Implementing Central Pattern Generators for Bipedal Walkers using Cellular Neural Networks

by

Bharathwaj Muthuswamy

B.S. (University of California, Berkeley) 2002

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of

Master of Science, Plan II

in

Engineering—Electrical Engineering

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee:

Professor Leon O. Chua
Professor Pravin P. Varaiya
Very Special Thanks to Professor Shankar S. Sastry

Spring 2005
The project report of Bharathwaj Muthuswamy is approved:


University of California, Berkeley

Spring 2005
ACKNOWLEDGMENTS

This project is the result of many people's efforts and it would be impossible to list them all here. First and foremost, many thanks to Dr. Leon O Chua and Dr. Shankar S. Sastry for giving me the opportunity to work on this project. Dr. Pravin P. Varaiya proofread my report on a very short notice and provided valuable insights. Dr. Istvan Szatmari helped me understand the basic concepts behind Cellular Neural Networks. Paolo Arena et. al. wrote the excellent Bioinspired locomotion book [10] that started this project. A million thanks to the people from my research group - Spencer Ahrens, Abhishek Mishra, Jonathan Tay, Tyler Lebrun, Eva Markiewicz, Melissa Li and Maryam Sanglaji for the SolidWorks model, the Paranoia simulation program and the countless hours spent assembling the robot. This project would have never taken off without them.

Thanks to the Electrical Engineering department at the University of California, Berkeley for providing our group with the facilities to work on this project. Ferenc, Tho, Pete, Khosrov, Ming, John and Changrui of the Electronic Support Group at the University of California, Berkeley provided timely debugging tips. Ben, Joe and others at the Machine Shop in Cory came up with the correct spacers at the right time! Cindy Palwick's timely help for ordering all the parts I requested was indispensable. John Secord took the time to debug my printed circuit board designs and corrected two fundamental mistakes. Sridhar Balasubramanian, Ian Roberston, Nathan Sentz and Vincent Howard of the Cal Robotics club corrected some of my insane assumptions and circuit designs! Ali El-Annan proofread my final thesis. Many thanks to Kevin Bowers for providing an excellent \LaTeX{} template on his homepage [8].

Sandra Sinisi, Andrea Payne, Sheila Ross and Ashwin Ganesan provided me with a lot of emotional support and patiently dealt with my frustrations and emotions during this project. And finally, a very special “thank you” to my parents and family for providing me support throughout my academic career. Thanks to them, I did not end up like a robot trying to take an insane exam to get into one of the Indian Institutes of Technology. Rather, I ended up working on the coolest project in THE best university on the planet.

Go Bears!
ABSTRACT

Implementing Central Pattern Generators for Bipedal Walkers using Cellular Neural Networks

by

Bharathwaj Muthuswamy

Master of Science, Plan II in Engineering—Electrical Engineering

University of California, Berkeley

Bipedal (two legged) robotics is a very interesting field in its own right. Besides the obvious joy of studying a complicated nonlinear (albeit very elegant) phenomenon, the main advantage of building bipedal walkers is to understand how humans walk. Nevertheless, designing and implementing a stable bipedal walker is a challenge because of the many degrees of freedom in the mechanisms, the intermittent nature of the contact conditions with the environment, and underactuation [12].

Our approach in solving this problem is biomimetic, that is, we want to mimic biology - mechanical construction and neural control are the main ideas - to build efficient bipedal walkers. To expand upon this, here is a summary of our goals:

1. Develop a simulation environment incorporating SolidWorks and SPICE for studying the dynamics of bipedal walking without sacrificing accuracy. SolidWorks [5] is an industry standard tool for solid modelling. Thus, SolidWorks provides us with a 3-d computer model of our robot complete with physical properties like centre of mass and moments of inertia. SPICE (Simulation Program With Integrated Circuit Emphasis) [6] is the industry standard simulation tool for circuits. By interfacing the SolidWorks model of our robot to the control circuitry model implemented in SPICE, we hope to study the motion of the bipedal walker before we even construct the real robot!

