2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)
July 6-11, 2014, Beijing, China

Real Time Fuzzy Controller For Quadrotor Stability Control

Pranav Bhatkhande
Department of Electrical and Computer Engineering
Michigan Technological University
Houghton, MI 49931 USA
Email: psbhatkh@mtu.edu

Timothy C. Havens
Department of Electrical and Computer Engineering
Department of Computer Science
Michigan Technological University
Houghton, MI 49931 USA
Email: thavens@mtu.edu

***Abstract*—In this paper, we develop an intelligent neuro-fuzzy controller by using *adaptive neuro fuzzy inference system* (ANFIS) techniques. We begin by starting with a standard *proportional-derivative* (PD) controller and use the PD controller data to train the ANFIS system to develop a fuzzy controller. We then propose and validate a method to implement this control strategy on *commercial off-the-shelf* (COTS) hardware. Using model based design techniques, the models are implemented on an embedded system. This enables the deployment of fuzzy controllers on enthusiast-grade controllers. We evaluate the feasibility of the proposed control strategy in a model-in-the-loop simulation. We then propose a rapid prototyping strategy, allowing us to deploy these control algorithms on a system consisting of a combination of an ARM-based microcontroller and two Arduino-based controllers.**

I. Introduction

Quadrotors (also called quadcopters) are flying vehicles with four vertically-mounted rotors that are typically found in a "plus" or a "X" frame. The four arm-mounted motors provide four thrust vectors to the system. The mechanical simplicity of the system is contrasted by the complexity of the problem of controlling these systems. Being under-actuated, and having no redundancy, the control problem is of utmost importance. Quadrotors have multiple uses; they're highly maneuverable and have the ability to reach places that might be dangerous to humans. Quadrotors can also be used as remote sensor pods. In disaster areas, Quadrotors can provide a high quality information bridge between the disaster zone and the rescue teams. They can also help in automated inspection of infrastructure. These platforms can also provide soldiers with high-quality, timely information in a combat situation. The low cost, high maneuverability, and easy of manufacturing make these machines very interesting.

Quadrotors are under-actuated, i.e., they have four motors to control six *degrees of freedom* (DOF). This makes the control problem quite complicated in many cases. In this paper, we first briefly discuss the dynamics of the quadrotor. For a detailed description of the system dynamics consult reference [1]. It must be noted that the Quadrotor dynamics here do not consider the coupling in very high speed maneuvers—we are primarily interested stable platforms for remote-sensing use.

We then introduce our proposed control strategy. We discuss how we developed this strategy using an ANFIS system [2]. We show that the ANFIS system is very useful in creating fuzzy controllers when the dynamics of the system are well known. With the vast amount of theory available for tuning PID controllers, a PD controller tuned to control the dynamic system can be used to initially train the fuzzy controller. We then show that by modifying the training data, we can improve the performance of the controller and incorporate features like robustness. Tuning a quadrotor is a difficult process when using PID controllers. We hope to reduce the tuning problems with the proposed fuzzy control strategy.

This paper also investigates a hardware implementation of our proposed fuzzy controller on an ARM-based micro-controller. To cut development time, we propose a method to rapidly develop fuzzy control algorithms and then implement these algorithms on COTS ARM-based components. We show that an enthusiast grade controller—the APM 2.5/APM 2.6—can be augmented to incorporate the more complicated controller. A system is developed where these Arduino-Mega based controllers can communicate over *user datagram protocol* (UDP) to ARM-based boards. The ARM-based chips handle the heavy processing, while the Arduinos are used as end actuators that provide the *pulsewidth modulation* (PWM) control signals.

We conclude by showing results of the fuzzy control strategy and comparing it with a PD controller. We then show the viability of the hardware implementation and propose method for hardware-in-the-loop simulator testing [3].

II. Quadrotor Dynamics

We begin by deciding upon a dynamic system model for the quadrotor system. The frame that is taken into consideration for developing our control strategy is shown in Fig. 1. It must be noted that the z-axis is taken in the downward direction—toward the ground or into the paper. This is especially important since it follows the aerospace convention. The directions for the motors are also as shown in Fig. 1. Reference [1] explains the dynamics frame in more detail. We follow the same expressions in [1] for rolling torque and pitching torque.

