}{J} f_x - \frac{g}{j} l \left(m_{mot} - \frac{m_l}{2}\right) Sin(\theta) \quad (4)$$

The system is nonlinear and therefore, we will predict the model using an identification technique. The system is put on action and a step is applied to its input as a control command to the BLDC motor. The response of the system to this step is measured. Fig.2. presented the scaled input step and the scaled response of the system. Again the system is commanded, but using a multi-sin signal and the response is measured. Fig.3. presented the scaled multi-sin

input and its measured response from the system. All inputs and outputs are scaled. The input which represents the control action to the BLDC motor is scaled to the range [0 to 1] that represents [0 to 1000] RPM The output angle of the system is scaled to the range [0 to 1] that represents the angles [0 to 90] degree.

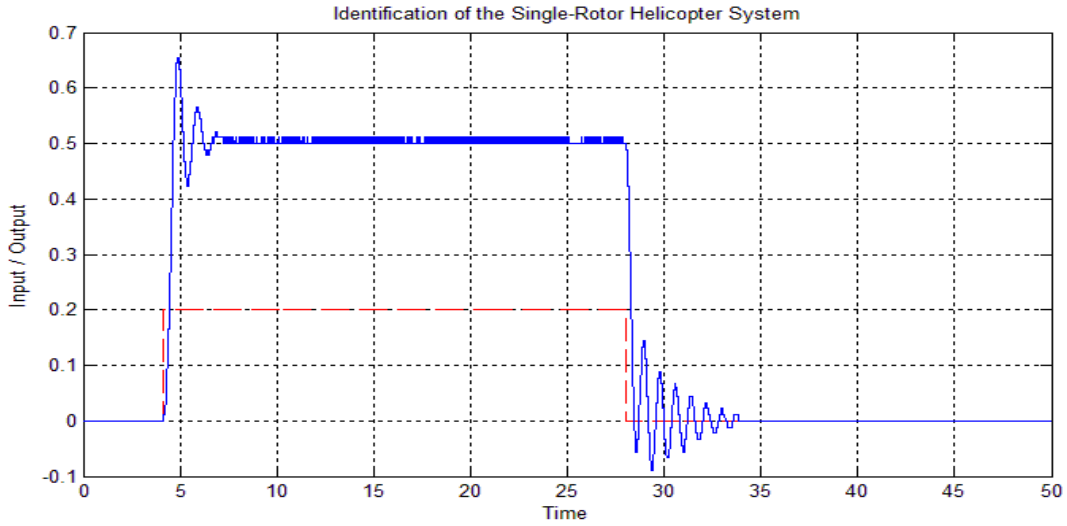


Fig.2. Input step and measured output angle of the system

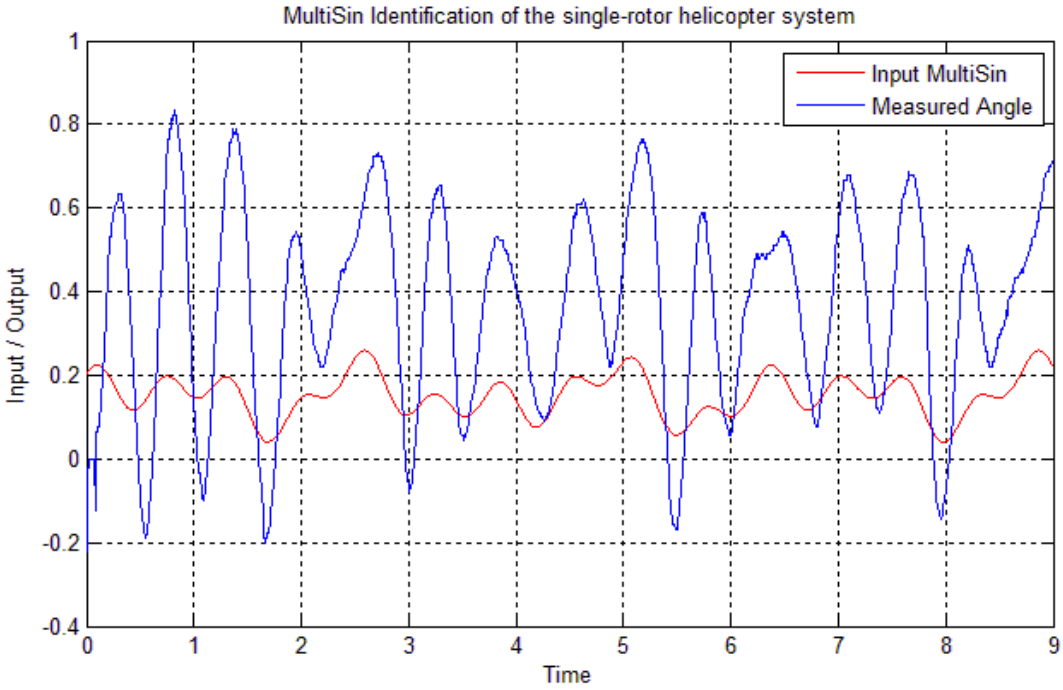


Fig.3. Input multi-sin and measured output angle of the system.