2. Design a nonlinear analog circuit (a cellular neural network or CNN) that models a Central Pattern Generator (CPG) capable of producing the patterns required for bipedal walking.

3. Design a biomimetic mechanical body.

4. Implement a nonlinear control law for the CNN that guarantees dynamic stability.

The goal of my Masters’ project is to complete (1) and (2) above; (3) and (4) are my Ph.D. topics at the University of California, Berkeley. For the latest information on all the goals above, please
visit our project homepage: http://robotics.eecs.berkeley.edu/~mbharat/raptor My Masters’ project report is organized as follows: In chapter 1, I talk about the theoretical ideas behind the CNN and the CPG. Chapter 2 deals with a very specific CPG model, the Chemical Synapse CPG that helps in generating patterns for walking motion. Chapter 3 shows an implementation of this CNN using op-amps and also gives some SPICE simulation results. Chapter 4 talks about the simulation environment we developed. Chapter 5 talks about our implementation. I conclude with a look at what we learned from this project and what we hope to accomplish for the future.
Contents

List of Figures v

List of Tables vi

1 Introduction to the CNN and the CPG 1
  1.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.2 A Brief Overview of a Cellular Neural Network (CNN) . . . . . . . . . . . . . . . . . . 2
  1.3 The Central Pattern Generator (CPG) . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 The Role of the Chemical Synapse CNN CPG in Bipedal Walking 6
  2.1 The Chemical Synapse CNN CPG Neuron . . . . . . . . . . . . . . . . . . . . . . . . . 6
  2.2 Oscillatory Behaviour of the Chemical Synapse CNN CPG . . . . . . . . . . . . . . . . 7
  2.3 MATLAB Simulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3 Circuit model of the Chemical Synapse CNN CPG 10
  3.1 The Chemical Synapse CNN CPG . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  3.2 The Integrator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  3.3 The Threshold Function Generator . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  3.4 Completing the Chemical Synapse CNN CPG Circuit . . . . . . . . . . . . . . . . . 13
  3.5 PSPICE Output Waveforms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

4 Simulation Results 16
  4.1 The SolidWorks Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
  4.2 Paranoia . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

5 Implementation Notes 21
  5.1 Lynxmotion Biped Scout Kit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
  5.2 Power Supply . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
5.3 CNN Board ................................................................. 22
5.4 Interface between CNN and Raptor Servos ....................... 22
5.5 Total Cost of the project ................................................. 23

6 Conclusion and Future Work ........................................... 24

Bibliography ..................................................................... 24
## List of Figures

1.1 A Block diagram of our project .................................................. 1  
1.2 A 3x3 CNN ................................................................. 3  
1.3 The Piecewise linear output function we will be using for a neuron in our CNN ... 4  

2.1 Genetic Algorithm Fitting ......................................................... 8  
2.2 CNN CPG output using parameters obtained from the genetic algorithm ........ 9  

3.1 CNN CPG Circuit Screen Shot .................................................. 11  
3.2 The Opamp Integrator ........................................................ 12  
3.3 The Piecewise Linear function ................................................ 13  
3.4 PSPICE Simulation Output ..................................................... 14  

4.1 SolidWorks Raptor Model ......................................................... 17  
4.2 Detailed modelling of the gears in the servos .................................. 18  
4.3 A screen shot of Paranoia, build 1 .......................................... 19  

5.1 The Biped Scout 1 kit from Lynxmotion ....................................... 21  
5.2 The Power Board lights up! ..................................................... 22  
5.3 Probing the CNN board ....................................................... 22  
5.4 Output Waveform from the CNN Board ...................................... 22  
5.5 CNN board is too big .......................................................... 23  

