**Self-balancing robot using Neural Networks and simulation on Vrep software**

Yashad Samant Yash Chopra

Master’s of Electrical & Computer Engineering Master’s of Electrical & Computer Engineering

Colorado State University Colorado State University

Abstract —This paper references to the problems faced by a bipedal robot to balance itself without any human intervention. It gives us an insight on how to design an intelligent control system using fuzzy neural network and why it should be preferred over the traditional PID controller. It discusses the current trends of technologies used to control the robot, as to why we used the software and the hardware used for the project and the future scope of the project. It discusses the various types of configurations of neural networks we implemented, their response time and the memory used by the intensive neural network code. It also explains the implementation of the neural network code on relatively smaller processor and how we optimized the code for proper memory management within the processor.

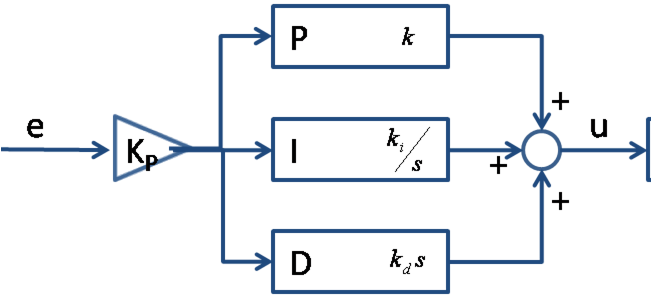
Keywords— Neural Networks, Fuzzy logic, Raspberry pi, self-balancing robot, Vrep simulation software

1. Introduction

While implementing a self-balancing object the first thing that comes to our mind is developing a PID controller to control the system irrespective of the calculations and the manipulations we have to do to design the system. PID stands for three different variables:

* Term **P** is proportional to the current value of the SP-PV error *e(t)*. For example, if the error is large and positive, the control output will be proportionately large and positive, taking into account the gain factor "K". Using proportional control alone in a process with compensation such as temperature control, will result in an error between the setpoint and the actual process value, because it requires an error to generate the proportional response. If there is no error, there is no corrective response.
* Term **I** accounts for past values of the SP-PV error and integrates them over time to produce the I term. For example, if there is a residual SP-PV error after the application of proportional control, the integral term seeks to eliminate the residual error by adding a control effect due to the historic cumulative value of the error. When the error is eliminated, the integral term will cease to grow. This will result in the proportional effect diminishing as the error decreases, but this is compensated for by the growing integral effect.
* Term **D** is a best estimate of the future trend of the SP-PV error, based on its current rate of change. It is sometimes called "anticipatory control" as it is effectively seeking to reduce the effect of the SP-PV error by exerting a control influence generated by the rate of error change. The more rapid the change, the greater the controlling or dampening effect.

Graphical representation of the model is as follows:}

}

But these different factors involve many calculations. Also, each have their different manipulations that we have to do. Also, when we are combining the all the three values, we have to be careful while implementing the filter. Thus, with all these manipulations in each phase of the PID controller adds errors in the output which accumulates in the final output which makes our response noisy. Also, D output adds a lot of noise in the output. Hence, we need a methods which is easier to implement and is intelligent enough to complement the response from the PID controller without the noise.

We achieve all this by using fuzzy logic controller which is a non-linear controller as the PID, very easy to implement and has a lot of documents associated to it. The only problem with the fuzzy logic is that it’s not as accurate as PID, hence we achieve that accuracy by implementing Neural Networks on top of fuzzy logic.

Also, we achieve this design of neural network on a small micro-controller.

1. Hardware

## Raspberry pi

* Our objectives included running storing a huge data from accelerometer and gyroscope and store it in memory of the processor.
* We were to design an intensive neural network algorithm to control the torque of the robot so that it can balance itself without any human intervention. One of our objective also involved managing the memory requirement of the network so that it can be implemented on a relatively small micro-controller.
* Raspberry pi 3 is the third generation of pi and a computer in itself. It has his own operating system Raspbian which makes it easy to interface the GPIO pins. It has Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAM, BCM43438 wireless LAN 40-pin extended GPIO which makes it a perfect platform for our robot.
* Thus, as it satisfies all the necessary requirements we chose to go forward with Raspberry Pi 3.

