EE290O Final Project - Ramasubramanian Group

Rose Abramson, Jason MacDonald, Phillippe Phanivong, Jaimie Swartz

1 Summary of Ramasubramanian et al.

The purpose of [1] was to implement a more accurate converter interfaced generation (CIG) model in a positive sequence powerflow software (specifically, PSLF) for large scale power system models for multi-machine network transient stability studies. This paper was motivated by the issues in the CIG source model that currently exist in PSLF, referred to by the authors as the "boundary current representation". In the paper, Ramasubramanian et al. propose a new converter model, validate it against a point on wave simulation, compare both the boundary current representation and the new model against the point on wave simulation, and then simulate CIG sources using both models in a modified IEEE 9-Bus system and a modified WECC 2012 system.

Ramasubramanian et al. propose a CIG model based on a Thévenin voltage source. This model replaces the current injection with a generated voltage behind an impedance representing either a filter inductor or a transformer. The authors also built a simple controller for this model but stressed the control architecture was intentially simple and generic to focus on the development of a positive sequence model.

To validate their new CIG model, Ramasubramanian et al. also built a point on wave model in PLECS. This PLECS model was then compared to both the boundary current and the new Thévenin equivalient representations. The PLECS model was used for calibration of an individual CIG source and then used in the modified IEEE 9-Bus system to compare the response of each model. The authors found the Thévenin equivalient representation followed the PLECS simulation better than the boundary current representation for each of the disturbances.

Once validated, the authors tested their CIG source model on a modified WECC 2012 system. This system had 18,205 buses, 13670 lines, and 2592 generators. 528 generators were replaced with CIG sources and several disturbances were simulated to compare the new Thévenin equivalient representation to the boundary current representation. The authors found their new model did not add significant amounts of computation time to the simulation and was able to converge for all disturbances simulated. In comparison, they found the boundary current representation simulations did not always converge for some of the disturbances they simulated. However, this result was not a focus of this paper.

Ramasubramanian et al. conclude their Thévenin equivalient representation of CIG sources is an improvement on CIG models for a positive sequence powerflow simulations. Future work suggested from this paper included modification to time constants used in the model to more accurately match the PLECS model. This paper also was extended further in Ramasubramanian's PhD thesis [2].

There were several shortcomings of the approach used in this paper. The CIG model used several time constants without clear explanation of their meanings. Although the authors stressed the control architecture was built to be generic, it was not clear why certain decisions were made in the model or if it was based on a standard control structure. The WECC 2012 system model placed all of the CIG sources in one region instead of dispersing the generators across the system. Also, the better fault convergence could have been explored more.

2 Semester Project Objectives

The objectives of this semester's project was to replicate the CIG models built by Ramasubramanian et al. in MATLAB and build a positive sequence model of a network for any CIG or synchronous generator model to interface with. This required the following work to be completed:

1. Build a model of the boundary current converter
2. Build a model of the Thévenin voltage source converter
3. Build a model of synchronous generator
4. Connect each source model to an infinite bus to model a change in load and/or line impedance
5. Compare the performance of the Thévenin voltage source with the boundary current converter
6. Connect the CIG model with the synchronous generator in a two bus system
7. Build a model of the network
8. Connect the CIG model into the network model

3 Detailed Model Description

3.1 CIG Models

The following section describes both the boundary current and the Thévenin voltage source models used for CIG sources.

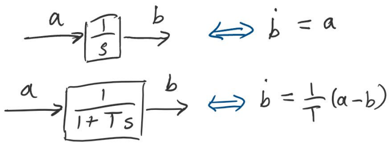
3.1.1 Converter Modeling Levels

The simplest model of a converter is that it is a nodal power injection. This perspective may be taken by transmission level operators who are focusing on economic dispatch rather than technical or stability concerns. One level deeper models a resource interfaced with a converter as an ideal constant voltage or constant current AC source. See figure 9 and 10. In reality, inverters are electrical devices that converter DC power to AC power, and there can be models for the DC circuitry, AC circuitry, and control loops. The third level is thus the use of measurements of grid frequency and voltage to adjust the AC power output, while ignoring the DC side. The fourth level would be detailing the DC side to model non-ideal switching that produces the AC waveform, as well as modeling a phase lock looped controller that estimates the grid frequency to make real power control decisions. The fourth level of modeling detail is advantageous for seeing how different designs reduce harmonics and switching loses. Power electronics literature tends to have the 4th level of detail, while power systems papers tend to focus more on control loop design and may make simplifying assumptions on the DC side. In this work, [1] adopts the third level of detail, while other works such as [3] adopt the fourth level.

3.1.2 Boundary Current Model

The boundary current converter model was designed for use in General Electric's phasor time-domain simulation software, PSLF or PSS/e. The model represents a aggregation of converters, encompassing those on a solar farm for example, to understand how it would impact or be impacted by transmission level disturbances.Because the model is meant for commercial software, it is structured to allow user written blocks and many parameters so that different users of the software can scale or modify the model for their application.

[1] and [4] use signal diagrams to represent the boundary current converter model, which is helpful for understanding how variables interact and can be implemented in visual software such as Simulink, but for implementation in MATLAB we convert the necessary diagrams into differential equations. Two common signal-diagram-to-equation conversions used are shown in figure 1. Additionally, variables g_i are internal states that only exist within each converter sub-block for use in building the equations.



$a \to \boxed{\frac{1}{s}} \to b \iff \dot{b} = a$
$a \to \boxed{\frac{1}{1+Ts}} \to b \iff \dot{b} = \frac{1}{T}(a-b)$

Figure 1: Relationship between laplace transform signal diagram piece and differential equation for two common blocks.

A high level diagram of the boundary current converter model, and those of a similar converter modeling level, is shown in Figure 2. The lefthand two blocks contain control loops and logic for converter decision making, while the righthand two blocks contain equations related to the physical converter device and power network circuit. It is well known that due to the transmission grid lines having a high X/R ratio, there is a decoupling that causes recative power flows to impact voltage magnitude, and real power flows to impact voltage phase angle and frequency. Hence this converter uses Q-V droop control to regulate the terminal bus voltage through equation 1.

First a measurement of the terminal voltage magnitude V_{term}, is inputed to the Q-V droop block. It is passed through a first order transfer function to represent measurement delay of T_r = 0.02s. IN practical implementation inverters tend to have this measurement capability internally built into the device. In the next two equations, we compare the delayed voltage measurement,g_1 with the reference voltage, which is the desired terminal voltage we seek that our converter maintain. We pass this difference, $(V_{ref} - g1)$, through an integrator and a delay term, scaling each branch by controller gain parameters K_{iv} and K_{pv}. Finally, the sum of the two branches is delayed with time constant T_c to represent actuator delays, yielding Q_{cmd} at the output of the Q-V droop block.

The original current control block in 2a takes in Q_{cmd} from the Q-V droop block to produce a current command associated with the reactive power, I_{qcmd}, so that ideally the inverter generated reactive power is equal to $I_{qcmd} * V_{term}$. The auxiliary test signal is omitted as no model or reference was provided to define that term, and the "P,Q priority flag" is omitted, allowing the inverter to produce real and reactive power according to two separate control structures. Note that this converter model does not work with the dq reference and simplifies the variables accordingly. In the results section we scrutinize the design and modify it to improve results, yielding equation 3a. P_{ord} original has no control loop associated with it, and is the direct real power from the solar PV panel.

The physical converter block is originally dominated by uder-defined operation curves. The high voltage management curve reduces the reacive current injection to limit V_{term} to 120V. The low voltage management curve causes a linear reduction of active current injection for V_{term} below 0.8pu. The low voltage power logic has a similar function. These three operational curves were removed because of their nonlinearity and reliance on user-defined design decisions. The remaining part of the block only has a delay T_{pwm} associated with pulse width modulation, as shown in equation 4a.

Comparing the equations above to those in [4] and [1], Some complexities have been removed. Specifically, limit functions have been removed, as implementing these piece-wise nonlinear functions is non-trivial and distracting from understanding the core functionality of the inverter. We later add one set of limits back in the results section to understand its effect in isolation. There are also two extra control loops in the original model aid in practical inverter operation: The antiwindup loop, containing the term Vfrz, is meant to "freeze" or halt the operation of the integrators when the controller is in saturation, i.e. has hit its current output limits. This is to prevent windup which would make the controller sluggish in reacting to the new system state after coming out of saturation. The termed "Q Droop Function" from [4] modifies the voltage reference to coordinate multiple inverters acting on the same reference voltage.

P-f droop:

$$\dot{g}_1 = \frac{1}{T_r}(V_{term} - g_1) \tag{1a}$$

$$\dot{g}_2 = \frac{1}{T_v}(K_{pv}(V_{ref} - g_1) - g_2) \tag{1b}$$

$$\dot{g}_3 = K_{iv}(V_{ref} - g_1) \tag{1c}$$

$$\dot{Q}_{cmd} = \frac{1}{T_c}(g_2 + g_3 - Q_{cmd}) \tag{1d}$$

$$\dot{g}_1 = K_{qi}(Q_{cmd} - V_{term} I_{qterm}) \tag{2a}$$

$$\dot{I}_{qcmd} = K_{vi}(g_1 - V_{term}) \tag{2b}$$

$$I_{pcmd} = P_{ord}/V_{term} \tag{2c}$$

$$\tag{2d}$$

It seems as though the model is called "boundary current" because the current control block has a central limiter block that bounds currents in several places.

$$I_{qcmd} = Q_{cmd}/V_{term} \tag{3a}$$

$$I_{pcmd} = P_{cmd}/V_{term} \tag{3b}$$

$$\dot{P}_{cmd} = K_{wi}(P_t - P_{nom}) \tag{3c}$$

$$\tag{3d}$$

$$\dot{g}_1 = \frac{1}{T_{pwm}}(I_{qterm} - g_1) \tag{4a}$$

$$\dot{g}_2 = \frac{1}{T_{pwm}}(I_{pterm} - g_1) \tag{4b}$$

$$\tag{4c}$$

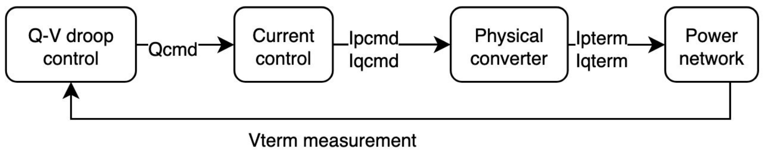


Figure 2: High level diagram of boudnary current inveter model block connections.

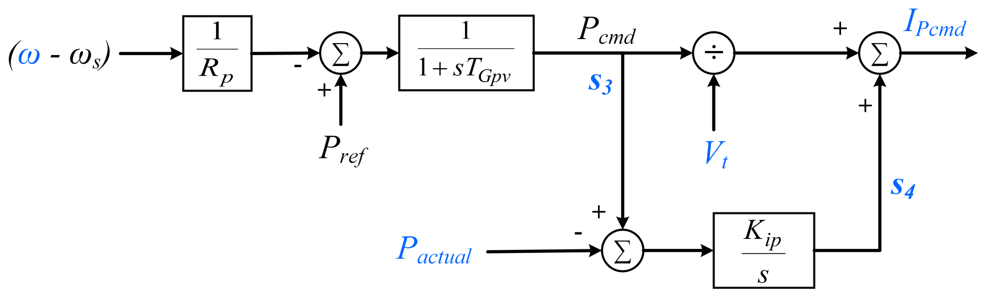
3.1.3 Thévenin Voltage Source Model

The Thévenin voltage source converter model consists of several parts, that have been broken up into separate functions in the MATLAB code. The models will first be introduced, followed by the coding methodology for setting up the DAEs in MATLAB.