6.1 The Raptor is almost finished .................................................. 24
List of Tables

3.1 Table showing the relationship between circuit and CNN parameters . . . . . . . . 12

5.1 Table showing the total product cost for our project . . . . . . . . . . . . . . . . . . . . 23
Chapter 1

Introduction to the CNN and the CPG

1.1 Introduction

Walking animals employ several distinct periodic patterns of leg movements called gaits [10]. Stereotyped movements (locomotion patterns) are also observed in swimming, flying and crawling animals [10]. The hypothesis [10] is that a Central Pattern Generator or CPG (a network of neurons), generates these patterns. A key feature of these networks is that the pattern of neural activity is mapped onto the pattern of locomotor activity [10]. This concept is useful in robotics since we can map the desired behaviour (i.e. a locomotion pattern) to a set of feedforward signals for driving the actuators of our robot [10]. A block diagram of our project is shown in the figure 1.1 below.

In figure 1.1, the dotted blocks have not been implemented. For my Master’s project, all we are doing is generating the patterns required for locomotion. We then map these patterns to the actuator using a transducer. In our case, we are using the JSTAMP [1] microcontroller from Parallax. To

Figure 1.1: A Block diagram of our project
ensure the robot does not fall, we prop the robot using a mechanical tether. For more information, refer to Chapter 5. Hence, this is a proof-of-concept project.

A subtle point to note in figure 1.1 is the feedback. Notice that unlike classic feedback, we do not have an error function going into the controller. Rather, the controller would tune the parameters of the CPG using sensory data.

In the next section, we talk about the Cellular Neural Network which we use a nonlinear oscillator to realize the CPG.

1.2 A Brief Overview of a Cellular Neural Network (CNN)

The Cellular Neural Network or CNN is simply an array of identical, locally interconnected, nonlinear dynamic circuits [10]. Mathematically, this definition translates into [9]:

\[
\begin{align*}
\frac{dx_{ij}}{dt} &= -x_{ij} + \sum_{kl \in N_r(ij)} A_{ij;kl}(y_{kl}(t), y_{ij}(t)) + \sum_{kl \in N_r(ij)} B_{ij;kl}(u_{kl}(t), u_{ij}(t)) + \alpha_{ij} \\
y_{ij} &= f(x_{ij})
\end{align*}
\]

In equation (1.1) above:

- \(ij\) refers to \(ij^{th}\) neuron on a 2-D grid.
- \(kl \in N_r(ij)\) is the \(kl^{th}\) neuron within a neighbourhood of radius \(r\) of the \(ij^{th}\) neuron.
- \(x_{ij}\) is the state of the \(ij^{th}\) neuron.
- \(A\) is the feedback vector or feedback cloning template, a function of both self-feedback \((y_{ij}(t))\) and cross-feedback with neighbouring neurons \((y_{kl}(t))\).
- \(B\) is the feedforward vector or feedforward cloning template. Inputs can come into the \(ij^{th}\) cell \((u_{ij}(t))\) or from neighbouring cells as well \((u_{kl}(t))\).
- \(\alpha_{ij}\) is a bias term, usually a constant for the entire neural network.

In equation (1.2) above:

- \(y_{ij}\) is the output of the \(ij^{th}\) cell.
- \(f\) is the nonlinear output function, also called as the threshold function.
You can see from equation (1.1) that a CNN is dynamical because its state evolves with time. The neurons in a CNN are interconnected with other neurons in a specified neighbourhood (hence *locally* interconnected). The neurons in a CNN are identical because each neuron’s state obeys the same dynamical equation (1.1). **However, notice that \( \vec{A} \) and \( \vec{B} \) may be different for each neuron.** If they are equal across all neurons in a CNN, then the cloning templates are said to be space invariant. If \( \vec{B} = 0 \) for each neuron, the CNN is said to be autonomous. Finally, the system is nonlinear because the threshold function in equation (1.2) is piecewise linear. A CNN can be visualized pictorially as shown in figure 1.2.