Consider the frame shown in Fig. 1. The vehicle has a thrust in the upward direction, the negative direction of the z axis. Let us denote the motor thrusts as T_i, the speed of each motor as ω_i, b is the lift constant, and $i \in \{1, 2, 3, 4\}$ represents the labels for the motors. Hence, the thrust from each motor can be calculated as

$$T_i = b\omega_i^2,\ i = 1, \ldots, 4.$$

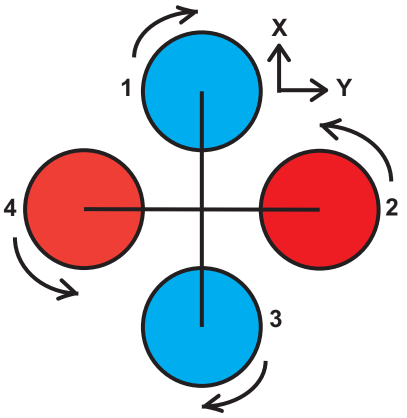

Fig. 1. 'Plus' quadrotor frame—x-axis points forward, y-axis to the right, and z-axis points down toward ground (into the paper in this figure)

This upward thrust is opposed by the force of gravity acting in the downward direction, i.e., $F_g = mg$. So for a vehicle of mass m, the dynamics are given by

$$F_t = mg - \sum_{i=1}^{4} T_i.$$

In order to *rotate* or yaw the vehicle, the controller uses a pairwise difference in the thrust of motors 1 and 3. In order to *roll* the vehicle, a correspondingly difference of force is input to motors 2 and 4.

Consider r is the distance between the center of the airframe, as seen in Fig. 1. We now define two torque values, τ_x and τ_y, as the *rolling torque* and *pitching torque* acting along the x and y axis respectively:

$$\tau_x = rb(\omega_4^2 - \omega_2^2); \tag{1a}$$
$$\tau_y = rb(\omega_1^2 - \omega_3^2). \tag{1b}$$

Now, we consider the *aerodynamic drag*, denoted as D, that acts to oppose thrust. The drag component corresponding to every T_i is denoted as D_i. The factor k depends on factors similar to the lift constant b. Thus, *aerodynamic drag* is defined as

$$D_i = k\omega_i^2.$$

This aerodynamic drag creates a reaction torque that acts to oppose the intended motion of each of the motors. This reaction torque is given by

$$\tau_z = D_1 - D_2 + D_3 - D_4. \tag{2}$$

As you can see, (1) and (2) describe the torque along each of the three axes of the vehicle given the four motor speeds.

Given a total torque vector

$$\xi = (\tau_x, \tau_y, \tau_z)^T,$$

the rotational equations of motion given by

$$I\dot{\mathbf{a}} + \mathbf{a} \times I\mathbf{a} = \xi, \tag{3}$$

where I is the inertia matrix and $\mathbf{a}$ is the angular velocity vector around each axis. I is diagonal for an ideal quadcopter model,

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}.$$

Now consider ω_T as the motor speed vector, we state the matrix A as

$$A = \begin{pmatrix} -b & -b & -b & -b \\ 0 & -rb & 0 & rb \\ rb & 0 & -rb & 0 \\ k & -k & k & -k \end{pmatrix},$$

We define ω_T as

$$\omega_T = \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix},$$

and γ as the thrust/torque vector

$$\gamma = \begin{pmatrix} \sum_{i=1}^4 T_i \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix}.$$

$$\omega_T = A^{-1}\gamma \tag{4}$$

The position is x, y and z and the pitch, roll and yaw angles are denoted as θ_r θ_p and θ_y The vehicle under-actuated - we need to generate a pitch angle- θ_p to create a forward velocity; Control over θ_p and θ_r enables control of the quadrotor. Modern enthusiast controllers like APM 2.5 (*Arducopter*) output a state vector. Note that, in order to calculate the x, y, z positions one has to calculate the appropriate rotation matrix $\Re$. More can be read about the dynamics of this vehicle in [1]. The final state vector of the vehicle is

$$\mathbf{x} = (x, y, z, \theta_r, \theta_p, \theta_y, \dot{x}, \dot{y}, \dot{z}, \dot{\theta_r}, \dot{\theta_p}, \dot{\theta_y}), \tag{5}$$

where $(x, y, z, \theta_r, \theta_p, \theta_y)$ is the 6 DOF pose of the vehicle (i.e., position and rotation) and $(\dot{x}, \dot{y}, \dot{z}, \dot{\theta_r}, \dot{\theta_p}, \dot{\theta_y})$ are the rates of change in each of the 6 DOF pose variables. In our real-world system, the state vector is provided to the controller by an *inertial measurement unit* (IMU) or some other collection of pose-estimate sensors.