## MPU 6050

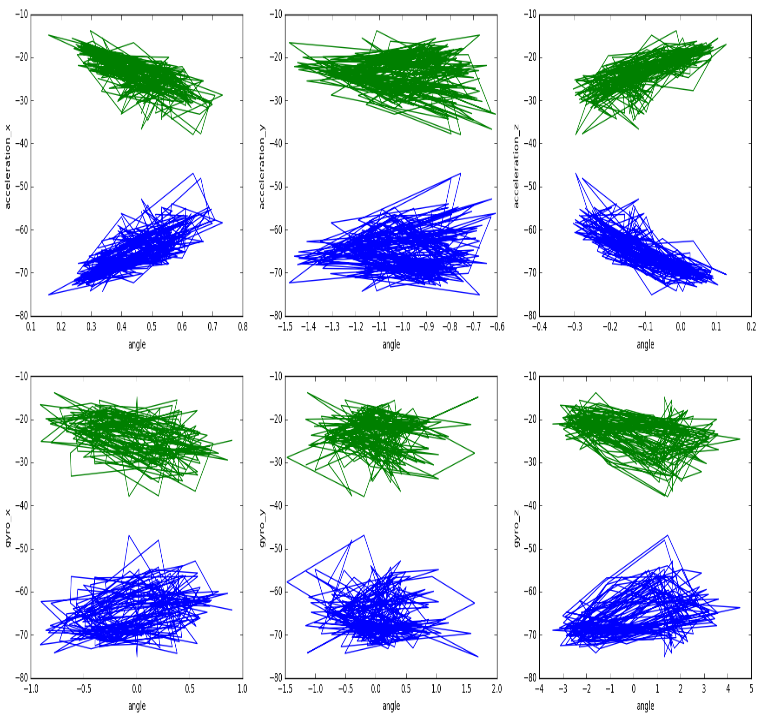
* The MPU-6050 devices combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die, together with an on-board Digital Motion Processor™ (DMP™), which processes complex 6-axis Motion-Fusion algorithms. We can obtain data (digital) from the sensor using SPI or I2C communication channel.
* MPU 6050 was an easy choice as it is known for its accurate and fast response as a 3-axis accelerometer and gyroscope and of course as both the sensors were on the same chip, interfacing becomes easier with the controller.
* Moreover, interfacing between pi 3 and mpu-6050 is documented online which made it easier for us to reference it and go forward with it.

## Interface

Interfacing of rasberry pi and mpu 6050 was done using the I2C channel on raspberry pi and SCI/ SPI on mpu 6050. Then found out the channel number for I2C on pi and obtained the values from accelerometer and gyroscope.

1. Software
   1. ***Data***

Though we were obtaining data from pi, for the neural network to train properly we needed ample of data and getting all that data from pi itself was not very convenient. That’s the reason we got our training data from Kaggle. It has the values of 3-axis accelerometer and 3-axis gyroscope taken from a iphone 5c. The relation between each input of data and output were as follows:

******

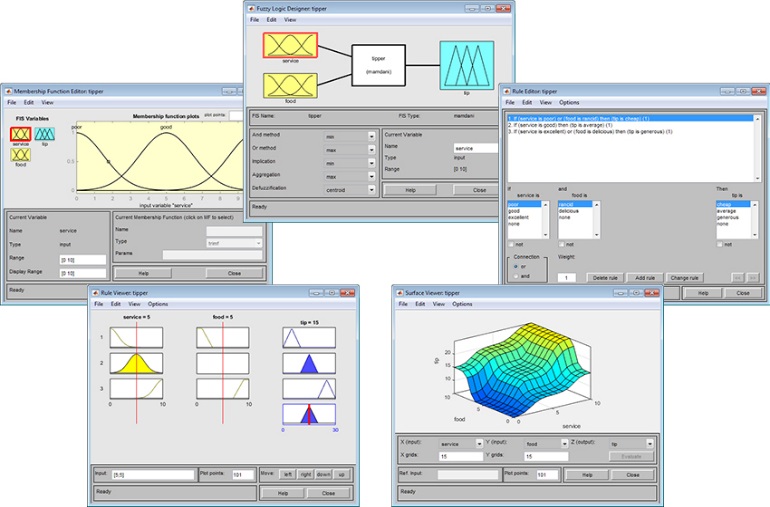
* 1. ***Fuzzy Logic***

Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1. By contrast, in Boolean logic, the truth values of variables may only be the integer values 0 or 1. Fuzzy logic has been employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. Furthermore, when linguistic variables are used, these degrees may be managed by specific (membership) functions.

