Digital Plasma Control System and Alcasim Simulation Code for Alcator C-Mod

by

Marco Ferrara

Laurea in Ingegneria Elettronica, Università di L'Aquila (2001)

Submitted to the Department of Nuclear Science and Engineering
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2005

Signature of Author ....
Department of Nuclear Science and Engineering
August 19, 2005

Certified by ....
Ian H. Hutchinson
Chair and Professor, Department of Nuclear Science and Engineering
Thesis Supervisor

Certified by ....
Stephen M. Wolfe
Principal Research Scientist, Alcator Project
Thesis Supervisor

Accepted by ....

Contents

List of Figures

Acknowledgements

My years as a student at the MIT Plasma Science and Fusion Center have been exciting. Fusion research is an outstanding example of how interesting physics and challenging engineering effectively meet. The collaboration with the people who supervised my work, Prof. I.H. Hutchinson, Dr. S.M. Wolfe and Mr. J.A. Stillerman, has been highly rewarding. Working with them has been always stimulating and a constant opportunity to learn.

The entire staff of people of the Plasma Science and Fusion Center has been equally supportive and important for the successful conclusion of my work.

My brother Luca and my parents Adele and Piero have been, as always, essential to my accomplishments. My friends Dominic, Thekla, David, Daniel, Sue, Jason, Seton, Luisa, Jenny and Roberto have made these years human and enriching.

To all of them I extend my grateful acknowledgement.

The present work was sponsored under the contract D.o.E. Coop. Agreement DE-FC02-99ER54512

Part I

DPCS Digital Plasma Control System

Chapter 1

Introduction

Alcator C-Mod is a compact tokamak at the MIT Plasma Science and Fusion Center. It is illustrated in figure 1-1. It comprises a toroidal field magnet (TF in the figure), whose 20 legs

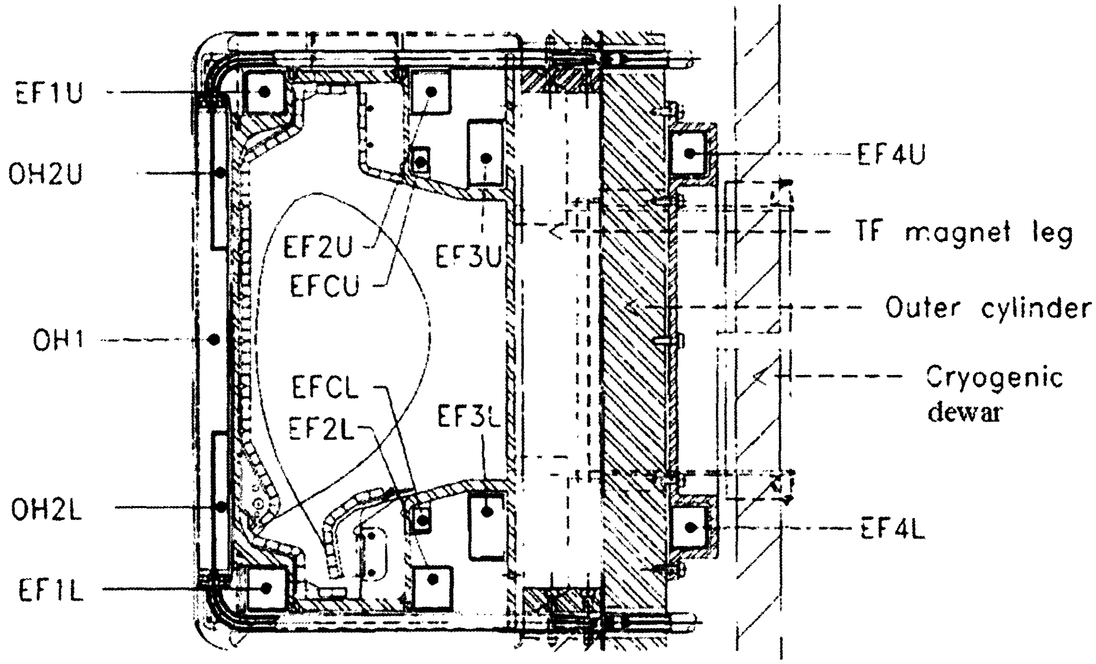


Figure 1-1: Illustration of the cross section of Alcator C-Mod

generate toroidal fields up to $8T$, 3 central coils, winding around the core of the machine (OH1, OH2U, OH2L) and 10 poloidal field coils (EF1U, EF1L, EF2U, EF2L, EF3U, EF3L, EF4U, EF4L, EFCU, EFCL). It is a feature of the C-Mod coil-set that essentially all the coils are used for both shape and position control, as well as inductive drive. This is in contrast to some designs, such as DIII-D, in which there is a "de-coupled" inductive drive coil that does not affect the shape. Part of the complication of the C-Mod control system in fact derives from this fact, that there is no one-to-one correspondence between parameters to be controlled and single (or small subsets) of coils to use as actuators. The plasma is formed inside the vacuum vessel and a current up to $2MA$ can be inductively driven by sweeping the currents in the active coils (mainly OH1 and the OH2 coils). The plasma is ohmically heated and the use of additional radio frequency heating allows the plasma to reach a core temperature of about $5keV$. In a magnetic confinement fusion device the exact knowledge of the topology of the magnetic field inside the vessel is essential for studying and controlling the plasma. Detailed information is available from the magnetic sensors. On Alcator C-Mod there are 21 full-flux coils, 6 partial-flux coils, and 118 poloidal pick-up coils. The full-flux coils run toroidally around the machine and measure the magnetic flux coupled with them. Similarly, the partial-flux coils measure the coupled magnetic flux, but they are located nearby ports, so they are not toroidally continuous. Their signals are combined in order to produce the total flux at locations of interest. The poloidal pick-up coils measure the poloidal component of the magnetic field at specific locations around the vacuum vessel. There are four sets of 26 coils, placed at different toroidal locations, plus 14 additional coils. Only one set of poloidal coils is used for plasma control, while the others are for post-processing and equilibrium reconstruction. The other magnetics diagnostics used for control are the plasma rogowski and the toroidal field measurement. Figure 1-2 illustrates the location of the magnetic diagnostics in Alcator C-Mod. The signals from the magnetic diagnostics can be linearly combined in order to obtain the magnetic field at specific locations and information on the shape and position of the plasma. Some of these quantities are chosen as the observables of the system and a number of them are controlled using a feedback loop. Figure 1-3 shows a set of fluxes evaluated at specific locations and used in Alcator C-Mod to calculate the relevant observables. In a standard C-Mod plasma discharge two phases are distinguished, the start-up phase, when the plasma is formed and its current ramped-up, and the flat-top phase, when the

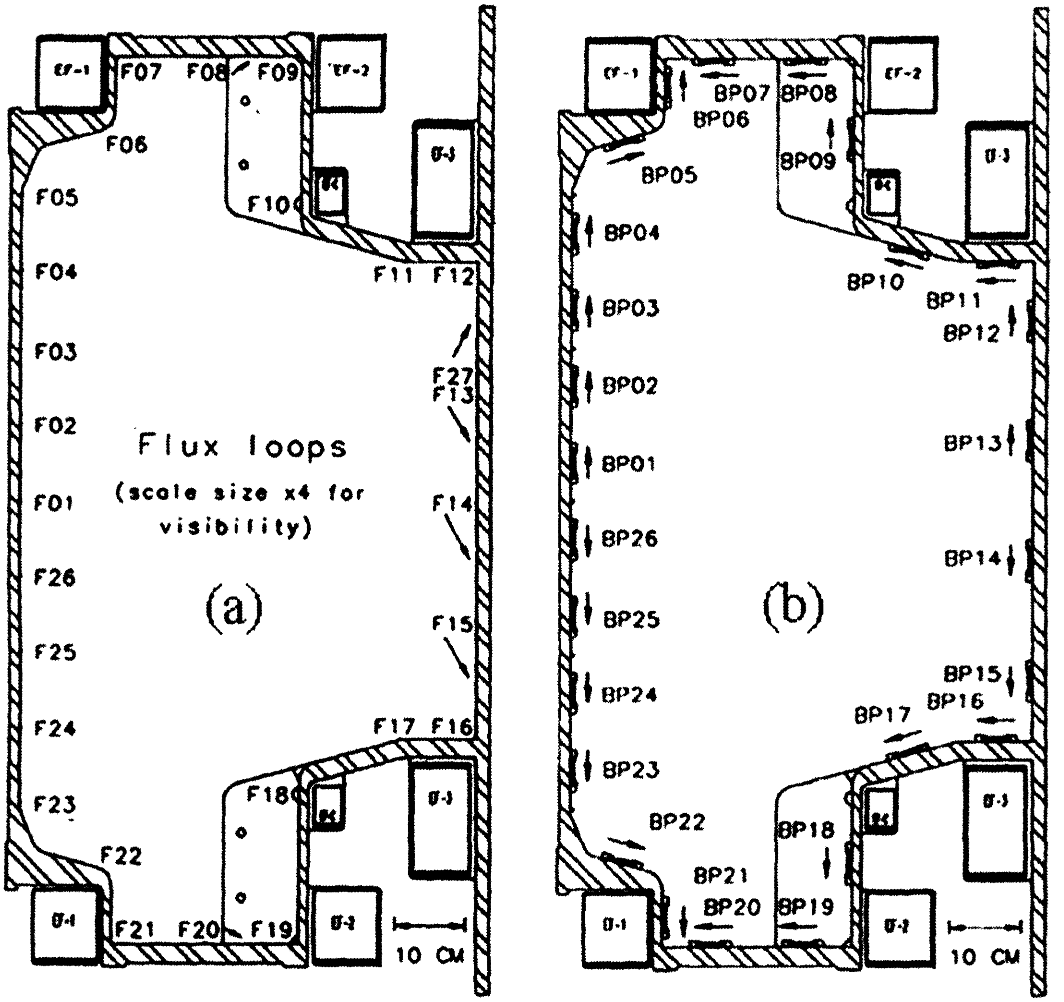
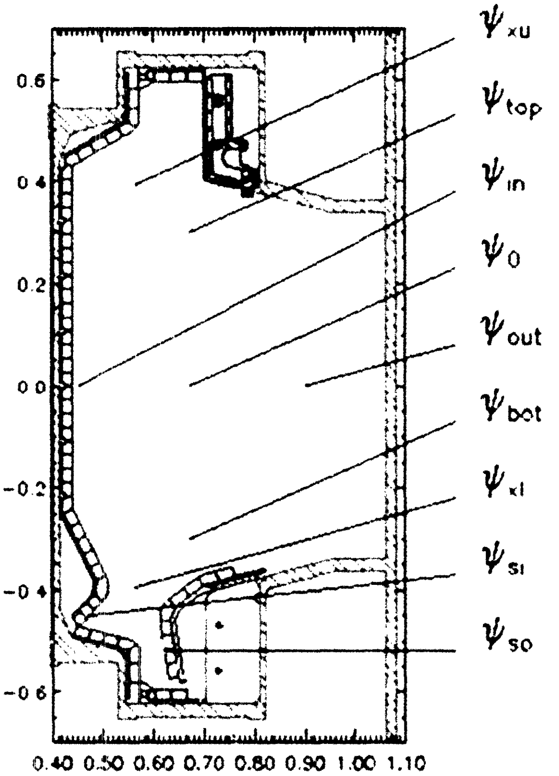


Figure 1-2: Cross section of Alcator C-Mod illustrating the position of the full-flux coils (panel a), the partial-flux coils (F08, F13, F14, F15, F20, F27) and the polidal pick-up coils (panel b)



Shape Parameters

$R_g \propto (\psi_{out} - \psi_{in})/I_p$

$Z_g \propto (\psi_{top} - \psi_{bot})/I_p$

$C_i \propto (\psi_{in} - \psi_{xl})/I_p$

$\rho_s \propto (\psi_{so} - \psi_{xl})/I_p$

$R_{xj} \propto (\alpha \frac{\partial \psi}{\partial Z}_{xj} - \beta \frac{\partial \psi}{\partial R}_{xj})/I_p$

$Z_{xj} \propto (\delta \frac{\partial \psi}{\partial Z}_{xj} - \gamma \frac{\partial \psi}{\partial R}_{xj})/I_p$

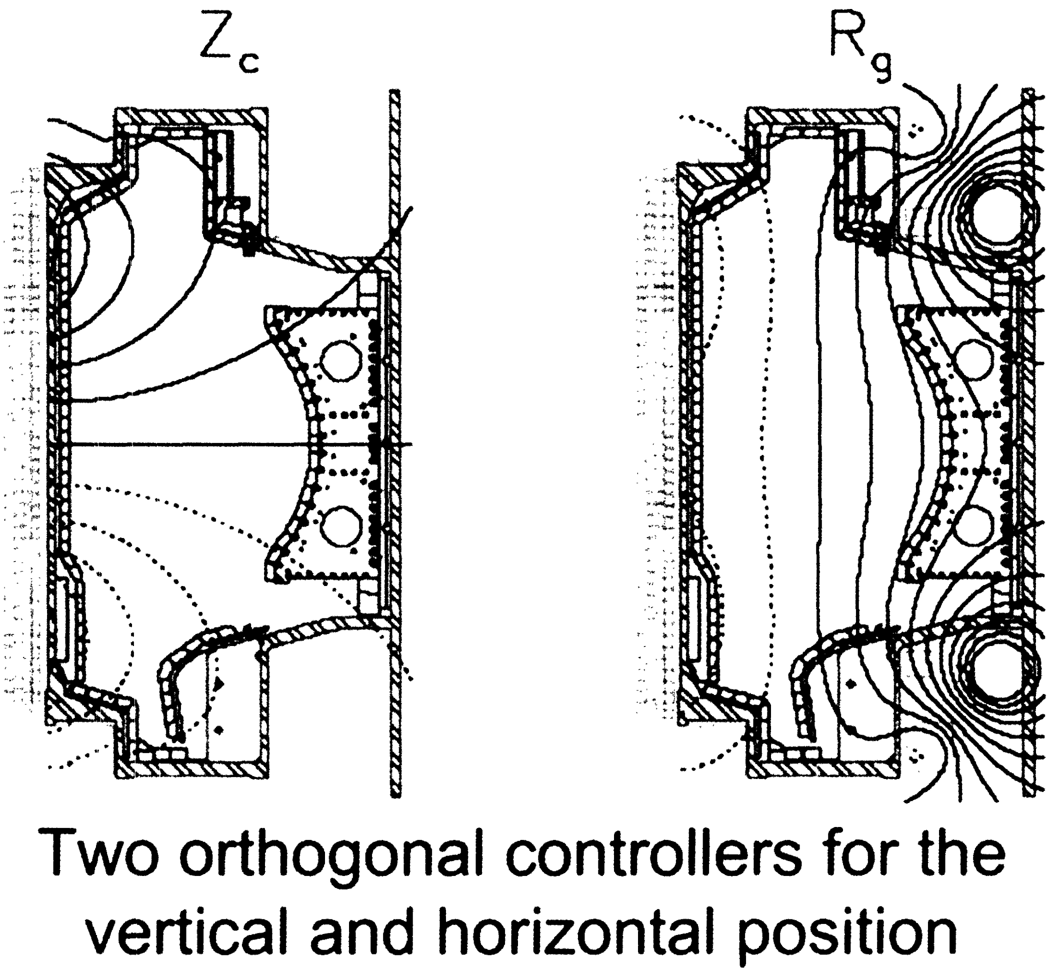
A possible set of shape parameters for the plasma

Figure 1-3: The magnetic fluxes at the locations pointed in the figure can be calculated with linear combinations of the responses of the magnetic sensors. These fluxes can then be combined to obtain information on the plasma position and shape. From [1]

plasma current is brought to a stationary value and further experiments are conducted. The switch time is usually $0.1s$. Different observables are used for these two phases. In particular, in a standard start-up phase of Alcator C-Mod, the only quantities under control are the radial component of the magnetic field at the plasma nominal centroid and the currents in the active coils. After the plasma has been formed, the quantities under control become the plasma radial and vertical position and the inner wall gap scaled by the plasma current (RCUR, ZCUR and CLEARIN respectively), the radial and vertical position of the lower and upper X-points (RXL, ZXL, RXU, ZXU), the plasma density (nl_04), the EF4 current, the plasma current (IP) and the average of the EF2 currents, when the EF4 current (more precisely, the voltage) is used as the corresponding actuator (EF2_BY_EF4). While the upper and lower x-point positions are often used as the observables, there are other options, including the strike point locations on the outer and inner divertor plates (STRKPSI and STRKIN) and the distance between primary and secondary separatrices (SSEP).

There is an important distinction between times before and after $0s$ in the first segment of the discharge. Before $0s$ the coil currents are controlled directly, and the only "field" quantity being controlled is BR0, the radial field. No positional or plasma quantities are under control before $0s$, because there is no plasma then. Shortly after $0s$, the gains on the ZCUR and RCUR are turned on, and many of the gains on the poloidal coils currents are reduced or turned off, in particular those on the OH coils. From $0s$ to $0.1s$ the feedback on RCUR and ZCUR is used to control the plasma position. The plasma current is normally not controlled by feedback during this time. The observable quantities involving gaps and positions are expressed as products with the plasma current because the products can be simply calculated as linear combinations of the magnetic signals.

The number of observables that can be ultimately controlled with a feedback loop is related to the number of actuators which can be independently operated. For the particular case of the shape and position of the plasma, the controllers will be the voltages applied to the poloidal coils or a combination of these voltages. The optimal synthesis of orthogonal controllers for a certain set of observables is a problem which admits purely formal solutions with the tools of control engineering, but an heuristic and physically-conscious approach can be more effective, even if leading to only a non-orthogonal set of controllers [1]. Figure 1-4 shows an example of combinations of coil currents that can be used to control the vertical and radial position of the plasma.