III. Control Strategy Design

A. Traditional control strategy

To stabilize the quadrotor system, the typical strategy is to have three PID control loops that continuously measure the current pitch, roll and yaw; given by (θ_r, θ_p, θy) and the change in the respective quantities ($\dot{\theta_r}$, $\dot{\theta_p}$, $\dot{\theta_y}$) relative to some desired pose. The request for the change in attitude is by the user in the form of remote control commands, by a radio, or predefined flight-plan. [4] Tuning the parameters is a very difficult task, for this under-actuated system. Although it might be theoretically possible to analytically tune the gains of the PD controller for the quadrotor, reforming this analysis for every new configuration of the quadrotor becomes difficult and tedious. Modified tuning techniques can also be used to tune the PD controller [5]. In our application, the PD controller is tuned using classical tuning methods for optimal response as described in [1].

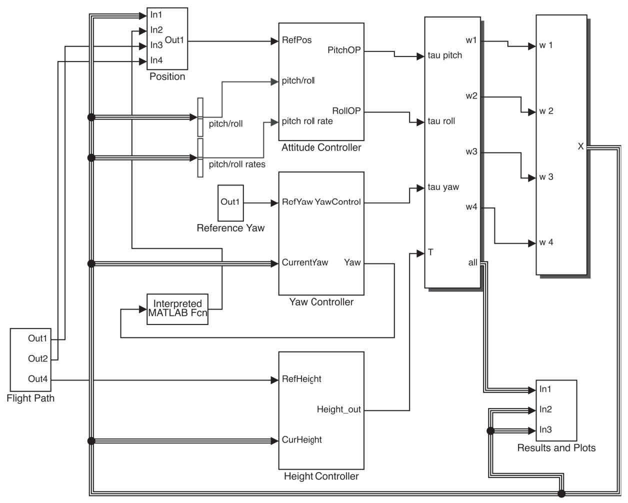

Fig. 2. Overall Control Loop

B. PD Controller

As noted in II, we need to control the *roll, pitch, yaw*; stated as $(\theta_r, \theta_p, \theta_y)$, and $(\dot{\theta}_r, \dot{\theta}_p, \dot{\theta}_y)$. We define the *Proportional gain values* for *roll, pitch, yaw* as $(K_{p_r}, K_{p_p}, K_{p_y})$ and derivative gain values as $(K_{d_r}, K_{d_p}, K_{d_y})$. Note that a feedforward constant C is added to the altitude controller to balance the weight of the quadrotor against the force of gravity. This is then given as

$$C = \sqrt{\frac{mg}{4b}}. \tag{6}$$

The control equations are given as

$$\tau_x = K_{p_r}(\hat{\theta}_r - \theta_r) + K_{d_r}(\hat{\dot{\theta}}_r - \dot{\theta}_r); \tag{7a}$$

$$\tau_y = K_{p_p}(\hat{\theta}_p - \theta_p) + K_{d_p}(\hat{\dot{\theta}}_p - \dot{\theta}_p); \tag{7b}$$

$$\tau_z = K_{p_y}(\hat{\theta}_y - \theta_y) + K_{d_y}(\hat{\dot{\theta}}_y - \dot{\theta}_y); \tag{7c}$$

$$T = K_{pZ}(\hat{Z} - Z) + K_{dZ}(Z - \dot{Z}) + C. \tag{7d}$$

C. Control splitting

The outputs generated by the four controllers above are split among the four motors. This is called control splitting. Let the contribution of each be denoted by f_r, f_p, f_y and f_z respectively for roll, pitch, yaw and altitude.f_x; $x = r, p, y, z$ are all *rpm values*.

$$\omega_1 = f_p + f_y + f_z \tag{8a}$$

$$\omega_3 = -f_p + f_y + f_z \tag{8b}$$

$$\omega_2 = -1(-f_r - f_y + f_z) \tag{8c}$$

$$\omega_4 = -1(f_r - f_y + f_z) \tag{8d}$$

Note that the output of the *altitude controller* is added equally to all the pairing of motors here; this allows the roll and pitch of the vehicle. This control splitting block is the same for the PD controllers well as the Fuzzy controller desribed in section III-D. In Fig. 2 we show the placement of the control splitting block.