Therefore, if you wanted to write the dynamical equation for neuron (2,2) (the neuron in the middle), it would be (assuming \( r=1 \)):

\[
\frac{dx_{22}}{dt} = -x_{22} + \sum_{1 \leq k \leq 2, 1 \leq l \leq 2} \vec{A}_{22;kl}(y_{kl}(t), y_{22}(t)) + \sum_{1 \leq k \leq 2, 1 \leq l \leq 2} \vec{B}_{22;kl}(u_{kl}(t), u_{22}(t)) + \alpha_{22} \tag{1.3}
\]

Notice that this CNN is **not** space invariant, the cloning templates for the neurons on the periphery are not square matrices. Hence, a CNN is capable of exhibiting very complicated nonlinear dynamics. We are going to restrict our CNN to be a single neuron autonomous (\( \vec{B} = 0 \)) CNN [10]:

\[
\frac{dx_1}{dt} = -x_1 + \alpha y_1 + \beta y_2 + i_1 \quad \tag{1.4}
\]

\[
\frac{dx_2}{dt} = -x_2 + \gamma y_2 + \delta y_1 + i_2 \quad \tag{1.5}
\]

and the output function is going to be a piecewise linear approximation to the sigmoid threshold function [10]:

\[
\frac{1}{2}(|x_{ij} + 1| - |x_{ij} - 1|) \quad \tag{1.6}
\]
If $x \geq 1$, then in equation (1.6) $|x_{ij} + 1| = (x_{ij} + 1)$ and $|x_{ij} - 1| = (x_{ij} - 1)$. Hence, we get:

$$\frac{1}{2}(|x_{ij} + 1| - |x_{ij} - 1|) = \frac{1}{2}(x_{ij} + 1 - x_{ij} + 1) = 1$$

If $0 \leq x \leq 1$, then $|x_{ij} + 1| = (x_{ij} + 1)$ and $|x_{ij} - 1| = -(x_{ij} - 1)$. Hence, we get:

$$\frac{1}{2}(|x_{ij} + 1| - |x_{ij} - 1|) = \frac{1}{2}(x_{ij} + 1 + x_{ij} - 1) = x_{ij}$$

A similar argument applies if $x \leq -1$ and if $-1 \leq x \leq 0$. The output function is plotted in figure 1.3.

In order to understand how to use this CNN, we need to understand what a CPG is. This is the topic of the next section.

1.3 The Central Pattern Generator (CPG)

As stated in the Introduction section, a CPG is a neural circuit that can produce a rhythmic motor pattern. It is mainly located in the spinal cord of vertebrates or in relevant ganglia in invertebrates [10]. Signals from the CPG directly control the effector organs (legs) while signals from higher control neurons are necessary only to initiate the locomotion, but not to generate the correct pattern of movements [10]. This fact has been demonstrated in a number of experiments, animals deprived of commands from high-level neural centres are still able to locomote, provided there is external stimuli to the CPG [10]. In our robot, this external stimulus is the power supply to the CNN.
circuitry realizing the CPG. Likewise, the presence of feedback from the environment is not necessary to generate a rhythmic pattern [10]. Obviously, we need feedback to guarantee dynamic stability of our bipedal walker, since our system is dynamically unstable [12]. Again, we will revisit the stability of our walker in the second phase of this project.

Unfortunately, unlike the CNN, many mathematical models exist for a Central Pattern Generator. A classic model used by Golubitsky et. al. [11] is the space-clamped Hodgkin-Huxley equations. However, this a system of four coupled nonlinear ordinary differential equations and are not feasible to implementation. But, based on the discussion in the preceding paragraph, the CPG we design needs to ensure a rhythmic nonlinear pattern generation for gait generation. Once such CPG is implemented as a CNN by Arena et. al [10], we study this so called Chemical Synapse CNN CPG 1 in the next chapter.