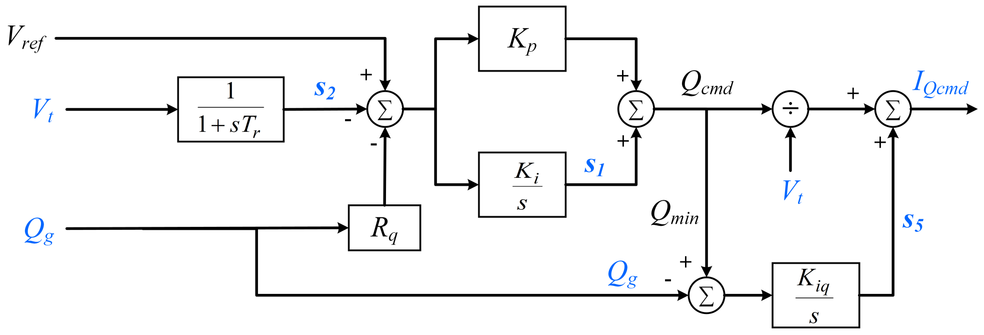
Power controller

As a main goal of the work was to model scenarios with close to 100% CIG penetration, there was motivation for a simple user defined control structure for the voltage converter [2]. This controller consists of a reactive power controller and real power controller. The effective real power order, in the form of a commanded I_{Qcmd}, is derived from the power setpoint, P_{ref} and the active power droop coefficient, R_p. The effective reactive power order, in the form of a commanded I_{Pcmd}, is derived from the voltage error along with the reactive power droop coefficient, R_q. The QV droop component is required for obtaining stable operation between converters when multiple converters are connected to the same bus [2].

A schematic control block diagram for the real and reactive powers are given in Fig. 3. These control blocks do not show the saturation limits imposed on P_{cmd} and Q_{cmd}.



(a) Real power controller



(b) Reactive power controller

Figure 3: User defined converter controller model

The differential equations describing this block diagram are given below. The following states for the power controller are given in Table 1, and are shown in blue in the control diagram. States s_1 - s_5 are intermediate states that are called out for ease of writing the differential equations to describe this system.

Table 1: Power Controller States

State	Description
s_1	Intermediate state
s_2	Intermediate state
s_3	Intermediate state
s_4	Intermediate state
s_5	Intermediate state
I_{Qcmd}	Commanded reactive current
I_{Pcmd}	Commanded real current
P_{actual}	Converter active power
Q_g	Converter generated reactive power
V_t	Terminal voltage (after coupling impedance)
ω	Converter angular frequency

The differential and algebraic equations that describe these states are given below.
Differential Equations:

$$\begin{aligned}
\frac{ds_1}{dt} &= K_i \cdot (V_{ref} - s_2 - R_q Q_g) \\
\frac{ds_2}{dt} &= \frac{1}{T_r} \cdot (V_t - s_2) \\
\frac{ds_3}{dt} &= \frac{1}{T_{Gpv}} \cdot (P_{ref} - (\frac{\Delta\omega}{R_p}) - s_3) \\
\frac{ds_4}{dt} &= K_{ip} \cdot (P_{cmd} - P_{actual}) \\
\frac{ds_5}{dt} &= K_{iq} \cdot (Q_{cmd} - Q_g)
\end{aligned}$$

Algebraic equations:

$$\begin{aligned}
I_{Qcmd} &= \frac{Q_{cmd}}{V_t} + s_5 \\
I_{Pcmd} &= \frac{P_{cmd}}{V_t} + s_4
\end{aligned}$$

where:

$$\begin{aligned}
P_{cmd} &= s_3 \\
Q_{cmd} &= s_1 + K_p \cdot (V_{ref} - s_2 - R_q Q_g)
\end{aligned}$$

In the full paper and thesis [1] [2], limits were imposed on the maximum active and reactive power and minimum reactive power deliverable. This was not implemented in the code for simplicity, but this has had significant impacts on the realism / correctness of the output plots.

Inner Current Loop

After the I_{Qcmd} and I_{Pcmd} current commands are generated, the actual i_q and i_d currents are derived from these commands by incorporating the time constants T_Q and T_D, as shown in Fig.

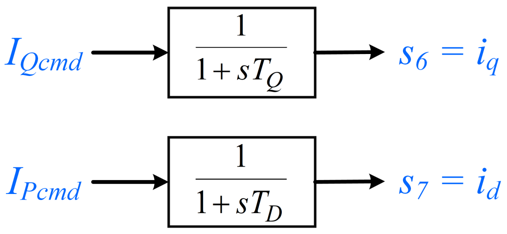


Figure 4: Inner current control loop

4. [2] also includes a limit I_{max} on i_q and i_d. None of the limits are shown in the control diagram as they were not implemented in the MATLAB code.

These time constants represent the effect of the inner current control loop of the converter control structure [2]. States s_6 and s_7 correspond to these actual i_q and i_d values, respectively. Table 2 summarizes the states involved in this subsection of the control diagram.

Table 2: Inner Current Loop States

State	Description
s_6	Actual i_q current
s_7	Actual i_d current

The differential equations that describe these states are given below. No algebraic equations are required.

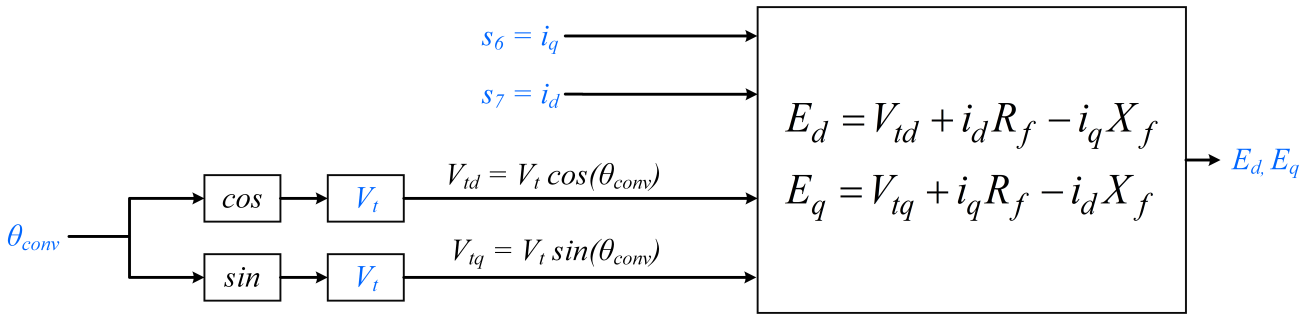
Differential Equations:

$$\frac{ds_6}{dt} = \frac{1}{T_q} \cdot (I_{Qcmd} - s_6)$$
$$\frac{ds_7}{dt} = \frac{1}{T_d} \cdot (I_{Pcmd} - s_7)$$

Voltage Source Converter

[1], [2] propose a voltage source representation of the converter, in order to include the effect of the converter coupling inductance in the simulation. The authors argue that this allows the model to depict the near instantaneous response that can be achieved from CIGs.

The converter is modeled as a Thévenin voltage source, as shown in Fig. 5. $R_f + X_f$ represents the coupling inductance, which could be either a filter inductor or a transformer. V_{td} and V_{tq} are the dq axis components of the terminal voltage, V_t. E_d and E_q represent the dq magnitudes of the developed converter voltage. Table 3 lists the states used in this control block.



$s_6 = i_q$
$s_7 = i_d$
$E_d = V_{td} + i_d R_f - i_q X_f$
$E_q = V_{tq} + i_q R_f - i_d X_f$
E_d, E_q
$V_{td} = V_t\, cos(\theta_{conv})$
$V_{tq} = V_t\, sin(\theta_{conv})$
θ_{conv}
cos
sin
V_t
V_t

Figure 5: Voltage Source Converter Model

Table 3: Voltage Source Converter Model States

State	Description
s_6	Actual i_q current
s_7	Actual i_d current
V_t	Terminal voltage
θ_{conv}	Converter angle
E_d	d-axis developed converter voltage
E_q	q-axis developed converter voltage

The following algebraic equations describe the voltage source converter model. There are no differential equations.

Algebraic Equations:

$$E_d = V_{td} + i_d R_f - iqX_f$$
$$E_q = V_{tq} + i_q R_f - idX_f$$

where:

$$V_{td} = V_t \cdot \cos(\theta_{conv})$$
$$V_{tq} = V_t \cdot \sin(\theta_{conv})$$

PWM Switching Delay Block

The effects of the PWM switching are also included in the converter model. For ease of reading, these effects have been split into two cascaded PWM stages: 1) the time delay due to the PWM switching process, and 2) the effects of the modulation index and dc voltage. The PWM Switching Delay Block deals with the first stage.

The delay in the PWM/switching process is represented by the time constants T_{ed} and T_{eq}, as shown in Fig. 6. The states involved are listed in Table 4.

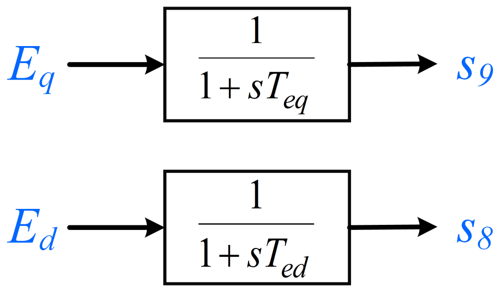


Figure 6: PWM Switching Delay Block

Table 4: PWM Switching Delay Block States

State	Description
s_8	Delayed E_d due to PWM switching
s_9	Delayed E_q due to PWM switching
E_d	d-axis developed converter voltage
E_q	q-axis developed converter voltage

This block is described by the following differential equations. There are no algebraic equations.

Differential Equations:

$$\frac{ds_8}{dt} = \frac{1}{T_{ed}} \cdot (E_d - s_8)$$
$$\frac{ds_9}{dt} = \frac{1}{T_{eq}} \cdot (E_q - s_9)$$

PWM Modulation Block

The PWM block includes the effect of the dc voltage and the amplitude modulation ratio m on of the pulse-width modulation control depicted in Fig7.

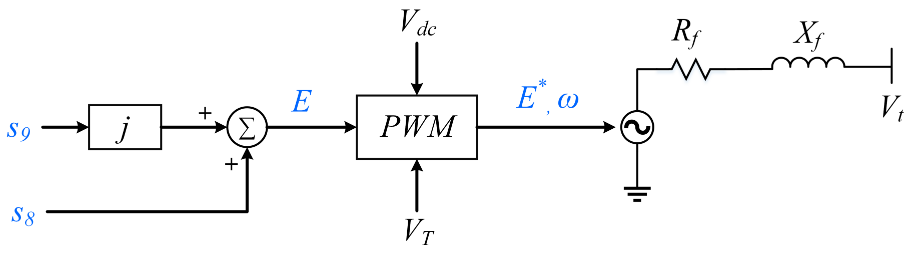


Figure 7: PWM Modulation Block

The following equations are used to initialize the carrier voltage V_T and dc voltage V_{dc}, as well as calculate the magnitude and angle of the developed converter voltage, E.

Setup equations:

$$V_T = \frac{\sqrt{E_d^2 + E_q^2}}{0.6}$$
$$V_{dc} = \frac{\sqrt{E_d^2 + E_q^2}}{0.5 * 0.6}$$
$$\|E\| = \sqrt{E_d^2 + E_q^2}$$
$$\phi = \tan^{-1}(\frac{E_q}{E_d})$$
$$m = \frac{\|E\|}{V_T}$$

While [2] writes the equations using E_d and E_q, from the diagram it appears as though the developed converter voltage and angle, ϕ, should use the delayed internal states s_8 and s_9. Therefore, this was how the code was implemented in MATLAB, so that the setup equations were modified to:

$$V_T = \frac{\sqrt{s_8^2 + s_9^2}}{0.6}$$
$$V_{dc} = \frac{\sqrt{s_8^2 + s_9^2}}{0.5 * 0.6}$$
$$\|E\| = \sqrt{s_8^2 + s_9^2}$$
$$\phi = \tan^{-1}(\frac{s_8}{s_9})$$
$$m = \frac{\|E\|}{V_T}$$

The three-phase voltages can then be constructed as follows:
Phase Voltages:

$$\begin{aligned} E_a &= \frac{1}{2} \cdot m \cdot V_{dc} \cos(\omega_s t + \phi) \\ E_b &= \frac{1}{2} \cdot m \cdot V_{dc} \cos(\omega_s t + \phi - \frac{2\pi}{3}) \\ E_c &= \frac{1}{2} \cdot m \cdot V_{dc} \cos(\omega_s t + \phi - \frac{4\pi}{3}) \end{aligned}$$

Using the Park Transformation, given below, the final developed converter dq-axis voltages (E_d^* and E_q^*) can then be calculated as given in the following algebraic equations.
Park transformation:

$$\begin{bmatrix} E_d \\ E_q \\ E_0 \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\omega t) & \cos(\omega t - \frac{2\pi}{3}) & \cos(\omega t + \frac{2\pi}{3}) \\ -\sin(\omega t) & -\sin(\omega t - \frac{2\pi}{3}) & -\sin(\omega t + \frac{2\pi}{3}) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} E_a \\ E_b \\ E_c \end{bmatrix}$$

Algebraic Equations:

$$\begin{aligned} E_d^* &= \frac{2}{3} \cos(\omega t) \cdot E_a + \cos(\omega t - \frac{2\pi}{3}) \cdot E_b + \cos(\omega t + \frac{2\pi}{3}) \cdot E_c \\ E_q^* &= -\frac{2}{3} \sin(\omega t) \cdot E_a - \sin(\omega t - \frac{2\pi}{3}) \cdot E_b - \sin(\omega t + \frac{2\pi}{3}) \cdot E_c \end{aligned}$$

Table 5 lists the states used in this block.