Two orthogonal controllers for the vertical and horizontal position

Figure 1-4: An example of two sets of coil currents which form the controllers for the vertical and radial position of the plasma. From [1]

Once the set of observables is defined and the target waveforms for these observables are drawn, the errors between the actual values and the desired values are computed and processed by the control algorithms to activate the corresponding controllers. Alcator C-Mod is currently using a linear control scheme, as shown in figure 1-5. The signals from the diagnostics, that is the magnetic signals, the currents in the coils, the density, the plasma current and other

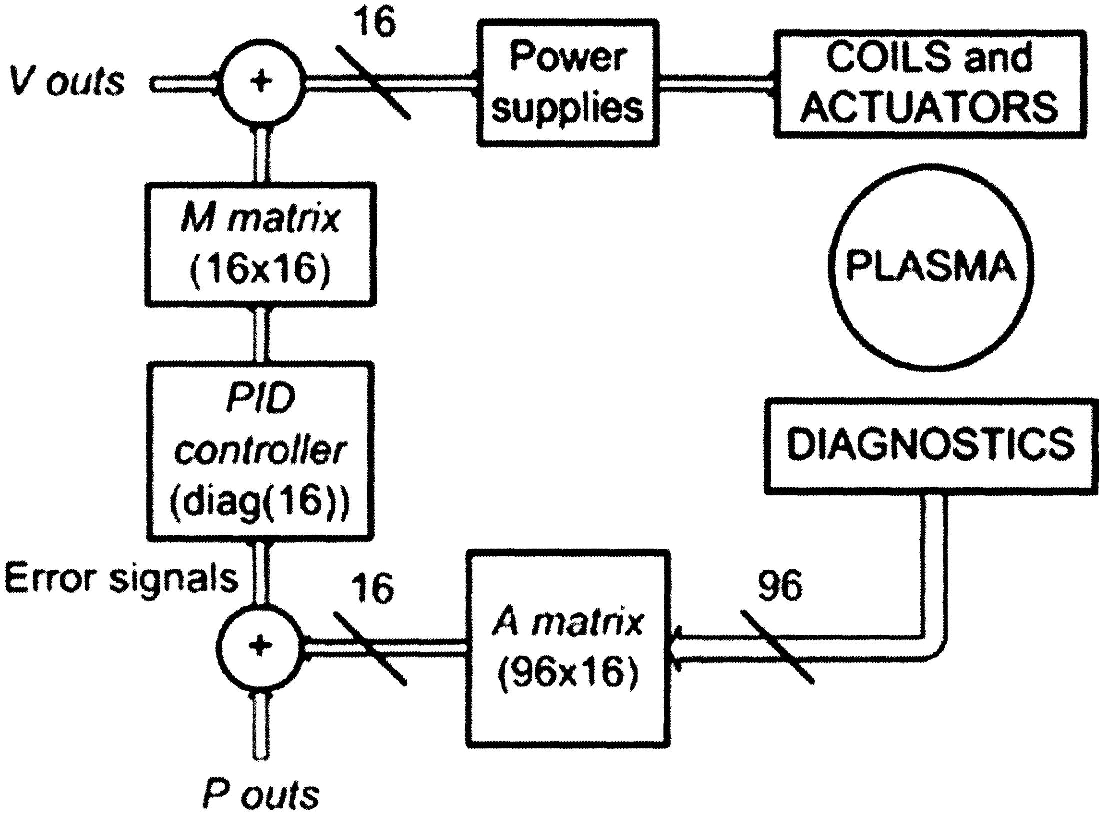


Figure 1-5: Simplified scheme of the linear control of Alcator C-Mod

relevant parameters, are input to the observer *A matrix*. The observables are computed as a linear combination of the inputs, then they are subtracted from the target waveforms *P outs*. Some of the target waveforms must be normalized by the plasma current in order to be consistent with the corresponding observables. The error signals are input to the PID controller. This is currently a set of 16 independent Proportional-Integral-Derivative controllers acting on each wire. The results of the computations are then input to the *M matrix*, which controls the power supplies of the poloidal field coils, and other relevant actuators, for example the gas valves, which feed-back on the plasma density. The output of the M matrix is corrected by adding the *V outs* feed-forward waveforms, which provide also the basic control when the loop is open. Detailed description of the plasma control strategies used in Alcator C-Mod have been previously published [2], [3].

In general, active control is essential for the vertical position of elongated plasmas. In fact, these configurations are obtained with the magnetic field "pushing" or "pulling" on the plasma, the field concavity is directed toward the outboard of the vessel and the zero of the radial field is a position of unstable equilibrium. Even a simple physical picture proves this assertion: if the plasma centroid is slightly moved from the zero of the radial field, the plasma will experience a force in the same direction of the displacement, instead of a restoring force. However, if the plasma is close enough to the wall, the passive currents induced in the metal structures will slow the instability down to their resistive decay time: this regime is called resistively unstable and provides a margin for active control, which would otherwise be impossible in the regime of ideal instability, where the vertical run is damped only by the natural inertia of the plasma, with typical times of $< 100\mu s$. In the case of a resistively unstable plasma, this time becomes a few ms. The fast poloidal coils EFC are used in Alcator C-Mod to compensate the vertical instability of the plasma: they are connected in anti-series and are fed by a chopper power supply. The small signal bandwidth of the supply is $1500Hz$, but falls to about $300Hz$ at full power.

The control scheme of Alcator C-Mod was originally implemented using a hybrid digital-analog system *Hybrid* [3]. During the 2003-2004 campaign and in the summer 2004 the new *Digital Plasma Control System* (*DPCS*) was implemented and successfully tested off-line. Digital control systems have been implemented on many tokamak machines, for example DIII-D, JET, JT-60U, ASDEX-U, TCV, because of their flexibility and reliability. Part of the interest in digital control comes from the development of the next generation tokamaks, among which the International Thermonuclear Experimental Reactor (ITER), whose size and power handling will require considerable safety measures and optimization procedures [4], [5]. Digital systems also allow to run advanced control strategies to stabilize high performance plasmas [6], [7], [8], [9].

The initial goals of DPCS were the consistency with the control signals produced by Hybrid, the compatibility with the *MDSplus* data structures created for previous shots, so that these shots could be reloaded and run with the new system, and the compatibility with the general control software *PCS* (*Plasma Control System*) and its graphical user interfaces. MDSplus is the database structure standard for fusion experiments [10].

At the start of the 2005 campaign DPCS was used to control the real machine and has been in operation since then. In fact, some advanced features have already been implemented or tested, such as the allowance for the loss of input samples, the real-time compensation of the input offsets and the reduction of the cycle rate for extra headroom for computation. The following chapters describe the architecture of the previous control system Hybrid and the digital control system DPCS. The off-line debugging of DPCS and the first operation of DPCS on Alcator C-Mod are discussed in detail.

Chapter 2

Implementation of Hybrid

Figure 2-1 illustrates both Hybrid and DPCS. Hybrid hardware is mainly composed of arrays of DACs used as programmable analog multipliers: the input signal of each multiplier is applied to

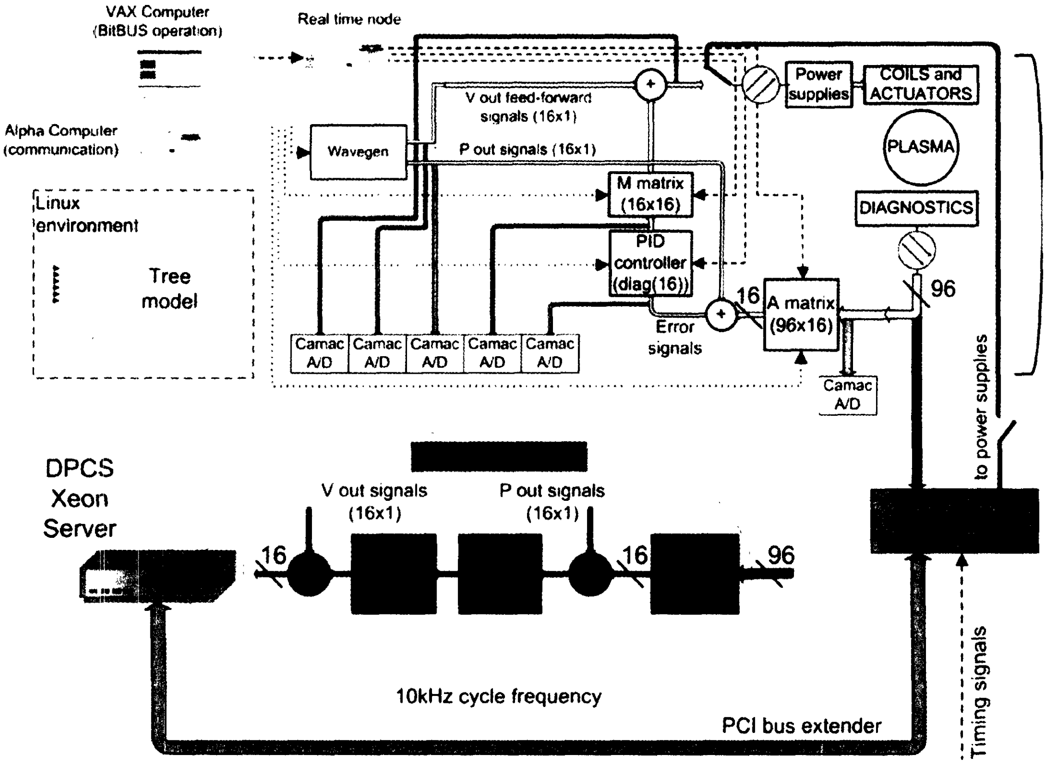


Figure 2-1: Complete schematic of Hybrid and the current implementation of DPCS

the reference pin and the output is the input multiplied by the digital word stored in the DAC. Some drawbacks exist with this implementation, for example the leakage between input and output at zero gain, or the presence of a finite output for zero input. The overall system has performed satisfactorily for its 14 years of operation. Among the advantages are a large analog bandwidth (about $20kHz$) and a low noise background. The bandwidth is largely exceeding the speed of the power supplies (always below $1kHz$, except for the fast *EFC* coils) and the time constant of the coils, with their large inductances and small resistances.

Hybrid is real-time, meaning that matrices and references are switched during the plasma discharge, but it is not adaptive, which means that the order of the matrices and references are decided and set before the shot, according to the particular plasma configuration under investigation. In theory, Hybrid could operate adaptively, but this feature has never been implemented.

The structure which contains the information to run a shot and stores the data from the shot is the *MDSplus tree.* MDSplus trees are used for control, data acquisition and analysis of essentially all aspects of the C-Mod experiment [10]. The matrices and the parameters to generate the *P outs* and *V outs* are pre-loaded into the multipliers and the wave-function generator through a *BitBUS* fieldbus. The fieldbus is managed by a *VAX computer* and the communication between the *Linux environment* and the VAX is operated by an *Alpha computer.* When a shot is run, a trigger signal is sent to the *timing node*, which takes care of sequencing the various phases of a plasma discharge by issuing the switching of the references and the matrices. The different instances of A, M and PID matrices are thus applied at different times during a plasma discharge. The system was originally developed by the TCV group as a clone of the hybrid control operating on TCV [11]. The data from a discharge are digitized with a sampling rate of $500Hz$ and stored into the corresponding tree.

Chapter 3

Implementation of DPCS

The decision to implement a new digital control system is consistent with the work done on big devices, such as JET, JT-60U, DIII-D, but also on smaller machines: a significant example is the TCV in Lausanne [12], whose earlier hybrid control system was a twin of C-Mod Hybrid.

The essential hardware of DPCS is extremely compact. It comprises two CPCI cards, a CPCI to PCI bus extender and a XEON Server. Each I/O card has 64 16-bit inputs and 16 analog outputs, totalling 128 inputs and 32 outputs. Cards with more channels or additional cards can be added should the need arise. The original hybrid control computer had a total of 96 inputs and 16 outputs. The server is a standard Intel® Xeon™ 3.20GHz with 2 Gigabytes of memory. It can easily be replaced or upgraded since it is completely standard. The computer is running the RedHat Linux distribution, booted diskless over NFS. The system boots using dhcp and PXE. A local disk is present for paging and swapping when not in real-time mode. The PCI/CPCI extender card transparently makes the CPCI cards appear as PCI peripherals on the host computer. The one in use is a 32 bit 33 MHz card from SBS Technologies, 64 bit 66 MHz cards are available should this prove to be a bottleneck in the future.

The DPCS software consists of many IDL routines grouped in the file *DPCS_startup.pro* [13]. The communication of DPCS with the main system supervising plasma discharges is illustrated in figure 3-1.

The software operation is summarized in the following:

1. A shell script called *dpcs* runs every morning on the pcdaqdpcs1 computer (and when the

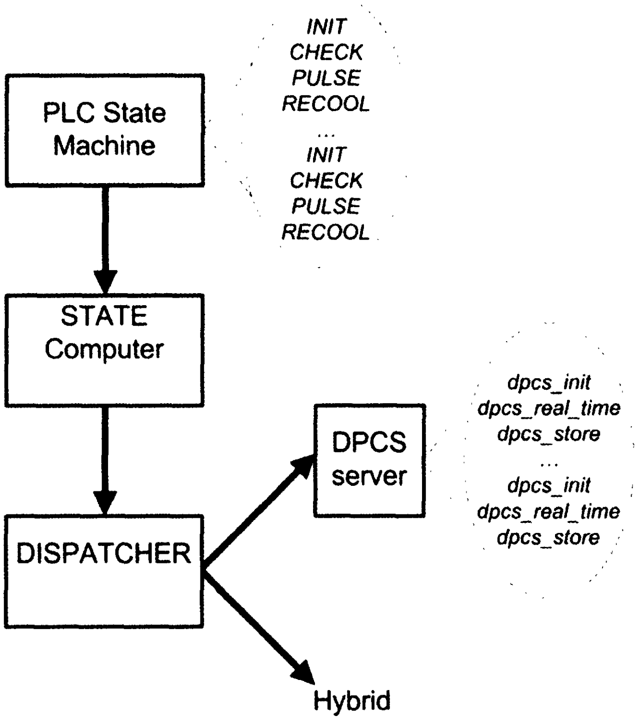


Figure 3-1: Schematic of the communication between the real time control routines of DPCS and the main system which supervises the plasma discharges

computer is restarted). This script initializes the digitizers, starts up an MDSplus server process, and starts an IDL process running *dpcs.pro.*

2. *dpcs.pro* compiles *dpcs_startup.pro*, the file that contains the main IDL routines. It then invokes the *dpcs_startup* procedure, which locks down memory, initializes the low-latency interface to the hardware, and exercises the interrupt disabling. On return from *dpcs_startup* the dpcs routine sets up the fifo buffer through which communication between the server and the IDL process is carried out.

   Steps (1) and (2) are normally done once a day. The remaining steps happen every shot.

3. During INIT (actually during the transition from RECOOL to INIT), the dispatcher sends the ACTION \HYBRID::TOP.HARDWARE.DPCS:DPCS:INIT_ACTION to the MdsPlus server running on pcdaqdpcs1. The server carries out the DPCS__INIT method which instructs the IDL process to execute the routine *dpcs_init.* This routine sets up the parameters used by the *dpcs_real_time* and *dpcs_store* routines, based on information read from the MdsPlus tree. The parameters for the real-time calculations, including matrices, target waveforms, etc., are all passed using IDL pointers to heap variables. The logical steps done in *dpcs_init* include:

   a Read from the tree the names of software packages to be run. This includes all calculations to be done other than the main routine which emulates the hybrid.

   b Clean up after previous cycles which may not have terminated correctly. This avoids potential memory leaks and dangling heap variables.

   c Read from the tree the parameters of the main calculation, i.e. the cycle time, the shape of the matrices that emulate the hybrid, and then the matrices and target waveforms themselves.

   d Call the initialization methods for the auxiliary routines, which set up all the parameters for those calculations. Also, compile all the real-time executables for those calculations.

   e Clean up temporary variables and pointers used during the init process for both the auxiliary and main calculations.

f Initialize array variables to hold the results of the real-time calculations at each time-step; these are eventually stored in the tree by the dpcs_store routine.

4. During CHECK (actually during the INIT to CHECK transition) the dispatcher tells the MDSplus server on pcdaqdpcs1 to execute the DPCS RT_ACTION method. This method causes the server to invoke the *dpcs_real_time* procedure in the IDL process. This procedure executes a dummy pass through the real-time calculation, which causes all the necessary instructions and variables to be faulted into memory. It then disables the interrupts and waits for a trigger before transferring data from the digitizers. The trigger and clock are fired in the PULSE state using CAMAC modules. The real-time loop is synchronized to the digitizer clock. If no trigger is received after a pre-set number of polling cycles the real-time code exits with a timeout error and re-enables the interrupts. Otherwise, the code executes the loop (write the output from a previous iteration, read the new input and perform the computation) until the termination condition is reached, at which the interrupts are turned back on and the real time routine is exited. For the detailed comment of the real time routine see also section 3.2.

5. During RECOOL (actually during the PULSE to RECOOL transition) the dispatcher tells the MDSplus server to execute the DPCS_STORE action and the IDL process runs the *dpcs_store* routine. This routine stores the digitized input data and the results of the real-time calculation to the MDSPlus tree. The routine also calls individual STORE routines (methods) associated with any auxiliary calculations carried out. Finally, the heap memory associated with the calculation parameters is freed.

6. Also during RECOOL the dispatcher tells the server to execute the RT_CHECK action which examines the results stored by the store action for discrepancies and optionally broadcasts a message if any are found.

3.1 Benchmarking and debugging DPCS

The initial requirements on DPCS were that it perfectly emulate Hybrid and reproduce the Hybrid signals during a plasma discharge. During the 2003-2004 campaign the signals digitized from Hybrid were compared with the corresponding signals from DPCS, in the case of successful plasma shots, power supply test shots and fizzle shots. Many simulators have been implemented in IDL to investigate particular aspects and separate the various contributions of the discrepancies between Hybrid and DPCS. At the same time, many MDSplus *dwscopes* have been designed to allow rapid estimation of those discrepancies. In our analysis a time-varying discrepancy of a few percent of the time-varying component of the corresponding Hybrid (or DPCS) signal was considered acceptable. Also, the analog offsets of Hybrid were subtracted from the Hybrid signals before comparing them with the results of DPCS[1]. The main results are summarized in the following and refer to an early implementation of the DPCS hardware comprising three 32 channel input digitizers and a Pentium 4 PC running the IDL code. No output cards were present and the output routine was not implemented. Despite the different configuration and performance of the hardware, this system was perfectly adequate to test the main features of the DPCS software[2].