The two measured responses of the system is used in a mixed-identification process using MATLAB identification toolbox. The process involves using the two responses to match a best-of-possible model of the system. An enhancement to this technique was applied by performing a brute-force identification of all possible mathematical models starting from 1 pole to 15 poles and 1 zero to 15 zeros. This produces 255 models. All models' responses are programmatically compared to the measured data. The found best-fit model is 5 poles and 4 zeros. Fig.4. and Fig.5 shows the response of this identified model compared to the measured data for both step response and multi-sin response.

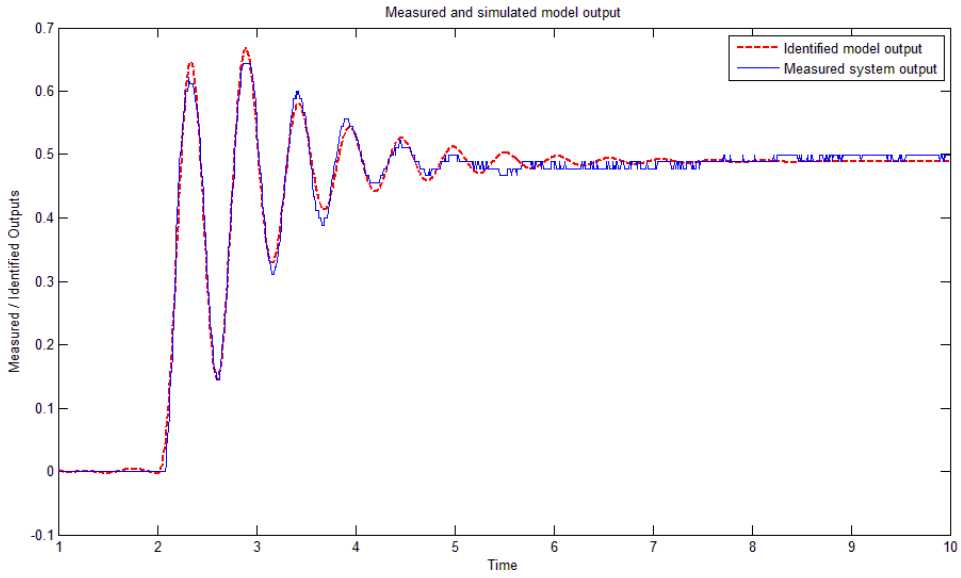


Fig.4. Measured and Identified responses of the system to a step input.

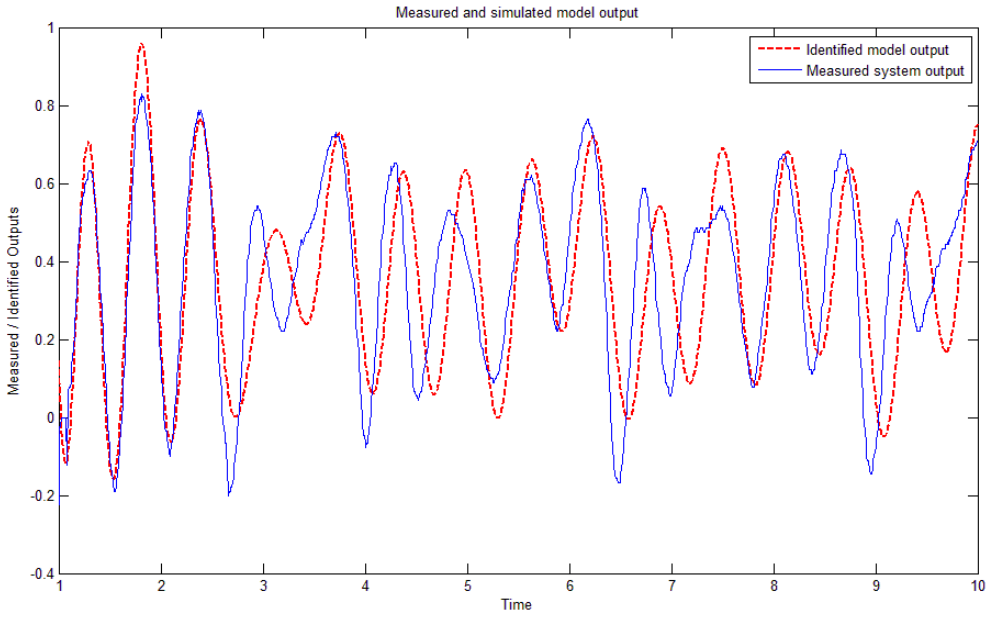


Fig.5. Measured and identified responses of the system to a multi-sin input.

The following is the transfer function of the model:

$$H(z) = \frac{0.002049\,z^{-1} - 0.005101\,z^{-2} + 0.00412\,z^{-3} - 0.001065\,z^{-4}}{1 - 4.913\,z^{-1} + 9.669\,z^{-2} - 9.529\,z^{-3} + 4.702\,z^{-4} - 0.9293\,z^{-5}} \quad (5)$$

Controller Design and Implementation

This section starts by introducing the Propeller multicore microcontroller that will be used to implement the control program. Then, we demonstrate how the feedback of the system, the angle from the helicopter, is measured and filtered before calculating the control action. Later,

the implementation of the controller within the introduced multicore microcontroller is presented.