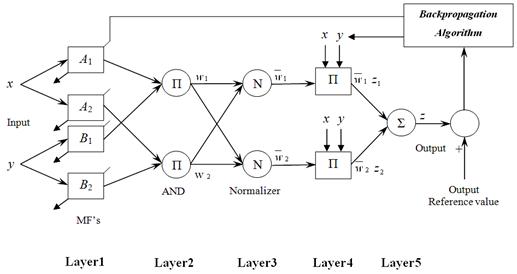
Since the fuzzy system output is a consensus of all of the inputs and all of the rules, fuzzy logic systems can be well behaved when input values are not available or are not trustworthy. Weightings can be optionally added to each rule in the rule base and weightings can be used to regulate the degree to which a rule affects the output values. These rule weightings can be based upon the priority, reliability or consistency of each rule. These rule weightings may be static or can be changed dynamically, even based upon the output from other rules.

* Fuzzify all input values into fuzzy membership functions.
* Execute all applicable rules in the rulebase to compute the fuzzy output functions.
* De-fuzzify the fuzzy output functions to get "crisp" output values.

Fuzzy logic in Matlab gives a good visual representation of it.

****

* 1. ***Fuzzy Neural Network***

****

We assume the fuzzy inference system under consideration has two inputs x and y and one output z. Suppose that the rule base contains two fuzzy if-then rules of Takagi and Sugeno’s type.

Rule 1: If x is A1 and y is B1, then fi = plx + q1y+ rl,

Rule 2: If x is A2 and y is B2, then f 2 = p2x + q2y+ 7-2.

The node functions in the same layer are of the same function family as described below:

* Layer 1: Every node i in this layer is a square node with a node function
* Layer 2: Every node in this layer is a circle node labeled Tz which multiplies the incoming signals and sends the product out.
* Layer 3: Every node in this layer is a circle node labeled N. The ith node calculates the ratio of the ith rule’s firing strength to the sum of all rules’ firing strengths:
* Layer 4: Every node i in this layer is a square node with a node function.
* Layer 5:The single node in this layer is a circle node labelled C that computes the overall output as the summation of all incoming signals.

Thus, here we have a general model of fuzzy neural network. We have used Mamdani method to design the membership function of the Fuzzy Neural Network system. It gives a non-linear approximation of all the input variables. We need not zero mean the input separately as the fuzzification process involves standardization which. Then, values obtained from different inputs are processed using the centroid method. But we have seen in the fuzzy logic section that there is no training involved and just calculation of final values based on the input. But in case of Fuzzy Neural Network we have target values, using backpropagation and/or least mean square error, we change the weights thus training the neural net as we can see in the above model.

Thus, it is a combination of fuzzy logic using which output is obtained and neural net where the net is trained using the target values.

## Complementary Filter

While implementing the filter to take the drift into consideration and getting accurate angles, we had two choices.

* Kalman Filter
* Complementary Filter

Thus, while looking for the best fit:

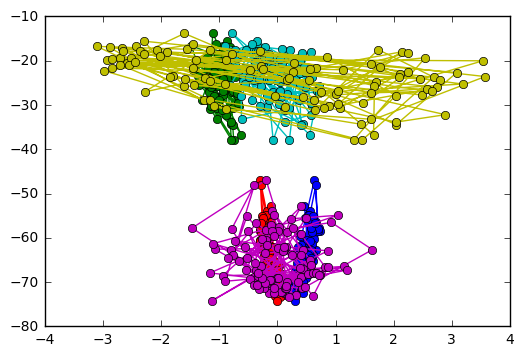
* We saw that Kalman filter was very accurate but it was very difficult to implement and moreover, the exceptional accuracy that it provides wasn’t necessary for our project thus, we went forward with complementary filter. It has appreciable accuracy, widely used (documentation available) and preferred over Kalman because of the easy implementation.
* Idea behind complementary filter isto take slow moving signals from accelerometer and fast moving signals from a gyroscope and combine them.Accelerometer gives a good indicator of orientation in static conditions.Gyroscope gives a good indicator of tilt in dynamic conditions.
* So the idea is to pass the accelerometer signals through a low-pass filter and the gyroscope signals through a high-pass filter and combine them to give the final rate.
* The key-point here is that the frequency response of the low-pass and high-pass filters add up to 1 at all frequencies.
* This means that at any given time the complete signal is subject to either low pass or high pass.