- A systematic time delay of $1ms$ was found between the Hybrid and DPCS A_in, that is the digitized versions of the input signals. This time delay is exactly half of the period of the CAMAC digitizers of Hybrid and is due to the fact that Hybrid was sampling on the falling edge of its clock, while the time stamp was taken on the rising edge. This detail is in itself insignificant, but must be compensated when the signals from Hybrid and DPCS are subtracted to evaluate their discrepancies.

- The P_outs of DPCS and Hybrid matched very well, as shown in figure 3-2. An important error was found in the algorithm which interpolated the DPCS references from the minimal data stored in the tree. The references are programmed in the PCS graphical user interfaces by adding points on x-y displays. These points are then interpolated in the init phase, before the discharge begins, to produce the continuous waveforms which

[1] In the present chapter we use the convention of calling the outputs of the A matrix A_outs, the reference signals P_outs, the sum of A_outs and P_outs *error signals* (or, simply, *errors*), the outputs of the PID matrix PID_outs, the outputs of the M matrix M_outs and the feed-forward signals V_outs. The inputs of the A matrix are called A_ins. The differences between PCS and DPCS signals are referred to as *discrepancies* and they usually have an *offset* and a *time-varying component*. Our conventions must not create confusion with the experimental data: in the tree A_out is used to indicate the error signals ($A\_outs + P\_outs$) and M_out is used to indicate ($M\_outs + V\_outs$).

[2] The name *Lowbrid* can be found frequently in the graphics and it stands for DPCS.

control the machine. However, during a plasma discharge, the observables are switched according to the particular phase, for example current rise or flat-top, and the target waveforms are accordingly changed. The event is called segment switch. Because some of the wires refer to completely unrelated quantities, there may be abrupt changes in the references. If these changes are interpolated across a segment switch, they will give rise to glitches. In order to fix the problem, an additional point is added one tick before each segment switch, before the linear interpolation is done.

- The A matrices were compared with the help of a simulator which applies the inputs of Hybrid to DPCS, performs the computation ($A\ matrix + P\_outs$) and evaluates the discrepancies between the real Hybrid error signals and these simulated error signals. The simulator points out the differences between the matrices, given the good matching of DPCS and Hybrid P_outs. The simulator demonstrated that a little change in the gains, as small as a few percent, may sum up in a big discrepancy between DPCS and Hybrid error signals, as high as 30%. However, the differences in the gains were of no concern because DPCS was going to replace Hybrid in the feedback loop. A leakage was also discovered in some outputs of the A matrix of Hybrid, where the gain is set to zero but the output has a small signal. These small signals, as well as the offsets of Hybrid, are eventually integrated over time in the PID controller and cause big errors at the output of the PID of Hybrid. Of course DPCS doesn't have this kind of flaw. Figure 3-3 illustrates this issue.

- The PID matrices worked very similarly, in spite of the simple discrete implementation of the derivatives and integrals in DPCS. A second simulator was implemented to test the PID matrices, which works in a way very similar to the A simulator, except for the fact that the Hybrid A_outs must be expanded on the DPCS time base, before applying them to the DPCS PID. This is necessary, because the DPCS PID calculates derivatives and integrals on the DPCS time base. The simulator allows the user to tune the time constant of the integrator τ_{int} and the best matching between Hybrid and DPCS was obtained with $\tau_{int} = 0.095$. The current value of τ_{int} in DPCS is 0.08. A major advantage of DPCS is that it does not saturate during the various stages of calculations, and it does not integrate offsets, because DPCS does not have the problem of analog offsets.

- The feed-forward signals V_outs of Hybrid and DPCS matched very well and the interpolation problem at segments switching was fixed for V_outs too.
- The M matrices were compared using the M simulator, which is similar to the A and PID simulators described above. The signals $(M\_outs + V\_outs)$ agree very well, but DPCS doesn't show the saturation problem of Hybrid.

All the simulators mentioned above use the IDL code of DPCS with the necessary modifications. The A and M simulators have also been implemented with a different and easier IDL code. The results from the different implementations agree perfectly and this proved to be a good test of the IDL code.

In conclusion, we found that DPCS was following Hybrid and was producing signals with the same shape, within a few percent, except for offsets, saturation and other Hybrid non-idealities.

3.2 DPCS real time code and time performance

The initial implementation of the IDL procedure *dpcs_ real_ time* used two nested *for* loops to read the samples and perform the corresponding calculations during the shot, with the appropriate time evolution of matrices, waveforms, and integrators and derivatives parameters. The period of each sample was fixed to $100\mu s$ ($10kHz$ sampling rate). This version of the code was tested for over three months without revealing any significant bug. In order to test its time performance, a number of performance parameters were added in the tree. The most important are:

- *NTIMES* is the number of input samples which have been read and processed by the IDL real time routine. NTIMES is the dimension of the vector *tlatch*, which contains the latch times of the input samples.
- $TLMAX$ is the max value of the vector tlatch (the time of the last sample).
- *TLMIN* is the min value of the vector tlatch (the time of the first sample).

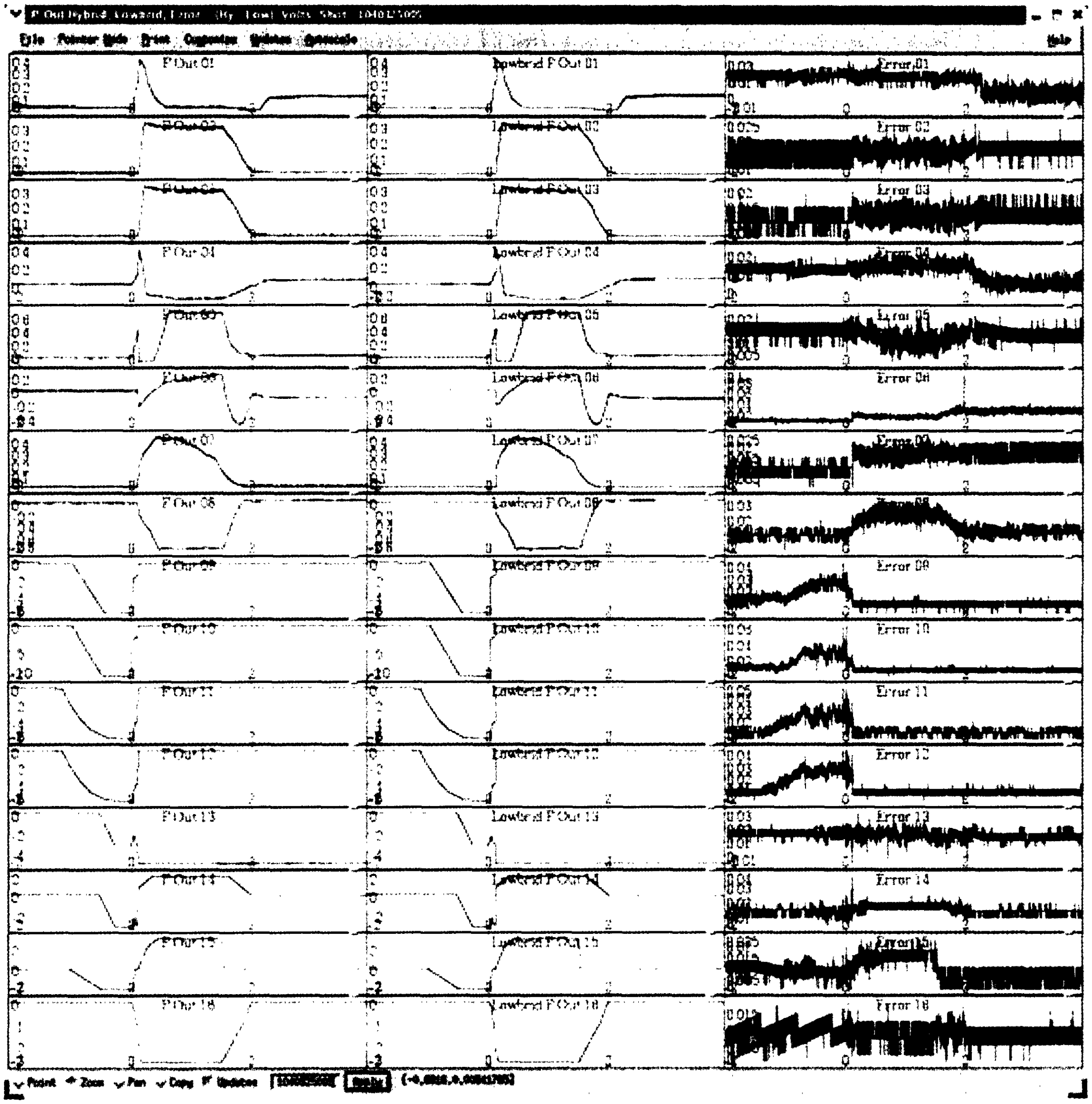


Figure 3-2: Hybrid and DPCS *P_outs* (reference signals) and the error between them for shot 1040325005

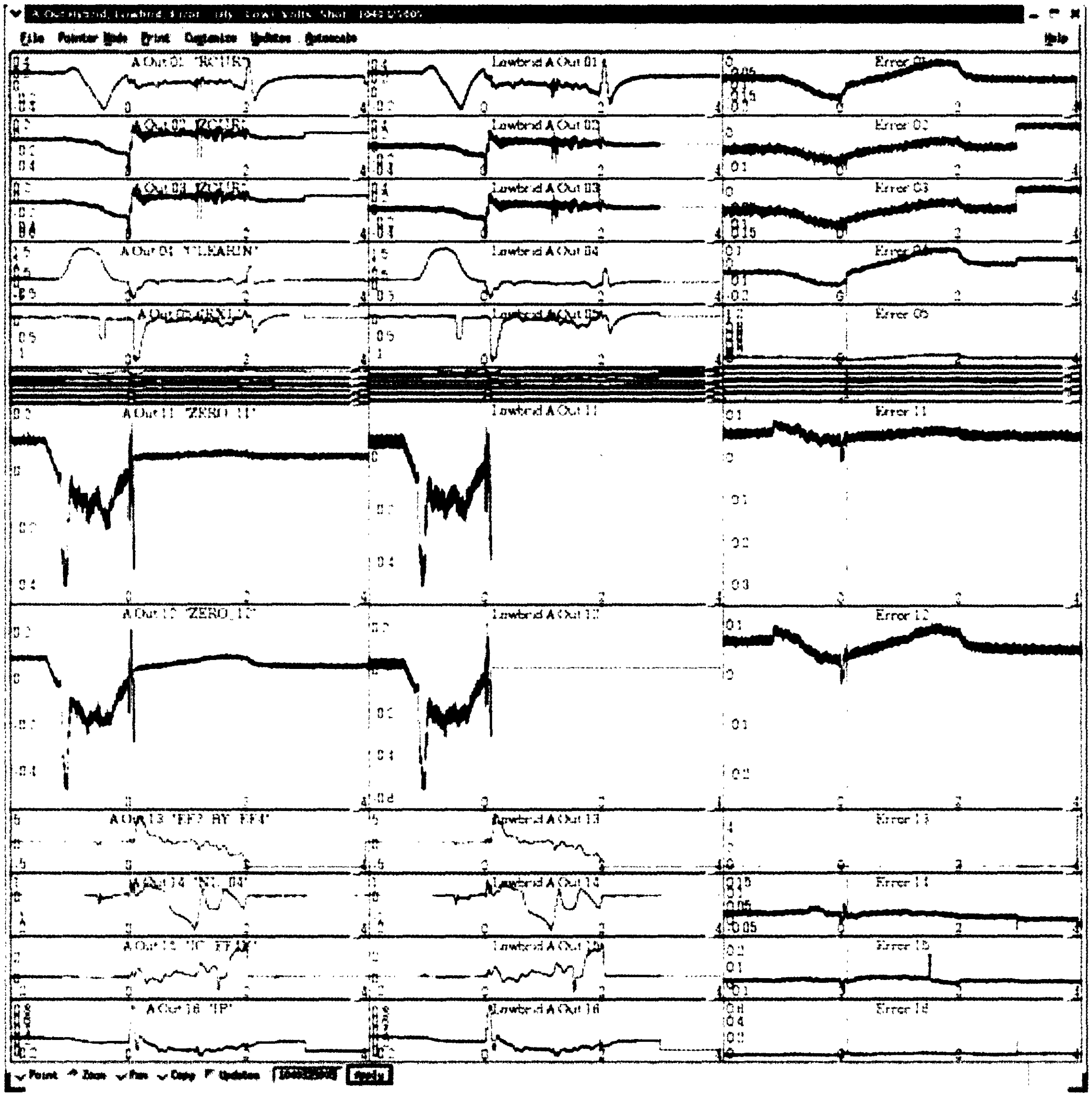


Figure 3-3: Note the leakage of Hybrid hardware during the second segment of the discharge, that is after 0.1s, in signals 11 and 12. The leakage causes the error signals to be non-zero even when the gain is set to zero (i.e. all the coefficients of the A matrix should be identically zero). DPCS is a digital system and does not have this kind of problem. Shot 1040325005

- *TINT_MEAN* is the average value of the time intervals between consequent input samples. In the case that no samples are lost, TINT_MEAN is exactly the clock period.

- *TINT_MAX* is the max value of the time intervals described in TINT_MEAN.

- *TINT_MIN* is the min value of the time intervals described in TINT_MEAN.

- *TINT_STDEV* is the standard deviation of the time intervals described in TINT_MEAN.

- *STATUS* is a variable used to check the loss of input samples, which occurs when the computations in the loop are too slow with respect to the rate of the digitizers. The variable STATUS is indeed the combination of three tests, namely $NTIMES > 0$ (i.e. at least some samples have been processed by DPCS), $TLMAX > TLMIN$ (i.e. no colossal errors in the time base and $TLMAX \neq TLMIN$) and $TINT\_MAX < 2 * ACT\_DELTA\_T$.

This last condition checks that no samples were lost. In fact, the sampling period of the digitizers is specified by the entry *DELTA_T* in the tree. Because the period can only be a multiple of the $1\mu s$ digitizers clock, a new variable *ACT_DELTA_T* is evaluated from DELTA_T and stored in the corresponding node in the tree. *STATUS* checks the loss of samples because this event happens only if there is at least one couple of processed samples whose time interval is larger than twice the digitizers period. The initial speed tests were done by changing DELTA_T in the tree. The system could get every sample and process it correctly up to a limit sampling period of $45\mu s$. With $40\mu s$ about 10% of the samples were lost and the output waveforms were seriously distorted (figure 3-4).

This happened because the calculations assumed the theoretical ACT_DELTA_T stored in the tree, while the actual time delay between the input samples was not constant.

Since the first implementation, the DPCS code and *dpcs_real_time* have undergone many upgrades in order to improve the robustness, efficiency, flexibility and readability. In particular:

- in order to make the system more robust with respect to the loss of input samples, *dpcs_real_time* takes into account their actual time stamp. Figure 3-5 shows an example of the *A_outs* waveforms calculated with the adaptive version of the code, when

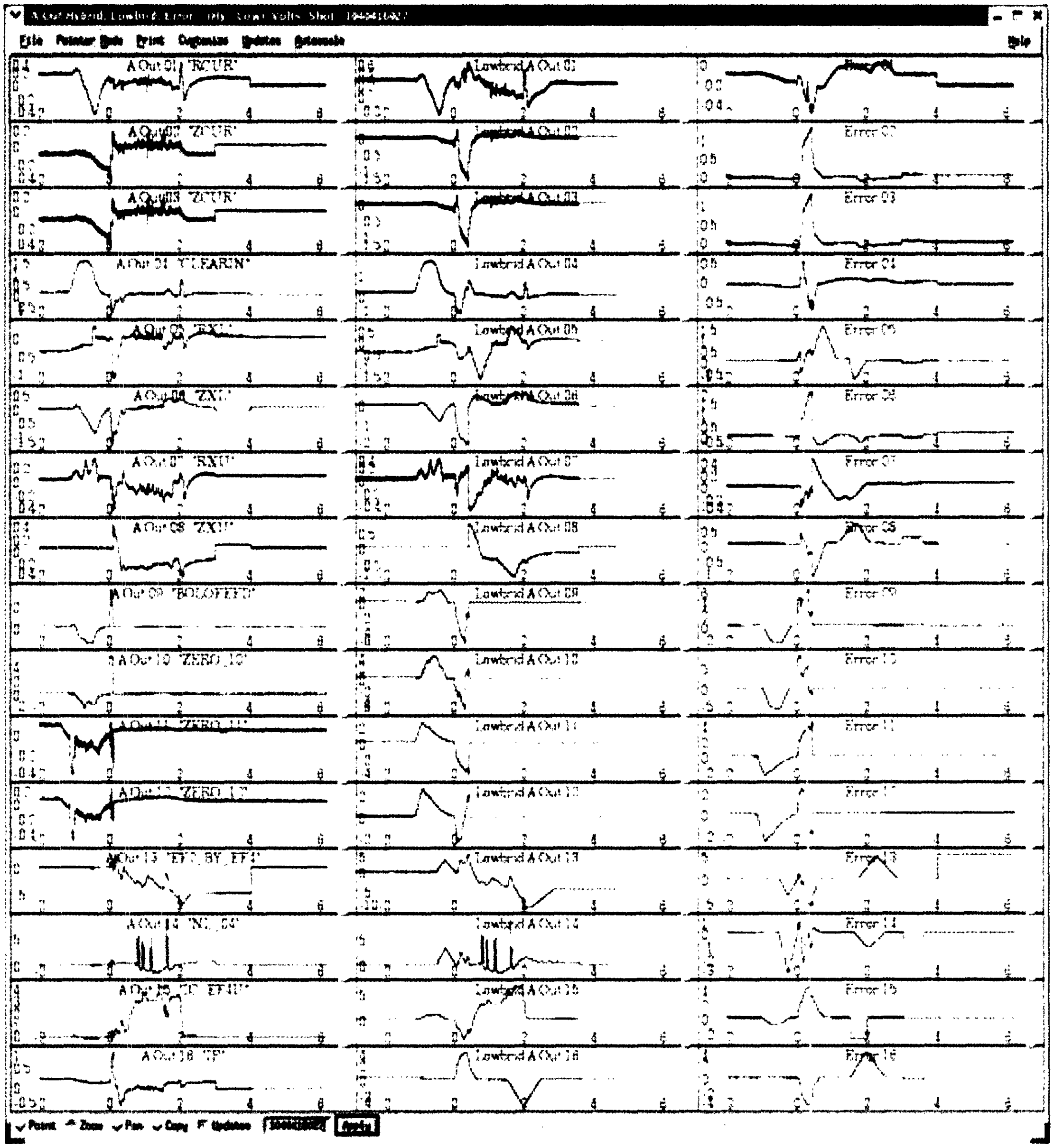


Figure 3-4: The waveforms resulting from the initial implementation of DPCS are distorted when the digitizer speed is 10% or more faster than the time required for calculations. The error signals in the third column show that the outputs of DPCS/Lowbrid (second column) are significantly different from the correct control signals (first column). Shot 1040416027

about 10% of the input samples is lost. The waveforms are correct and do not show the distortions in figure 3-4.

