Design and Implementation of a Self-Balancing Two-Wheeled Robot Driven by a Feed-Forward Backpropagation Neural Network

Mr. Rajvardhan Shendge[1], Mrs. Tejashree Shendge[2]

[1]Student, Computer Engineering, Ramrao Aidik Institute of Technology(India)
[2]Student, Electronics and Telecommunication Engineering, Fr. C. Rodrigues Institute of Technology(India)

***Abstract** - The design, manufacture, and implementation of a revolutionary control system for a two-wheeled self-balancing robot are covered in this research paper. A pair of DC motors, an Arduino Uno R3 microcontroller board, a 3-axis gyroscope, and a 3-axis accelerometer (for altitude measurement) are all part of the system design (employed for altitude determination). The two-wheeled robot that was constructed is essentially a real-time model of an inverted pendulum open-loop system that is inherently unstable without feedback control and has nonlinear dynamics. The recommended balancing solution entails utilizing the backpropagation algorithm to train a feedforward neural network to learn to balance the bot through supervised learning on the basis of a training routine that runs at startup. In addition, gyro drifts are compensated for using the linear quadratic estimation technique (LQE) and a complementary filter. Our findings demonstrate that the suggested feedforward neural network balancing approach can learn to balance the bot in a short amount of time, with far less oscillations around the equilibrium point than a traditional PID controller, resulting in a more elegant system. Our bot is a modest, low-cost prototype that demonstrates artificial neural networks' (ANNs) efficiency and complex-learning capabilities (ANNs).*

***Key Words**:* **Two-Wheeled robot; Self-Balance control; Proportional-Integral-Derivative Controller; Kalman Filter; Arduino Uno R3; Digital Motion Processing (DMP); Backpropagation Algorithm; Feedforward Neural Network.**

1.INTRODUCTION

Over the past decade, mobile robots have penetrated public spaces such as hospitals, schools, and ordinary homes, moving away from military and industrial corridors. While many of these robots for civil purposes are technically stable, such as 'Aibo,' the Sony robotic dog, or four-wheeled intelligent vacuum cleaners, the Segway personal transport is one that everyday onlookers would find awe-inspiring. It's a mechanically unreliable, two-wheel self-balancing mode of transportation that's been used by police enforcement, tourists, and individuals. The Segway might be classified as a robot since it can never stay upright without the sensory capabilities and advanced controls that come with a robot. While the Segway is a well-known commercial product, there has been a lot of study on how to operate such a mechanical device.

Aside from the Segway, there has been a lot of study on two-wheel self-balancing robots. JOE [1] and nBot [2], for example, are both early versions featuring inertia sensors, motor encoders, and onboard microcontrollers. Since then, there has been significant research on control design for such systems, including nonlinear backstepping controls, PID controllers, application of discrete Kalman filters, fuzzy-neural control, Linear-quadratic regulators (LQR), and combinations of the aforementioned [3][4]. To deal with the equilibrium problem posed by wheeled inverted pendulums, several intelligence approaches have been developed, including fuzzy control [5][6], control systems based on support vector machines [7], operant conditioning theory [8,] and so on.

The wheeled mobile robot and the inverted pendulum mechanism are combined in the two-wheeled robot [9]. It also entails the idea of establishing a mode of human transportation. The inverted pendulum is a dynamically unstable and nonlinear open-loop system with a single-input, multi-output system configuration (SIMO), in which the system's center of mass is located above the pivot point. As a result, this system has become a classic problem in dynamics and control theory, and it is often used as a benchmark for assessing control system solutions. The wheeled inverted pendulum is not self-propelled and requires human balancing to remain upright. It uses gyroscopes and accelerometers to detect the inclination of the vertical axis, and the controller provides torque signals to each motor to counteract the inclination and prevent the system from falling. It's a control system paradigm in which an item can only be changed by adding more weight to it. As a result, wheeled inverted pendulums have become a new topic, and developing balancing algorithms to solve their equilibrium problem has piqued the interest of many specialists.

Although there are many studies in this field, only a few have focused on making meaningful comparisons between traditional control system techniques like PID controllers and LQR feedback controllers and the relatively new artificial-intelligence based neural network controllers. That is precisely what we are aiming for. We will investigate the self-balancing system using dynamic theory, establish a mathematical model, and explain the behavior of the neural net in the most detailed and clear manner possible, as well as create a simulation that demonstrates its functionality and efficiency.