The Propeller Multicore Microcontroller

The Propeller MCU [13] is a cheap 40-pin chip with an 8-core multiprocessor architecture. It is designed to provide high-speed processing for embedded systems while maintaining low current consumption and a small physical footprint. In addition to being fast, the Propeller chip provides flexibility and power through its eight processors, called cogs, that can perform simultaneous tasks independently or cooperatively, all while maintaining a relatively simple architecture that is easy to learn and utilize. Many programming languages are available to program this chip: SPIN (a high-level object-based language), Propeller Assembly and C/C++. The price of a single propeller chip ranges from 4$ to 8$. Fig.6. shows a block diagram of the Propeller's architecture. It contains eight symmetric 32-bit processors (cogs) numbered 0 to 7. Each cog contains a processor block, local RAM, a video generator, I/O output register, I/O direction register, and other registers.

All eight cogs are driven from the system clock; they maintain the same time reference and all active cogs execute instructions simultaneously. They also all have access to the same shared resources; shared memory and IO ports. Cogs can be started and stopped at run time and can be programmed to perform tasks simultaneously, either independently or with coordination from other cogs through main RAM. Each cog has its own RAM, called Cog RAM, which contains 512 registers of 32 bits each. Each cog can operate with a speed up to 20 MIPS, resulting into a 160 MIPS total speed when it is running on an 80 MHz system clock. Moreover, cogs can execute local instructions or they can execute instructions in the shared memory space.

The resulting design of the Propeller frees application developers from common complexities of embedded systems programming because the memory map is flat, so, there is no need for paging schemes with blocks of code, data or variables. This is a great time-saving mechanism during application development. Moreover, asynchronous events are easier to handle than they are with devices that use interrupts. The Propeller has no need for interrupts as cogs can be assigned to individual, high-speed polling tasks. The result is a more responsive application that is easier to maintain.

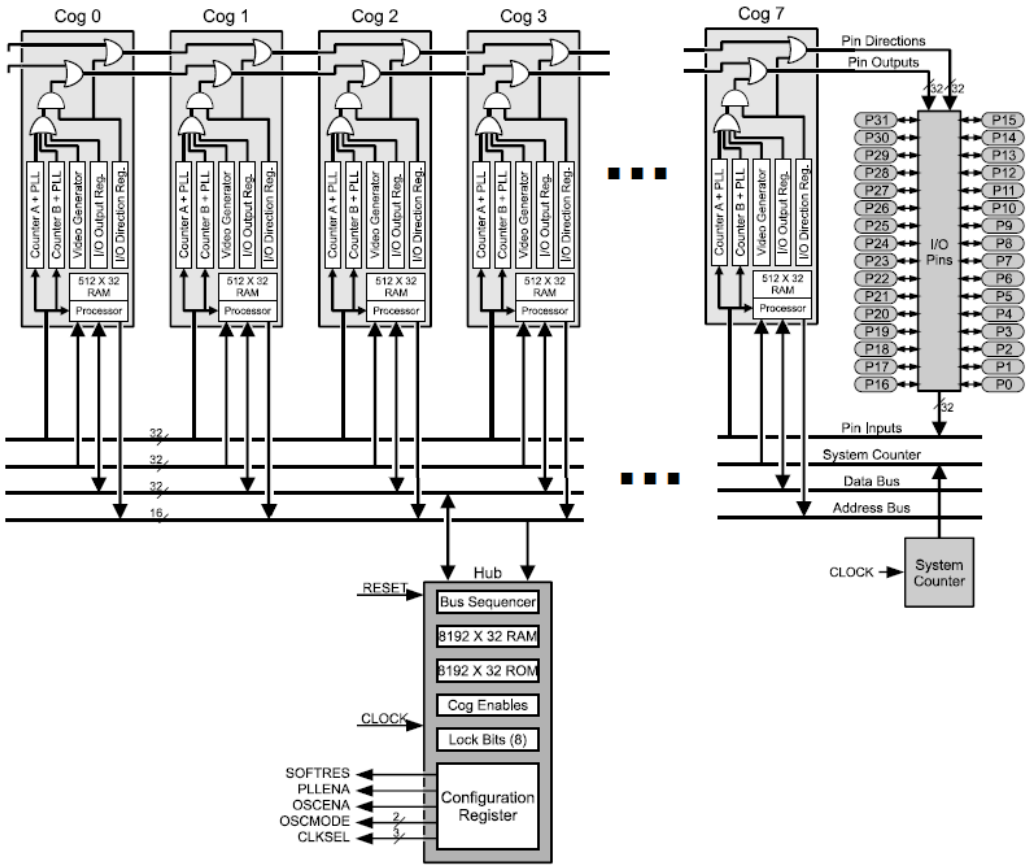


Fig.6. Architecture of the Parallax Propeller Multicore MCU.

Reading Rotation Angle from the Helicopter Model

For determining the inclination angle of our model to the horizontal position, the controlled variable, we use an IMU which includes a 16-bit 3-axis accelerometer MEMS along with a 16-bit 3 axis gyroscope for measuring acceleration and angular speed respectively in all 3 axis x, y and z. This unit is also equipped with a DMP that can be programed to synchronize and filter the sensors readings and also can be used to further process the data to give the rotation matrix, cosine matrix or calculate yaw, pitch and roll angles internally.
The inclination to the horizontal angle (α) which our controlled variable is calculated from its complementary angle (θ) from (6)

$$\alpha = 90 - \theta \tag{6}$$

Sensors data give us information about acceleration in x axis ($\ddot{x}$), acceleration in (y) axis ($\ddot{y}$) and acceleration in (z) axis ($\ddot{z}$) and also we have data given from gyroscope which is rotational speed around (x) axis ($\dot{w}_x$), rotational speed around (y) axis ($\dot{w}_y$) and rotational speed around z axis ($\dot{w}_z$).