\footnotesize{\textsuperscript{1}Which comes first, CNN or CPG? Well, it makes sense to say “a CNN that is modelling a CPG” since CNNs are a more general concept than CPGs.}
Chapter 2

The Role of the Chemical Synapse CNN CPG in Bipedal Walking

2.1 The Chemical Synapse CNN CPG Neuron

Let us revisit the second order CNN CPG from the previous chapter, however we are going to impose the condition that $\beta = -\beta$, $\gamma = \alpha$ and $\delta = \beta$. This gives the definition of the CNN CPG Neuron considered by Arena et. al. [10]:

**Definition 1 (CNN CPG Neuron)**

\[
\begin{align*}
\frac{dx_1}{dt} &= -x_1 + \alpha y_1 - \beta y_2 + i_1 \\
\frac{dx_2}{dt} &= -x_2 + \beta y_1 + \alpha y_2 + i_2
\end{align*}
\]

How we determined the values of $\alpha$, $\beta$, $i_1$ and $i_2$ will be covered in section 3. However, a fundamental element in the CPG design is the way in which CNN neurons are interconnected [10]. Considering a typical human walk, we see that the two legs are always out of phase with each other. Hence, if we have two neurons that are each described by equations above, then the problem boils down to coupling the two neurons. The intuitive way is to use the output of one neuron to bias the other neuron. This leads to the so-called flexor-extensor model proposed in [10]:

**Definition 2 (Chemical Synapse CNN CPG Neuron)**

\[
\begin{align*}
\frac{dx_{1a}}{dt} &= -x_{1a} + \alpha y_{1a} - \beta y_{2a} + i_1 + \epsilon y_{1b} \\
\frac{dx_{2a}}{dt} &= -x_{2a} + \beta y_{1a} + \alpha y_{2a} + i_2 \\
\frac{dx_{1b}}{dt} &= -x_{1b} + \alpha y_{1b} - \beta y_{2b} + i_1 + \epsilon y_{1a}
\end{align*}
\]

\[\text{1 I have slightly modified the definition in [10]: } 1+\mu \equiv \alpha \text{ and } s \equiv \beta\]}
\[ \frac{dx_{2b}}{dt} = -x_{2b} + \beta y_{1b} + \alpha y_{2b} + i_2 \]  

(2.6)

The synapse is called inhibitory if \( \epsilon < 0 \) and excitory if \( \epsilon > 0 \) [10]. Now, let us analyze the conditions that lead to the oscillatory behaviour of the neurons above.

### 2.2 Oscillatory Behaviour of the Chemical Synapse CNN CPG

We just need to analyze one neuron, so let us rewrite Definition 1 using matrix notation:

\[ \vec{x} = -\vec{x} + \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix} \vec{y} + \vec{i} \]  

(2.7)

Here:

- \( \vec{x} \) is the vector of states = \([x_1 \ x_2]\)
- \( \vec{y} \) is the output vector = \([y_1 \ y_2]\)
- \( \vec{i} \) is the bias vector = \([i_1 \ i_2]\)

Analyzing the dynamics of the CNN CPG neuron can be simplified if we divide the problem into different cases depending on the output function. We have three possible cases for the output function: \( y = -1 \), \( y = x \) and \( y = +1 \). We have two output functions: \( y_1 \) and \( y_2 \). Therefore, we have a total of nine possibilities. An excellent detailed analysis is done in [10]. Since we are looking for oscillatory dynamics, we need to figure out the equilibrium points first. We can apply the Poincare-Bendixson theorem ([13] and [15] have some great examples) to determine if limit cycles exist. Again, I don’t want to repeat the excellent analysis in [10].

### 2.3 MATLAB Simulation

In the previous section, we saw how we could guarantee oscillatory dynamics for the Chemical Synapse CNN CPG. This till begs the question, how do we determine the parameters required for oscillation? Jonathan Tay from our research group came up with a solution: he coded a Genetic Algorithm in MATLAB that helps us determine values for \( \alpha, \beta, i_1, i_2 \) and \( \epsilon \) based on a waveform data. His code and relevant documentation will be available on our project homepage [4] in early June 2005. We also hope to get a publication out of it! The procedure for using his code is very simple:
1. Place a single period of the waveform in a comma separated file called rawdata.csv (first column is time, second column is data). The genetic algorithm will attempt to guess the $\alpha$, $\beta$, $i_1$, $i_2$ and $\epsilon$ values for this waveform. Note that the value of $\epsilon$ will ensure the phase difference between the state outputs of the two neurons will be 180 degrees.