2. CONSTRUCTION OF THE ROBOT

1. ACRYLIC SHEET CHASIS: Our frame is made out of acrylic sheets. To hold our three-tiered structure in place, we used metallic rods with a diameter of 3mm. The gyroscope sensor and 9V batteries are housed in the upper half. The Arduino Uno R3 microcontroller board and another 9V battery are located on the middle sheet. Finally, the driver motor controller and two DC geared motors are housed in the bottom layer.

2. MOTORS: The motors we've selected are standard DC geared motors that run on 12V and provide 1.5 kg-cm of torque at 300 rpm.

3. MOTOR CONTROLLER: The L298N motor controller is a dual H-bridge motor driver that allows us to regulate the speed and direction of two DC motors at the same time. The module can run DC motors with voltages ranging from 5 to 35V and peak currents of up to 2A.

4. GY-521 MODULE: This module measures the robot's tilt angle. The values of the same are called using a function. The MPU-6050 MEMS (Microelectromechanical system) has a 3-axis gyroscope, a 3-axis accelerometer, a digital motion processor (DMP), and a temperature sensor. The GY-521 module is a breakout board for the MPU-6050 MEMS (Microelectromechanical system). Complex algorithms may be computed directly on the board using the digital motion processor. Typically, the DMP runs algorithms that convert the sensor's raw data into accurate location data. The sensor data is gathered via the I2C serial data bus, which only requires two wires (SCL and SDA) (SCL and SDA). We used the MPU6050 Arduino library, which has reverse-engineered functions that calculate yaw, pitch, and roll using the DMP on-board the MPU6050.

5. ARDUINO UNO R3: The Arduino Uno R3 is a microcontroller board that uses the ATmega328P (datasheet) microcontroller (datasheet). There are 14 digital input/output pins (six of which may be used as PWM outputs), six analog inputs, a 16 MHz quartz crystal, a USB connection, a power connector, an ICSP header, and a reset button on the board.