Accelerometer reads acceleration in all 3 axis all the time, if the sensor is in horizontal position the z axis reading ($\ddot{z}$) is affected by the earth's gravity (g) and the sensor should read the full gravity acceleration on ($\ddot{z}$) in horizontal position.

As the sensor is inclined to the horizontal, the effect of gravity is distributed on two axis y and z. Performing analysis of the effect of gravity on the two axis y and z in case that the sensor is inclined to the vertical by θ as in Fig.1. We get that

$$(\ddot{z}) = gcos(\theta) \tag{7}$$

$$(\ddot{y}) = gsin(\theta) \tag{8}$$

From equation (7) and (8), we get that

$$\tan(\theta) = \frac{(\ddot{y})}{(\ddot{z})} \tag{9}$$

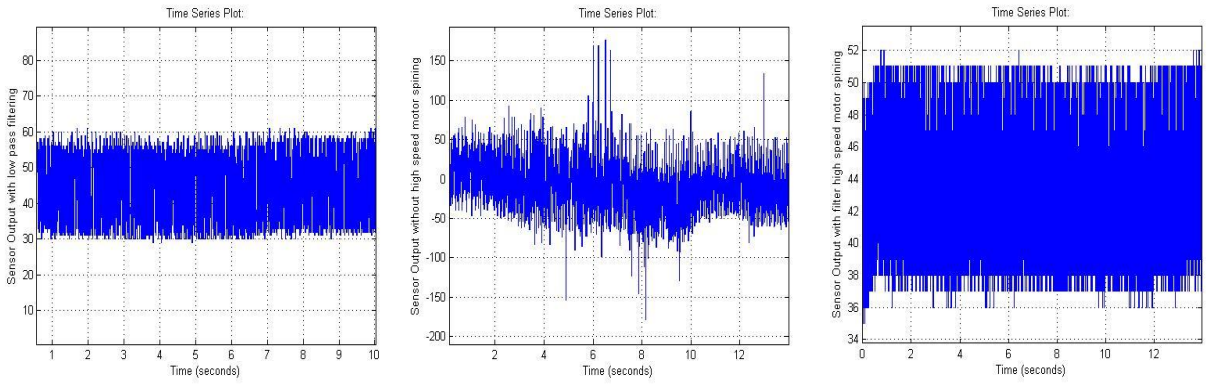
By using equation (9) we can calculate the inclination angle to the vertical (θ) and use Equation (1) to calculate (α). The gyroscope measures the rotational speed around all 3 axis. Simply according to the way we mounted the IMU on our model we get that.

$$\dot{\theta} = \dot{w}_x \tag{10}$$

Integrating the above equation, we can get (θ) and accordingly our controlled variable (α).

The implementation of an accurate angle estimation system is challenging. The accelerometer readings suffer from jitter noisy readings that happen randomly with different amplitudes. The gyroscope zero readings always drift with time and needs to be recalibrated constantly. The sensor suffers from a significant vibration caused by the high speed rotor that is in the model. This vibration effect is increased because of the fixed hinge and un-canceled gyroscopic effect and aerodynamic torque from the BLDC motor.

Examining the sensors readings of the inclination angle while the rotor is running at its minimal hovering speed, we get that there is a vibration in readings with frequency of 14 Hz. To overcome this vibration in angle reading, a low pass filter is used and implemented on the (DMP) at frequency 10 Hz to remove any vibration caused by running motor. Note that the vibration frequency increases as the motor speed increases, so designing a low pass filter of the frequency of vibration caused by minimal hovering speed will filter out any vibrations caused by the motor at higher speeds. Fig.7.a. shows the sensor reading before low pass filtering and Fig.7.b. shows the vibration of the reading at higher motor speed and Fig.7.c. shows the sensor reading after low pass filtering, examining the frequency of the vibration we clearly see that the frequency now is about 16 Hz i.e. the frequency increases as the motor speed increases.



(a) (b) (c)

Fig.7. The sensor reading

Sensor fusion involves collecting data from various sensors and applying data fusion techniques to obtain optimal measurements. Sensor fusion can overcome the gyroscope drifting problem as its bias will be continuously recalibrated based on estimated error and accelerometer readings measurement. To address this sensor fusion problem the Kalman filter solution is proposed [15], however due to the involvement of a non-linear system of equations, the Extended Kalman filter version has been employed to simplify implementation. Due to the microcontroller limitation and in favor of speeding up the calculation as fast as possible the equations here are simplified whenever it is possible. The following state-space system represents the Kalman estimation technique.