- Custom procedures can be called within the loop to perform additional computations at any stage, for input conditioning, observers estimation, PIDs correction or controllers synthesis. All these operations are implemented within a unified syntax, the *call_procedure* instruction, whose parameters are appropriately designated.
- The code is thoroughly commented.

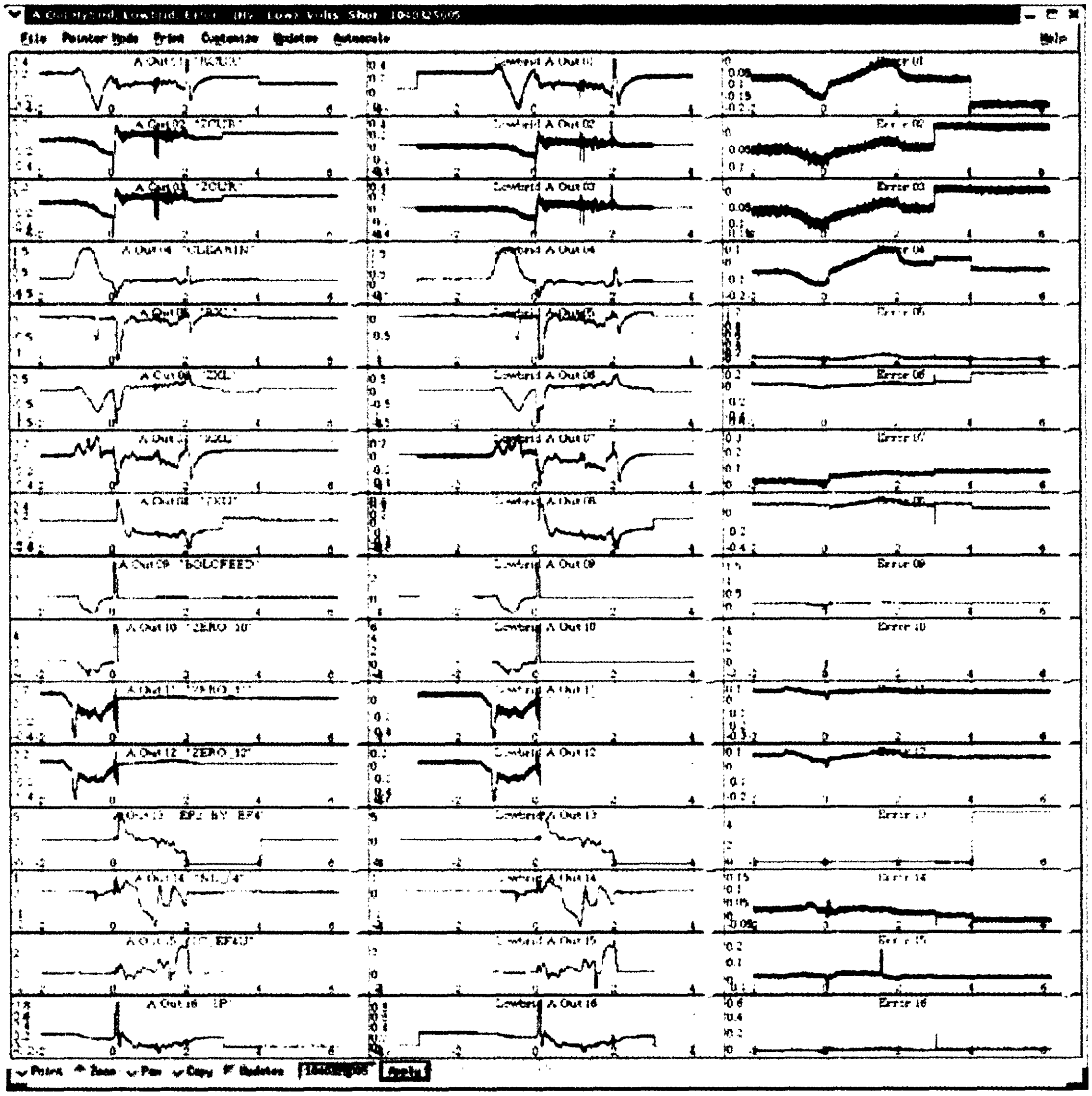


Figure 3-5: The calculation of the control signals is correct even if some input samples are lost, thanks to the adaptive evaluation of the time stamp of the samples and the use of this information in the PID calculations. In the case illustrated above 10% of the input samples was lost because of the delay in the loop calculations, but the DPCS/Lowbrid signals (second column) are still consistent with the correct control signals (first column), as the error signals (third column) show

The dpcs_real_time code, as in use in June 2005, is included and commented below.

```
pro dpcs_real_time,max_loops
;
common hybrid_params,MAX_STEPS,dt1,dt2,NW,NX,NM,NSEG,SEG_NM
common DPCS_params,Nbuf,Ndacs,Ncards,ecm
common hybrid_wavegen,pP,pV
common hybrid_output_info,X_OUT,Y_OUT,Z_OUT,U_OUT,P_OUT,V_OUT,LAST_STEP, $
  tlatch,tinst,tprocess,hb_poll
common hybrid_init_c,psw_common,sw_union,interrupts
common hybrid_daq_data,buf,dacs
common DPCS_software,llImage
common dpcs_functions,input_funcs,observer_funcs,controller_funcs
common dpcs_procedures,input_pros,observer_pros,pid_pros,controller_pros, $
  test_pros,Allpros
common dpcs_procedure_params,input_params,observer_params,pid_params, $
  controller_params,test_params
```

The initial part is used for declaring common variables. These are organized according to their specific functions: hybrid_params contains the definitions of the parameters of the Hybrid-like control, that is the maximum number of time steps during a discharge MAX_STEPS (this is evaluated on the basis of the trigger time, the stop time and the DPCS sampling period), the derivative and integral time scalings dt1 and dt2, the number of wires (observables) NW, the number of inputs of the A matrix NX, the total number of inputs Nbuf (NX and Nbuf used to agree on Hybrid, but this is not necessarily the case with DPCS), the number of controllers NM, the number of segments active in a plasma discharge NSEG and their numeric identifiers SEG_NM. DPCS_params contains parameters specific of the hardware of DPCS, the total number of inputs Nbuf, the total number of outputs Ndacs, the number of I/O cards Ncards and the number of ticks of the clock of the digitizers corresponding to a DPCS cycle ecm. pP and pV are the pointers to the target and feed-forward waveforms (P outs and V outs). X_OUT, Y_OUT, Z_OUT, U_OUT, P_OUT, V_OUT, LAST_STEP, tlatch, tinst, tprocess, hb_poll are used to store momentarily the signals that will be written in the tree by another routine, dpcs_output: X_OUT, Y_OUT, Z_OUT, U_OUT, P_OUT and V_OUT store the inputs, error signals, PID outputs, controller outputs and target and feed-forward waveforms for each time step. The time step itself is in tlatch. The other three time parameters, tinst, tprocess, hb_poll, are used for debugging. hybrid_init_c contains the definition of the switching times of the matrices and waveforms during a discharge. interrupts is used for disabling the interrupts in the real time loop. hybrid_daq_data contains variables used to pass inputs and outputs to and from dpcs_input. llImage is the low-latency routine used to operate the input cards in fast data acquisition mode. Finally, dpcs_functions and dpcs_procedures contain the definitions of the procedures and functions that can be called to perform custom computation at each stage of the control loop.

```
; first time though fault in all of the code
message, /reset_error_state
sw = sw_union
psw = psw_common
nj =n_elements(sw)-1
; Initialize internal variables for more efficient assignments in loop
y=fltarr(NW)
y1=y
y1_dum=y \qquad \qquad \qquad ;NEED THIS FOR REAL TIME STEP
y_last=y
y2=y
y2_last=y
t_last=long(-ecm) \qquad ;NEED THIS FOR REAL TIME STEP
```

```
real_step=1. \qquad \qquad   ;NEED THIS FOR REAL TIME STEP
z=y
x=fltarr(Nbuf)
NXm1 = NX-1 ; Note x is now dimensioned to conform to the shape of buf, not A
p = fltarr(NW)
q = p \qquad ; Make a dummy var for just the observer, same size as P
v = fltarr(Ndacs) ; Could take U and V to be fltarr(Ndacs). It is not NW
u = v \qquad ; U and V must have same dimension, (or n_U>n_V) to avoid out-of-range
w = v       ; Make a dummy of the same length
sp = size(*pP) ; *pP and *pV are dimensioned in dpcs_init and may have different lead
sv = size(*pV) ; dim than local 1-dim variables P and V
Np = sp[1] & Npm1=(Np<n_elements(p))-1L
; Used for subscripting. Upper bounds must avoid out-of-range
Nv = sv[1] & Nvm1=(Nv<n_elements(v))-1L
;
tl = lonarr(Ncards) ;tl and ti are now local variables passed to dpcs_input
ti = lonarr(Ncards)
tp = lonarr(Ncards)
hbpoll = lonarr(Ncards)\qquad
; Implement a synchronous loop over switch times
; Initialize counters and flags:
step=0l
first = 1\qquad ; Set flag indicating first pass through RT loop
i0 = sw[0]  ; This is probably always zero?
i = sw[0]
j = 0
exited_early = 0
EXIT_I = sw[nj]-1
EXIT_S = MAX_STEPS-1
max_loops = long(max_loops)
```

```
; Get info on external real_time procedures
Nin_p = n_elements(input_pros) & Nin_pm1 = Nin_p-1 & $
  call_inputs = (Nin_p gt 0L)
Nobs_p = n_elements(observer_pros) & Nobs_pm1= Nobs_p-1 & $
  call_observers = (Nobs_p gt 0L)
Npid_p = n_elements(pid_pros) & Npid_pm1= Npid_p-1 & $
  call_pids = (Npid_p gt 0L)
Ncont_p = n_elements(controller_pros) & Ncont_pm1=Ncont_p-1 & $
  call_controllers = (Ncont_p gt 0L)
Ntest_p = n_elements(test_pros) & Ntest_pm1=Ntest_p-1 & $
  call_tests = (Ntest_p gt 0L)
```

After the initial definition of the common variables, some of them are initialized and the external ones are copied into local variables to make the code faster. The counters and flags for the loop are initialized and the information on custom procedures is retrieved.

```
REPEAT BEGIN
  x[0] = dpcs_input(first, max_loops, timeout, tl, ti, tp, hbpoll)
  ; Need to avoid out-of-range for NX<Nbuf below,
  ; in which case, not subscripting the LHS is more efficient
  IF (not timeout) THEN BEGIN
    inc = tl[0]/ecm
    i = inc+i0
```

The real time loop is contained in the REPEAT-UNTIL instruction. dpcs_input reads the array of 128 input samples (in the current implementation of DPCS there are 128 input channels, each digitizing at 16 bit resolution). This instruction calls a custom C routine, llAcquire, which waits for the inputs and performs a time-out check by polling for the trigger for a maximum number of iterations max_loops. The inputs are passed through the common variable buf. If the trigger doesn't show, the variable timeout is set to 1 and the loop is exited. The time-out

check cannot be based on the internal clock of the DPCS computer, because the interrupts are disabled during real-time operation. The reason for doing this will be explained later in the comment of the code. The input instruction also returns the time stamp of the samples, tl, and other timing parameters ti, tp, hbpoll. ti is tl plus the transfer time from the input cards to the computer. In the case dpcs_input doesn't time-out, an incremental index is calculated on the basis of the time stamp tl (inc = tl[0]/ecm). dpcs_input also sets the analog output values in the call to llAcquire. The common variable dacs is used to pass the outputs to llAcquire.

```
IF (i gt EXIT_I) THEN begin
  exited_early = 1
  goto, EXIT_EARLY
ENDIF
```

The target waveforms, feed-forward waveforms and matrices will be accessed using the real time indices inc and i, instead of a simple incremental pointer. The adaptiveness with regard to the real timing of the inputs is one of the most interesting features of the code and has many advantages: for example, a number as large as 10% of the total input samples can be lost and the control signals are still correct. It required careful implementation though, for example the branch instruction EXIT_EARLY had to be introduced to account for the case when some samples are lost and the maximum number of iterations EXIT_S is not completed in the time of a discharge. The exit early condition is based on the real time pointer i.

```
WHILE (i ge sw[j+1]) DO j++
```

The processing of input samples according to their real time stamp brings another issue, when the real time index i is pointing to a time past a reference or matrix switch, but the matrices and waveforms haven't been updated yet. The

solution to this problem employs a WHILE cycle. The real time index i is compared with the set of indices which mark the times at which the matrices and the waveforms have to switch. The pointer to the matrices and waveforms j is updated until i becomes smaller than the next switching index.

```
; Index into P and V arrays, based on real timestamp and lead dim of array
sNp = inc*Np
sNp1 = sNp+Npm1
sNv = inc*Nv
sNv1 = sNv+Nvm1
```

The target waveforms and feed-forward waveforms are read using the real time indices $sNp \div sNp1$ and $sNv \div sNv1$.

```
; Optional input conditioning step
IF (call_inputs) THEN $
  FOR k=0L,Nin_pm1 DO call_procedure,input_pros[k],*input_params[k,j],x
; Update the target vector from the P waveform
; The reference could be generalized
p[0] = (*pP)[sNp : sNp1] * ([10.,x[55]])[*psw[6,j]]
; Observer Step
q[0] = *psw[0,j] # x[0:Nxm1] ; Multiply the inputs by the A-matrix
IF (call_observers) THEN $
  FOR k=0L,Nobs_pm1 DO $
    call_procedure,observer_pros[k],*observer_params[k,j],x,q
y[0] = q + p ; form the error signal
real_step = (tl[0]-t_last)/ecm
y1[0] = (real_step ne 0L) ? (y - y_last)/dt1/real_step : y1_dum ; Derivative value
y2[0] = (y2_last +real_step*dt2*y * *psw[3,j]) * *psw[5,j] ; Integral value
y_last[0] = y
y2_last[0] = y2
```

```
t_last = tl[0] ;NEED THIS FOR REAL TIME STEP
Z[0] = *psw[2,j] * y + *psw[4,j] * y1 + y2 ; PID step (default linear gains)
IF (call_pids) THEN $
  FOR k=0L,Npid_pm1 DO call_procedure,pid_pros[k],*pid_params[k,j],y,y1,y2,Z
v[0] = (*pV)[sNv : sNv1]\qquad \qquad \qquad ; This step's V (of whatever length)
w[0] = *psw[1,j] # Z ; Product of M#pid_out, perhaps with trailing zeros
IF (call_controllers) THEN $
  FOR k=0L,Ncont_pm1 DO $
    call_procedure,controller_pros[k],*controller_params[k,j],Z,w
U[0] = v + w ; The sum is of the right length (nu=nv=nw=Ndacs)
IF (call_tests) THEN $
  FOR k=0L,Ntest_pm1 DO $
    call_procedure,test_pros[k],*test_params[k,j],x,p,q,y,y1,y2,Z,v,w,U
```

The computations of the PID are reported above. Note how the time step of integrals and derivatives is including the information on the real time step. Another important feature of DPCS is the ability to accommodate custom computations at various stages, for input conditioning, observers estimation, PIDs correction or controllers synthesis. All these features are implemented within a unified syntax, the call_procedure instruction, whose parameters are appropriately designated. In particular, the input conditioning routine is used to compensate the input offsets.

```
dacs[0] = 10.< U >(-10.) ; use [0] subscripting for dacs vector allocated in dpcs_init
dpcs_output, step,tl,ti,x,y,Z,U,p,v, tp, hbpoll
; Could try doing these directly, using pointers?
step++
```

The final output is constrained within the power rails of the output cards. The content of dacs will be output at the next call of dpcs_input. The output routine dpcs_output saves the current results in the common variables LAST_STEP, tlatch, tinst, X_OUT, Y_OUT, Z_OUT, U_OUT, P_OUT, V_OUT, tprocess and hb_poll.

```
    ;
    ; after the first time through the loop disable the interrupts
    ;
    IF (first) THEN BEGIN
      IF interrupts THEN josh = call_external(llImage, 'llDisableInts')
      first = 0
      ; step-- ; step back to zero, so dpcs_output overwrites dummy values
    ENDIF
  ENDIF ELSE GOTO, time_out
```

Deterministic time performance is obtained on a PC running a standard Linux operating system by disabling the interrupts: no instruction in the control loop will be interrupted and it will take a deterministic time to complete. In order to avoid problems of page faults, a first cycle through the loop must be completed and the relevant instructions and variables loaded in the internal memory of the processor, before the interrupts are disabled.

```
ENDREP UNTIL (step gt EXIT_S) ; Normal Exit when last step
; all done so enable the interrupts
; and leave low latency mode
EXIT_EARLY: ; Exit when get to last timeslice earlier than num steps
time_out: ; Timed out waiting for trigger
IF (not first) THEN begin
  IF interrupts THEN begin
    dummy = call_external(llImage, 'llEnableInts')
    spawn,'/etc/rc.d/init.d/ntpd restart > /dev/null' ; reset the date
  ENDIF
  IF (timeout) THEN print,'Timed out'
  IF (exited_early) THEN print,'Exited Early'
ENDIF
```

The interrupts must be re-enabled only in the case that two or more cycles of the real time loop were completed.

```
dummy = call_external(llImage, 'llDone')
; Clean up Heap Variables
ptr_free,pP,pV,psw
;
LAST_STEP = STEP-1 ; Needed so that Store knows how many to store
print,'LAST_STEP=',LAST_STEP,'EXIT_I=',EXIT_I,'EXIT_S=',EXIT_S
print,'i=',i,'step=',step
help, /structure, !error_state
end
```

The final cleaning and log operations after the control loop.