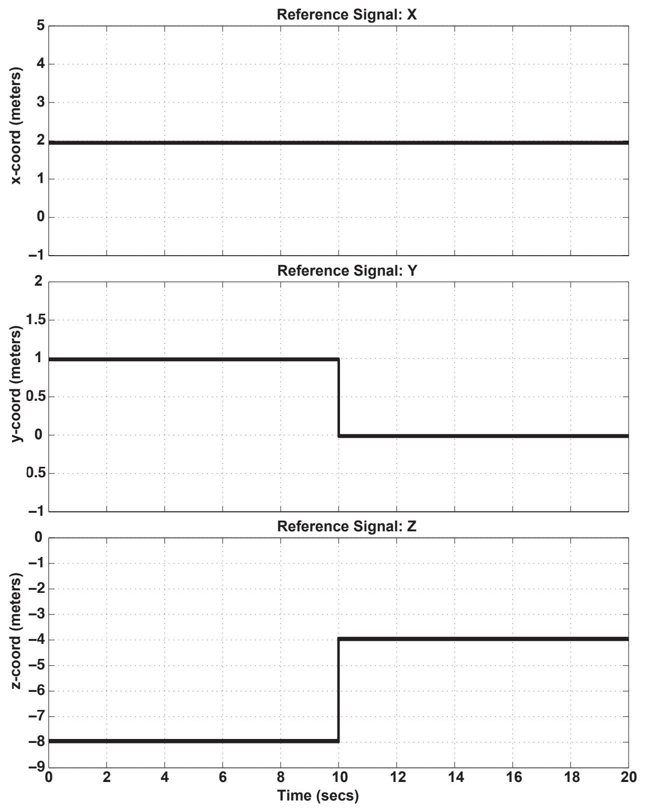

Fig. 3. Experimental (x, y, z) signals

D. Fuzzy control strategy

In this section, we develop a fuzzy control strategy to control the quadrotor as described in Sec. II. We propose a strategy based on the ANFIS system [2]. We first set up an experiment, collect data from this experiment and then create a controller from the training data obtained. The derived controller is used to control the quadrotor.

1) Experimental Setup: The goal of the experiment is to create a closed loop scenario, in which we can test control algorithms against an approximate dynamics model described in Sec. II. We define x, y and z coordinates, the set (x, y, z) is where we could like our quadrotor to go. In the absence of *Radio Control* (RC) commands, these serve as a good replacement. For illustration, consider Fig. 3; here we keep the value of x constant and request changes in the y and z coordinates. Various input conditions are investigated in an effort to create better controlling data as explained.

2) Generating training data: We first log data from the experiment set up above. The experiment is first run for the z controller, in this case, we first train the system to go from a height of 0 to a maximum step height, thus simulating the step response. We log data for *z*, *dz* and *rpm change* due to the z controller.

A similar process is repeated for the attitude control and yaw control of the vehicle. Data logged from this process is then fed into the ANFIS system.

3) Learning controller from training data: ANFIS combines a neural network with fuzzy logic and thus achieves a