$$\dot{x} = Ax + Bu \tag{11}$$

$$Z = Hx \tag{12}$$

Where x is our state vector x = [θ, γ] and consequently $\dot{x} = [\dot{\theta}, \dot{\gamma}]$, where $\dot{\theta} = gyro\ measurement - \gamma$ and $\dot{\gamma} = 0$, where θ is the inclination angle and γ is the gyro bias. A is the Jacobian of $\dot{x}$ with respect to the states:

$$A = \begin{bmatrix} \frac{d(\dot{\theta})}{d(\theta)} & \frac{d(\dot{\theta})}{d(\gamma)} \\ \frac{d(\dot{\gamma})}{d(\theta)} & \frac{d(\dot{\gamma})}{d(\gamma)} \end{bmatrix} \tag{13}$$

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{14}$$

The H matrix is a 1x2 (measurements x states) matrix that is the Jacobian matrix of the measurement value with respect to the states:

$$H = \begin{bmatrix} \frac{d(\theta_{measured})}{d(\theta)} & \frac{d(\theta_{measured})}{d\gamma} \end{bmatrix} \tag{15}$$

$$H = [1 \quad 0] \tag{16}$$

Since there is no relation what so ever between the measured angle from the accelerometer and the gyro bias. The state update (state estimation) can be represented using the following equation:

$$\theta_{k+1} = \theta_k + \dot{\theta} \times dt \tag{17}$$

And the update the state covariance matrix is

$$P_{k+1} = A \times P_k \times A' + Q \tag{18}$$

Where Q is the process noise covariance matrix. The Kalman gain is calculated from state covariance matrix P, observation matrix H and the measurement noise covariance matrix R.

$$K = P_k \times H'(H \times P_k \times H' + R)^{-1} \tag{19}$$

The state covariance matrix P, is updated by the Kalman gain K, and the observation matrix H

$$P_{k+1} = P_k - K \times H \times P_k \tag{20}$$

The state estimate is updated from the Kalman gain and the error between the calculated sensor output and the actual sensor output (measured at this point).

$$x_{k+1} = x_k + K(\theta_{estimated} - \theta_{measured}) \tag{21}$$

Fig.8. shows the output of the IMU after Kalman filtering, it shows a great improvement as it successfully filter out the noise due to accelerometer jitter effect, gyro bias drifting and the

most effecting parameter the vibration due to rotor spinning at high speed. The average error of the raw measurement is almost (80 deg.) which is very large and unacceptable. The average error of the LP filtered measurement is (30 deg.). The average error of the estimated measurement using Kalman filter and the LP filtered data is (2 deg.). The IMU sensor generates its output with a frequency of (125 MHz) and hence, for a digital controller that tracks the system and captures the best of its dynamics, the control loop time should be less than (0.008 sec.).

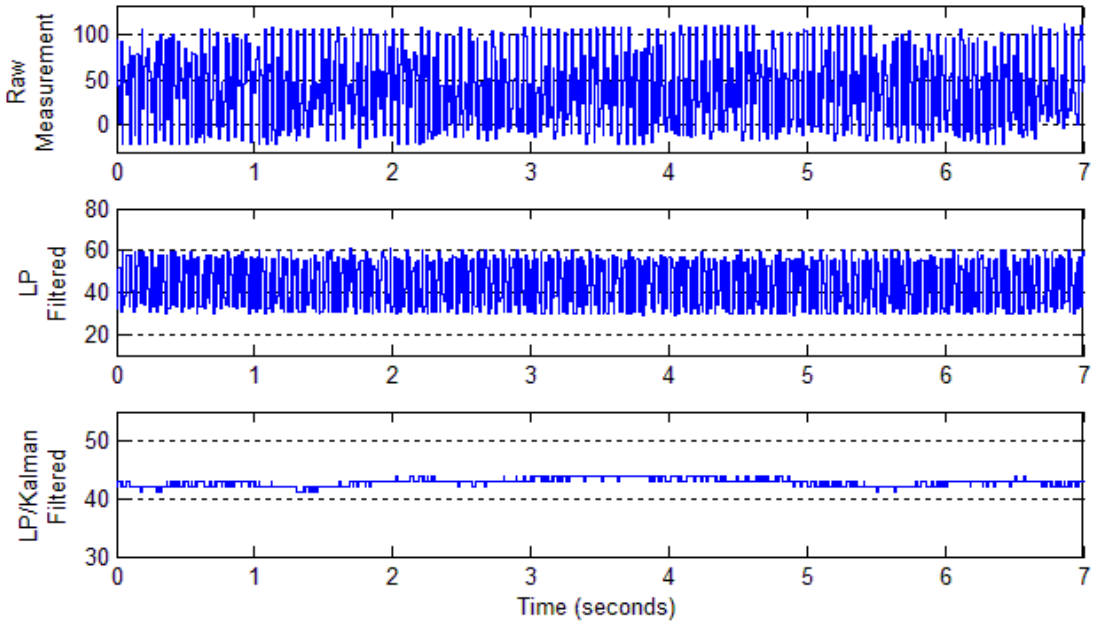


Fig.8. Measured and filtered data of the roll angle (deg.).

Controller Design and Implementation

The system is to be controlled using a digital PID controller. The proportional, integral, derivative, or more popularly, the PID, is probably one of the most popular controllers in use today [16]. Equation (X) describes the basic operation of the digital PID controller:

$$U(k) = K_p E(k) + K_i I(k) + K_d D(k) \tag{22}$$

The system is represented in a sampled-data form. E (k) is the input to the controller and the k^{th} sample of the error signal. The output of the controller is called control command, U (k). I (k) represents the integral the k^{th} error sample while D (k) represents the derivative of the same error sample. K_p, K_i and K_d are the proportional, integral and derivative controller parameters. Fig.9. describes the structure of the digital PID controller.