2. Set limits on your parameters

3. Run the genetic algorithm to get the parameter values.

We ran Jonathan’s genetic algorithm with a test waveform we obtained from human gait data\footnote{Unfortunately, Jonathan returned the book to the library before he noted down citation details!} to see if we can get parameters for our CNN CPG. Figure 2.1 show the results.

The original waveform is shown in blue and the fitted data is shown in red. We ran the genetic algorithm for 1 generation only, generally more generations the better the refinement, but longer the time! Again, more details will be available on our project website in early June 2005.

Figure 2.2 shows the outputs of the CNN CPG to the genetic algorithm parameters. Istvan wrote the MATLAB code for simulating the CNN CPG, this code is already available on our website. Notice the limit cycles in the phase portraits.

In conclusion, we have proved the Chemical Synapse CNN CPG is very robust. The next chapter deals with actually implementing this CNN using an electric circuit.
Figure 2.2: CNN CPG output using parameters obtained from the genetic algorithm.
Chapter 3

Circuit model of the Chemical Synapse CNN CPG

This chapter heavily relies on concepts from circuit theory, an excellent reference is [14] (chapters 5 and 7). Although the circuitry is similar to Arena’s implementation in [10], it is slightly different. We use standard power supply voltages (+12 V, -12 V) and resistor values in kilo-ohms rather than mega-ohms.

We used PSPICE from Cadence to simulate the circuit, you can download the relevant files from our project website [4]. PSPICE is a graphical front-end to SPICE. However, notice the student version of PSPICE cannot be used to simulate the circuit below. The circuit exceeds the nodal limit of the student version, you need the full version of PSPICE to simulate the circuit. However, if you have SPICE then you can just take the netlist from our source online and use that to run the simulation.

Another point to note: the numerical simulator in PSPICE is not as robust as MATLAB. So, you may (will?) need to poke around in the Simulation Settings in PSPICE for this circuit to work. The details are beyond the scope of this report, shoot us an email if you have sustained difficulty. Refer to our project homepage: [4] for our contact information.

3.1 The Chemical Synapse CNN CPG

Figure 3.1 shows a screen shot of one neuron. We need eight op-amps, 4 op-amps to implement one differential equation. The functionality of each op-amp circuit is described in the sections below.
This example controls the hip tilt motor of the robot. The relationship between COCNN parameters and the circuit is:

\[ \alpha_1 = \frac{\lambda_1}{\lambda_1 + \beta_1} \delta_1 \]

\[ \alpha_2 = \frac{\lambda_2}{\lambda_2 + \beta_2} \delta_2 \]

In the expressions above, \( \alpha_1 \) and \( \alpha_2 \) are \( 10k \) pot, \( \beta_1 \) and \( \beta_2 \) are \( 1k \) pot, \( \lambda_1 \) and \( \lambda_2 \) are \( 550 \) \( \Omega \), and \( \delta_1 \) and \( \delta_2 \) are \( 1.5 \) \( \Omega \).

Figure 3.1: CNN CPG Circuit Screen Shot
Table 3.1: Table showing the relationship between circuit and CNN parameters

<table>
<thead>
<tr>
<th>Circuit Parameter</th>
<th>CNN Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1+\alpha$</td>
<td>$\frac{R_{t}}{\beta C_{11}}$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\frac{R_{t}}{\alpha C_{11}}$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>$\frac{R_{t}}{\beta C_{11}}$</td>
</tr>
<tr>
<td>$i_{1}$</td>
<td>$\frac{1}{R_{t}C_{11}}$</td>
</tr>
</tbody>
</table>

3.2 The Integrator

The core of the CNN CPG circuit is the integrator, shown in figure 3.2.