## Vrep Simulation software

V-REP's strength comes from several features:

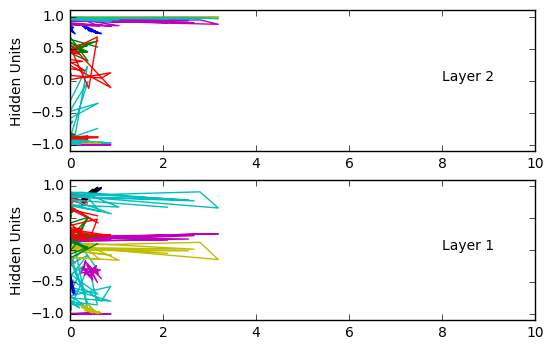
* V-REP provides a unified framework combining many powerful internal and external libraries that are often useful for robotics simulations. This includes dynamic simulation engines, forward/inverse kinematics tools, collision detection libraries, vision sensor simulations, path planning, GUI development tools, and built-in models of many common robots.
* V-REP is highly extensible. V-REP developers provide an API that allows one to write custom plugins that add new features. You can embed Lua scripts directly into a simulation scene that, for example, process simulated sensor data, run control algorithms, implement user interfaces, or even send data to a physical robot. They also provide a remote API that allows one to develop standalone applications in many programming languages that are able to pass data in and out of a running V-REP simulation.
* V-REP is cross-platform, mostly open-source, and provides a free educational license.

1. Results
   1. ***Fuzzy Neural Network***



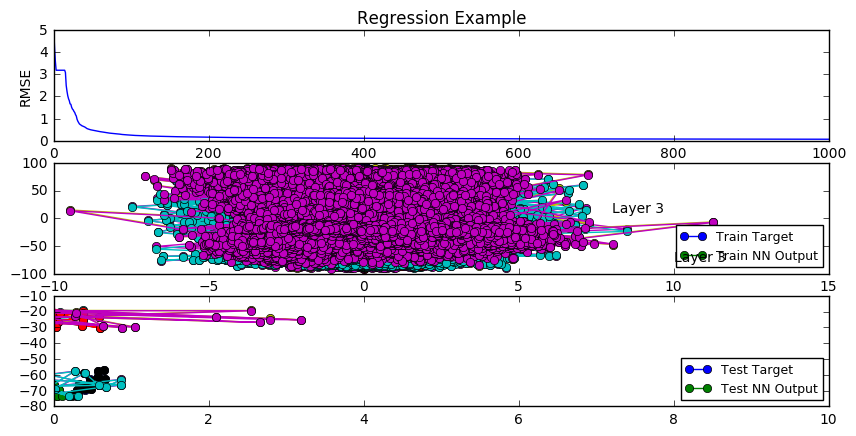
*Fig 1: Train output data*

In this graph, we have plotted angle v/s sensor response. We can observe the trend that training data observed. The six colors represent the data from 3-axis accelerometer and 3-axis gyroscope making it six inputs for the neural network. The output for the training data comes from the complementary filter. Using the data from the filter and feedback loop from the neural network, we have calculated the change in actual and desired angle and on the basis of the root-mean-square error value, we have achieved change in torque.



*Fig 2: TRAINED WEIGHTS*

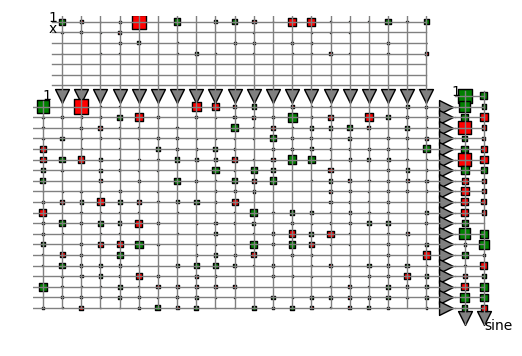
In the plot above, we observe the trained weights in layer one and layer two. We can see the weights are significantly big and equally scaled for all the inputs. This makes it easier for us to use the weights as it is, otherwise we had to scale the inputs to be relatively close to each other to avoid the probem of unequal scale of weights (where a input having higher arithmatic value will obtain higher weights thus increasing it’s effect on the output).



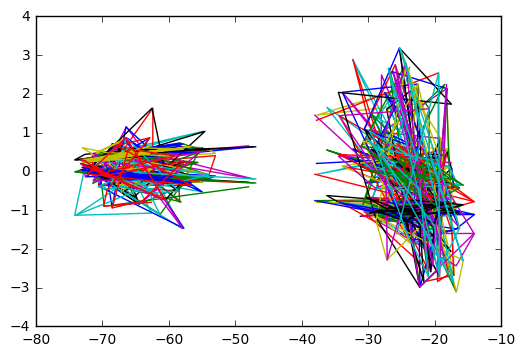
*FIG 3: RMSE ERROR, TRAINED NN OUTPUT, TESTNN OUTPUT*

We have plotted three plots above which gives the summary and the success of the code.