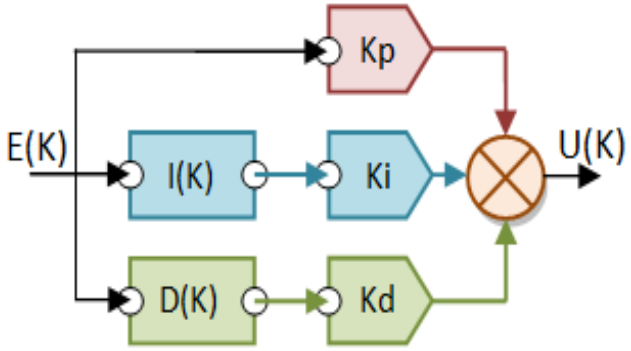


Fig. 9. Structure of the digital PID controller.

The three branches of the system are distributed calculations. The first branch is computed by simply multiplying the error sample with the K_p parameter. The second and third branches take more time to execute as they need to compute the integral and derivative of the error sample. The digital PID control algorithm is usually implemented with a sequential algorithm.Fig.10. Describes the digital PID control algorithm.

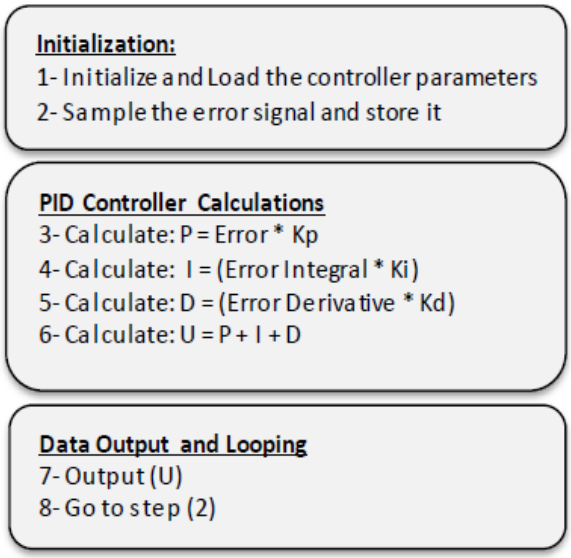


Fig. 10. Operations within the sequential PID algorithm.

Using the identified model of the system, the parameters PID controller was estimated using the PID tuning tool in MATLAB. The tuned PID parameters are used as base parameters for the control system. Variations over the three parameters are tested and recorded to be used for a fuzzy-logic tuning FLT program that monitors the system state and selects the best combination of parameters for the next control action. Fig.11. shows the simulation model of the control system including the FLT using Matlab.

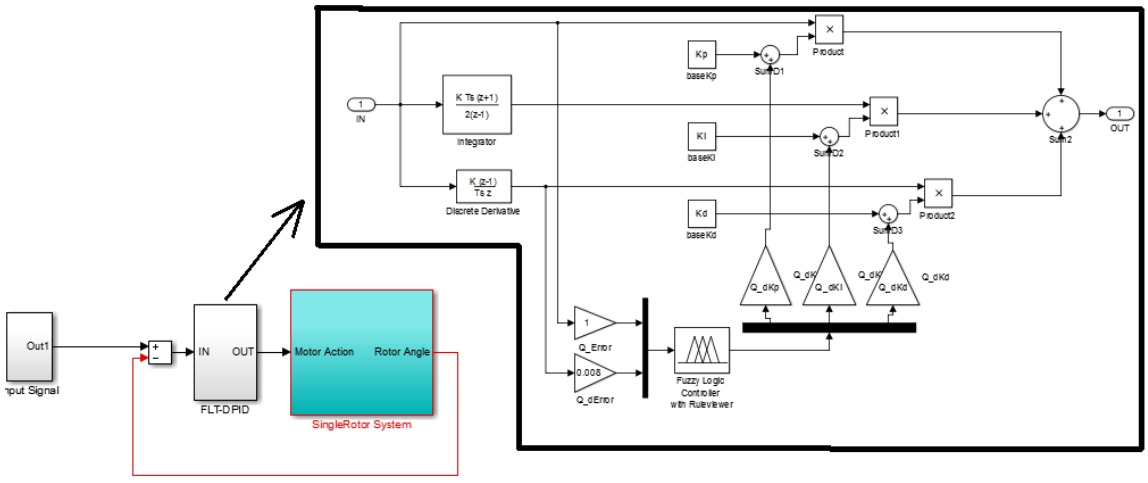


Fig. 11. Control system and FLT simulation model

The primary role of an embedded controller's software for this system is to measure the roll angle, to filter it, to calculate an action using a PID algorithm, and to interface the BLDC motor. Moreover, additional tasks are assigned to this control software. A self-tuning task for the PID parameters and an information logging task are also considered. Each task is separated and its time is measured. Table II shows the execution time of each of the tasks. All tasks are implemented within the Propeller MCU using SPIN language to optimize code size.

It is clear that if the tasks are executed sequentially, the 0.008 second time frame will be missed. Hence, the Propeller multicore MCU is used to execute the tasks simultaneously where each tasks is executed within a single cog.