6. JUMPER WIRES: Wires that connect males to males and females to males.

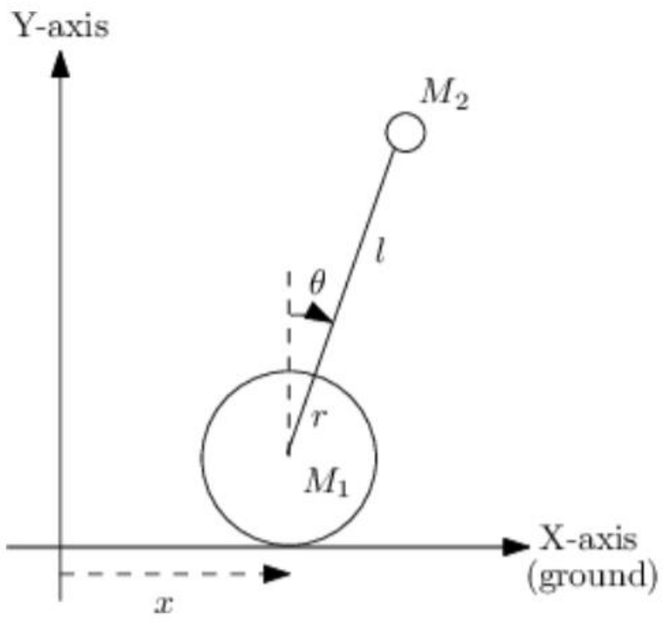

Fig. 1: Schematic model for a wheeled inverted pendulum

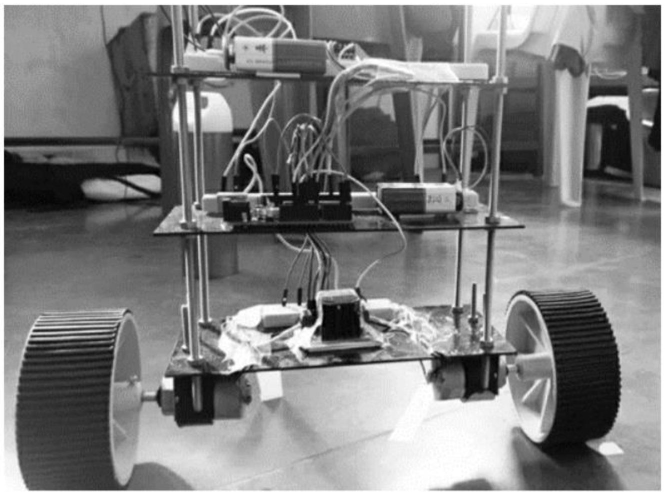

Fig. 2: Front view of constructed robot

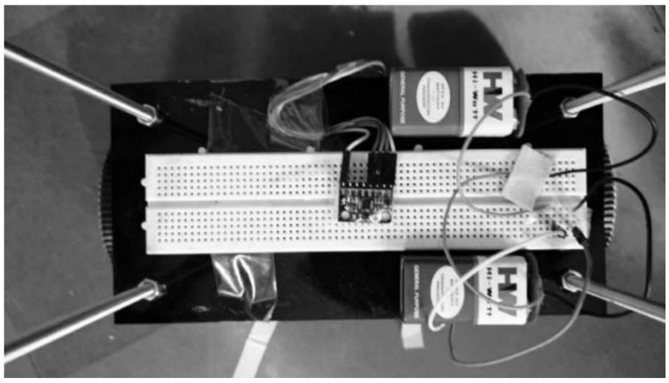

Fig. 3: Top view of constructed robot

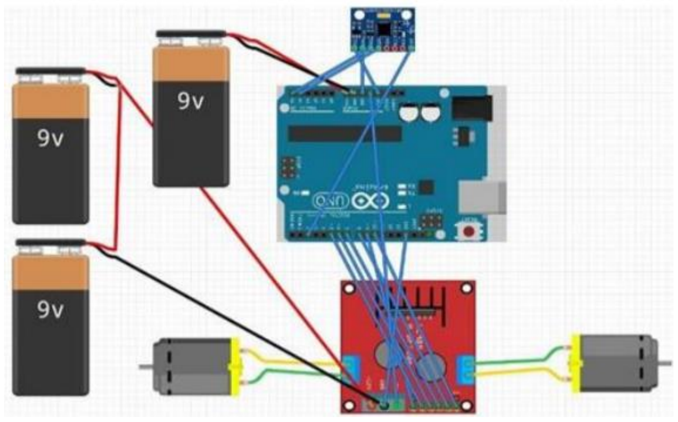

Fig 4: Circuit Schematic

3. ALGORITHMS USED

3.1 PID CONTROLLER APPROACH

We investigate two strategies for keeping the robot upright. The first method is to use a PID controller. The Proportional Integral Derivative controller (PID) is a feedback control loop mechanism that is widely used in industrial control systems and a variety of other applications that need continually modulated control [10]. The PID controller continuously calculates an error value as the difference between the desired setpoint (SP) and the measured process variable (PV) and initiates a correction based on proportional, integral, and derivative terms, which are denoted by P, I, and D, respectively, and give the controller its name. Technically, it is a control function that transmits accurate and responsive correction. PID is made up of three sections: (1). The proportional component is used to change the response's magnitude directly. The system becomes unstable if the value of Kp is too high or too low. (2). The integral part is used to smooth out prior inaccuracies and accelerate the process's progress toward the setpoint. (3). The derivative element predicts future errors based on previous error variations. Simply adding these three variables together yields the final control value. The following is the formula:

$$u=k_p e(t)+k_i \int e(t)dt+k_d \frac{de(t)}{dt}. \quad (1)$$

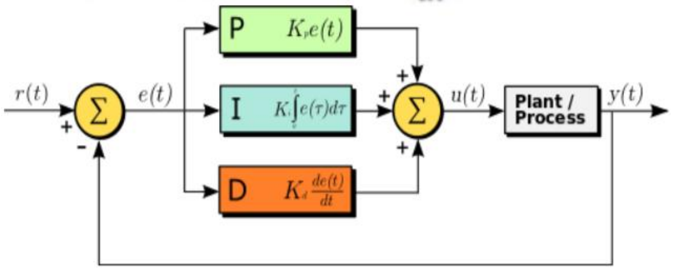

Fig 5: PID Controller Block Diagram

Algorithm to adjust Kp, Ki and Kd values: -

- We modify the value of P after setting the I and D terms to 0 so that the robot begins to oscillate around the mean position. P should be large enough for the robot to move about in, but not too large that it hinders smooth movement.