In spite of the large number of input and output channels, the basic PID controller cycle uses only $48\mu s$, which correspond to an headroom of about $50\mu s$ for extra computation at the nominal rate of $10kHz$. This controller was used in the power supplies test day (run 1050204) and in the early conditioning of the machine. With various upgrades, DPCS has been on-line since the start of the 2005 experimental campaign. An initial tweak-up of the programming of previously run discharges was necessary but the new config files are likely to work on DPCS consistently in time, as the implementation of DPCS is inherently free from drifts and other problems of analog circuitry. The run 1050210 was in part dedicated to test one of the new features of DPCS, namely the possibility of changing the cycle time to verify the limit speed at which the plasma can be controlled against vertical instability. At the nominal speed of $10kHz$, DPCS has an equivalent bandwidth below $5kHz$, while the EFC coils, which compensate for the vertical instability of our plasmas, have an analog bandwidth of about $3kHz$. We tried to reduce the DPCS cycle time for a rather unstable plasma. The ratio of the decay index and the single mode critical index for this plasma was $nxxc = 1.15$ and the elongation was $k = 1.68$: these values are close to the limit of controllable plasmas in Alcator C-Mod. The peak current was $0.8MA$. The plasma was successfully controlled at 10 and $5kHz$ (discharges 1050210021 and 1050210022), but disrupted at $2.5kHz$ (discharge 1050210023), as shown in figure 3-6 ([14]).

Notably, the plasma could almost be controlled at $2.5kHz$, thus we expect the minimum cycle

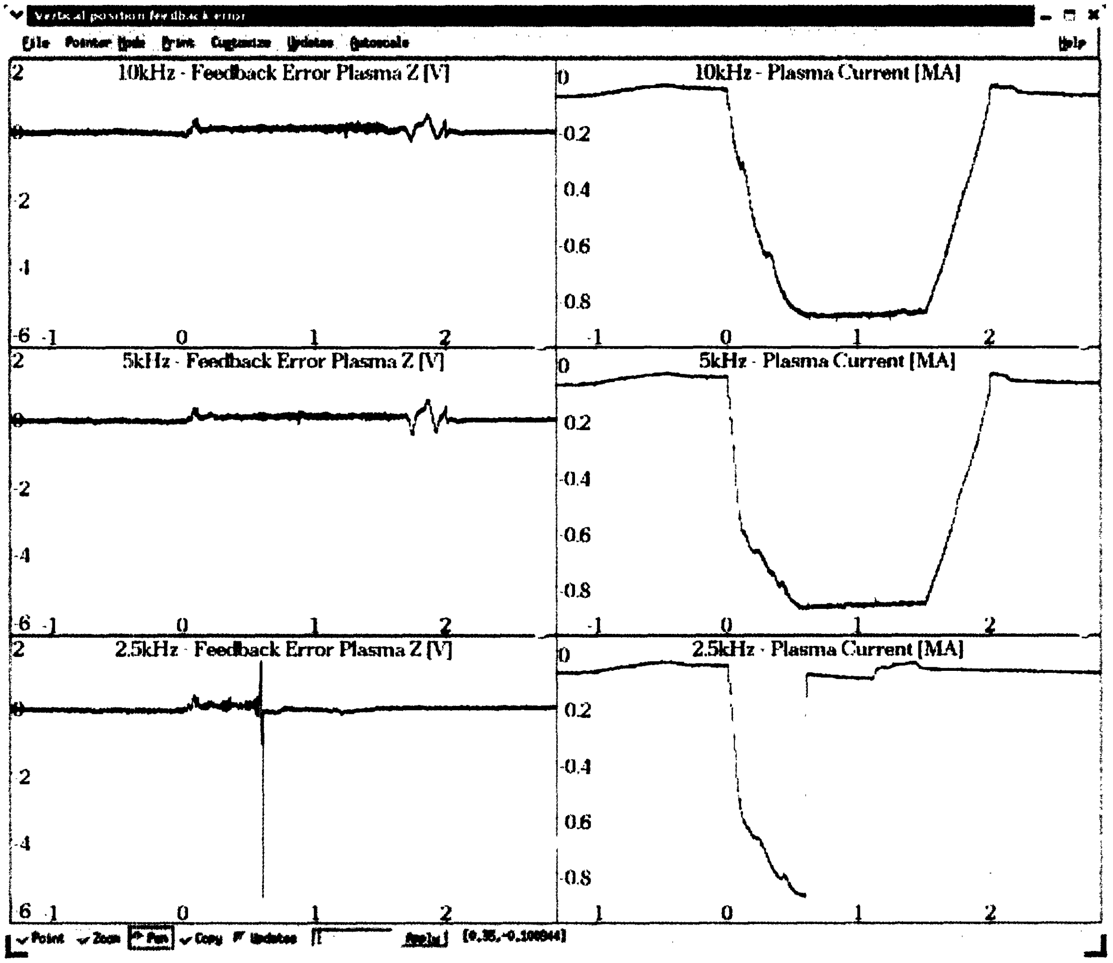

Vertical position feedback error
File Pointer Node Print Customize Updates Autoscale Help
10kHz - Feedback Error Plasma Z [V]
2
0
-2
-4
-6
-1 0 1 2
10kHz - Plasma Current [MA]
0
0.2
0.4
0.6
0.8
-1 0 1 2
5kHz - Feedback Error Plasma Z [V]
2
0
-2
-4
-6
-1 0 1 2
5kHz - Plasma Current [MA]
0
0.2
0.4
0.6
0.8
-1 0 1 2
2.5kHz - Feedback Error Plasma Z [V]
2
0
-2
-4
-6
-1 0 1 2
2.5kHz - Plasma Current [MA]
0
0.2
0.4
0.6
0.8
-1 0 1 2
Point Zoom Pan Copy Updates Apply
[0.35,-0.100944]

Figure 3-6: Plasma control at different cycle rates of the digital control DPCS. The plasma is controlled against vertical instability when DPCS cycle is $100\mu s$ (top panel, shot 1050210021) and $200\mu s$ (central panel, shot 1050210022), but disrupts when the cycle is $400\mu s$ (bottom panel, shot 1050210023)

rate for this type of plasma to be slightly above $2.5kHz$. At $5kHz$ the headroom for extra computation is considerable, about $150\mu s$.

Part II

Alcasim simulation code for Alcator C-Mod

Chapter 4

Introduction

Alcasim is a code developed in Matlab-Simulink with the intention of providing a flexible tool for the analysis of control strategies for a tokamak. The application of the powerful duo Matlab-Simulink for modeling a plasma discharge in a tokamak and studying the associated control issues is not new, for example [15] and [16], where the DINA simulation code is integrated for modeling the tokamak and the plasma. In particular, the suite of tools developed by people at DIII-D provides a powerful environment for design, simulation and study of tokamak control, both for devices that already exist, like DIII-D , NSTX, and MAST and for those which are in the design/construction phase, as demonstrated in the case of KSTAR, EAST and ITER [17].

Nevertheless, in the cases mentioned above, the code is intended to simulate the control of a plasma in equilibrium, while incorporating a detailed description of the physics of the plasma and the various components of the machine. Alcasim uses a discretized electromagnetic model of Alcator C-Mod and a single or multi-filament plasma, as discussed in [18] and [19], and can simulate full discharges, comprising the current ramp-up and current ramp-down. The results are consistent with the experiments of C-Mod, when the same setup and control parameters are used. With Alcasim we tried to meet the following goals:

- Draw a model of the machine from basic building blocks;
- Read the data concerning with the configuration of the diagnostics from the corresponding tree of a real shot and simulate the relevant diagnostics;
- Use all the conversion factors used in the DPCS control system;

- Apply the same control algorithm, with the target wave-forms and the control matrices loaded from the corresponding tree of a real shot;
- Model the power supplies appropriately, with all the setup parameters read from the corresponding tree of a real shot;
- Simulate the closed loop evolution of the system during the entire discharge, using a simple model of the plasma;
- Compare the results with the data from the real experiment.

The simulation of the current rise is particularly important in the case of shots which attempt to form high current plasmas. In the current rise phase the demand on the power supplies can be too large and they can saturate, producing the consequent disruption of the plasma. Alcasim might prove useful to study control strategies for preventing these events.

Alcasim is equally flexible: it is not intended for use on a specific machine, but the user interfaces and the data structures allow to draw a generic toroidal device and simulate the poloidal equilibrium of the plasma running in it. Alcasim moves very easily from the rapid prototype of the machine to the Matlab electromagnetic simulation or the Simulink simulation of the tokamak and the control system. Another important feature is the modularity of the simulator: the block-diagram language of Simulink and the implementation of Alcasim allow to replace a subsystem, for example a power supply, with minimal adaptation. In the following we discuss how Alcasim is programmed and how its graphical interfaces work. We also report on the first results from Matlab open-loop simulations and Simulink closed-loop simulations.

Chapter 5

Alcasim Graphical User Interfaces

Alcasim makes an extensive use of Graphical User Interfaces (GUIs). Matlab provides an environment to design and program user interfaces, *GUIDE* (GUI Development Environment). This environment is very similar to what is offered by NI CVI (LabWindows) or other software for data acquisition and analysis. The programmer designs an interface as a *.fig* file and the code is automatically generated as a *.m* file. The *.m* file contains the code for the graphical objects and the various tools (buttons, knobs, sliders, axes). The programmer must introduce his pieces of code to perform specific actions when an interface command is triggered. The programmer can also call GUIs from other GUIs, to create a complete graphical application.

The code of a generic Matlab GUI consists of an opening section, where general definitions are located (this part should not be edited), an opening function, which performs all the tasks before the GUI is run, a number of user defined event-functions, which are executed when an interface command is triggered, and an output function. The instructions *uiwait* and *uiresume* and the output function are necessary when the GUI is returning something, otherwise they are not used.

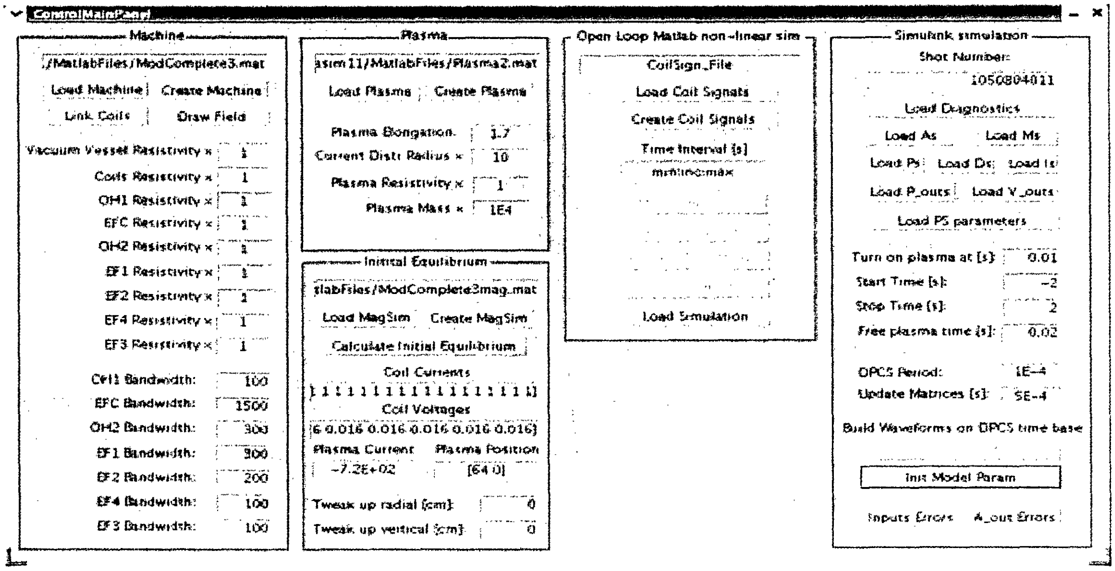
In order to share the variables among user interface functions, Matlab GUIs provide a useful tool, the *struct* variable *handles*, where every partial result can be stored and any information about the user interface axes, commands and general settings can be retrieved. Because this structure is by default passed to all the functions, it is the most convenient way to share information. The following code illustrates the general code of a GUI. It refers to the GUI used to remove coils from a model of the machine. This GUI returns the modified model to the calling GUI *Machine* and demonstrates the use of the commands *uiwait* and *uiresume* and the output function *RemoveCoilPanel_ OutputFcn*:

```
function varargout = RemoveCoilPanel(varargin)
% REMOVECOILPANEL M-file for RemoveCoilPanel.fig
% REMOVECOILPANEL, by itself, creates a new REMOVECOILPANEL
% or raises the existing singleton*.
% Last Modified by GUIDE v2.5 26-May-2004 17:49:32
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
  'gui_Singleton', gui_Singleton, ...
  'gui_OpeningFcn', @RemoveCoilPanel_OpeningFcn, ...
  'gui_OutputFcn', @RemoveCoilPanel_OutputFcn, ...
  'gui_LayoutFcn', [] , ...
  'gui_Callback', []);
if nargin & isstr(varargin{1})
  gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
  [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
  gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
%---------------------------------------------------------------------------
% Executes just before RemoveCoilPanel is made visible.
%---------------------------------------------------------------------------
function RemoveCoilPanel_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments to RemoveCoilPanel (see VARARGIN)
% Choose default command line output for RemoveCoilPanel
handles.Output = hObject;
% Define the local Model
handles.PurgedCoils = varargin{1};
handles.NextCoilPointer = varargin{2};
handles.ReferenceAxes = varargin{3};
handles.scaleM = varargin{4};
% Update handles structure
guidata(hObject, handles);
% Update popup menu with coil names
UpdatePopup(handles);
% UIWAIT makes RemoveCoilPanel wait for user response (see UIRESUME)
uiwait(handles.RemoveCoilPanel);
%-----------------------------------------------------------------------
% This function updates the popup list with the names of the coils
%-----------------------------------------------------------------------
function UpdatePopup(handles)
.....
% ----------------------------------------------------------------------
% Outputs from this function are returned to the command line.
% ----------------------------------------------------------------------
function varargout = RemoveCoilPanel_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.Output{1};
varargout{2} = handles.Output{2};
delete(handles.RemoveCoilPanel);
% -----------------------------------------------------
% Executes on button press in Remove_Coil.
% ---------------------------------------------------------
function Remove_Coil_Callback(hObject, eventdata, handles)
.....
% ---------------------------------------------------------
% Draw_Model_Remove function.
% ---------------------------------------------------------
function Draw_Model_Remove(hObject,handles)
% This function plots the coils of a Model in the Section View axes.
.....
% ---------------------------------------------------------
% Executes on button press in Exit_Remove.
% ---------------------------------------------------------
function Exit_Remove_Callback(hObject, eventdata, handles)
handles.Output = {handles.PurgedCoils, handles.NextCoilPointer};
guidata(hObject, handles)
uiresume(handles.RemoveCoilPanel);
%---------------------------------------------------------------------------
% Executes during object creation, after setting all properties.
% --------------------------------------------------------------------------
function Coil_Selection_CreateFcn(hObject, eventdata, handles)
.....
% --------------------------------------------------------------------------
% Executes on selection change in Coil_Selection. Not used.
% --------------------------------------------------------------------------
function Coil_Selection_Callback(hObject, eventdata, handles)
.....
```

Alcasim makes extensive use of the functions that Matlab provides to work with data structures such as structures (*struct*), arrays of cells (*cell*) and string and numeric matrices.

5.1 ControlMainPanel GUI

The *ControlMainPanel* GUI is illustrated in figure 5-1. It is divided in sub-panels, that is *Machine*, *Plasma*, *Initial Equilibrium*, *Open Loop Matlab non-linear sim* and *Simulink simulation*. *Machine* allows the user to build and tweak up the model of the machine, while *Plasma* has the controls to introduce a simple plasma model: in the current implementation the plasma is a single current plasma. Pressing the button *Create Machine* opens the GUI *Machine*, which is described later. The button *Load Machine* allows the user to load a model of the machine which has been previously designed and saved. The files are saved as Matlab *.mat* files. They are Matlab structures. In the particular case of a model of the machine, the file is labeled with a general name *Model* and has the following fields:



./MatlabFiles/ModComplete3.mat

Vacuum Vessel Resistivity × 1
Coils Resistivity × 1
OH1 Resistivity × 1
EFC Resistivity × 1
OH2 Resistivity × 1
EF1 Resistivity × 1
EF2 Resistivity × 1
EF4 Resistivity × 1
EF3 Resistivity × 1

OH1 Bandwidth: 100
EFC Bandwidth: 1500
OH2 Bandwidth: 300
EF1 Bandwidth: 300
EF2 Bandwidth: 200
EF4 Bandwidth: 100
EF3 Bandwidth: 100

asim11/MatlabFiles/Plasma2.mat

Plasma Elongation 1.7
Current Distr Radius × 10
Plasma Resistivity × 1
Plasma Mass × 1E4

tlabFiles/ModComplete3mag.mat

Coil Currents
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Coil Voltages
[6 0.016 0.016 0.016 0.016 0.016]

Plasma Current -7.2E+02
Plasma Position [64 0]

Tweak up radial [cm]: 0
Tweak up vertical [cm]: 0

CoilSign_File

Time Interval [s]
min:inc:max

Shot Number: 1050804011

Turn on plasma at [s]: 0.01
Start Time [s]: -2
Stop Time [s]: 2
Free plasma time [s]: 0.02

DPCS Period: 1E-4
Update Matrices [s]: 5E-4

Figure 5-1: The Graphic User Interface (GUI) *ControlMainPanel*

- *Inductances*, which is the matrix of self and mutual inductances of the coils.