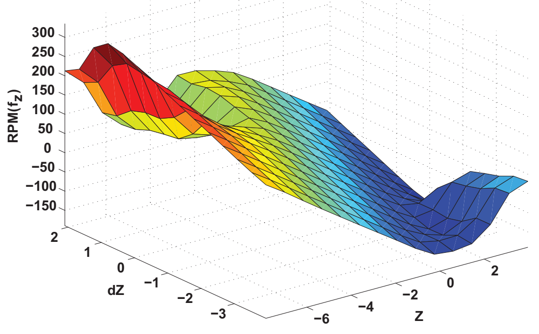

Fig. 4. Surface view: Height controller

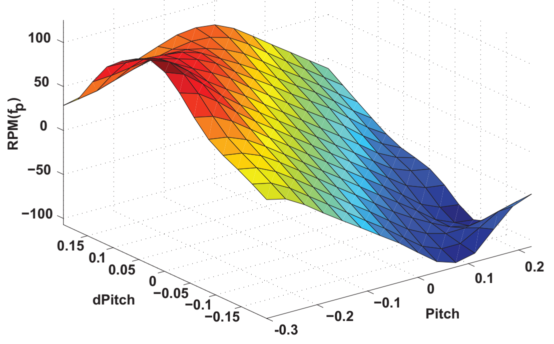

Fig. 5. Surface view: Pitch controller

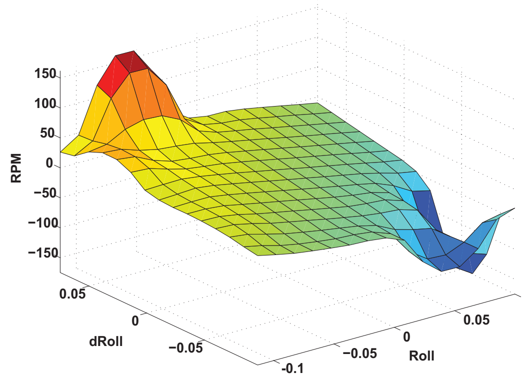

Fig. 6. Surface view: Roll controller

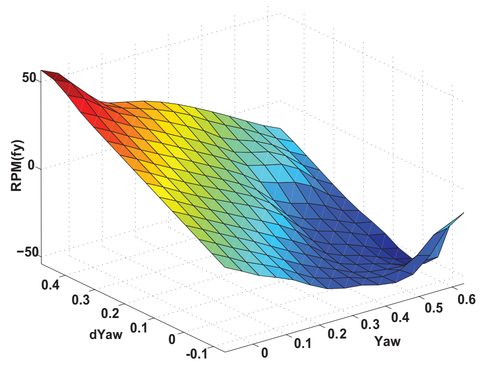

Fig. 7. Surface view: Yaw controller

learning mechanism for a fuzzy rule base. It is widely regarded as an universal estimator [6]. We propose that the controller only has to learn once, in a simulation or a hardware-in-loop test, and the code deployed to the embedded hardware would perform well compared to a PD or PID controller. In this effort, we collect training data from the above experiment and feed it into the ANFIS system [2]. The follow parameters are used in the ANFIS system:

- number of inputs: 2
- number of outputs: 1
- number of rules: 25
- type of membership functions: Gaussian Bell functions
- fuzzy inference system: Sugeno
- intersection: product
- union: max
- defuzzification : weighted average

Figures 4–7 show the surface views for the four learned controllers. The ANFIS system has the ability to leverage neural networks and fuzzy rules to create a fuzzy inference system. We do this for all our sets of the training data, and create the rule bases for our controllers.

While ANFIS systems are very good at producing high quality fuzzy rule bases (as a universal estimator), they are computationally complex. However, hybrid learning algorithms [7] could be used to produce good control rule bases more efficiently.

IV. Hardware implementation

In this section we investigate a proposed hardware implementation for the control strategy described above. The Arduino Mega based Arducopter system [8] cannot process a complex controller like this one; thus, we implemented a hardware solution by having an additional microcontroller board, the GUMSTIX Overo FIRESTORM COM. This is a dual core ARM Cortex A8 board with 512MB of memory. Figure 8 illustrates a schematic of the implementation on the proposed hardware.

A. *Challenges in hardware implementation*

Fuzzy controllers are fundamentally more complex to implement or process; Most enthusiast grade microcontrollers work on Arduino-based boards. Top of the line Arduino Boards do not have the ability to implement a fuzzy controller; this limitation is due to the memory and the processor architecture. Our initial experiments on the APM 2.5 (Arduino-Mega derived flight controller) showed that the APM 2.5 board failed to implement fuzzy controller.