TABLE I. TIMING DETAILS OF INNER TASKS

Task	PID	IMU I2C Interfce	LP/Kalman. Filter	BLDCM Interface	FLT *	Logging
Time (m. s.)	1.5	2.1	3.2	1.0	3.2	3.6
*: A minimized version to tune the K_i parameter only.						

Fig.12. shows the complete controller block diagram. The design consumes 7 cogs of the Propeller MCU. Each task within the controller's software is executed within a dedicated cog. A single cog is dedicated to 32-bit floating point operations for better performance. The communication between tasks is represented with arrows and implemented using shared memory variables. The logging task has access to all shared variables and this helps collecting logging information. It has access to a serial interface to broadcast the logging data to a connected computer. The total control loop time is the time of the longest task which is 3.6 milliseconds. This makes a speedup of almost 75% compared to a sequential implementation of all tasks. A delay at the end of the control loop is added to synchronize the control loop with the sensor data feed. However, the remaining time could be used for more tasks within each cog. The gathered logging data is used to capture the response of the system during different development stages. Fig.13. shows the response of the system to a changing step input. The response is gathered twice; with the FLT disabled and with it enabled.

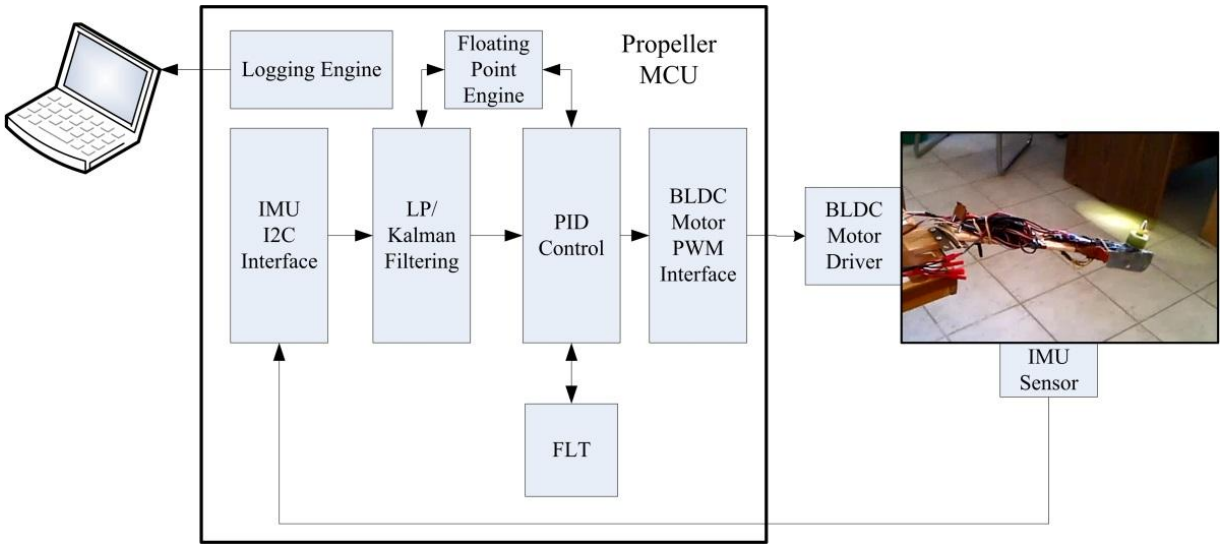


Fig.12. The complete embedded multicore controller.

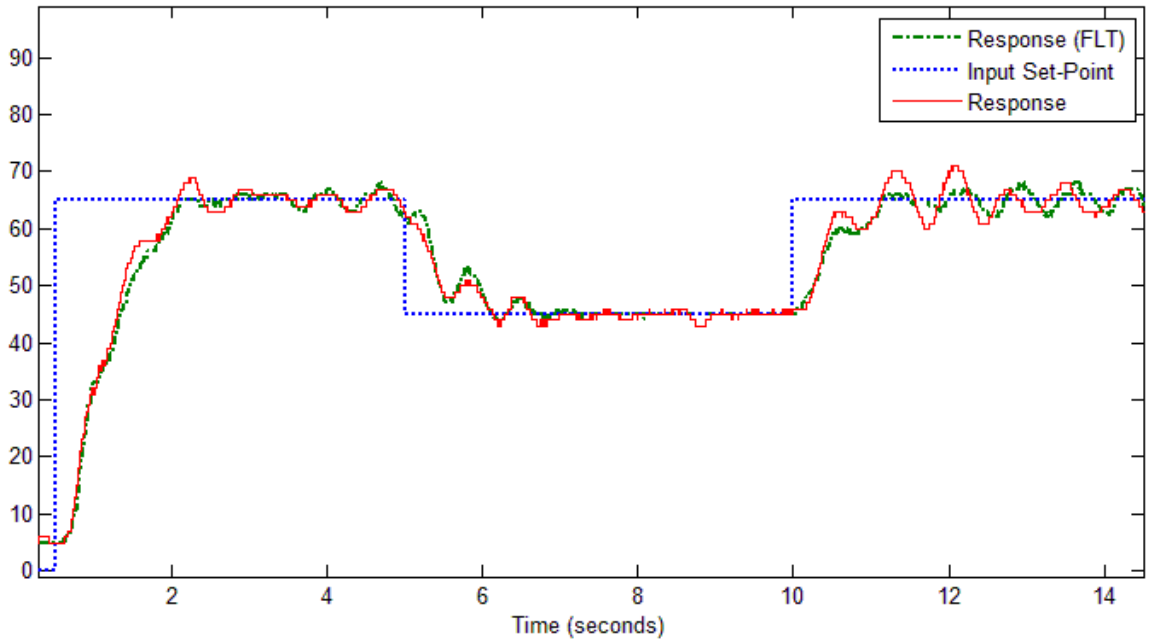


Fig. 13. Roll angle (deg.) measurement of the system under control.