* In the first plot, we can observe that the rmse error is exponentially going to 0. This means that we have achieved a good response from our supervised neural network where the actual output have converged well towards the target output. We ran this code for 1000 iterations but we can see that we obtained respectable results in 200 iterations itself.
* Second plot is a bit strenuous to understand as it consists of approximately 90000 instances from the data set. Though it has many instances, it’s the weight trained for the third layer of the network.
* We can observe the weights from the test data here which are similar to weights from layer 1 and 2.

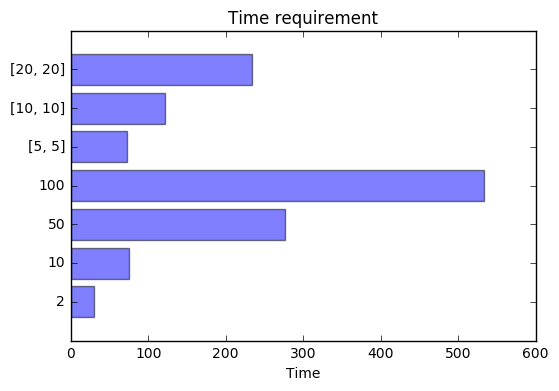
*FIG 4: TRAINED WEIGHTS*

This plot above gives the structure of the neural net where red are weights which negatively impacts the neural network while green impacts positively.



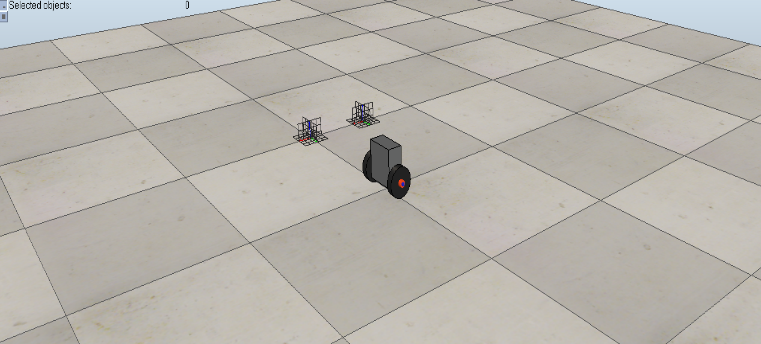
*Fig 5: TEST OUTPUT*

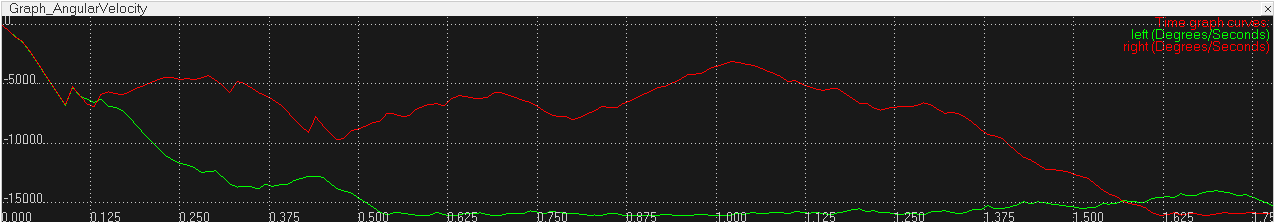
In this plot above, we see that this is the test output from layer 5. We can observe that we have achieved a good response when it comes to the trained output in fig1 and test output here. Thus, from this graph we become sure that the neural networks is working fine and is replicating the results we achieved from the trained network.



*Fig 6: TIME REQUIRED BY DIFFERENT CONFIGURATIONS OF NEURAL NETWORK*

* In this plot, we have designed neural networks when it comes to hidden units in each layer and the number of hidden layers. Thus, the x coordinate indicates the number of layers and y suggests the number of units in a hidden layer. The single digit indicates the number of neurons in each of the five layers
* We can observe the time requirement by each of the layers. We can see that 100 hidden units in each layer took the maximum amount of time. [20, 20] had the best performance of the layers and we can see that we code was completed in around 200 seconds. The data plotted above is for [20, 20] as it had the lowest rmse error.
  1. ***Vrep Simulation software***
* In the simulation software, we designed the robot and created a .ttt file which is native to the software. We created the robot as we can see in the picture below.
* We also implemented a python script to control the angular velocity and torque of the motors so that the robot can balance itself.
* As we can see from the second picture, we can see the change in angular velocity at every instant.
* Only problem we suffered was when there was uneven surface, we couldn’t control the robot. As neural network was not designed to get such quick change in the angles. Also, there was a jerk factor with involving of rocks and mountains in the simulation which neural network wasn’t sure of which is pretty much expectable.
* After researching about the irregular environment, we found out that we can implement reinforcement learning with the neural network code to take into consideration the irregular environment. More details about it are mentioned in the future scope.