- When P is correctly recognized, I is increased, causing the robot to accelerate more quickly when it is off balance.

The robot should be able to balance itself for at least a few seconds if P and I are adjusted correctly.

- Next, the value of D is chosen so that the robot may move about its balanced position in a more friendly manner. There should also be no significant overshoots.

- If the first attempt does not provide satisfactory results, reset the PID settings and repeat the technique above until you get a satisfactory result.

- During fine tuning, the PID values are confined to nearby values, and their effects are seen in real-world situations.

3.2 FEEDFORWARD NEURAL NET APPROACH

A feedforward neural network is used in our proposed approach to control a two-wheeled self-balancing robot. The robot is incapable of doing anything other than self-balancing, and there is no way to regulate its direction. The program uses a neural network with a single input node, two hidden nodes, and a single output node to get around the Arduino Uno's 2K SRAM limitation. The program starts with a training procedure that takes just a few seconds and may be repeated every time the robot is launched. The bot must be kept upright when the backpropagation approach is used to teach the neural network routine. The input sensor values (the robot's tilt angle) are converted to a number between 0 and 1. This value is fed into the neural network model, which subsequently generates a sigmoid activation layer output that ranges from 0 to 1. Finally, we transform these output numbers to a useful value for DC motors, in this case from -255 to 255.

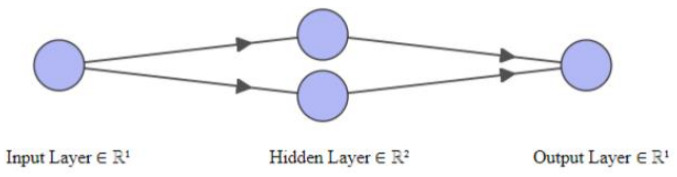
Input Layer $\in \mathbb{R}^1$
Hidden Layer $\in \mathbb{R}^2$
Output Layer $\in \mathbb{R}^1$

Fig 6: Feedforward Neural Network Architecture

4. METHODOLOGY

In order to reverse-engineer the DMP (Digital Motion Processing) algorithm of the GY-521 Breakout Board, which uses the InvenSense MPU-6050 6-Axis Accelerometer and Gyroscope Module, we use the Kalman filter, also known as the Linear Quadratic Estimation (LQR) algorithm, in conjunction with the Complementary filter to obtain information about the robot's spatial orientation. TheDMP procedures are called and used to get precise and accurate data for our mechanically unstable system's Yaw, Pitch, and Roll.

The Pitch value is then used to interpret and calculate the robot's vertical tilt angle, allowing the robot to restore its alignment afterwards. This is accomplished by using Pulse-Width Modulated Signals (PWM) to control the speed of the motors based on the tilt angle. Because PWM signals may be sent in the range of -255 to 255, a PIDcontroller is used to decide the value to be delivered as PWM. To compute the PWM value, we must first submit an input and a Setpoint to the PID controller, which will calculate the error (Setpoint – Input) and then provide an output that will reduce the error. As an input, we provide the robot's tilt angle while keeping it upright. Because the input tilt angle varies depending on the surface where the robot is housed, we must alter the value in the code.

However, none of these parameters are required in our neural network application. We simply use the back-propagation method and add a single input node, two hidden nodes, and a single output node in the neural network.

Rather of meddling with all of the Kp, Ki, and Kd parameters required by the PID Controller, we just call a training procedure at the robot's starting during which the bot must be kept upright and steady. The bot uses this vertical tilt angle as the training set's input angle and calculates the error.

The Sigmoid Activation Function is used for the neural net in this situation. The function's graph is as follows:

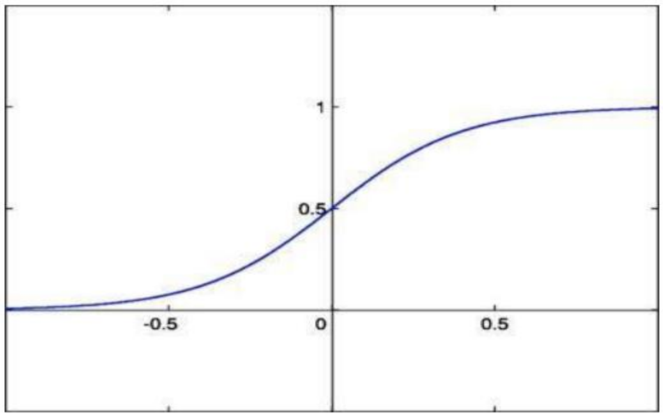

Fig. 7: Sigmoid Activation Function

The hyperparameters set for the neural net are as follows

Parameter	Value
Learning Rate	0.3
Momentum	0.9
Initial Weights	0.5
Success	0.0015