Table 5: PWM Modulation Block States

State	Description
s_8	Delayed E_d due to PWM switching
s_9	Delayed E_q due to PWM switching
E_d^*	Final d-axis developed converter voltage
E_q^*	Final q-axis developed converter voltage
ω	Converter angular frequency
optional (E_d)	d-axis developed converter voltage
optional (E_q)	q-axis developed converter voltage

3.2 Synchronous Generator Models

Several synchronous generator models were built with increasing levels of detail. First, a second-order model was built using only the machine dynamics per [5]. Then the complete classical model was built per [6]. Finally, a fourth-order generator model was built.

3.2.1 Second-Order Synchronous Generator

The first synchronous generator modeled built was the classical or second-order model:

$$\dot{\delta} = \Omega_b(\omega - 1) \tag{5a}$$

$$\dot{\omega} = (p_m - p_e - D(\omega - 1))/2H \tag{5b}$$

However, (5a, 5b) only describe the mechanical dynamics of the synchronous generator. The complete classical model requires the following equations per [6]:

$$
\begin{aligned}
\dot{\delta} &= \Omega_b(\omega - 1) && \text{(6a)}\\
\dot{\omega} &= (p_m - p_e - D(\omega - 1))/2H && \text{(6b)}\\
0 &= (v_q + r_a i_q)i_q + (v_d + r_a i_d)i_d - p_e && \text{(6c)}\\
0 &= v_q + r_a i_q - e'_q + x'_d i_d && \text{(6d)}\\
0 &= v_d + r_a i_d - x'_d i_q && \text{(6e)}\\
0 &= v_h \sin(\delta - \theta_h) - v_d && \text{(6f)}\\
0 &= v_h \cos(\delta - \theta_h) - v_q && \text{(6g)}\\
p_h &= v_d i_d + v_q i_q && \text{(6h)}\\
q_h &= v_q i_d - v_d i_q && \text{(6i)}
\end{aligned}
$$

3.2.2 Fourth-Order Synchronous Generator

It can be inferred from the [1] that the synchronous machine models used in the nine-bus example were of the fourth order or two-axis models. These include transient dynamics on both the d and q axes. A complete fourth-order model was built per [6]:

$$
\begin{aligned}
\dot{\delta} &= \Omega_b(\omega - \omega_s) && \text{(7a)}\\
\dot{\omega} &= (\tau_m - \tau_e - D(\omega - 1))/2H && \text{(7b)}\\
\dot{e}'_q &= (-e'_q - (x_d - x'_d)i_d + v_f)/T'_{d0} && \text{(7c)}\\
\dot{e}'_d &= (-e'_d + (x_q - x'_q)i_q)/T'_{q0} && \text{(7d)}\\
0 &= (v_q + r_a i_q)i_q + (v_d + r_a i_d)i_d - \tau_e && \text{(7e)}\\
0 &= v_q + r_a i_q - e'_q + x'_d i_d && \text{(7f)}\\
0 &= v_d + r_a i_d - e'_d + x'_d i_q && \text{(7g)}\\
0 &= v_h \sin(\delta - \theta_h) - v_d && \text{(7h)}\\
0 &= v_h \cos(\delta - \theta_h) - v_q && \text{(7i)}\\
p_h &= v_d i_d + v_q i_q && \text{(7j)}\\
q_h &= v_q i_d - v_d i_q && \text{(7k)}
\end{aligned}
$$

3.2.3 Type II Governor

A turbine governor was added to the synchronous generator models to better control them when in parallel. The model used was a type II governor per [6]:

$$
\begin{aligned}
\dot{x}_g &= (\frac{1}{R}(1 - \frac{T_1}{T_2})(\omega^{\text{ref}} - \omega) - x_g)/T_2 && \text{(8a)}\\
\hat{\tau}_m &= x_g + \frac{1}{R}\frac{T_1}{T_2}(\omega^{ref} - \omega) + \tau_{m_0} && \text{(8b)}\\
\tilde{\tau}_m &= \begin{cases} \tau^{\max} & \text{if } \hat{\tau}_m > \tau^{\max} \\ \hat{\tau}_m & \text{if } \tau^{\min} \le \hat{\tau}_m \le \tau^{\max} \\ \tau^{\min} & \text{if } \hat{\tau}_m < \tau^{\min} \end{cases} && \text{(8c)}\\
0 &= \tilde{\tau}_m - \tau_m && \text{(8d)}\\
0 &= \omega_0^{\text{ref}} - \omega^{\text{ref}} && \text{(8e)}
\end{aligned}
$$

3.2.4 Automatic Voltage Regulators

An automatic voltage regulator (AVR) model was built to control the field voltage of the generator. The model used was the type I, a DC exciter, model per [6]:

$$0 = \tilde{v}_f - v_f \tag{9a}$$
$$0 = v_0^{ref} - v^{ref} \tag{9b}$$
$$\dot{v}_m = (v_h - v_m)/T_r \tag{9c}$$
$$\dot{v}_{r1} = (K_a(v^{ref} - v_m - v_{r2} - \frac{K_f}{T_f}\tilde{v}_f) - v_{r1})/T_a \tag{9d}$$
$$\dot{v}_{r2} = -(\frac{K_f}{T_f}\tilde{v}_f + v_{r2})/T_f \tag{9e}$$
$$\dot{\tilde{v}}_f = -(\tilde{v}_f(K_e + A_e e^{B_e|\tilde{v}_f|}) - v_r 1)/T_e \tag{9f}$$

3.3 Network Model

3.3.1 Infinite Bus

An infinite bus is one that has constant voltage and frequency, regardless of the current or power that is pulled from it or sent to it. It can be thought of as representing the connection to an infinitely large grid that is not affected by power flows that the grid component being simulated produces. Typically one tests a new model for a grid component by connecting it to an infinite bus, as shown in Figure 8, where the grid component is a voltage source with internal impedance Z_f. For this work we use a similar setup but add a load to the grid component terminal bus Z_{load} so that we can examine the converter disturbance rejection capability in response to load changes, as shown in Fig 9. The same network is setup for the voltage source converter (see Fig 10) so that the two converter models can be compared.

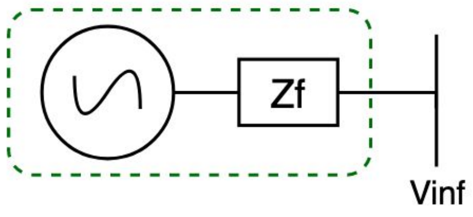


Figure 8: High level circuit diagram for voltage source converter connected to an infinite bus.

The powerflow equations for a two bus system can be modeled as just the power transfer equation plus any additional load at the buses. For a single source connected to an infinite bus, if only the machine dynamics for a synchronous machine (5a, 5b) or the only the real power out of the CIG are used then only the real power component is required (10). However if (6a-6i) or a complete CIG model are used, both the real and reactive power components need to be accounted for (11a, 11b).

$$P_e = \frac{V_g V_s}{z}\sin(\delta - \theta) + P_{load} \tag{10}$$

$$P_e = \frac{V_g V_s}{z}\sin(\delta - \theta) + P_{load} \tag{11a}$$
$$Q_e = \frac{V_g^2}{z} - \frac{V_g V_s}{z}\cos(\delta - \theta) + Q_{load} \tag{11b}$$

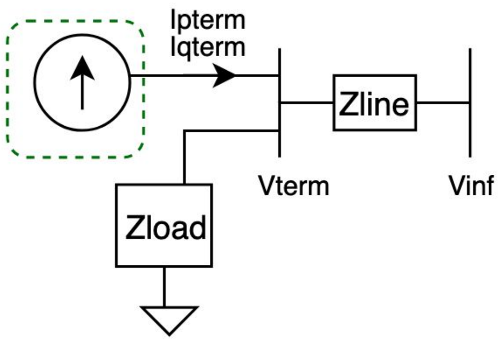


Figure 9: High level circuit diagram for current source converter connected to an infinite bus and load on terminal bus.

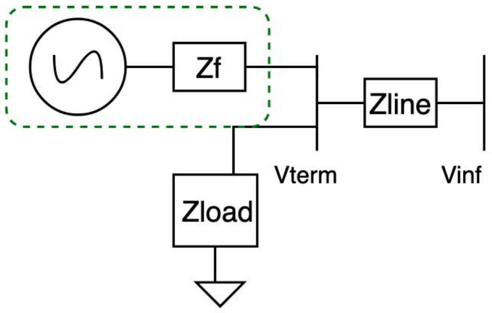


Figure 10: High level circuit diagram for voltage source converter connected to an infinite bus and load on terminal bus.

3.3.2 Infinite Bus for Voltage Source Converter

The infinite bus was set up to a similar setup to that described by Fig. 10, so that the following algebraic equations described the power flow. The first two equations are used to solve for V_t and θ_{conv}, while the last two equations are used to solve for Q_g and P_{actual}. V_∞ and θ_∞ describe the terminal voltage and angle of the infinite bus that the converter is connected to.

Algebraic equations:

$$0 = \frac{V_t V_\infty}{X} \cdot \sin\left(\theta_{conv} - \theta_\infty\right) + Real(S_{load}) - P_{actual}$$

$$0 = \frac{V_t^2}{X} - \frac{V_t V_\infty}{X} \cdot \cos\left(\theta_{conv} - \theta_\infty\right) + Imag(S_{load}) - Q_g$$

$$0 = V_{tq} i_d - V_{td} i_q - Q_g$$
$$0 = V_{td} i_d + V_{tq} i_q - P_{actual}$$

where X is the line impedance (completely reactive) and S_{load} is the load reactance (can be real or reactive). The last state, ω, is defined to have a derivative equal to 0, so it is described by the differential equation below.

Differential equtions:

$$\frac{d\omega}{dt} = 0$$

The states are summarized in Table 6

Table 6: Infinite Bus States

State	Description
V_t	Terminal voltage (after coupling impedance)
θ_{conv}	Converter angle
P_{actual}	Converter active power
Q_g	Converter generated reactive power
ω	Converter angular frequency

3.3.3 Two Machine System

For two sources connected in parallel, both generators will need either (10) or (11a, 11b) depending on the model. When using DAEs it is useful to split the power transfer equations into the power being transmitted to the other bus (P_{tf_h}) and power balance of the generator. For example, (10) becomes (12a-12d).

$$P_{tf_1} = \frac{V_{g_1}V_{g_2}}{z}\sin(\delta_1 - \delta_2) \tag{12a}$$

$$P_{e_1} = P_{load_1} + P_{tf_1} \tag{12b}$$

$$P_{tf_2} = \frac{V_{g_1}V_{g_2}}{z}\sin(\delta_2 - \delta_1) \tag{12c}$$

$$P_{e_2} = P_{load_2} + P_{tf_2} \tag{12d}$$

3.3.4 Multi-Bus Network

The standard power flow equations are shown in equations 13a. As shown in standard power systems courses. P_h, Q_h, V_h, and δ_h denote the bus real power, reactive power, voltage magnitude, and phase angle. For implementation, it is important to note that phasor notation is not supported well by the DAE solver functions. Hence equations for the power network should be kept with explicit sin and cosine terms with complex and real parts separates into real and reactive power equations.

$$P_h = \sum_{k=1}^{N} |V_h||V_k|(G_{hk}cos\theta_{hk} + B_{hk}sin\theta_{hk}) \tag{13a}$$

$$Q_h = \sum_{k=1}^{N} |V_h||V_k|(G_{hk}sin\theta_{hk} - B_{hk}cos\theta_{hk}) \tag{13b}$$

4 Computing Process

Each of the models described in 3 were built in MATLAB as separate DAE models to validate their performance. For each model, ODE15s was used to solve the DAEs. The following describes the general simulation methodology for each device. Several modifications were performed and described for each specific device below.