To address this, we first attempted an implementation on the Raspberry Pi. The Pi performs well with a single fuzzy controller (e.g., the height controller), but struggles to keep

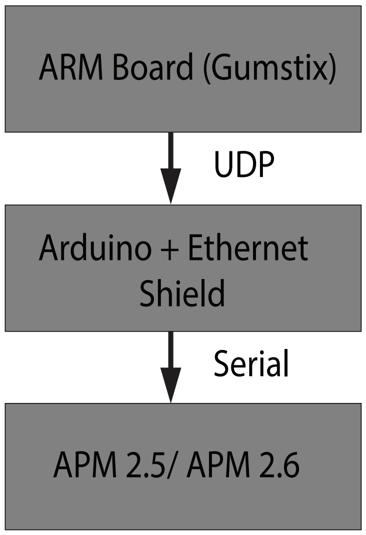

Fig. 8. Hardware Implementation

up when all four controllers are implemented. The Gumstix system was found to be better performing; this is due to it being a dual core chip, in addition to that it is also clocked higher.

The method of communication between the Arduino and the Gumstix is decided upon to be *User Datagram Protocol* (UDP). The reasons for this is as follows. The GPIO pins of the Gumstix can only read and write logical values, they are not useful for sending IMU information. The MATLAB implementation for both microcontrollers supports UDP; hence, UDP is decided upon for its universal nature and fast processing.

Programming various types of microcontrollers in various languages leads to a huge development overhead. This time can be cut down by a rapid prototyping strategy. We use a strategy based on that proposed in [9]. Similar strategies are used in automobiles for programming Electronic Control Units [10].

B. Rapid implementation strategy

A workflow [10] is developed to implement the control algorithm on hardware. First, we develop the control algorithm in Matlab/Simulink. We generate C/C++ code using code generation capabilities within Matlab. After the code is generated, the native code is exported to the microcontrollers. The build system for Arduinos required modification to be used with Arducopter [8]. The Gumstix code was generated directly from Matlab. Note that with this process, three microcontrollers are programmed to perform various tasks. These are listed as below and illustrated in Fig. 8. More can be read about such a strategy in [9].

1) Gumstix Overo FIRESTORM – heavy processing, filtering, processing fuzzy controller;
2) Arduino UNO – basic relay between APM 2.5 [8] and Gumstix microcontroller;
3) Arduino Ethernet Shield – UDP packet bridge between Gumstix and Arduino.

C. Data-flow

A custom build Arduino Board with an onboard IMU (APM 2.5) [8] generates the Yaw, Roll, and Pitch, and the difference in all those quantities per time step. These data are sent to the Arduino UNO board, and the Arduino UNO board acts as a relay between the APM controller and the Gumstix Microcontroller. We use the Arduino Ethernet Shield to transfer the data from the APM to the Arduino. The Arduino then sends the data via UDP to the Gumstix microcontroller. The Gumstix microcontroller returns the control data via the reverse loop, enabling us to send commands to the APM controller. The APM controller is connected to the speed controllers. Figure 8 shows this process.

D. Hardware actuation

The motor control hardware consists of the APM 2.5 controller. This board has an onboard IMU unit in addition to all the features provided by an Arduino Mega board. The motor speed controllers are connected to the analog output channels of the APM 2.5 board. Hence, the end speed control request is sent to the speed controllers over the analog output channel of the APM 2.5 board. The motors are connected to these speed controllers, controlling the thrust of the vehicle. At this, the *(roll, pitch, yaw)* and difference in roll, pitch, yaw *(droll, dpitch, dyaw)* are noted, this information is sent back to the Arduino, which sends it back to the Gumstix, the fuzzy controllers takes this input, and the process repeats.

V. Results

A. Controller evaluation

We first evaluate the performance of our controller in simulation. The ANFIS derived fuzzy controller performs just as well as a fully tuned PD controller when it is given the same goals. The viability of the control strategy to control the quadrotor is established. Both the PD controller and the ANFIS-derived fuzzy controller show very similar rise time, and similar overshoots. The fuzzy controller for z control has a lower overshoot compared to the PD controller. The results prove that the fuzzy controller is easily capable of controlling the quadrotor. Furthermore, the controller is automatically tuned, as opposed to the PD controller which was manually tuned for optimal operation.

B. Hardware implementation testing

The computational complexity of an ANFIS system is the major hurdle in implementing these controllers on embedded hardware. We solved this problem by creating an architecture as seen in Fig. 8. We verify these results by implementing the four fuzzy controllers on the Gumstix hardware and communicating between the three microcontrollers. In our preliminary hardware test, we were able to show that the Gumstix can efficiently implement the FIS rule base and communicate with the other microcontrollers; the next step would be to create a true hardware-in-loop test [3]. We save this analysis for the sequel to this paper.