- *CoilsResistances*, which is the array of the electrical resistances of the coils.

- *NextCoilPointer*, which is used to keep trace of the number of coils in the model.

- *Coils*, which is an array of structures, that is the coils. Each coil is a structure which is created by the GUI *NewCoilPanel*.

The button *Link Coils* allows the user to define series and anti-series connections among coils. If some coils are connected in series or anti-series, the order of the electromagnetic problem is consequently reduced. The button *Draw Field* runs the GUI *DrawFieldPanel* which draws the field lines for a certain set of coil currents.

The buttons *Create Plasma* and *Load Plasma* allow to create a plasma file, through the GUI *PlasmaPanel*, and to load the corresponding *.mat* file. The file of the model of a plasma is labeled with the general name *Plasma*. The additional controls in the frame *Machine* are edit fields, which can be used to modify the resistivity of the passive and active coils in the model. Note also the edit fields in the *Plasma* sub-panel which allow to change the mass of the plasma in the massive simulation or modify its resistivity in the electromagnetic simulations.

The sub-panel *Initial Equilibrium* is used to evaluate an initial configuration where the plasma is in equilibrium. The button *Create MagSim* opens the GUI *CalcMagFieldPanel* to create a *.mat* file of the type *MagneticSim*. This file contains the magnetic field generated by each coil of the model on a user-defined grid. *Load MagSim* loads a *MagneticSim* file. The button *Calculate Initial Equilibrium* launches the GUI *DrawFieldPlasmaPanel*, which uses the information in *MagneticSim* to evaluate the equilibrium: the radial field is minimized for a user-defined radial position of the plasma (this operation determines the vertical position as well), then the vertical component of the magnetic field in the position of the plasma is read and the plasma current is calculated so that the plasma stays in equilibrium in that position. Once the equilibrium position and the current of the plasma are evaluated for a particular set of coil currents, all the relevant quantities are reported in the corresponding fields in the sub-panel *Initial Equilibrium*. The voltages across the coils are evaluated using the resistances of the coils.

There are two simulation options, the Matlab open-loop simulation, which was initially used to validate our linear piece-wise electromagnetic simulation code, and the Simulink simulation. The commands inside the sub-panel *Open Loop Matlab non-linear simulation* set the duration and the time step of the Matlab simulation, run the code which simulates the open loop evolution of a perturbed plasma and plot the results as single plots or as playbacks of the plasma discharge. The single plots are editable as conventional Matlab plots. The plotting buttons are *Plot* and

Movie. Save Simulation and *Load Simulation* perform the corresponding tasks. A simulation is saved as a Matlab structure in a *.mat* file, whose general label is *Simulation.* The structure contains a field *TimeInterval* with the time base, a matrix *Data* with the physical quantities stored during the simulation (the coil currents and various plasma parameters, organized as columns in the matrix) and the cell array *Labels* of the labels of the signals in Data.

Mass Simulation launches the Matlab script with the massive simulator (*TimeDomain.m*) and *Massless Simulation* launches the Matlab script with the massless simulator (*TimeDomainMassless.m*). Before either a massive or a massless simulation can be run, it is necessary to define the signals which will perturb the coils. These signals are the inputs of the simulator. The GUI panel dedicated to the coil signals is *CoilControlSignPanel* and is run with the button *Create Coil Signals. Load Coil Signals* loads the corresponding file. Again, the coil signals are saved in a *.mat* file labeled with the general name *CoilSign.* It is an array of structures, each structure being the signal applied to a coil.

The sub-panel *Simulink simulation* contains the commands to interface with the *MDSplus* database of the experiments, download the relevant data and initialize the variables needed for the Simulink simulation. The use of the buttons and the work-flow for running a Simulink simulation will be described in section 6.2.

5.2 Mathematical model of the tokamak and plasma

The open loop simulations use a mass or massless model of the plasma, while the Simulink simulations are generally performed using the massless model of the plasma. The present section illustrates the relevant equations for each model and how they are linearized. Both the open loop simulations and the closed loop simulations are linear piece-wise. The matrices are only updated at appropriate time intervals, during which a sensible displacement of the plasma can happen. This time usually corresponds with the decay time of the passive currents in the metal structures, that is a few milliseconds. The parameter *Update Matrices* in the control main panel allows the user to tune the update frequency. The plasma is modeled as a single element of current whose minor radius changes to conserve the toroidal flux, but in the Simulink simulations the coefficients of the linearized vertical equilibrium equation are computed as an average over six filaments conveniently placed in the cross-section of the plasma. The discretized electromagnetic model of the machine and a single or multi-filament plasma and the incorporation of the equilibrium equations are also discussed in detail in [18] and [19].

5.2.1 Mass simulation

The mass simulation uses the circuit equations combined with the dynamic equations for the vertical and radial displacement of the plasma and the equation of the minor radius of the plasma, which changes in order to conserve the toroidal flux through the plasma. In the case of N coils, the equations are as follows:

$$\sum_{j=1}^{N}\left(M_{ij}\frac{\partial I_j}{\partial t}\right)+M_{ip}\frac{\partial I_p}{\partial t}+\frac{\partial M_{ip}}{\partial r_p}\frac{\partial r_p}{\partial t}I_p+\frac{\partial M_{ip}}{\partial z_p}\frac{\partial z_p}{\partial t}I_p+R_iI_i=V_i\ ,\quad i=1...N$$

$$\sum_{j=1}^{N}\left(M_{pj}\frac{\partial I_j}{\partial t}+\frac{\partial M_{pj}}{\partial r_p}\frac{\partial r_p}{\partial t}I_j+\frac{\partial M_{pj}}{\partial z_p}\frac{\partial z_p}{\partial t}I_j\right)+L_p\frac{\partial I_p}{\partial t}+\frac{\partial L_p}{\partial r_p}\frac{\partial r_p}{\partial t}I_p+R_pI_p=V_p$$

$$m_p\frac{\partial^2 z_p}{\partial t^2}=-2\pi r_pI_p\cdot\sum_{j=1}^{N}B_{rj}$$

$$m_p\frac{\partial^2 r_p}{\partial t^2}=2\pi r_pI_p\cdot\left(\sum_{j=1}^{N}B_{zj}-B_{eq}\right)$$

$$a=a_0*\sqrt{\frac{r_p}{r_{p0}}}\ (a_0 \text{ and } r_{p0} \text{ are initial quantities})$$

z_p is the vertical position of the plasma; r_p is the radial position of the plasma; m_p is the mass of the plasma:

$$m_p=2\pi r_p\cdot\pi a^2\cdot n_p\cdot m_a$$

where a is the minor radius of the plasma, n_p is the particle density and m_a the atomic mass of the species used. L_p is the self inductance of the plasma and is evaluated as:

$$L_p=L_p(r_p)=\mu_0 r_p\left[\ln\left(8\frac{r_p}{a}\right)-1.75\right]$$

M_{ij} is the mutual inductance between two concentric coils i and j of radii r_i and r_j and vertical distance h_{ij}, and is evaluated through the complete elliptic integrals of the first and second types K and E:

$$M_{ij}=\mu_0*\sqrt{r_i*r_j}*\tfrac{2}{k}*\left[\left(1-\frac{k^2}{2}\right)*K\left(k^2\right)-E\left(k^2\right)\right],$$

$$k = \sqrt{\frac{4 * r_i * r_j}{\left[(r_i + r_j)^2 + h_{ij}^2\right]}}$$

The plasma is treated as a coil and M_{pj} is its coupling with the $j - th$ coil. B_{ri} is the radial component of the magnetic field produced by the current in the $i - th$ coil at the position of the plasma. It is computed through elliptic integrals:

$$B_{ri} = B_0 * \frac{g}{pi * \sqrt{Q}} * \left[\frac{1 + \alpha^2 + \beta^2}{Q - 4\alpha} * E(k) - K(k)\right]$$

$$\alpha = \frac{r_p}{r_i},\ \beta = \frac{z_p - z_i}{r_i},\ g = \frac{\beta}{\alpha},\ Q = \left[(1 + \alpha)^2 + \beta^2\right],\ k = \frac{4\alpha}{Q},\ B_0 = \frac{\mu_0 I_i}{2r_i}$$

B_{zi} is the vertical component of the magnetic field produced by the current in the $i - th$ coil at the position of the plasma:

$$B_{zi} = B_0 * \frac{1}{pi * \sqrt{Q}} * \left[\frac{1 - \alpha^2 - \beta^2}{Q - 4\alpha} * E(k) + K(k)\right]$$

$$\alpha = \frac{r_p}{r_i},\ \beta = \frac{z_p - z_i}{r_i},\ g = \frac{\beta}{\alpha},\ Q = \left[(1 + \alpha)^2 + \beta^2\right],\ k = \frac{4\alpha}{Q},\ B_0 = \frac{\mu_0 I_i}{2r_i}$$

B_{eq} is the magnetic field necessary to keep the plasma in equilibrium:

$$B_{eq} = -\frac{\mu_0}{4\pi} * \frac{I_p}{r_p} * \left[\beta_p + \frac{l_i}{2} + \ln\left(8\frac{r_p}{a}\right) - 1.5\right]$$

The voltage V_p across the plasma is introduced as an initial condition of the open loop simulator, which is used to perturb a plasma from an initial equilibrium and study its evolution. It is set to zero when the entire discharge is simulated in Simulink. The system can be linearized and put into matricial form. In the case of explicit linearization, the matrices are as follows:

$$\begin{pmatrix} \overleftrightarrow{M_{ij}} & \overrightarrow{M_{ip}} & \overrightarrow{\frac{\partial M_{ip}}{\partial z_p}} I_p & \overrightarrow{\frac{\partial M_{ip}}{\partial r_p}} I_p & 0 & 0 \\ \overrightarrow{M_{pj}} & L_p & \frac{\partial M_p}{\partial z_p} & \frac{\partial M_p}{\partial r_p} & 0 & 0 \\ \overrightarrow{0_{1\times N}} & 0 & m_p & 0 & 0 & 0 \\ \overrightarrow{0_{1\times N}} & 0 & 0 & m_p & 0 & 0 \\ \overrightarrow{0_{1\times N}} & 0 & 0 & 0 & m_p & 0 \\ \overrightarrow{0_{1\times N}} & 0 & 0 & 0 & 0 & m_p \end{pmatrix} * \begin{pmatrix} \overrightarrow{\delta I_i} \\ \delta I_p \\ \delta z_p \\ \delta r_p \\ \delta \dot{z_p} \\ \delta \dot{r_p} \end{pmatrix} = \begin{pmatrix} \left(\overrightarrow{V_i} - R_i \overrightarrow{I_i}\right) \delta t \\ (V_p - R_p I_p)\, \delta t \\ m_p z_p \delta t - \pi r_p I_p B_r \delta t^2 \\ m_p r_p \delta t + \pi r_p I_p (B_z - B_{eq}) \delta t^2 \\ -2\pi r_p I_p B_r \delta t \\ 2\pi r_p I_p (B_z - B_{eq}) \delta t \end{pmatrix}$$

where $\overleftrightarrow{M_{ij}}$ is the matrix of the self and mutual inductances, $\overrightarrow{M_{ip}}$ is the column vector of the mutual coupling of the plasma with the coils, $\overrightarrow{M_{pj}}$ is the transpose of $\overrightarrow{M_{ip}}$, $\overrightarrow{\delta I_i}$ and $\overrightarrow{I_i}$ are the column vectors of the currents in the coils, $\overrightarrow{V_i}$ is the column vector of the voltages applied to the coils and $\overrightarrow{\frac{\partial M_{ip}}{\partial z_p}}$ and $\overrightarrow{\frac{\partial M_{ip}}{\partial z_p}}$ are column vectors where:

$$\frac{\partial M_{ip}}{\partial r_p} = \frac{2\pi}{I_i} r_p B_{zi}(r_p, z_p) \ , \quad i = 1...N$$

$$\frac{\partial M_{ip}}{\partial z_p} = -\frac{2\pi}{I_i} r_p B_{ri}(r_p, z_p) \ , \quad i = 1...N$$

At last:

$$\frac{\partial M_p}{\partial r_p} = \frac{\partial L_p}{\partial r_p} I_p + \sum_{j=1}^{N} \frac{\partial M_{jp}}{\partial r_p} I_j$$

$$\frac{\partial M_p}{\partial z_p} = \sum_{j=1}^{N} \frac{\partial M_{jp}}{\partial z_p} I_j$$

The initial conditions are given by the vector:

$$[I_{10}, I_{20}, ..., I_{N0}, I_{p0}, z_{p0}, r_{p0}, \dot{z}_{p0}, \dot{r}_{p0}]$$

and the independent variables are the voltages applied to the coils:

$$[V_1, V_2, ..., V_N]$$

In the case of coils in series or anti-series the dimensionality of the problem is reduced and the problem is solved. At each step the matrix and the coefficient vector are evaluated and the linear system is solved for the increments of the physical quantities of interest.

5.2.2 Massless simulation

The massless simulation uses the circuit equations combined with the dynamic equations for the vertical and radial displacement of the plasma. In these equations the mass of the plasma is zero, which means that the plasma will always occupy an equilibrium position and the eddy currents will contribute in determining the corresponding equilibrium field. If the equilibrium is unstable, it will be unstable on the time scale of the eddy currents induced in the active and passive coils (i.e. the passive structures). In the case of N coils, the equations are as follows:

$$\sum_{j=1}^{N} \left(M_{ij} \frac{\partial I_j}{\partial t} \right) + M_{ip} \frac{\partial I_p}{\partial t} + \frac{\partial M_{ip}}{\partial r_p} \frac{\partial r_p}{\partial t} I_p + \frac{\partial M_{ip}}{\partial z_p} \frac{\partial z_p}{\partial t} I_p + R_i I_i = V_i \ , \quad i = 1...N$$

$$\sum_{j=1}^{N}\left(M_{pj}\frac{\partial I_j}{\partial t}+\frac{\partial M_{pj}}{\partial r_p}\frac{\partial r_p}{\partial t}I_j+\frac{\partial M_{pj}}{\partial z_p}\frac{\partial z_p}{\partial t}I_j\right)+L_p\frac{\partial I_p}{\partial t}+\frac{\partial L_p}{\partial r_p}\frac{\partial r_p}{\partial t}I_p+R_pI_p=V_p$$

$$-2\pi r_pI_p\cdot\sum_{j=1}^{N}B_{rj}=0$$

$$2\pi r_pI_p\cdot\left(\sum_{j=1}^{N}B_{zj}-B_{eq}\right)=0$$

$$a=a_0*\sqrt{\frac{r_p}{r_{p0}}}\ (a_0 \text{ and } r_{p0} \text{ are initial quantities})$$

The first order approximation of the last equation is:

$$\delta a=0.5*\frac{a}{r_p}*\delta r_p$$

The voltage V_p across the plasma is introduced as an initial condition of the open loop simulator, which is used to perturb a plasma from an initial equilibrium and study its evolution. It is set to zero when the entire discharge is simulated in Simulink. The force equations are linearized as follows:

$$\sum_{j=1}^{N}\left(\frac{B_{rj}}{I_j}\delta I_j+\frac{\partial B_{rj}}{\partial z_p}\delta z_p+\frac{\partial B_{rj}}{\partial r_p}\delta r_p\right)=0$$

$$\sum_{j=1}^{N}\left(\frac{B_{zj}}{I_j}\delta I_j+\frac{\partial B_{zj}}{\partial z_p}\delta z_p+\frac{\partial B_{zj}}{\partial r_p}\delta r_p\right)-\frac{B_{eq}}{I_p}\delta I_p+\left(\frac{B_{eq}}{r_p}+10^{-7}\frac{I_p}{r_p^2}\right)\delta r_p-\left(10^{-7}\frac{I_p}{r_pa}\right)\delta a=0$$

The system can be linearized and put into matricial form. In the case of explicit linearization, the matrices are as follows:

$$\begin{pmatrix} \overleftrightarrow{M_{ij}} & \overrightarrow{M_{ip}} & \overrightarrow{\frac{\partial M_{ip}}{\partial z_p}}I_p & \overrightarrow{\frac{\partial M_{ip}}{\partial r_p}}I_p & 0 \\ \overrightarrow{M_{pj}} & L_p & \frac{\partial M_p}{\partial z_p} & \frac{\partial M_p}{\partial r_p} & 0 \\ \overrightarrow{\frac{B_{rj}}{I_j}} & 0 & \frac{\partial B_r}{\partial z_p} & \frac{\partial B_r}{\partial r_p} & 0 \\ \overrightarrow{\frac{B_{zj}}{I_j}} & \frac{-B_{eq}}{I_p} & \frac{\partial B_z}{\partial z_p} & \frac{\partial B_z}{\partial r_p}+\frac{B_{eq}}{r_p}+10^{-7}\frac{I_p}{r_p^2} & 10^{-7}\frac{-I_p}{r_pa} \\ \overrightarrow{0_{1\times N}} & 0 & 0 & 0.5\frac{a}{r_p} & -1 \end{pmatrix} * \begin{pmatrix} \overrightarrow{\delta I_i} \\ \delta I_p \\ \delta z_p \\ \delta r_p \\ \delta a \end{pmatrix} = \begin{pmatrix} \left(\overrightarrow{V_i}-R_i\overrightarrow{I_i}\right)\delta t \\ (V_p-R_pI_p)\,\delta t \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

where

$$\frac{\partial B_r}{\partial z_p}=\sum_{j=1}^{N}\frac{\partial B_{rj}}{\partial z_p},\ \frac{\partial B_r}{\partial r_p}=\sum_{j=1}^{N}\frac{\partial B_{rj}}{\partial r_p},\ \frac{\partial B_z}{\partial z_p}=\sum_{j=1}^{N}\frac{\partial B_{zj}}{\partial z_p},\ \frac{\partial B_z}{\partial r_p}=\sum_{j=1}^{N}\frac{\partial B_{zj}}{\partial r_p}$$

and $\frac{\overrightarrow{B_{rj}}}{I_j}$, $\frac{\overrightarrow{B_{zj}}}{I_j}$ are row vector. The derivatives of the magnetic field are approximated as finite differences over a small distance. The default value is $5mm$. The initial conditions are given by the vector:

$$[I_{10}, I_{20}, ..., I_{N0}, I_{p0}, z_{p0}, r_{p0}, a_0]$$

and the independent variables are the voltages applied to the coils:

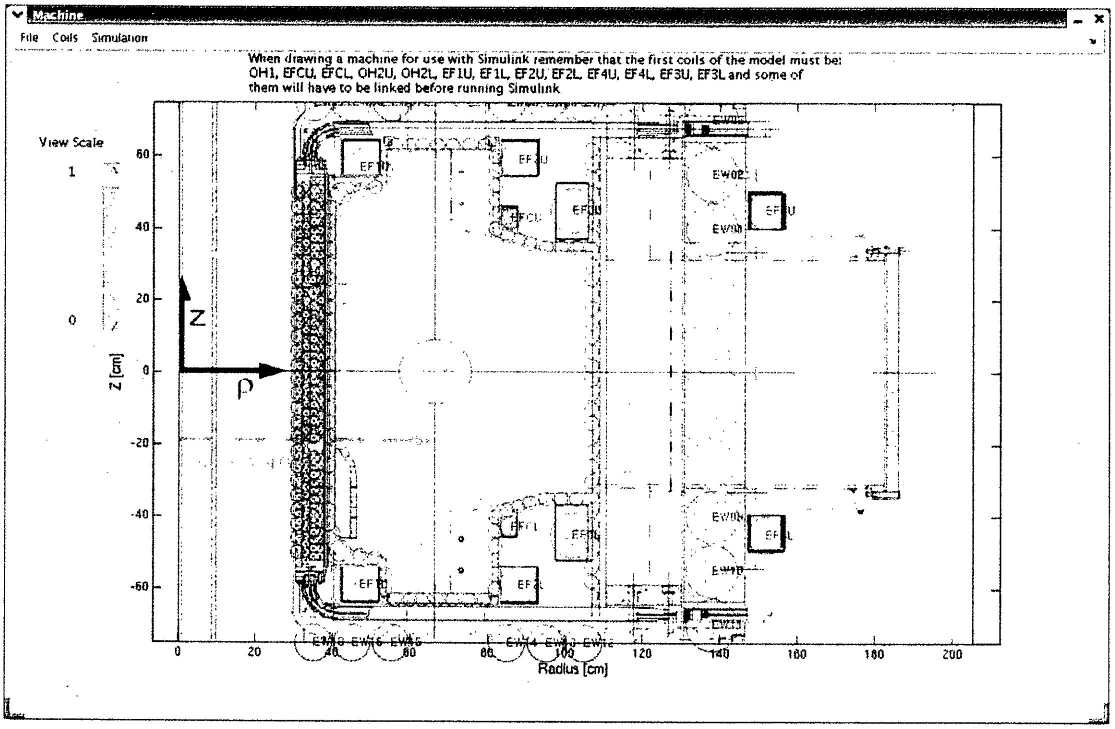
$$[V_1, V_2, ..., V_N]$$

In the case of coils in series or anti-series the dimensionality of the problem is reduced and the problem is solved. At each step the matrix and the coefficient vector are evaluated and the linear system is solved for the increments of the physical quantities of interest.

5.3 Machine GUI

The GUI *Machine* is opened with the button *Create Machine.* Another option is running the GUI directly from the Matlab console. The GUI is shown in figure 5-2. This GUI draws a section view of Alcator C-Mod in the background, which serves as a reference to place the coils of the equivalent model of the machine used in the simulations. The GUI offers a slider control for the scale of the drawing and many menus. The menu *File* collects the functions to open a model, to start a new model or to save and save-as the current model. A model is built by adding new coils. A specific GUI is used to add new coils, the *NewCoilPanel* GUI, which allows the user to fill all the fields of the data structure of a coil. These are:

- *CoilName*, which is a string and is used to uniquely identify the coil. Duplication of coil names is not allowed.
- *MajRad*, which is a double and specifies the major radius of the coil. The unit is *cm.*
- *MinRad*, which is a double and specifies the radius of a section of the coil. The unit is cm.
- *Zpos*, which is a double and specifies the vertical position of the coil, with respect to the midplane. The unit is *cm.*



When drawing a machine for use with Simulink remember that the first coils of the model must be: OH1, EFCU, EFCL, OH2U, OH2L, EF1U, EF1L, EF2U, EF2L, EF4U, EF4L, EF3U, EF3L and some of them will have to be linked before running Simulink

Figure 5-2: The GUI *Machine* with a model of the vacuum vessel, the surrounding structures and the active coils

- *NumTurns*, which specifies the number of turns of the coil.
- *CoilType*, which is a character, "a" or "p", to signify that the coil is active or passive.

Other fields are used to draw the coil:

- *DrawLabel*, which is the string that appears as a label beside the coil in the drawing.
- *DrawStyle*, not implemented.
- *Color*, the color of the coil, specified by a string which represents a RGB code.

Once the fields are filled, the user presses the button *Add Coil* and the coil is added to the temporary model which is created inside *NewCoilPanel*. The coil is also drawn in the figure. When the *Exit* button is pressed, the current model is returned to the main GUI *Machine*. Note that *Machine* doesn't know of any addition or modification of coils and continues to work with the previous model until *NewCoilPanel* is exited.

Coils can be removed from the model using the GUI *RemoveCoilPanel*. As with *NewCoilPanel*, *RemoveCoilPanel* updates the drawing but the modified model is not returned until the GUI is exited. The popup selector allows the user to choose the coil to purge among those in the model.

Coils can be also tweaked up without deleting and redrawing, using the GUI *ModifyCoilPanel*. All the fields of the coil can be changed. The popup selector allows the user to choose the coil to modify.

The menu *Simulation* of the GUI *Machine* collects the functions to evaluate the electromagnetic parameters of the model. *Calculate Inductance* calculates the matrix of self and mutual inductances of the coils in the model. Because this matrix is saved together with the array of structures which represent the coils, its matrix must be re-evaluated any time that the model is modified by adding, removing or editing some coils, before the model is saved. *Calculate Inductance* uses the formulas with elliptic integrals to evaluate the mutuals and the logarithmic approximation for the self inductances. *Calculate Resistance* calculates the array of the resistances of the coils in the model. *Magnetic Field* opens the GUI *CalcMagFieldPanel*. This GUI is used to define a grid and evaluate the corresponding magnetic field produced by the various coils for unitary currents. The resulting matrix has dimensions given by the number of points in the radial direction multiplied by the number of points in the vertical direction multiplied by the number of coils. The file is stored as a *.mat* file with the general label *MagneticSim.*

5.4 Draw Field Lines

This command opens the GUI *DrawFieldPanel*, which is shown in figure 5-3. This GUI is used to represent the magnetic topology on a user defined grid, for a generic set of user-defined currents. It uses a *MagneticSim* file, which has been previously created and saved. *DrawFieldPanel* presents many controls for scaling the drawing and the field lines. The command *Set Coil Currents*, under the menu *Currents*, opens the *CurrentsPanel* GUI, which is used to set the

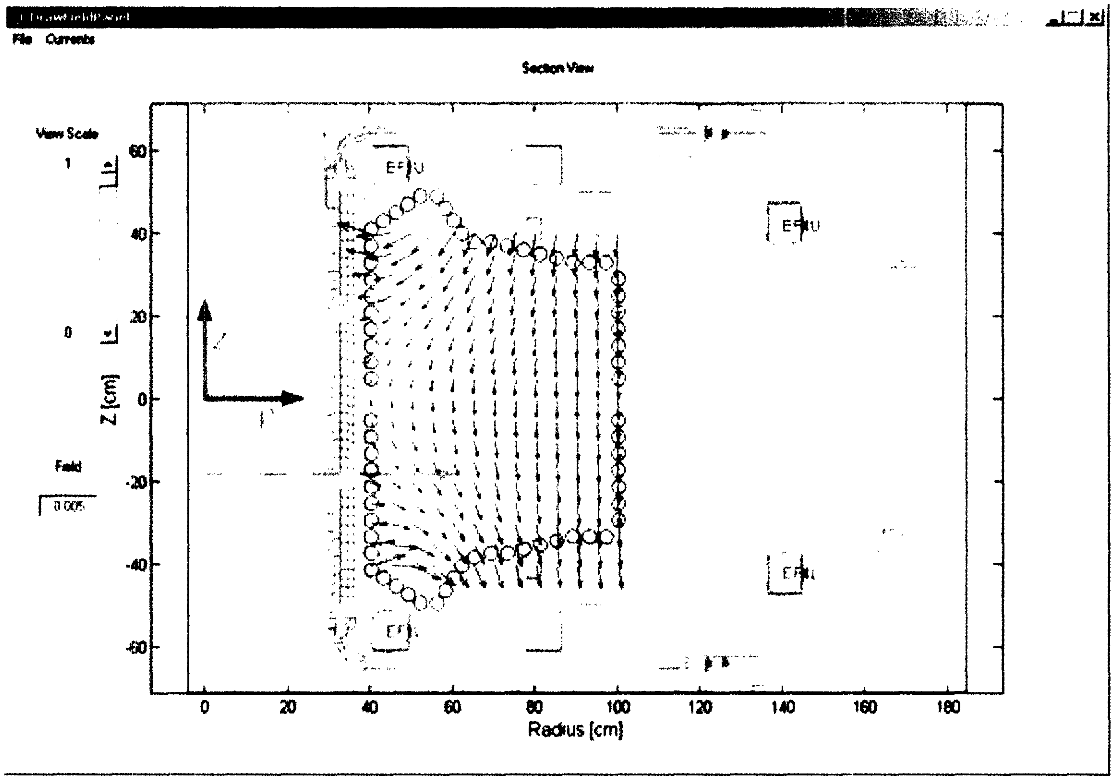


Figure 5-3: The GUI *DrawFieldPanel.* Also shown is the magnetic field on a coarse grid (5 *cm* step). The currents in this case are zero for passive coils, $5A$ for EF1s and $-3.5A$ for EF4s

currents in the active and passive coils. The currents must be introduced in the order given in the list and with the syntax of a Matlab vector. The convention is that a positive current flows into the picture.

5.5 LinkCoilsPanel GUI

This simple GUI allows the user to link two or more coils in series and couples of coils in anti-series. The panel is very similar to *CurrentsPanel.* In *LinkCoilsPanel* an array of labels is introduced which corresponds to the list of coils at the top of the GUI. The default is all independent coils. An independent coil is labeled with an "i", while "sx" is used for groups of coils in series, where x is a univocal numerical identifier, and "ax" is used for couples of coils in anti-series, where x is a univocal numerical identifier. Similarly, "px" is used for coils in parallel.

5.6 PlasmaPanel GUI

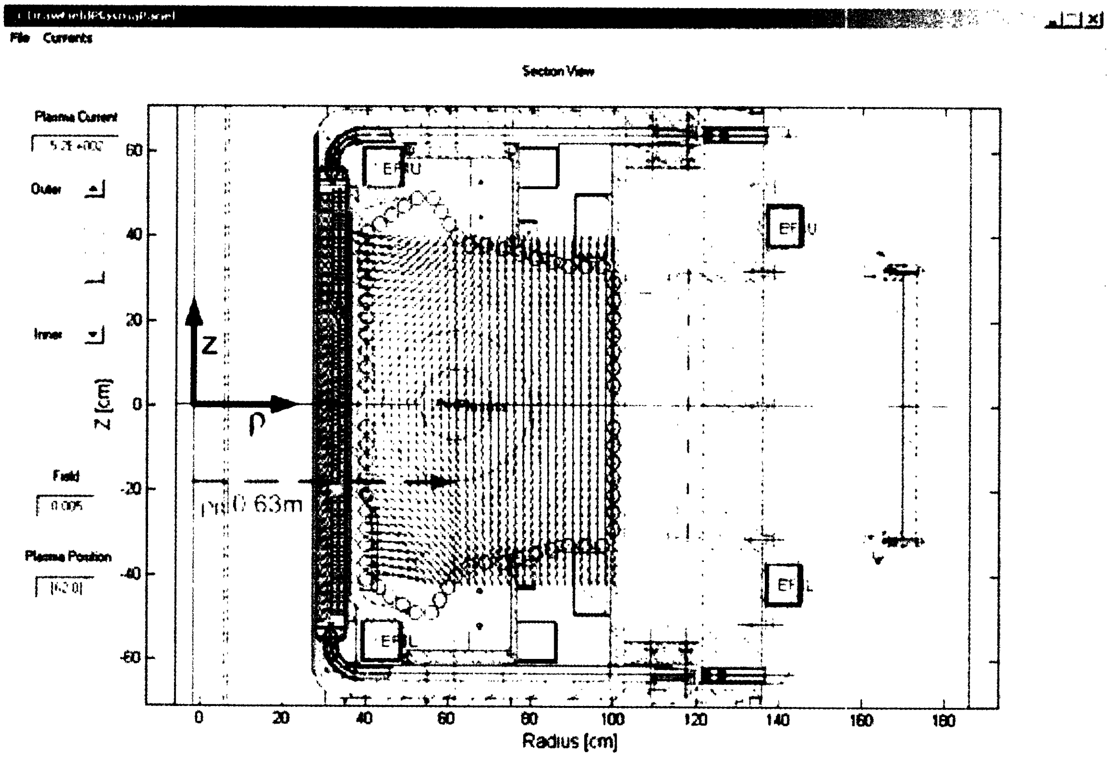
The *PlasmaPanel* GUI allows the user to define the parameters of the model of the plasma. In the present implementation of Alcasim, the only model available is a simple single current plasma with the poloidal beta and the internal inductance (l_i) held constant during the simulation. The fields of the data structure representing the plasma are:

- *PoloidalBeta.*
- *InternalInductance.*
- *AspectRatio.*
- *MinorRadius*, with units *cm.*
- *Resistivity*, with units $\Omega * cm$.
- *Density*, with units $particles/m^3$.

A plasma model is saved as a *.mat* file with the generic label *Plasma.*

5.7 DrawFieldPlasmaPanel GUI

With this GUI, which is illustrated in figure 5-4, the user finds an equilibrium position for the plasma, given a set of currents in the coils (active and passive). The equilibrium position is evaluated by minimizing the radial field for a user-defined radial position of the plasma (this operation determines the vertical position as well), then the vertical component of the magnetic field in that position is read and the plasma current is calculated so that the plasma will stay in equilibrium. When more than one zeros of the radial field are present, the one closest to the midplane is chosen. Because the initial equilibrium is evaluated on a grid as a procedure of minimization of the radial field, the finer is the grid, the smaller is the error in defining an initial equilibrium. Initial equilibria at the boundary of the user-defined grid should be avoided. The GUI uses the *MagneticSim* file loaded in the *ControlMainPanel* GUI. The



Plasma Current: 5.2E+002

Field: 0.005

Plasma Position: [62 0]

Figure 5-4: The initial equilibrium in the GUI *DrawFieldPlasmaPanel*

slider on the left allows the user to select a radial position for the plasma, while the edit fields display the plasma position and the plasma current at equilibrium, for a user-defined set of coil currents. The currents can be changed with the GUI *CurrentsPanel*, which is accessible from the menu *Currents.* The data at equilibrium are returned to *ControlMainPanel* when *DrawFieldPlasmaPanel* is exited.

5.8 CoilControlSignalPanel GUI

This GUI is used to define the external perturbations applied to the coils in the open loop non-linear simulations. The coil signals are saved in a *.mat* file with the general label *CoilSign.* A *CoilSign* file is an array of structures. Each structure defines the signal for one particular coil and contains the following fields:

- *CoilName.*
- *SignType*, which, in the present implementation, can be "Step", "Sine", "Ramp", "Sawtooth" or "White Noise".
- *Ampl*, which is the maximum amplitude of the signal (in the case of white noise, it defines the RMS value per unit bandwidth).
- *FreqDuration*, which defines the frequency/time duration/bandwidth of the signal, according to the particular type of signal selected.
- *StartTimePhase*, which defines the start time for a ramp and sawtooth and the initial phase for a sine wave.

The coils of the model are passed to *CoilControlSignalPanel* as an input. *CoilControlSignalPanel* allows the user to save *CoilSign* files and to retrieve previously saved files for editing.

5.9 SimulationMoviePanel GUI

The GUI *SimulationMoviePanel* allows the user to view single frames of a plasma discharge: the coil currents, the magnetic field, the poloidal field generated by the plasma, and the plasma, are all drawn together. This GUI is illustrated in figure 6-3. It includes a number of functions. *Start*, *Step On* and *Step Back* are used to move between the frames. *Skip Frame* is used to sample only some of the frames. The edit displays report relevant parameters of the plasma and the time of a particular frame. *SimulationMoviePanel* is called from *ControlMainPanel*, using the button *Movie*, after a *Simulation* file is produced or loaded by the button *Load Simulation.*

Chapter 6

Alcasim User's Guide

In this chapter we describe how Alcasim can simulate a plasma discharge with a user-defined model of the tokamak. We will demonstrate the use of both the Matlab open loop simulator and the Simulink simulator. The difference between them is that the Simulink simulator incorporates the Matlab simulation code of the tokamak and the plasma in a comprehensive model also comprising the power supplies, the diagnostics and a functional equivalent of the DPCS digital control system.