4.1 Boundary Current MATLAB Model

For the boundary current model, a state-space representation of the model was built and solved using ODE15s. The coding structure is modular in allowing the switching out of different sets of differential equation blocks and allowing easy modification of parameters and simulation settings, but requires rigor in setting up. For example, the initial condition vector, state vector, and dxdt vector must all have the same dimensions and be ordered in direct correspondence. Therefore every state must have an equation that defines how it evolves dynamically or algebraically.

Because the DAE solver operates on a state vector only, every variable is either a parameter which does not change throughout the simulation, or a state. Inputs and outputs do not fit into this coding structure, and thus inputs an outputs between internal blocks are considered states. A poorly chosen initial condition is often the cause of errors such as the DAE solver failing to integrate past a certain timestep, or the initial condition being undefined. Using fsolve is a good way to check initial conditions, which if chosen correctly will cause fsolve inputs and outputs to almost or completely match.

To create a disturbance, you cannot pass a timeseries structure into the DAE solver, as the solver integrates at every timestep to *generate* the timeseries of all state variables. Instead, one way to create a disturbance, in this case a shift in load, is to simulate up to the disturbance start time, save the state values, reassign the load parameter, then initialize the DAE solver again with the saved state values. Essentially you are causing an event to occur outside of the numerical integration.

An alternative coding structure was considered that was more aligned with the transfer function block diagrams in [1]. MATLAB's control systems toolbox has a set of function associated with state space, such as ss, series, parallel. However, these functions assume the equations are linear and time-invariant, which is typically not the case for the nature of the inverter and network equations. Hence we focus on translating the transfer blocks to equations and building a DAE vector dxdt.

Simplifications

The first simulation aims to be as aligned with the original boundary current model as possible. It includes the original current block that has a double integrator, as well as suggested parameters from [4]. All limit blocks are removed to better see the converter behavior.

First we provide steps for building up the complexity of a simulation, then present our results:

1. Turn off all controllers, i.e. by making the controller gain variables equal to zero
2. run fsolve to see check for any syntax errors. Once fsolve runs successfully, compare the initial condition x0 passed in to the output of fsolve, x00. If your system is plausible and your initial condition is suitable, x0 and x00 should match or be very close. fsolve runs only the algebraic part of your DAE, so it can be used to produce an initial condition for passing into the DAE solver
3. Run the DAE solver. Check that the first timestep of state variables is similar to the initial condition, there should not be a huge jump. If there is, there may be a discontinuity between the algebraic. Because the controller gains are off, there should be no actuation.
4. Repeat steps 1-3 with controller gains on but without any changes to load. The inverter should not actuate.
5. Repeat steps 1-3 with controller gains on and with a step change in load real power t1 seconds in, then a change in reactive power at t2 seconds into the simulation. To simulate the step change in load, you must reassign the load parameter. Run ODE15s with the first parameter to the time of your step change, save the ODE15s output, reassign the param, then use the saved output as the initial condition for a new ODE15s run

4.2 Thévenin Voltage Source MATLAB Model

The modeling process was similar to that described in the inverter boundary current modeling setup. For ease of writing th code, all saturation blocks / limits were left out of the code. As well be shown later, however, this meant that the gains used in [2] could cause various states to quickly blow up to unstable values.

The code organization is as follows:

- **solveDAE.m**: the main file, which imports parameters and calls the ode15s solver function. It also sets the Mass Matrix which identifies which elements of the column vector input to the ode solver correspond to algebraic vs. differential equations.
  - **parameters.m**: sets all the various parameters (such as time constants, gains, reference values)
  - **VoltageSource_InfBus.m**: calls the sub-blocks corresponding to those described above, and creates the complete column vector of DAE equations.
    - **power_controller.m**: implements the power controller block
    - **inner_current_loop.m**: implements the inner current loop block
    - **voltage_source.m**: implements the voltage source block
    - **PWM_switching_delay.m**: implements the PWM switching delay block
    - **infBusNwk.m**: implements the infinite bus network
    - **PWM_block_internal_states.m**: implements the PWM modulation block, using s_8 and s_9 instead of E_d and E_q as described above. Note, **PWM_block.m** implements the same equations using E_d and E_q, but is not currently used.

The main file **solveDAE.m** performs the following steps:

1. imports parameters
2. solves for an initial condition using fsolve
3. uses that initial condition as an input to the ode15s solver, which also calls the main system of DAEs described in the model "VoltageSource_InfBus.m" It also sets the Mass Matrix based on whether the state is defined by an algebraic equation (so the corresponding diagonal entry = 0) or a differential equation (so the corresponding diagonal entry = 1).
4. performs step changes in load conditions by running the system of equations for a certain amount of time, stopping to adjust parameters, such as S_{load}, and then restarting the simulation using the previous ending condition as the new initial condition

4.3 Second-Order Synchronous Generator MATLAB Model

For the second-order synchronous generator model, a state-space representation of the model was built and solved using ODE15s. Our first simulation only modeled the mechanical dynamics of the generator connected to an infinite bus. Since this model only had 3 DAEs, a single matlab script was written with the ODE function embedded at the bottom of the script. This allowed for easy reading and modifying of the system without needing to create multiple functions. For two-bus system models, all parameters and states were modeled in Per Unit with the system and generator at the same S base to simplify calculations. All other parameters for all of the synchronous generator simulations were taken from [6].

After this model was verified, the full classical synchronous generator model was built and several disturbances were tested on it. As described above for the boundary current model, the parameters of the system that were not states could not be modified while the ODE was solving. To simulate load changes or line losses, two separate ODEs were used with the first to calculate initial conditions of the system and the fed its output into the second ODE that had the change in load or line impedance hard-coded in the parameters.

Once the full classical synchronous generator model was validated to perform as expected against an infinite bus, two were connected in parallel across a two bus system with a load at each bus. During this testing, the synchronous machine was found to swing vigorously, leading us to implement a Type II governor model.

4.4 Fourth-Order MATLAB Synchronous Generator MATLAB Model

With the fourth order, or 2-axis, synchronous generator model, the emphasis of the coding exercise was on integration as much as it was on including the transient dynamics that were missing from the second order model. Again, we modeled the generator connected to an infinte bus via a simple transmission line. The system of equations was solved via ODE15s in Matlab.

The focus on integration into larger systems lead the coding design toward a modular structure. Similar to the methods employed on the inverter models, parameters [6] were hard coded into a separate file and could be read into models, structured as functions. This allows the system to use the same function for any generator while supporting a different parameters without having to rewrite the dynamic equations of each.

Turbine Governors and Automatic Voltage Regulators (AVRs) are treated similarly in the code. Each are defined as functions that have their own set of parameters and the code could be applied to any synchronous generator, without altering the generator code or reproducing the same set of equations from a different generator. The Turbine Governors and AVRs each replace an algebraic equation in the 2-axis synchronous generator model. To support this, and test just the generator model, the equations that are superceded needed to be removed from the base generator model and added to their own functions. These functions are called "NoGovernor" and "NoAVR" because they are added to a syncronous machine's model if that model will not include either a governor or an AVR. This structure allows the generator model to remain unchanged no matter whether there exists a functioning AVR model or not, and allows for testing the models independently.

The two-axis Synchronous Machine and the associated AVR and Governor models are located in the "SynchMachina" folder of our repository. To combine the models with algebraic power flow equations that link a single generator to an infinte bus, the "InfBus_to_2axis.m" function concatenates the synchronous machine, governor, AVR, and infinite bus functions that define their equations. Then "main.m" is used to initialize and run the ODE15s and plot the system. To initialize the system, fsolve is used, which finds the systems stablee operating point. However, since dynamics would not be particularly interesting if the system starts at a stable condition and is never perturbed, we slightly perturb the initial conditions given by fsolve so that the system can dynamically find that stable operating point in simulation. Due to time constraints, the 2-axis synchronous machine model was only implemented connected with an infinite bus.

4.5 MATLAB Network Model

The first network models were simply the source connected to an infinite bus. A two bus system with a load at each end was also built and tested against the classical synchronous generator model but not the other devices. For the IEEE 9-Bus system, it was found when reading the appendix of [2] that the M parameter of one of the generators was unreasonable high when compared to values from [7]. After discussing with other groups, we decided to not try to build the IEEE 9-Bus but instead use the IEEE 14-Bus as described in [6]. MATLAB code for solving a N-Bus system with Newton-Raphson was written for previous steady-state powerflow work and we anticipated adapting it for the IEEE 14-Bus. Although we never reached this step, the code for the powerflow is included in this project for completeness.

5 Modeling Results

5.1 Boundary Current MATLAB Model

The first simulation aims to be as aligned with the original boundary current model as possible. It includes the original current block that has a double integrator, as well as suggested parameters from [4] . All limit blocks are removed to better see the converter behavior. As shown in 11. there is a highly oscillatory response in the reactive power commands and terminal voltage, which makes sense from the double integrators present in that part of the loop. In fig 12, a single set of limits I_{qmin} and I_{qmax} are added with parameter values from [4]. The oscillations are truncated, but this seems like a crude way to address the oscillatory issue.

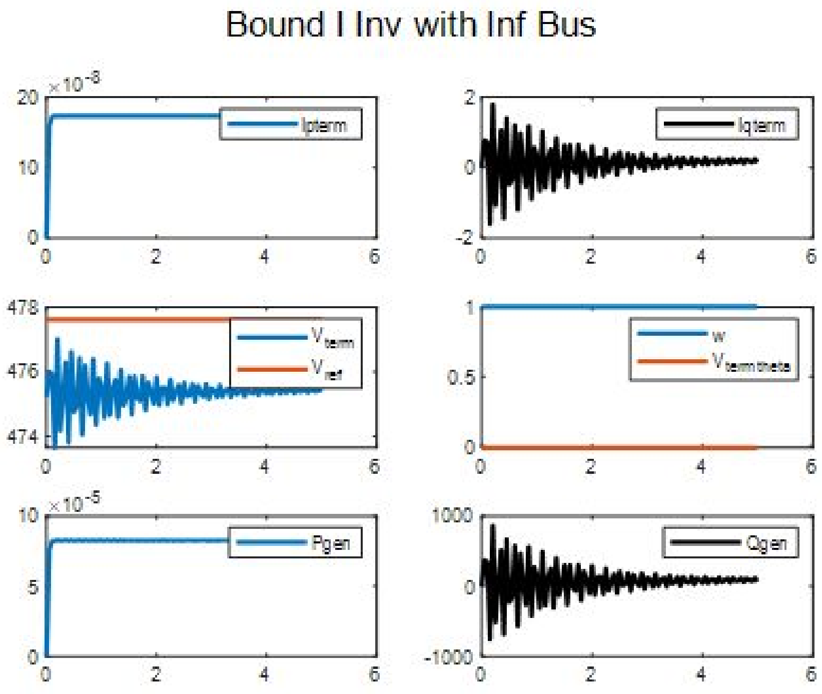


Figure 11: Boundary current model: response with original current block implemented without any limit functions.