Assuming the op-amp is in the linear region \(^1\), KCL at the inverting input gives (let $i_{1} = R_{t}$):

$$C_{11} \frac{dx_{1}}{dt} + \frac{x_{1}}{R_{11}} + \frac{-y_{1}}{\alpha C_{11}} + \frac{y_{2}}{\beta C_{11}} + \frac{V_{SS}}{R_{t}} + \frac{y_{1} \text{ RightHip}}{\text{synapse}_{R}} \frac{1}{R_{11} C_{11}} = 0 \quad (3.1)$$

Simplifying yields:

$$\frac{dx_{1}}{dt} = \frac{1}{R_{11} C_{11}} (-x_{1} - \frac{-y_{1} R_{11}}{\alpha C_{11}} - \frac{y_{2} R_{11}}{\beta C_{11}} - \frac{V_{SS} R_{11}}{R_{t} C_{11}} - \frac{y_{1} \text{ RightHip}}{\text{synapse}_{R} C_{11}}) \quad (3.2)$$

Here, $x_{1}$ is the output voltage of the op-amp. You can see the equation above models the first order CNN state equation if $R_{11} C_{11} = 1$. However, since resistors are usually in kilo-ohms, this means we would need a millifarad capacitor. These caps are huge. But, experimentally, changing $C_{1}$ affected the timescale of the CNN waveform. Therefore to get periods in the order of seconds we used $C_{1} = 22 \mu F$. Table 3.1 shows the relationship between the circuit parameters and the CNN parameters.

Now, we discuss how to generate the nonlinear threshold function.

---

\(^1\)This is a valid assumption since the voltage inputs to the opamp ($y_{1}$ and $y_{2}$) never go beyond $+1$ V or $-1$ V because of the threshold function generator. Hence, the integrators output never reaches $+12$ V or $-12$ V
Looking at figure 3.1, we see that op-amp U2x1 is the core of the threshold function generator. This is a non-inverting amplifier, the output is related to the input $x_1$ by:

$$\text{out} = (1 + \frac{11}{1})x_1$$

(3.3)

The transfer characteristic of this circuit is shown in figure 3.3. Notice how we exploited the railing of the op-amp to “flatten out” the curve for input voltages greater than 1 V and less than -1 V. However, the output of the circuit when saturated should be 1 V. This is precisely the function of the voltage dividers formed by $R_{71}$ and $R_{81}$. Hence, $y_{1_{left}}$ is precisely the threshold function of the CNN CPG.

### 3.4 Completing the Chemical Synapse CNN CPG Circuit

- The inverting amplifier U4x1 generates $-y_1$ for cross-feedback and self-feedback.
- Duplicating this circuit gives rise to the 2nd state in the CNN CPG Neuron.
- The synapse connection in the integrator goes to an identical CNN on another page. The resistor impedes the flow of current into the op-amp and thus acts as a very simple inhibitor.

### 3.5 PSPICE Output Waveforms

Figure 3.4 shows the output voltages across the capacitors for the coupled neurons.

In conclusion, PSPICE has shown the circuit model of our CNN CPG is capable of producing sustained oscillations. It would be great if we could see how our robot would move before we designed
Figure 3.4: PSPICE Simulation Output
the actual circuit. This is where the simulation environment we developed comes in, the subject of the next chapter.
Chapter 4

Simulation Results

This chapter is very short, it just describes our simulation environment. You have already seen the PSPICE model in the previous chapter. Let us look at the SolidWorks modelling environment.

4.1 The SolidWorks Model

The mechanical engineers in our group developed a SolidWorks [5] model of our robot, a screen shot is shown in figure 4.1. The distinguishing feature of this model is the level of detail. For instance, figure 4.2 shows the gears inside the servo.

SolidWorks has two very important uses:

1. COSMOSMotion is an add-on to SolidWorks that helps to do motion analysis. You can pick joints, assign rotary or linear motion to them and do torque calculations etc. For details, refer to our project homepage [4].

2. Open API. The SolidWorks Application Programming Interface is thoroughly documented. Using Visual Basic, a user can interface to a running SolidWorks program and execute commands like “rotate part X by 27 degrees about axis Y”. This proved invaluable in developing the interface between PSPICE and SolidWorks, Paranoia 1.