The result is a fuzzy controller that has the ability to train from look-up tables and PD controller data. The concepts are seen widely in robotics, especially industrial robotics, where

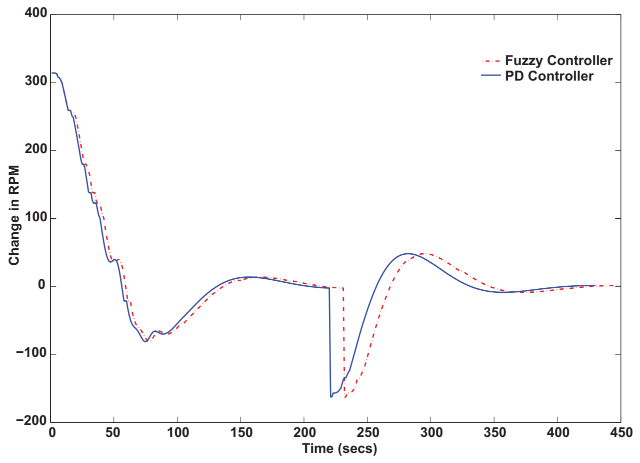

Fig. 9. Fuzzy vs PD Height Controller

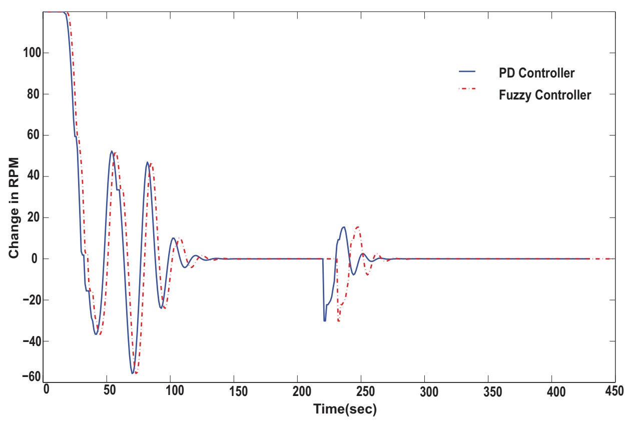

Fig. 10. Fuzzy vs PD Pitch Controller

robots are taught a motion, and they repeat it. In our case, however, by building the controller, we can not only mimic previous motions, but also very well approximate new goals. The next goal is to add a pre-filter to the reference input and improve the transient response of our baseline controller. This, in addition to filtering the jitter in the output signal, should generate better training data, helping us to outperform the PD controller.

Quadrotors are very sensitive to weight. One can design very complicated and sophisticated controllers, but fail to implement them due to logistical challenges. Our choice of materials includes the fiberglass frame—280g, Turnigy Aerodrive Sk3 motors (brush-less DC), each one weighs 31g, Lithium-Polymer batteries - 2200mAh 3S batteries, which weigh 163g each. The Arduino Uno board; and the Gumstix controller together weigh in at about 70g. Given the current battery setup, we have a flying time of about 15 to 20 minutes. The trade-off here is between weight and processing capability, a challenge in any power and weight limited robotics problem.

VI. Conclusion

In this paper, we developed a fuzzy controller for controlling a quadrotor UAV using the ANFIS technique. We evaluated the performance of the learned controller and verified its satisfactory performance. At this point, the fuzzy controller is as good as the PD controller, and is able to outperform the PD controller in certain conditions. With minor modifications to the training data, and addition of control logic, we should see an improvement in controller performance. We also proposed a hardware implementation of the controller, investigated its feasibility and implemented the hardware fuzzy controller on a dual-core ARM Cortex A8 board. We then evaluated a strategy where we can implement these controllers on existing enthusiast-grade hardware. The controller performance is found to be good in a preliminary hardware-in-loop test (Gumstix Overo connected to a Simulink model). [11] confirms that this strategy will work with real hardware.

In summary, our system provides a flexible hardware implementation for controlling multicopter aircraft. The hardware we have proposed is proved feasible and, furthermore, is flight-worthy as it is small, light, and exhibits low power usage. The next steps in our investigation to focus on improving the controller performance, tweaking the implementation, and finally performing flight tests.