The limit functions are removed and gain parameters are detuned to better examine the affect of the line impedance on the oscillations. The line and load impedance in Fig 13 are $Z_l = j$ and $Z_L = 100 + 100j$ respectively., while in Fig 14 they are $Z_l = 0.001j$ and $Z_L = 100 + 100j$ respectively. Notably, the oscillations that are originally damped and stable become unstable when the line impedance is reduced. This is likely due to the fact that while the load is a constant power load, the line is the only component in this small circuit that has impedance, which naturally causes a dampening affect. When the impedance value is scaled back, it's dampening capability

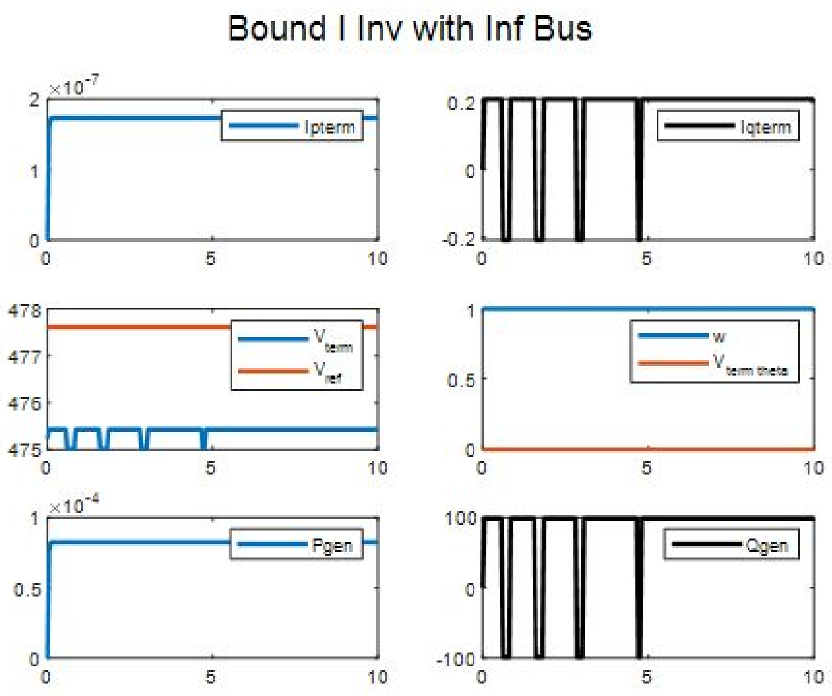


Figure 12: Boundary current model: response with original current block implemented and current limits added.

is as well. This concept can be generalized to how small isolated power networks are prone to instability more than some larger grids; the impact of aggregated generators or loads can shift flows dramatically without there being much impedance or inertia to dampen shifts in the network.

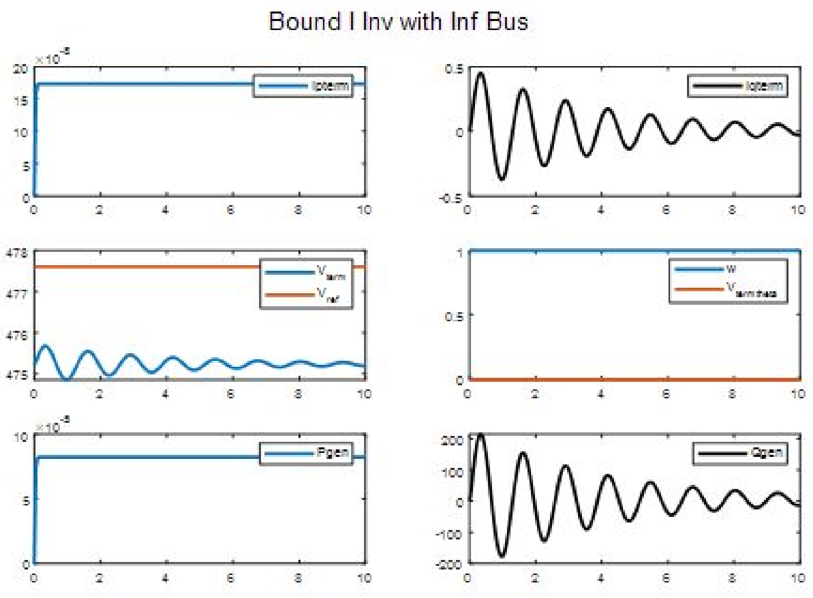


Figure 13: Boundary current model: reactive power response when controller gains are detuned and line impedance is relatively large.

From a design standpoint the boundary current model's gain parameters may be overtuned to ensure fast responses, but then several limit functions are added across multiple inverter blocks to allow for users of the PSLF software to enforce safety limits after the fact. Hence a proposed

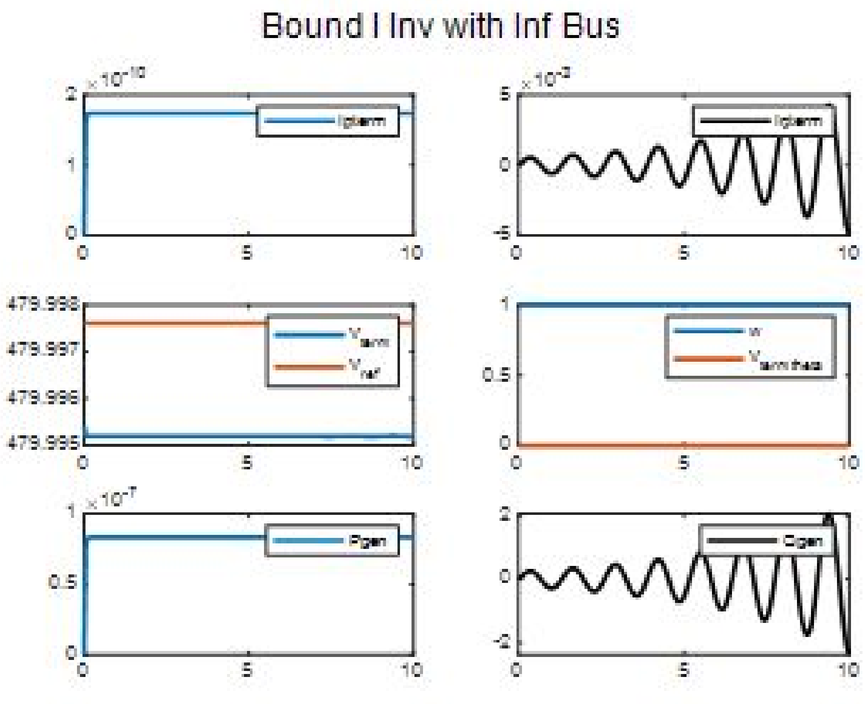


Figure 14: Boundary current model: reactive power response when controller gains are detuned and line impedance is relatively small, resulting in unstable behavior.

alternative to the boundary current model current block is in 3a. The new block has a basic real power control according to deviations of the terminal real power output from the nominal point as defined in the initial condition x0. With this new current block implemented, we create a disturbance in Fig 15 by changing the load on the infinite bus network by 50W and 20VARs, as shown in Fig 9. First we turn the controller gains off by setting them to zero to verify the converter does not respond to the load change. Next we turn the gain parameters on and tune them to achieve a reasonable response as shown in Fig 17. Both the Q-V droop and P-f control loops respond to shifts in the terminal voltage and nominal terminal node power to return to the initial state of the system after changes in the load.

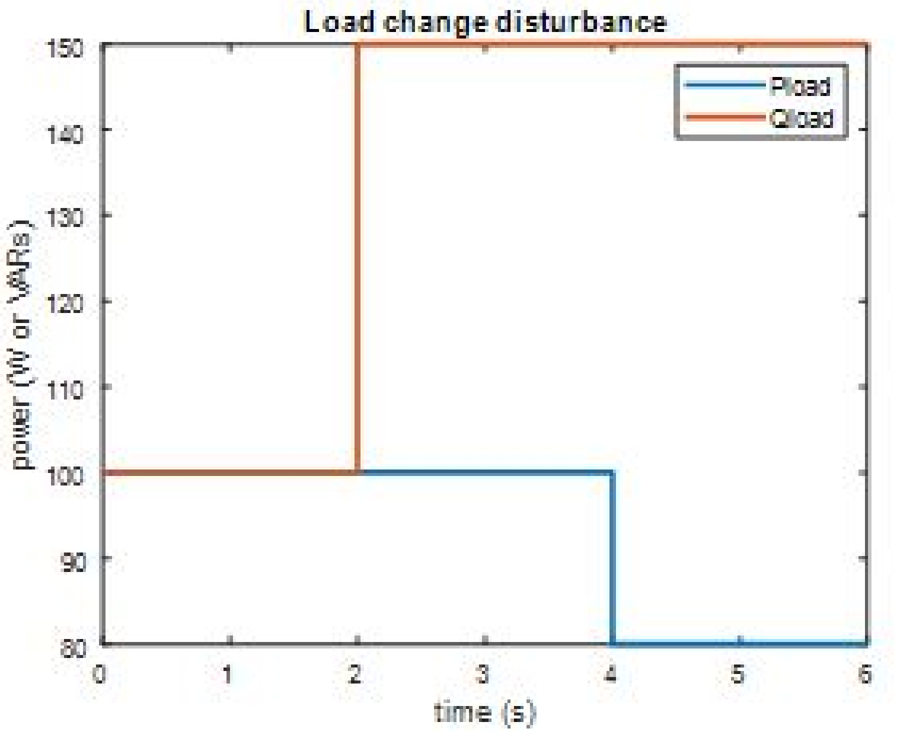


Figure 15: Boundary current model: disturbance caused at termina bus by changing the constant power load's real and reactive power

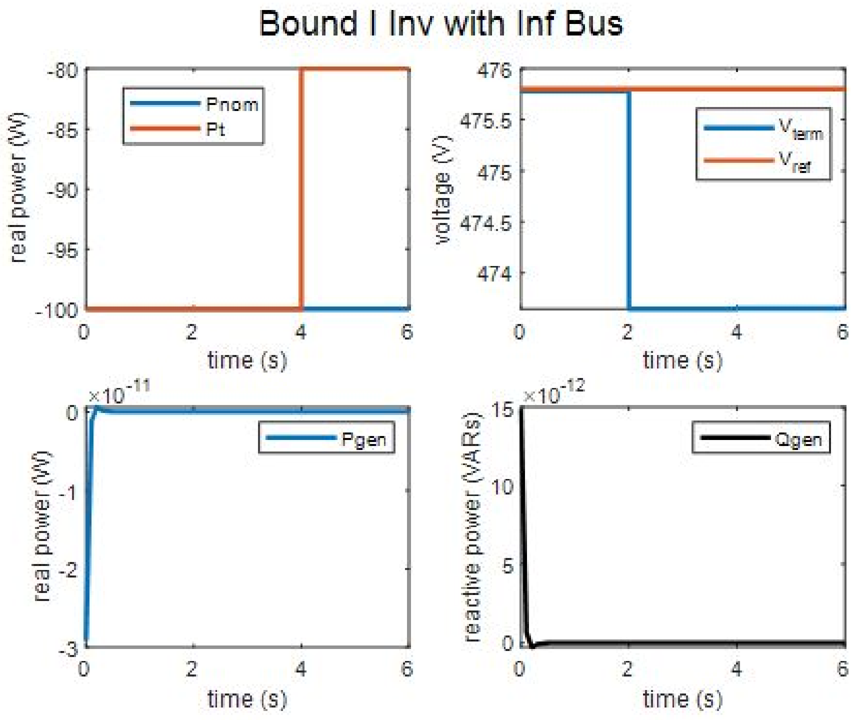


Figure 16: Boundary current model: after implement new current control block turn controller gains off to verify no converter generation

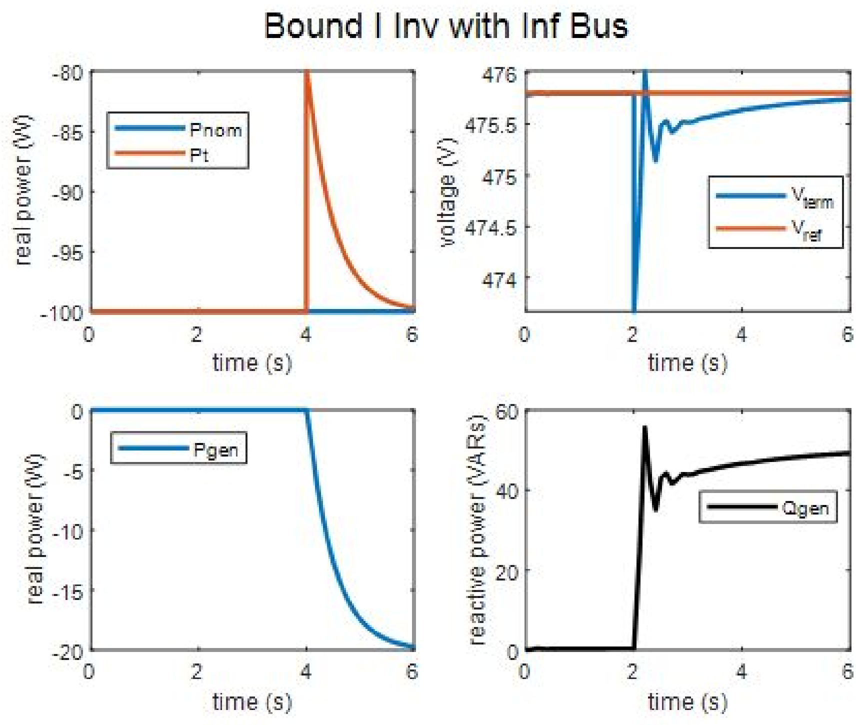


Figure 17: Boundary current model: response to load change disturbance after implement new current control block

5.2 Thévenin Voltage Source MATLAB Model

The voltage source converter model proved to be complex enough that only basic simulations were able to be run, so the boundary current and voltage source converters could not be compared in more detail. More work would also need to be done on turning gains and time constants (especially those related to the PWM switching behavior, as this was an interesting area that was not fully expanded on in the paper).