4.2 Paranoia

A screenshot of Paranoia is shown in figure 4.3. Paranoia is the work of Abishek Misra, an EECS undergrad in our group. Paranoia takes the output file from PSPICE in the form of tab delimited text data (time,voltage values) and “actuates” the SolidWorks model. In other words, Paranoia is

1I asked the CS guru in our group, Abhishek Mishra to pick a cool sounding name for the tool, he picked Paranoia. I liked it, so it stuck
Figure 4.1: SolidWorks Raptor Model
Figure 4.2: Detailed modelling of the gears in the servos
Figure 4.3: A screen shot of Paranoia, build 1
our simulated actuation. Hence, if we are actuating servos, then Paranoia outputs a square wave as an intermediate text output file for debugging purposes. It then actuates the SolidWorks model using this data.

That's the end of the simulation part of our project. We finished the simulation by the middle of March 2005, the implementation and debugging took us another two months!
Chapter 5

Implementation Notes

In this chapter, we give a “brain-dump” of our implementation.

5.1 Lynxmotion Biped Scout Kit

The source of our SolidWorks model is the Biped Scout kit from Lynxmotion [2]. A picture of the assembled robot is shown in figure 5.1. It took me a total of 6 hours to assemble the robot.

![Figure 5.1: The Biped Scout 1 kit from Lynxmotion](image)

5.2 Power Supply

The power supply for our battery is the 7.4 V, 60A Lithium Polymer battery from Thunder Power [7]. We designed a printed circuit board (PCB) for our power supply, it can be found online at our homepage [4]. The supply has two +/- 12 V DGP12U5D12 DC-DC converters from Power One [3] for the op-amp neural network (one DC-DC converter per leg). We also had 6 LM1084 Low Drop Out voltage regulators for the servos and the electronics. A screen shot of our working power board is shown in figure 5.2. A cost breakdown of our power board is given in table 5.1 in the last section of this chapter.
5.3 CNN Board

Figure 5.3 shows us connecting a digital scope to the CNN board. Figure 5.4 shows the actual output. We adjusted the frequency to be on the order of 60 Hz.

5.4 Interface between CNN and Raptor Servos

However, one problem which we hoped wouldn’t occur, did: the board was way too big for the robot. Figure 5.5 says it all.

But, because of our detailed simulation environment, we already had the transducer implemented
in the form of Paranoia! Hence, we used the neural network data from the PSPICE circuit and copied it onto the JSTAMP. The JSTAMP was to interface the actual CNN to the robot. But, since the board was too big we used the data from the PSPICE simulation instead, since we experimentally verified the equality of the two methods.

5.5 Total Cost of the project

The data in the table below was obtained from our purchase orders. The prices exclude tax and shipping. Yet, you can see the implementation easily costs less than $2500!

<table>
<thead>
<tr>
<th>Category</th>
<th>Cost (US $)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Supply</td>
<td>315</td>
</tr>
<tr>
<td>CNN Board</td>
<td>800</td>
</tr>
<tr>
<td>Lynxmotion body</td>
<td>600</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1715</strong></td>
</tr>
</tbody>
</table>

Table 5.1: Table showing the total product cost for our project

The final chapter is mainly about where we plan to go from here.
Chapter 6

Conclusion and Future Work

Figure 6.1: The Raptor is almost finished

The robot is complete, we are just waiting to build a mechanical prop so the robot does not fall down as it takes a few steps. We will be uploading movies of our robot on our website in early June 2005.

Our short term goals (Summer 2005) are:

• Design a biomimetic body.

• Reimplement the CNN using FPGA (Field Programmable Gate Arrays).

• Start working on a control law.

Our very long term goals are to do away with motors and use dynamically compliant actuation systems like artificial muscle fibers.

\footnote{Recall: the controller has not been designed yet, nor are there any sensors on the robot}
Bibliography