* 1. ***Memory Management***

While running the neural network on a desktop was not a problem. It had a huge memory at its disposal and strong processor to complement it. But we were running our code pi 3 which has just 1Gb ram. Initially, when we tried the code on pi without any optimizations, it gave us memory error. But we achieved our objective using two factors:

* Firstly, we created a swap file. A virtual memory where we convert the external memory to RAM. That helped us to convert some memory, in our case (500Mb) to RAM.
* Secondly, we started working on code optimizations. As we know from the structure of neural net, we had membership functions for every input. We had six inputs and two outputs, around 90000 instances of data. Thus, it used to create a six 2D structures with 90000x2 instances for every layer. Thus, depending on the number of layers, it used to calculate data every time and store it into a different matrix. To avoid this problem, we initialized a single matrix which could handle all these computations and we averaged it out every three iterations, thus we don’t save the all the matrices for every iteration thus reducing memory usage.

1. Conclusions

* Developed a system which takes time to train but while testing gives output at least 0.3 seconds earlier than PID controller.
* System is easy to design and there are no manipulations when compared to PID.
* Designed a 5-layer neural network code based on fuzzy inference system with variable hidden layers and units in each layer.
* Implemented a self-balancing robot using neural networks and ran the code on a pi which has only 1GB RAM. It took some modifications in the code but it worked.
* Also, a 500MB swap file was created which creates a virtual memory partition on the external memory card and considers it as RAM.
* Obtained simulation results on Vrep software by controlling the motors of the robot using the angles from the neural networks.

1. Future scope

* Design of this code only controls the balancing of the robot by changing the torque of the motors using the neural network for faster response time than PID controllers.
* If I were to obtain the result by using Fuzzy Neural Network, I would have gained more flexibility and obtained much more accurate results. Fuzzy logic handles the non-linearity constraint better than tradition neural networks. If I would got more time, I would have worked a way through the memory error and obtained the results.
* As mentioned earlier, the control of the robot is solely based on the angle between the axis of the bot and normal. But if we use reinforcement learning on top of neural networks, we can obtain convergence quicker and it will also take into consideration the abnormalities in the environment.

Acknowledgment

We are glad to have Dr. Sudeep Pasricha as our mentor in this project. We thank him for letting us going forward with the project, helping us where we missed the minute detail and also helping us in obtaining the hardware.

We also thank Dr. Chuck Anderson for his insights on the machine learning code.

References

1. S. M. Metev and V. P. Veiko, *Laser Assisted Microtechnology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.
2. J. Breckling, Ed., *The Analysis of Directional Time Series: Applications to Wind Speed and Direction*, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.
3. S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, “A novel ultrathin elevated channel low-temperature poly-Si TFT,” *IEEE Electron Device Lett.*, vol. 20, pp. 569–571, Nov. 1999.
4. M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, “High resolution fiber distributed measurements with coherent OFDR,” in *Proc. ECOC’00*, 2000, paper 11.3.4, p. 109.
5. R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, “High-speed digital-to-RF converter,” U.S. Patent 5 668 842, Sept. 16, 1997.
6. (2002) The IEEE website. [Online]. Available: http://www.ieee.org/
7. M. Shell. (2002) IEEEtran homepage on CTAN. [Online]. Available: http://www.ctan.org/tex-archive/macros/latex/contrib/supported/IEEEtran/
8. *FLEXChip Signal Processor (MC68175/D)*, Motorola, 1996.
9. “PDCA12-70 data sheet,” Opto Speed SA, Mezzovico, Switzerland.
10. A. Karnik, “Performance of TCP congestion control with rate feedback: TCP/ABR and rate adaptive TCP/IP,” M. Eng. thesis, Indian Institute of Science, Bangalore, India, Jan. 1999.
11. J. Padhye, V. Firoiu, and D. Towsley, “A stochastic model of TCP Reno congestion avoidance and control,” Univ. of Massachusetts, Amherst, MA, CMPSCI Tech. Rep. 99-02, 1999.
12. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11, 1997.