Additionally, the array of initial conditions could have huge impacts on the convergence of the solution, or the value of the end result. At times, better performance was obtained by not using fsolve to 'reset' the initial conditions, as this allowed those perturbations to show up in the data plots rather than being removed before the simulation started.

The load step used for the Thevenin Voltage source model was the same as for the boundary converter model, as shown in Fig. 18.

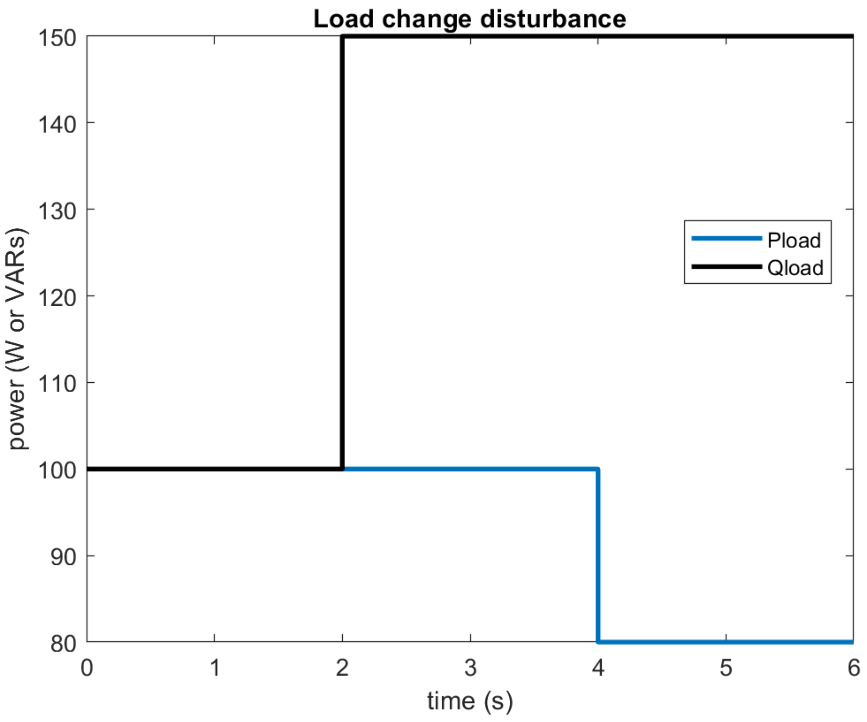


Figure 18: Voltage source model: disturbance caused at the terminalbus by changing the constant power load's real and reactive power

5.2.1 All gains set to 0

With all gains set to zero and the following initial condition vector, the following plots for ω, V_t, P_{actual}, and Q_g are obtained.

```
      s1            0
      s2    480.00000
      s3            0
      s4            0
      s5            0
   IQcmd            0
   IPcmd            0
  s6(iq)     -0.20833
  s7(id)      0.20833
```

```
          Ed     480.00000
          Eq             0
          s8     480.00000
          s9             0
          Vt     480.00000
  theta_conv             0
          Qg             0
     Pactual             0
       omega       1.00000
     Ed_star     480.00000
     Eq_star             0
```

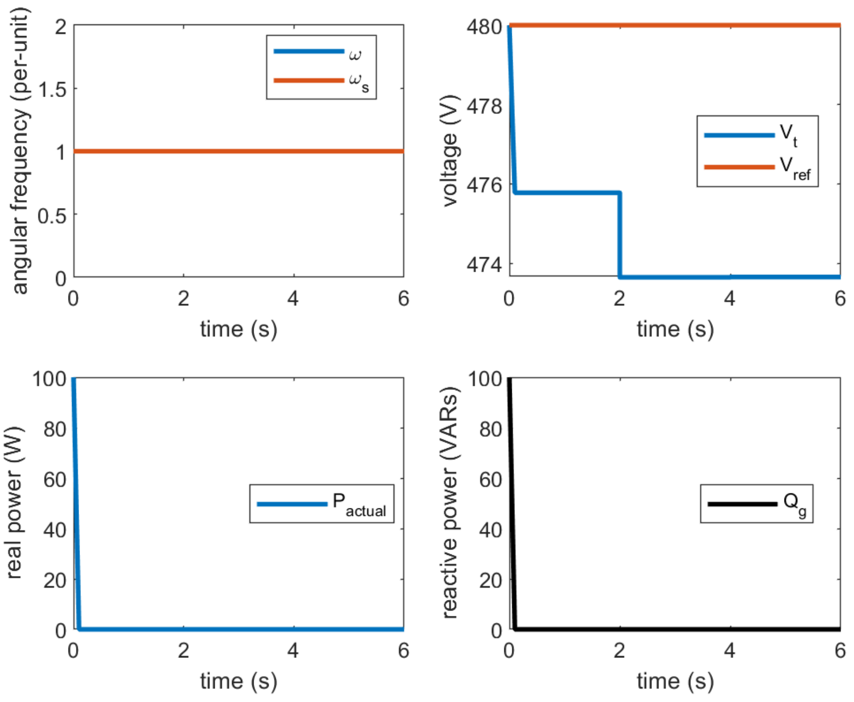


Figure 19: Voltage source model: plots for ω, V_t, P_{actual}, and Q_g with all gains set to 0

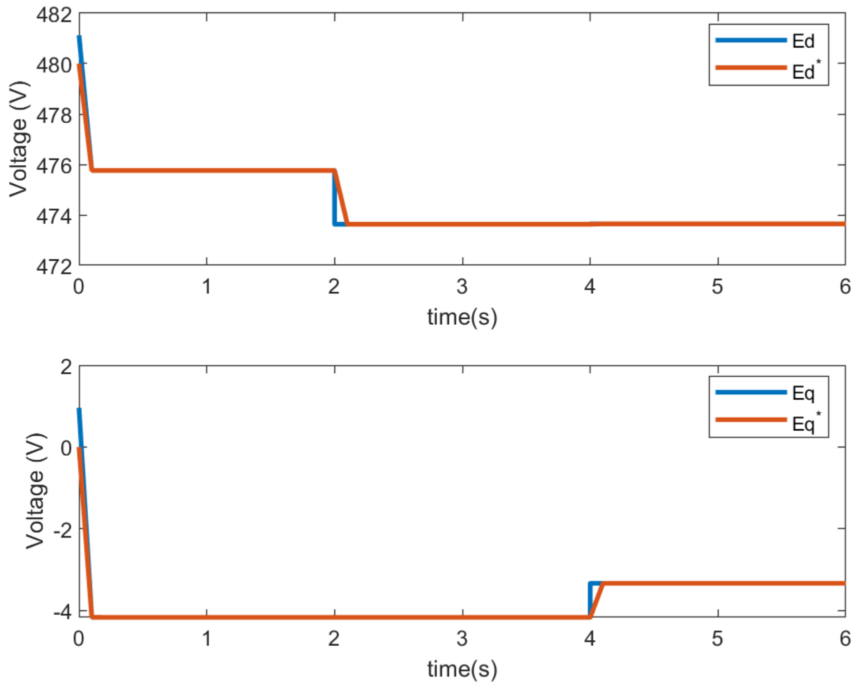


Figure 20: Voltage source model: plots for E_d, E_d^*, E_q, and E_q^* with all gains set to 0

5.2.2 $K_i = 10.0$

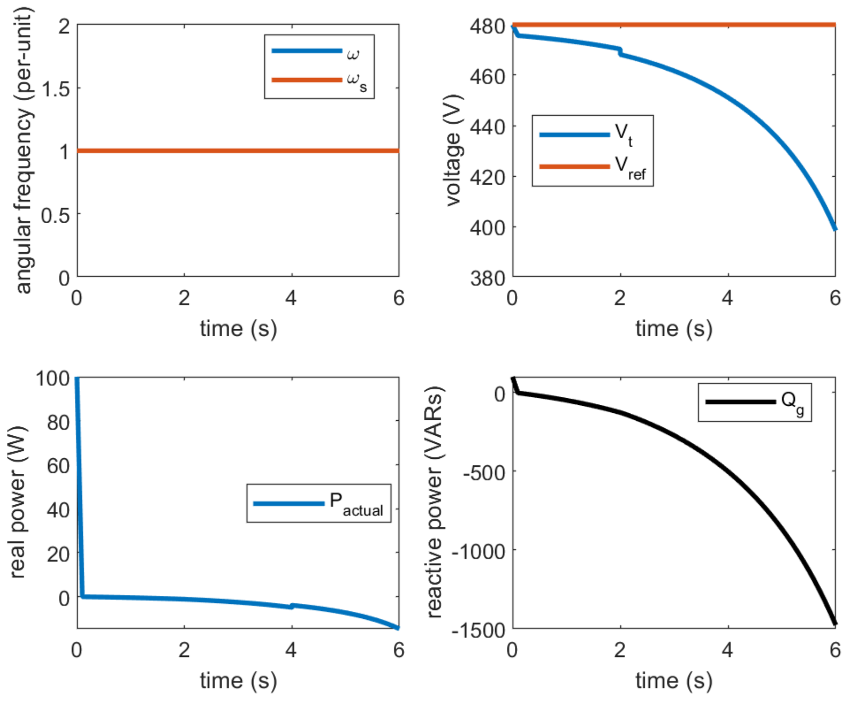


Figure 21: Voltage source model: plots for ω, V_t, P_{actual}, and Q_g with $K_i = 10.0$ and all other gains set to 0

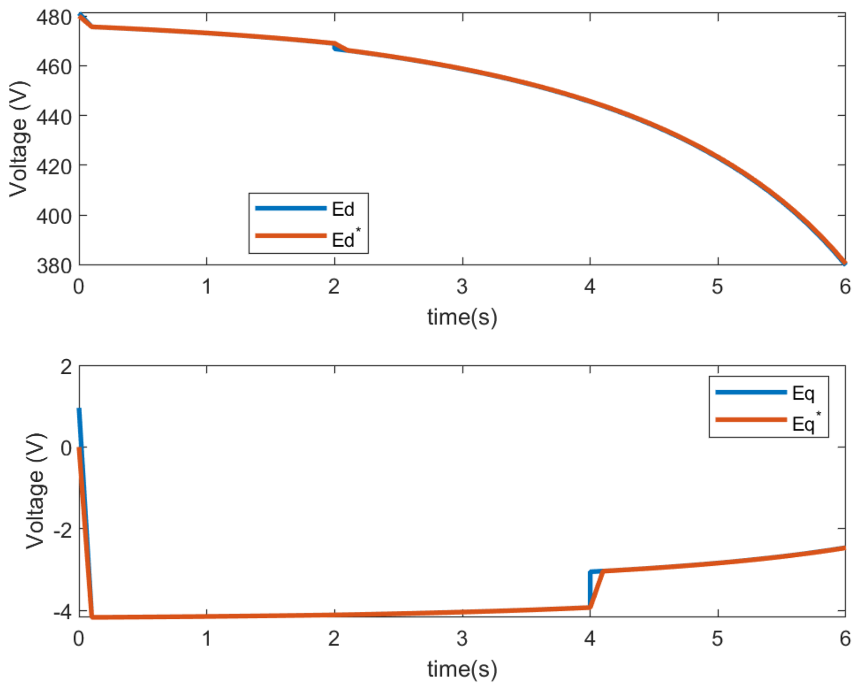


Figure 22: Voltage source model: plots for E_d, E_d^*, E_q, and E_q^* with $K_i = 10.0$ and all other gains set to 0