Conclusions

In this article, a model of single rotor helicopter system based on Newtonian mechanics was developed. The system is nonlinear. Using black-box approach is represented the dynamic properties of the system (identified using MATLAB Identification toolbox). We have described the design of the controller. We have designed the linear controller using PID algorithm to stabilize hovering the one DOF helicopter system. Parameters PID controller was estimated using the PID tuning tool in MATLAB and then the tuned PID parameters are used as based Parameter for fuzzy logic tuning. The sensors readings of the inclination angle while the rotor is running at its minimal hovering speed. Given the outputs provided by the available sensors, a Kalman filter was developed capable of estimating the angular velocity and Euler angle of the single-rotor helicopter system. The implemented control system uses a multicore microcontroller to execute all tasks simultaneously and hence improves performance and functionality. Finally, we get from this article show that the simplified model of single-rotor helicopter can be controlled using multicore microcontrollers, it reduced design-time, implementation-time and cost while keeping higher performance rates and the execution time of the algorithm reduced by 75%.

References

[1] UBM Tech Electronics, "2014 Embedded Market Study," SAN FRANCISCO, April 21, 2014.

[2] H. A. Youness, M. Moness and M. Khaled, "MPSoCs and Multicore Microcontrollers for Embedded PID Control: A Detailed Study," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2122-2134, Nov. 2014.

[3] S. Chon, "What it Takes to do Efficient and Cost-Effective Real-Time Control with a Single Microcontroller: The C2000™ Advantage (SPRY-157)," Texas Instruments Inc., Dallas, Texas, Jan. 2011.

[4] M. Moness, H. A. Youness and M. Khaled, "Direct Mapping of Digital PID Control Algorithm to a Custom FPGA-Based MPSoC," in *In proceedings of 8th International Conference of Electrical Engineering (ICEENG), At M.T.C*, April 2012.

[5] S. Chon and B. Novak, "Performance without Compromise: Implementing Real-Time Control and

Communications with a Dual-Subsystem Microcontroller (SPRY-174A)," Texas Instruments Inc., Dallas, Texas, Sep. 2012.

[6] STMicroelectronics Inc., "32-Bit Power Architecture® Microcontroller for Automotive SIL3/ASILD Chassis and Safety Applications," STM Inc., DS9374, Data Sheet, Geneva, Switzerland, Nov. 2012.

[7] R. Marau, P. Leite, M. Velasco and P. Marti, "Performing flexible control on low-cost microcontrollers using a minimal real-time Kernel," *IEEE Trans. Ind. Informat.*, vol. 4, no. 2, p. 125–133, May 2008.

[8] Y.-D. Hong, C.-S. Park and J.-H. Kim, "Stable bipedal walking with a vertical center-of-mass motion by an evolutionary optimized central pattern generator," *IEEE Trans. Ind. Electron.*, vol. 61, no. 5, p. 2346–2355, May 2014.

[9] M. Idirin, X. Aizpurua, A. Villaro, J. Legarda and J. Melendez, "Implementation details and safety analysis of a microcontroller-based SIL-4 software voter," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, p. 822–829, Mar. 2011.

[10] H. G. d. Marina, F. J. Pereda, J. M. Giron-Sierra and F. Espinosa, "UAV attitude estimation using unscented Kalman filter and TRIAD," *IEEE Trans. Ind. Electron.*, vol. 59, no. 11, p. 4465–4474, Nov. 2012.

[11] B. Akin, M. Bhardwaj and S. Choudhury, "An integrated implementation of two-phase interleaved PFC and dual motor drive using single MCU with CLA," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, p. 2082–2091, Nov. 2013.

[12] G. Han, H. Zeng, M. D. Natale, X. Liu and W. Dou, "Experimental evaluation and selection of data consistency mechanisms for hard real-time applications on multicore platforms," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, p. 903–918, May 2014.

[13] Parallax Inc., "Parallax Propeller: The electronics industry with highquality robotics and revolutionary microcontrollers," 2014. [Online]. Available: http://www.parallax.com.

[14] XMOS Inc., "XMOS xCore: Multicore microcontrollers," 2014. [Online]. Available: http://www.xmos.com.

[15] T. Chingozha, O. Nyandoro and A. Malani, "Low cost controller for small scale helicopter," in *8th IEEE Conf. Ind. Electron. Appl. (ICIEA'13)*, Jun. 19–21, 2013,.

[16] J. Ledin, Embedded Control Systems in C/C++: An Introduction for Software Developers Using MATLAB, CMP Books, January 12, 2003.