Sample Time (mS)	0.005
Loop Timer (mS)	4
Input Node	1
Hidden Nodes	3
Output Node	1

Table 1: Optimized Hyperparameter values of FNN

The value of the option 'Success' has been set to 0.0015. The difference between the requisite Set Point (vertical tilt angle) and the Current Angle (Real-Time Tilt Angle) is anticipated to be less than the value of success in order to keep the robot upright, and the neural net provides the necessary output. This value is then linearly translated into the 0-255 range. The magnitude of the PWM signal is then calculated using this number. As a result, the greater the value provided to the PWM signal, the greater the torque created by the motor, allowing the bot to maintain its upright position.

5. RESULTS AND DISCUSSION

We employed both approaches on our constructed robot, in order to compare and infer as to which algorithm worked better.

The first approach was the PID Controller approach where the following steps were taken-

- The desired setpoint was set as 6.10 (degrees from the vertical).
- The value of Kp after extensive experimenting, was set to 90 as it was optimal and made the bot oscillate about the balance position.
- The value of Ki was set to 250 as it made the balancing more efficient and provided more torque to the motors at higher tilt angles.
- The value of Kd was set at 2.15 since any value more than that made the robot jitter and any value less than that made the robot fall.

The next approach was the Neural Net approach where the following steps were taken-

- The desired setpoint was set as 6.10 (degrees from the vertical).
- The value of Learning Rate was tinkered with, and it was found that a value of 0.3 made the bot balance perfectly without oscillating at high frequencies and causing large deviations.
- A high learning rate made the output change frequently and this caused the motors to frequently change speed, thus making the robot oscillate and jitter.
- The momentum and initial weights were set as 0.9 and 0.5 respectively based on hit and trial.
- The Success value was kept at 0.0015 since any number higher or lower than that resulted in a significant change in the error computation and caused the DMP function to be called more often than required, causing the robot to jitter excessively.

In addition, in order to examine and assess the motor control signals under wheel synchronization in Fig. 8, we used the serial graph plotter included in the Arduino IDE Software. The wheel synchronization can clearly be seen to be operating.

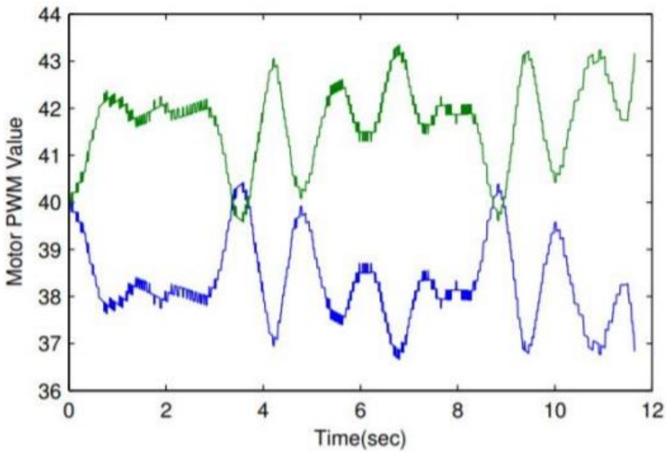

Fig. 8: PWM Motor Control Signals

The Serial Graph feature of the Arduino IDE Software was also used to create a graph between the tilt angle and the time elapsed. These graphs were shown for both approaches, and it can be shown in Fig. 9 that PID control has low stability. Meanwhile, the Neural Net provides substantially more enhanced stability, and the tilt angle deviates far less when using the Neural Net.