5.2.3 $K_p = 10.0$

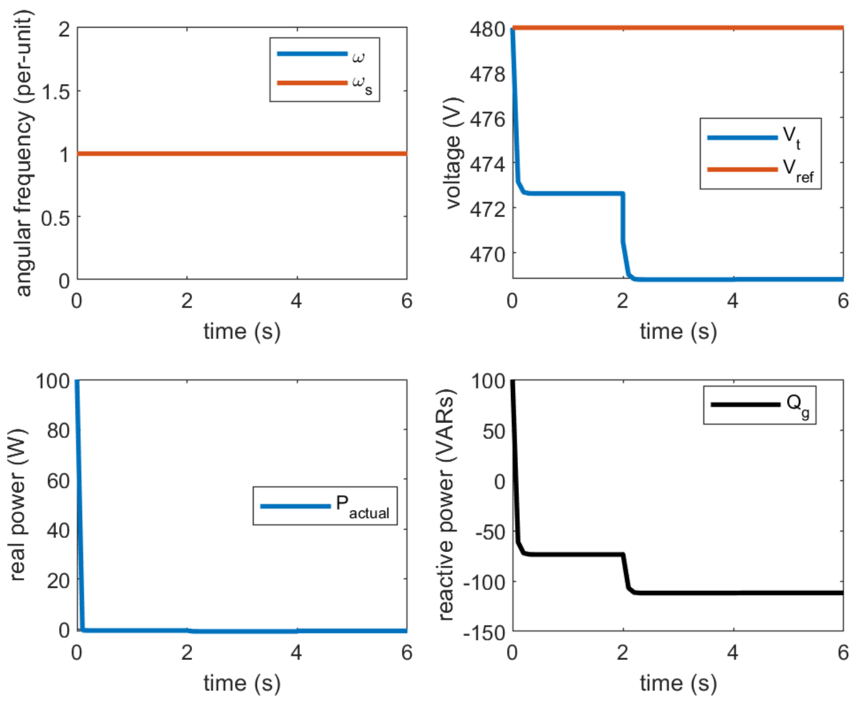


Figure 23: Voltage source model: plots for ω, V_t, P_{actual}, and Q_g with $K_p = 10.0$ and all other gains set to 0

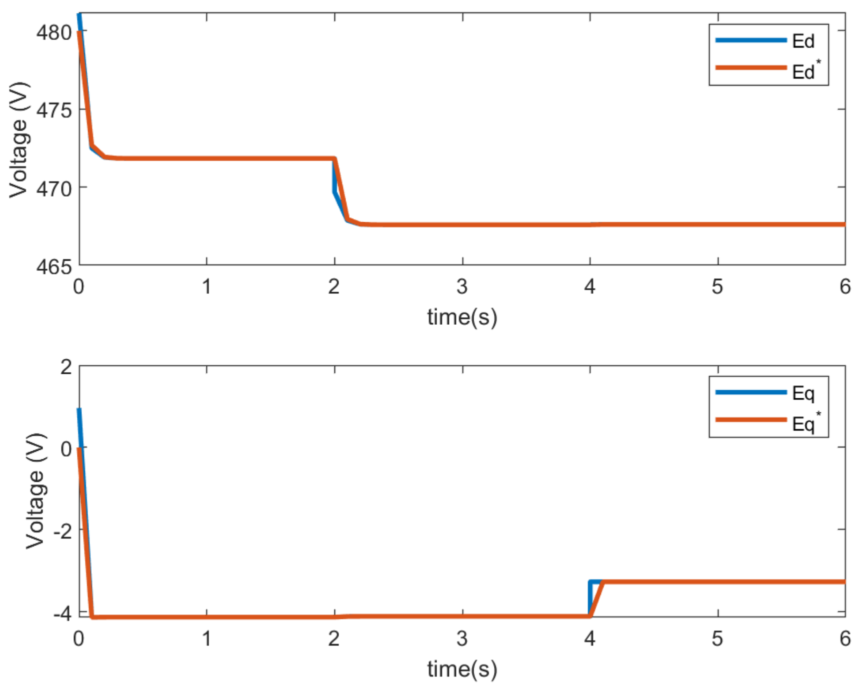


Figure 24: Voltage source model: plots for E_d, E_d^*, E_q, and E_q^* with $K_p = 10.0$ and all other gains set to 0

5.2.4 $K_{iq} = 0.001$

Note, this gain causes Q_g to blow up very easily (if K_{iq} ¿ 0.001 sometimes). This possibly could be fixed by putting the saturating blocks in place to limit the Q_{cmd}.

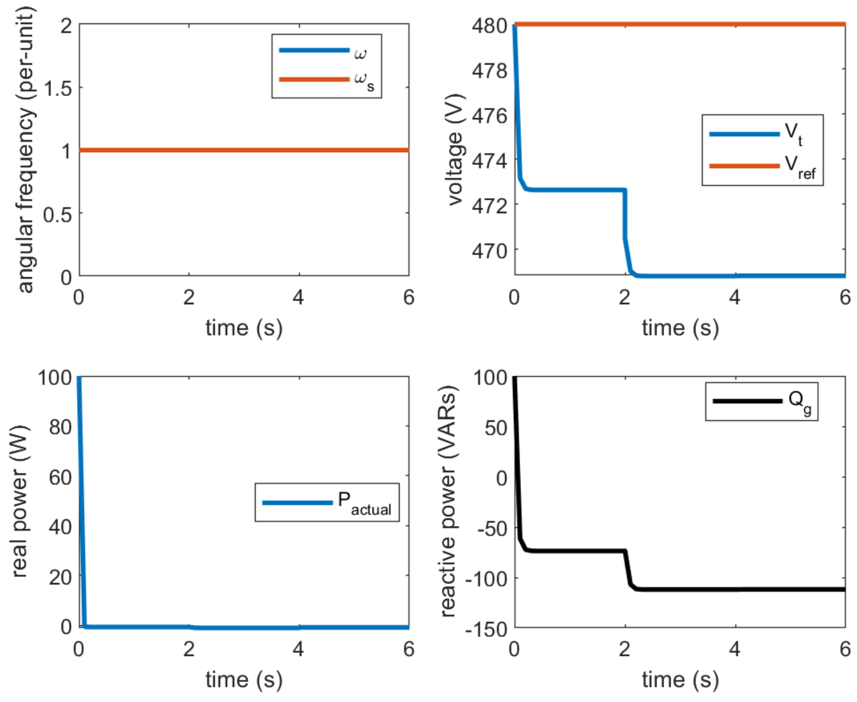


Figure 25: Voltage source model: plots for ω, V_t, P_{actual}, and Q_g with $K_{iq} = 0.001$ and all other gains set to 0

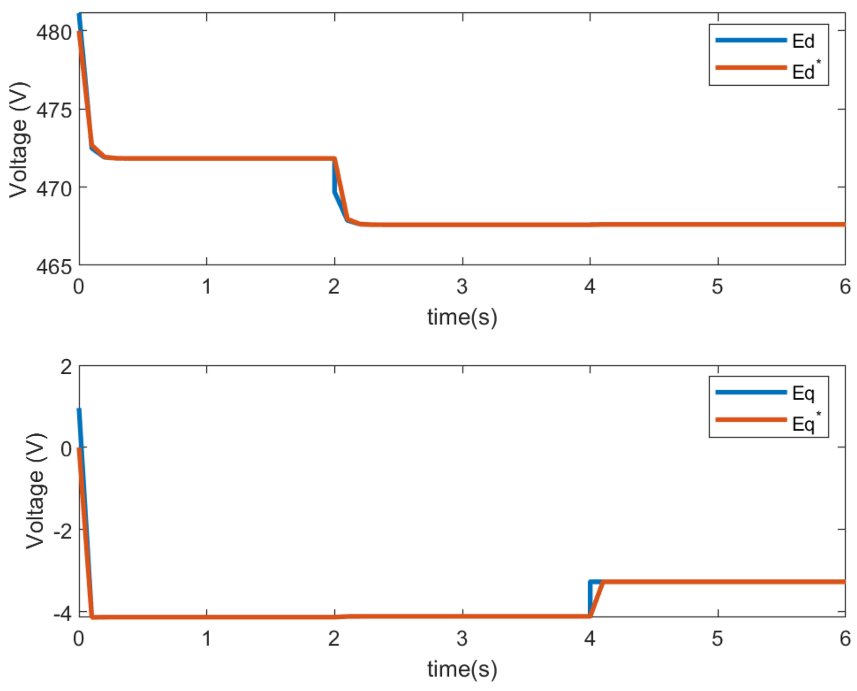


Figure 26: Voltage source model: plots for E_d, E_d^*, E_q, and E_q^* with $K_{iq} = 0.001$ and all other gains set to 0

5.2.5 Combining gains

Combining multiple gain settings (which required $K_{iq} = 0$, the following plots were obtained for $K_i = 10.0$, $K_p = 10.0$, and $K_{ip} = 10.0$.

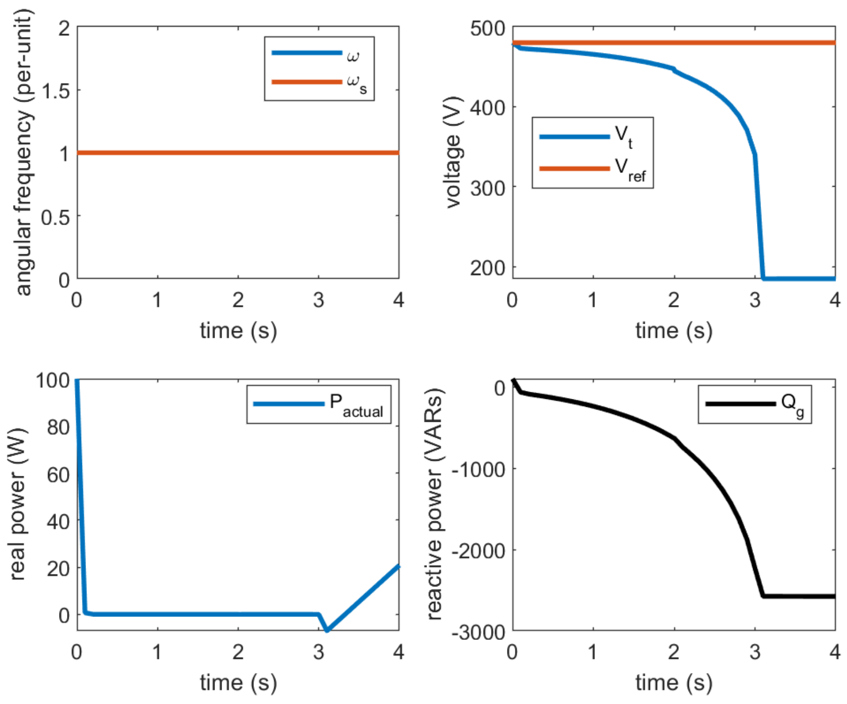


Figure 27: Voltage source model: plots for ω, V_t, P_{actual}, and Q_g with $K_p = K_i = K_{iq} = 10.0$ and $K_{iq} = 0$

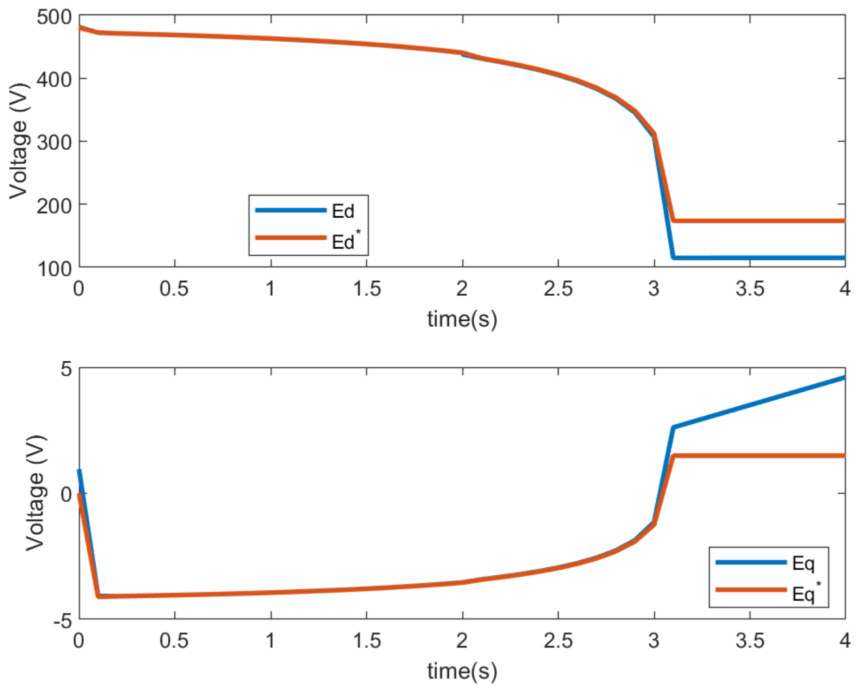


Figure 28: Voltage source model: plots for E_d, E_d^*, E_q, and E_q^* with $K_p = K_i = K_{iq} = 10.0$ and $K_{iq} = 0$

These plots, while showing that the DAEs can run for reasonably long time periods with load step changes without diverging, are not very informative about the overall converter behavior. More in-depth simulation is required to not only check that the DAEs are in fact set up right, but to fine tune the gains and time constants. Limits should also be added as described in [2].