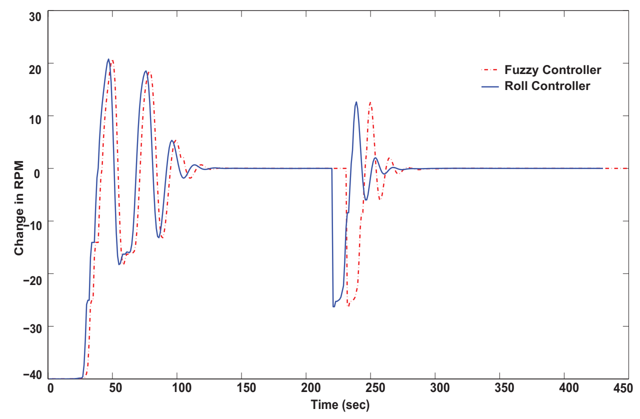

Fig. 11. Fuzzy vs PD Roll Controller

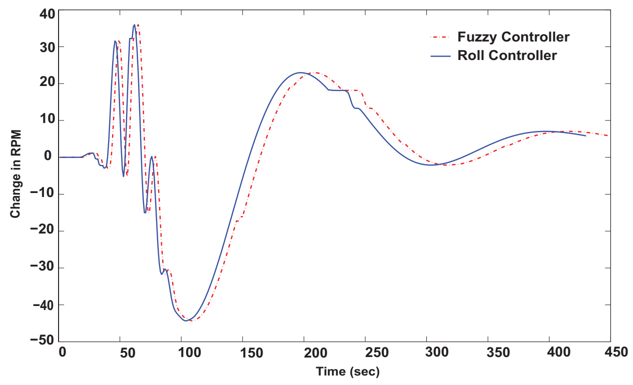

Fig. 12. Fuzzy vs PD Yaw Controller

Acknowledgements

This work was supported by a Michigan Technological University Research Excellence Fund award. Superior, a high performance computing cluster at Michigan Technological University, was used in obtaining results presented in this publication.

References

[1] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer, 2011.

[2] J.-S. Jang, "Anfis: adaptive-network-based fuzzy inference system," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, no. 3, pp. 665–685, 1993.

[3] Z. Bo, X. Bin, Z. Yao, and Z. Wei, "Hardware-in-loop simulation testbed for quadrotor aerial vehicles," in *Control Conference (CCC), 2012 31st Chinese*, 2012, pp. 5008–5013.

[4] I. Dikmen, A. Arisoy, and H. Temeltas, "Attitude control of a quadrotor," in *Recent Advances in Space Technologies, 2009. RAST '09. 4th International Conference on*, 2009, pp. 722–727.

[5] P. M. Meshram and R. Kanojiya, "Tuning of pid controller using ziegler-nichols method for speed control of dc motor," in *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*, 2012, pp. 117–122.

[6] O. Lutfy, S. B. M. Noor, and M. Marhaban, "A genetically trained simplified anfis controller to control nonlinear mimo systems," in *Electrical, Control and Computer Engineering (INECCE), 2011 International Conference on*, 2011, pp. 349–354.

[7] A. Al-Hmouz, J. Shen, R. Al-Hmouz, and J. Yan, "Modeling and simulation of an adaptive neuro-fuzzy inference system (anfis) for mobile learning," *Learning Technologies, IEEE Transactions on*, vol. 5, no. 3, pp. 226–237, 2012.

[8] P. Wallich, "Arducopter parenting," *Spectrum, IEEE*, vol. 49, no. 12, pp. 26–28, 2012.

[9] R. Toulson, "Advanced rapid prototyping in small research projects with matlab/simulink," in *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, 2008, pp. 1–7.

[10] S.-H. Seo, S.-W. Lee, S.-H. Hwang, and J. W. Jeon, "Development of platform for rapid control prototyping technique," in *SICE-ICASE, 2006. International Joint Conference*, 2006, pp. 4431–4435.

[11] G. E. M. Mahfouz, M. Ashry, "Design and control of quad-rotor helicopters based on adaptive neuro-fuzzy inference system," *International Journal of Engineering Research and Technology*, vol. 2, December 2013.