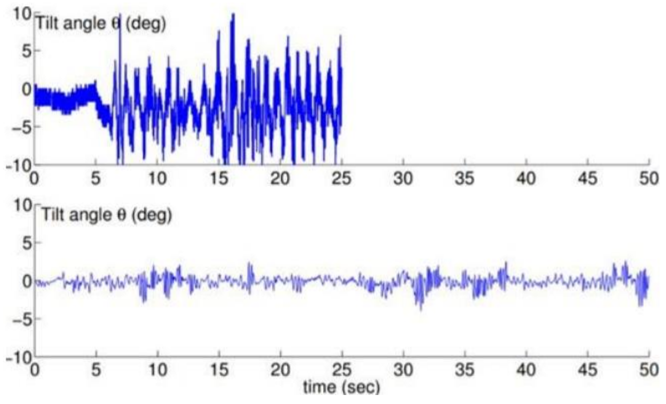

Fig. 9: Experimentally obtained history of tilt angle: PID controller (top), Neural Net (Bottom)

6. CONCLUSIONS

Neural Net approaches for self-balancing are not as commonly employed as PID controllers, nonetheless based on our complete analysis we can deduce that the Feedforward Neural Network was able to balance the robot much better than the PID Controller. The Kp, Ki, and Kd parameters of the PID Controller need to be adjusted significantly, as shown by the data. These values are unique to a system and vary from one to the next. As a result, the identical settings for one robot cannot be used for another. The parameters created in the neural net, on the other hand, are universal, hence this approach does not need any adjustments. Once the neural net code is loaded into the Arduino Uno Microcontroller, the robot will begin to balance itself. Finally, the Neural Net approach is a fascinating and trustworthy way to balance the robot since it causes the bot to vary less around the balancing point and so keeps it steady for a longer period of time. Meanwhile, when the PID controller is subjected to manual disturbances/imbalances, it performs better. When the bot is supplied with a tiny push and forced to unbalance, the PID controller is able to deliver the larger torque required and balances the bot whereas the Neural Net often fails to do so, allowing the robot to tumble. Finally, using low-cost components, we were able to develop a compact and cost-effective two-wheeled selfbalancing robot. A fine-tuned PID controller and a 3-layered feedforward neural network trained using the backpropagation method were used to handle the selfbalancing problem, and pertinent comparisons were made between them.

REFERENCES

[1] F. Grasser, A. D. Arrigo, and S. Colombi, “JOE: A mobile, inverted pendulum,” IEEE Trans. Ind. Electron., vol. 49, no. 1, pp. 107–114, Feb. 2002.

[2] http://www.geology.smu.edu/~dpa-www/robo/nbot/

[3] Nguyen Gia Minh Thao, Duong Hoai Nghia and Nguyen Huu Phuc, "A PID backstepping controller for twowheeled self-balancing robot," International Forum on Strategic Technology 2010, Ulsan, 2010, pp. 76-81, doi: 10.1109/IFOST.2010.5668001.

[4] W. An and Y. Li, "Simulation and control of a twowheeled self-balancing robot," 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO), Shenzhen, 2013, pp. 456-461, doi: 10.1109/ROBIO.2013.6739501.

[5] Q. Yong, L Yanlong, Z Xizhe, and L. Ji, "Balance control of two-wheeled self-balancing mobile robot based on TS fuzzy model," 2011 6th International Forum on Strategic Technology (IFOST), pp.406,409, Aug 22–24, 2011.

[6] J. Wu, S. Jia, "T-S adaptive neural network fuzzy control applied in two-wheeled self-balancing robot," 2011 6th International Forum on Strategic Technology (IFOST), pp.1023, 1026, Aug 22–24, 2011.

[7] L. Jiang, M. Deng, and A. Inoue, "Support vector machinebased two wheeled mobile robot motion control in noisy environment", J. of Systems and Control Engineering, vol. 222, no. 7, pp. 733–743, 2008.

[8] J. Cai and X. Ruan, "Bionic autonomous learning control of a two-wheeled self-balancing flexible robot," J. of Control Theory and Applications, vol. 9, no. 4, pp.521–528, 2011.

[9] Kim, Y., Kim, S.H. & Kwak, Y.K. Dynamic Analysis of a Nonholonomic Two-Wheeled Inverted Pendulum Robot. J Intell Robot Syst 44, 25–46 (2005). https://doi.org/10.1007/s10846-005-9022-4

[10] S. Cong and Y. Liang, "PID-like neural network nonlinear adaptive control for uncertain multivariable motion control system," IEEE Trans. Ind. Electron., vol. 56, no. 10, pp. 3872–3879, Oct. 2009