5.3 Second-Order Synchronous Generator MATLAB Model

The following are plots taken from modeling the second-order synchronous generator models as DAEs in MATLAB. 29 shows the time series results of the full classical synchronous generator model connected to an infinite bus with a 0.5pu load on it. The time-series results for the mechanical dynamics second-order synchronous generator model and the full classical model were identical. 30 shows the same generator with a Type II governor.

These results showed the behavior we expected and encouraged us to build the two-axis model and proceed with testing of two classical generator models in parallel. The results for the two generators in parallel show us how the generators respond to a load change. We found that the generators could accept load changes when the load was still balanced but if the load wasn't exactly 2 per unit, the generator δ would run away. This was less evident in small signal changes but δ would still increase or decrease. 31 shows the results of a balanced change in load (the load on each bus changes but the sum of the load on the system is still 2 pu). 32 shows the same load change but with Type II governors on the generators. A line fault was simulated between two generators with Type II governors. The generators performed as expected with the fault 33.

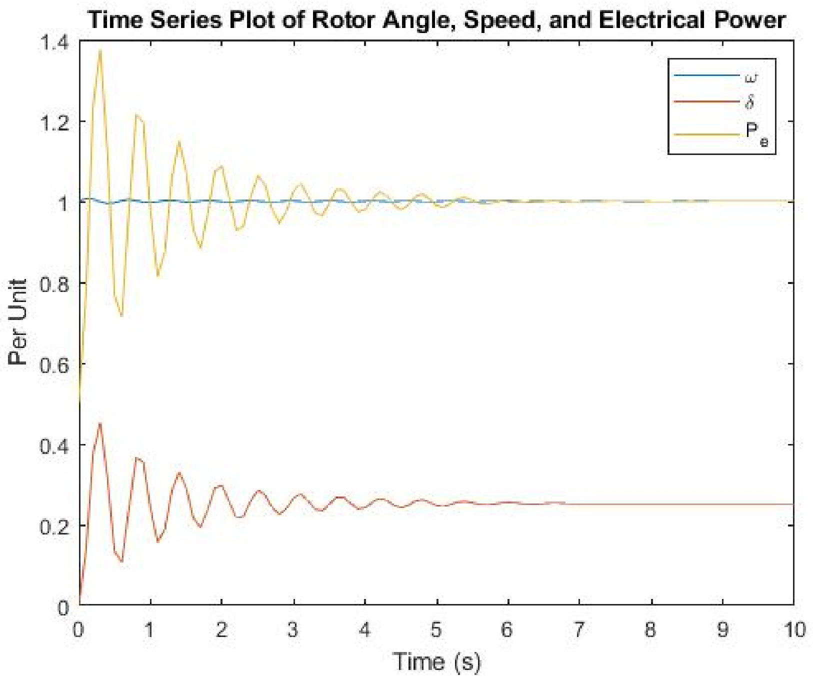


Figure 29: Single synchronous generator connected to an infinite bus

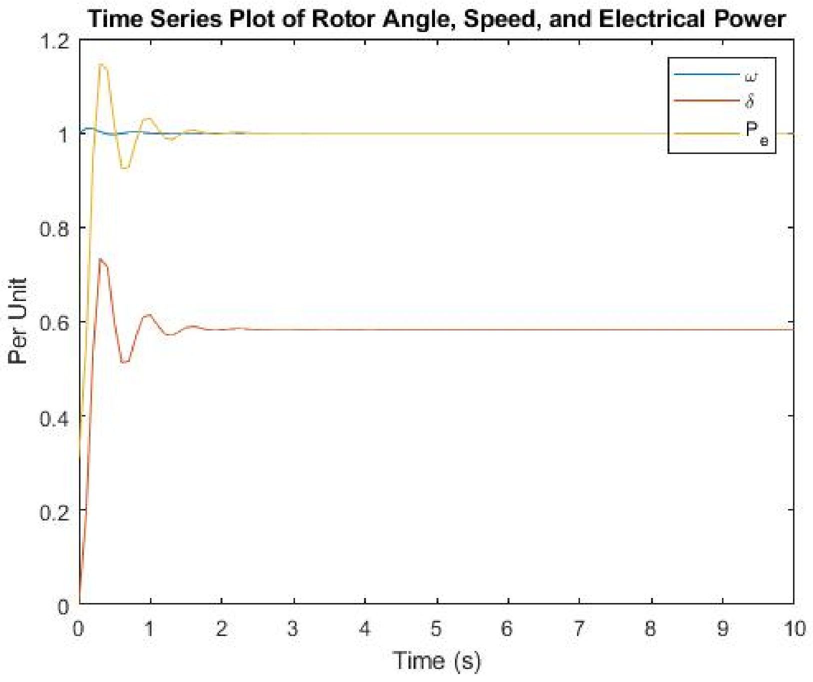


Figure 30: Single synchronous generator with a Type II governor connected to an infinite bus

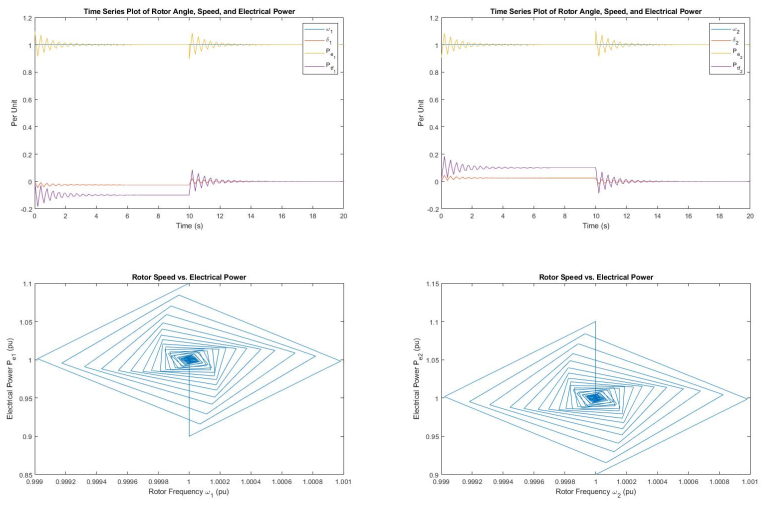


Figure 31: Two synchronous generators in parallel with a balanced change in load

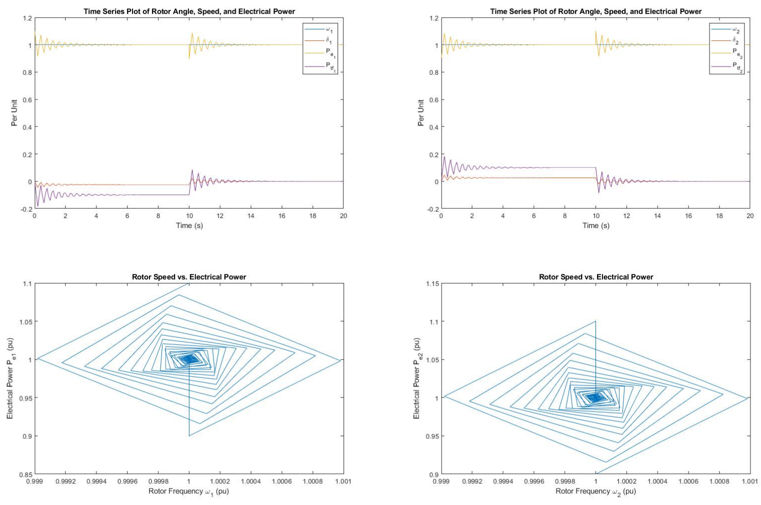


Figure 32: Two synchronous generators with governors in parallel with a balanced change in load

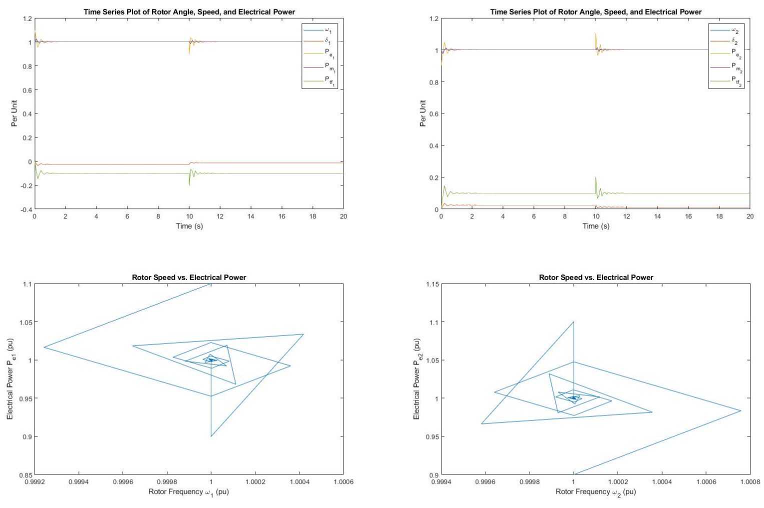


Figure 33: Two synchronous generators with governors in parallel with a line fault

6 Lessons Learned

There were many things learned by all group members throughout this project. Many gaps in knowledge and assumed understanding of principles were uncovered as we all started to build the DAE models. However, the following are useful takeaways we wish to pass on to future teams:

- Making sure the order of states in the DAE array exactly matched the order of states in the Mass Matrix (which should also match the type of equation - algebraic or differential), and in initial condition arrays
- Performing parameter steps by simulating over a certain time period, changing the parameter, and then using the ending state of the first time period as the new initial condition for the next period was effective
- Gain parameters set in the reference papers often assumed there were limits on various states. Without these limits, those gain parameters were often not appropriate
- Initial conditions could have a big impact on convergence or steady state values
- Breaking components of the overall control blocks into individual m-files made the code more modular and easier to read and edit
- Give more time for every aspect of the coding than you expect
- Start with models from references that have complete examples or code

7 Team Member Contributions

As a team we decided to divide up the modeling duties by device to different members. Jaimie Swartz built the boundary current model and led the organizing of many of the team meetings. Rose Abramson built the Thévenin Voltage Source Model and led the writing of the PowerPoint presentations, Jason MacDonald Phillippe Phanivong both worked on the synchronous generator models with Jason focusing on the two-axis model and Phillippe working on the two bus network model. Jason curated the GitHub repo for all of the team code and created the framework for this document in LaTeXwhile Phillippe organized final draft. All members contributed to the content.

References

[1] D. Ramasubramanian, Z. Yu, R. Ayyanar, V. Vittal, and J. Undrill, "Converter model for representing converter interfaced generation in large scale grid simulations," *IEEE Transactions on Power Systems*, vol. 32, no. 1, pp. 765–773, Jan. 2017, ISSN: 0885-8950. DOI: 10.1109/TPWRS.2016.2551223.

[2] D. Ramasubramanian, "Impact of converter interfaced generation and load on grid performance," PhD thesis, Arizona State University, 2017.

[3] U. Markovic, J. Vorwerk, P. Aristidou, and G. Hug, "Stability analysis of converter control modes in low-inertia power systems," in *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, Oct. 2018, pp. 1–6. DOI: 10.1109/ISGTEurope.2018.8571583.

[4] K. Clark, N. W. Miller, and R. Walling, "Modeling of GE solar photovoltaic plants for grid studies," GE Energy, General Electric International, Inc., One River Road, Schenectady, NY 12345, Tech. Rep., Apr. 2010.

[5] J. Machowski, J. Bialek, and J. Bumby, *Power system dynamics: stability and control.* John Wiley & Sons, 2011.

[6] F. Milano, *Power System Modelling and Scripting*, 1st. Springer Publishing Company, Incorporated, 2010, ISBN: 9783642136689.

[7] P. M. Anderson and A. A. Fouad, *Power system control and stability.* Ser. IEEE Press power engineering series. Piscataway, N.J. : IEEE Press : Wiley-Interscience, ©2003, 2003, ISBN: 9780470545577.