6.1 Open Loop Matlab simulations

6.1.1 Building the Machine

The first step is running the script *LaunchScript.m*, which launches *ControlMainPanel* . The script also contains the definition of common variables useful for the Simulink simulations. It is convenient to have all the files of Alcasim in the same directory. The GUI *ControlMainPanel* looks as in figure 5-1, where some fields have already been filled. The next step is building a machine. We then run the GUI *Machine* and add, modify and remove coils to a new or retrieved model as needed. It is important to remember that the changes applied when adding, removing or modifying coils take place only after the corresponding GUIs are exited. Once the model is complete, we must update the matrix of inductances and the array of resistances using the commands *Calculate Inductance* and *Calculate Resistance* in the *Simulation* menu of *Machine*, then we save the model with a custom name or the name of the original file. Figure

5-4 illustrates a simple example with an approximate model of the vacuum vessel and the EF1 and EF4 coils. In this picture, from the GUI *DrawFieldPlasmaPanel*, the field lines and the plasma are also shown. We evaluate the magnetic field using *CalcMagFieldPanel*, which is launched by *Magnetic Simulation* in the *Simulation* menu or by *Calculate MagSim* in the *ControlMainPanel* GUI. The model of the machine must be loaded in *ControlMainPanel* and the name of the file is displayed in the corresponding field. Many buttons, which operate on the model of the machine, and were initially inactive, are now functional. One of these is the *Link Coils* button, which allows the user to link groups of coils in series (this option is particularly useful for modeling the large OH coils) or couples of coils in anti-series (this option is necessary to model the EFC coils). In the present example, we don't link any coils, so we leave the default setting "i" (independent) for every coil.

6.1.2 Defining the Plasma Model

The GUI *PlasmaPanel* is launched from the GUI *ControlMainPanel*, using the button *Create Plasma.* In the present example we use the following parameters:

- *PoloidalBeta* $= 0.1$
- *InternalInductance* $= 1$
- *AspectRatio* $= 3$
- *MinorRadius* $= 21\ cm$
- *Resistivity* $= 1.6 * 10^6\ \Omega * cm$
- *Density* $= 10^{20}\ /m^3$

Once the single current model of the plasma has been saved, we load the file in *ControlMainPanel.* The name of the file is displayed.

6.1.3 Evaluating an Initial Equilibrium

The initial equilibrium consistent with the force balance equations for a certain set of coil currents is evaluated through a convenient graphical user interface *DrawFieldPlasmaPanel.* A

MagneticSim file must be loaded before *DrawFieldPlasmaPanel* is lauched. We load the file previously created, with the fine $2cm$-step grid, and set the currents $5A$ for the EF1s and $-3.5A$ for the EF4s. Then we choose an initial radial position, in this example 62 cm, and the program evaluates the vertical position (which in this symmetric case must be 0) and the plasma current at equilibrium. We can change the currents through the *CurrentsPanel* GUI previously discussed. Figure 5-4 shows the final result of these operations. We *Exit&Save* and the equilibrium parameters are reported in the corresponding fields. The voltages across the coils are evaluated using the coils resistances.

6.1.4 Setting the options for the open loop simulations

A *CoilSign* file must be loaded, before a simulation is run. This file contains the description of the signals which perturb the voltages sustaining the equilibrium currents in the coils. In the present example, we decide to apply a very small step perturbation, namely 1% of the DC value, to the upper EF1 coil, and we expect to see the vertical instability of the plasma developing as an exponential drift toward the wall, with the time constant determined by the resistivity of the coils which form the vacuum vessel. We launch *CoilControlSignalPanel* and set the EF1U signal as a step with start time $0.1s$ and amplitude $0.004V$. Then we input the time interval and the time step in the corresponding fields of *Open Loop Matlab non-minear sim* and run the simulation. We may try both the massless and the massive simulation for comparison. Once a simulation is completed, it becomes the current simulation and the data from previous simulations are lost, if they haven't been saved using the button *Save Simulation.*

6.1.5 Viewing simulation results from the open loop Matlab simulator

After a simulation is completed, or after it's been loaded using the button *Load Simulation*, it can be viewed as single plots or movie frames of the plasma discharge. The *Plot* button plots the single physical quantities. Each Matlab figure contains four plots, each of which is editable through the standard Matlab interfaces *Figure Properties* and *Axes Properties.* Each figure can be saved as a Matlab *.fig* file. Figures 6-1 and 6-2 show the evolution of some quantities during our massless and massive simulations. A movie frame drawn by *SimulationMoviePanel* looks like figure 6-3 or 6-4, where two instants are illustrated which correspond to the initial

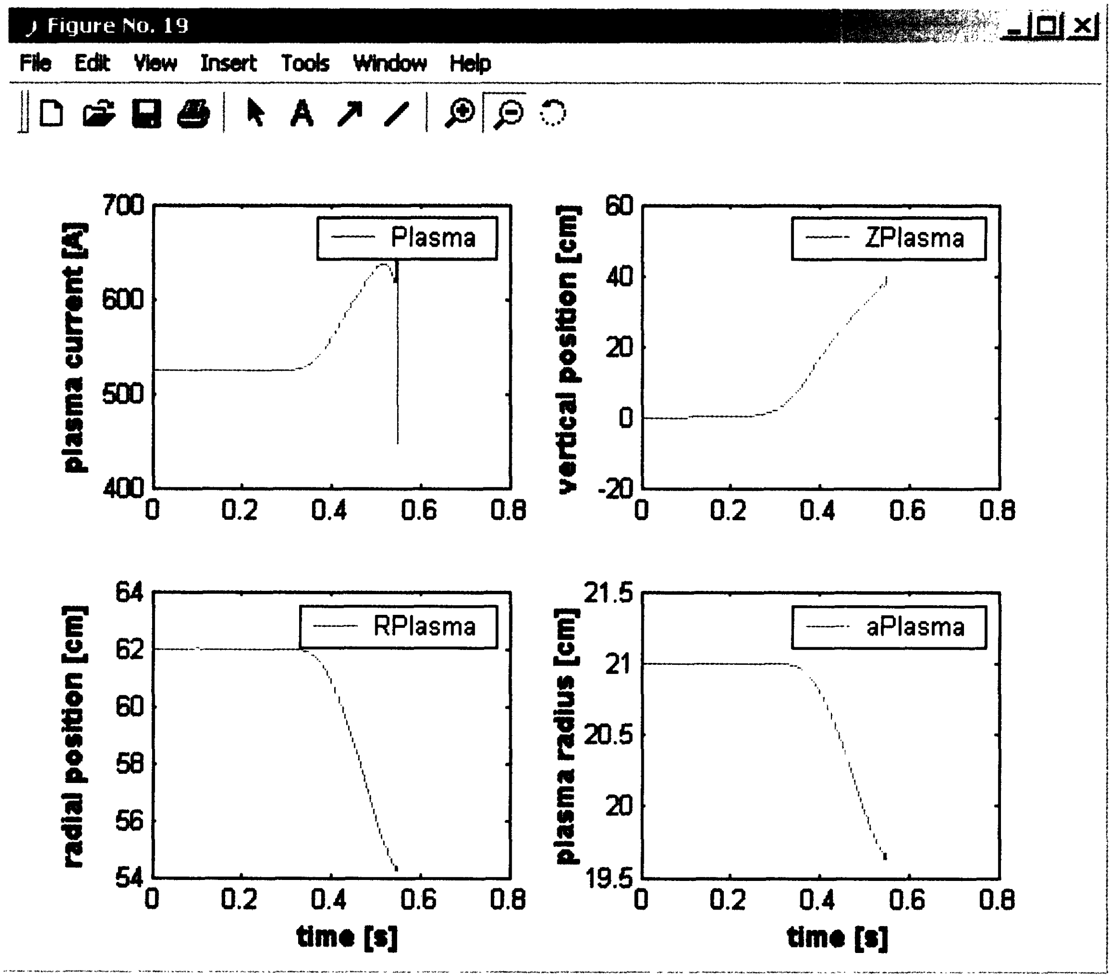


Figure 6-1: Evolution of some physical quantities in the massless simulation

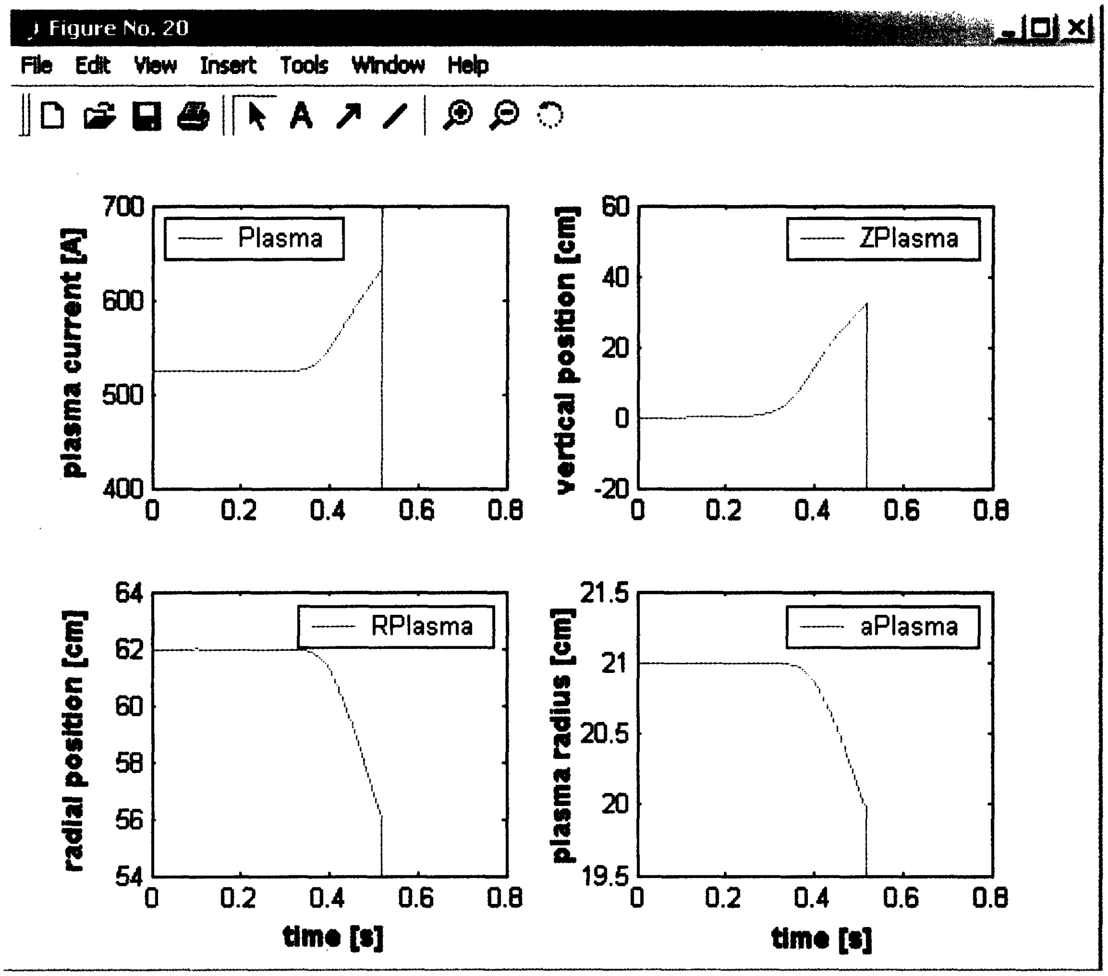
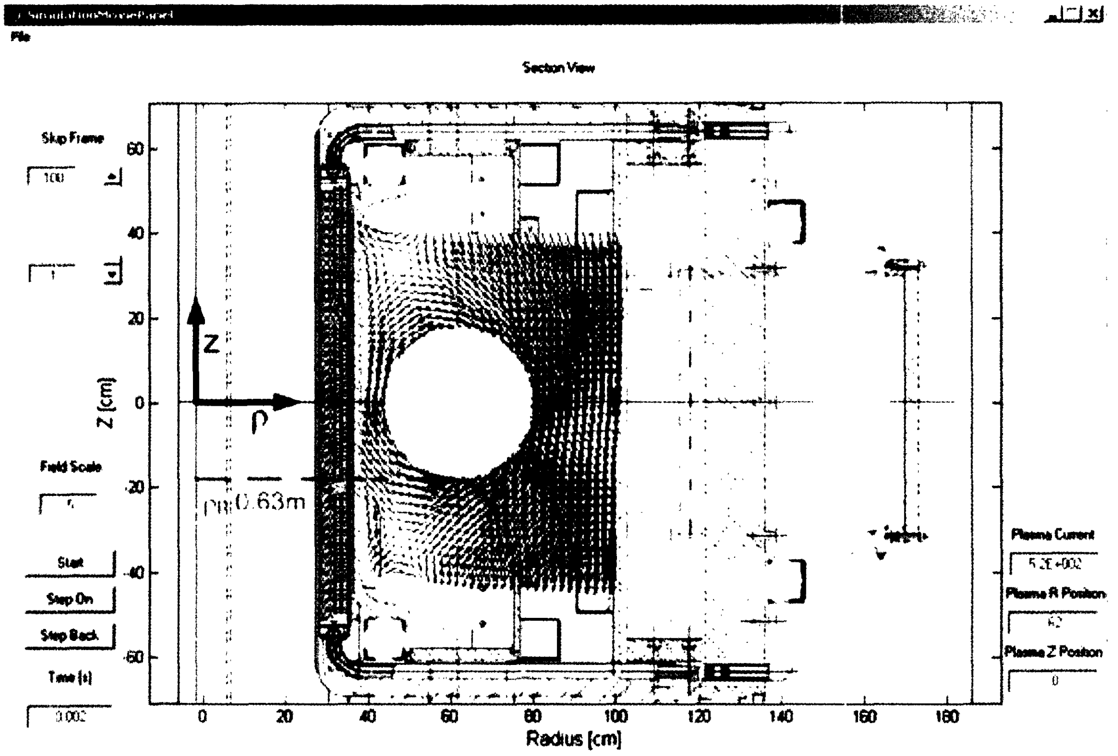


Figure 6-2: Evolution of some physical quantities in the massive simulation. The initial density of the plasma in this simulation is $10^{22}/m^3$

equilibrium and the plasma disruption against the upper wall.



Skip Frame 100

1

Field Scale 5

Time (s) 0.002

Plasma Current 5.2E+002

Plasma R Position 62

Plasma Z Position 0

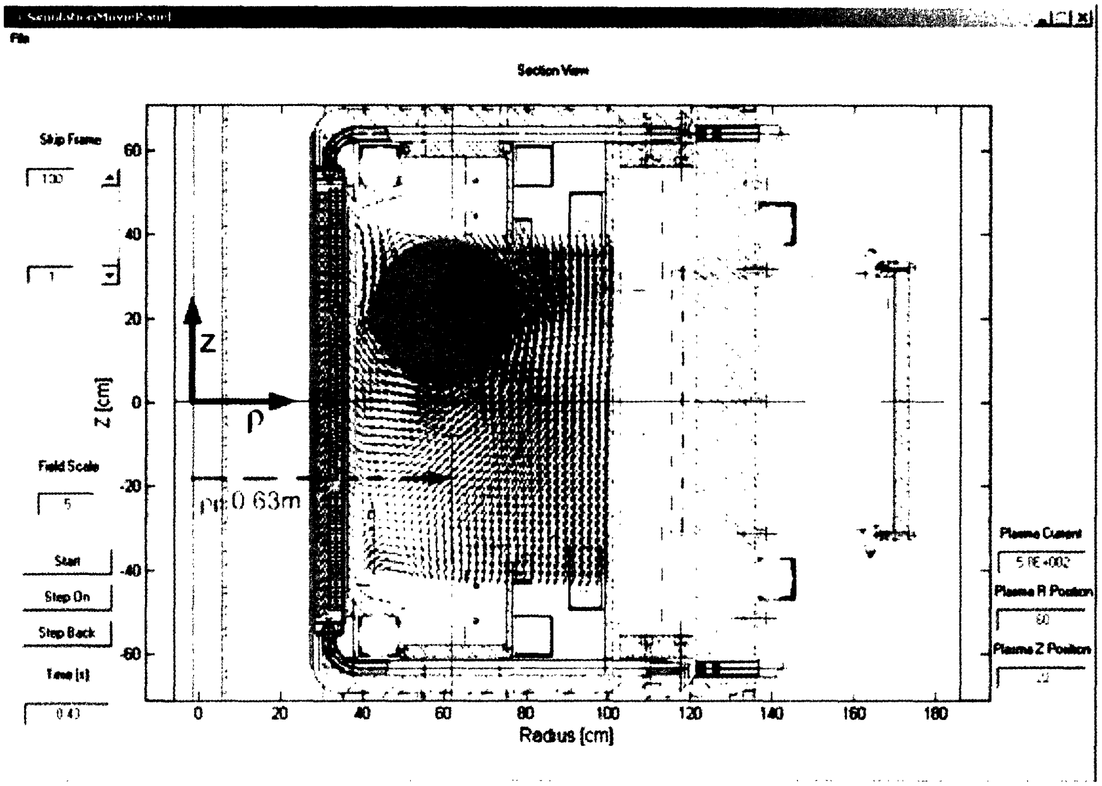
Figure 6-3: The initial equilibrium in our simulations

6.2 Simulink simulations

The Simulink block diagram is shown in figure 6-5. The blocks corresponding to the tokamak, the diagnostics and DPCS are Matlab sFunctions, namely *sCMod.m*, *Diagnostics.m* and *sControl.m*. The tokamak sFunction implements the same linearized massless model described in section 5.2. At appropriate time intervals the relevant matrices are updated and the derivatives of the state vector are computed as:

$$\frac{\partial \overrightarrow{X}}{\partial t} = \left(\overleftrightarrow{A}\right)^{-1} * \left[\overrightarrow{V} - \overrightarrow{R} * \overrightarrow{I}\right]$$

However, we found that the single-filament model of the plasma is too unstable for use in the



Skip Frame: 100

1

Field Scale: 5

Time [s]: 0.43

Plasma Current: 5.8E+002

Plasma R Position: 60

Plasma Z Position: 22

Figure 6-4: The disruption of the plasma toward the upper wall in our simulations

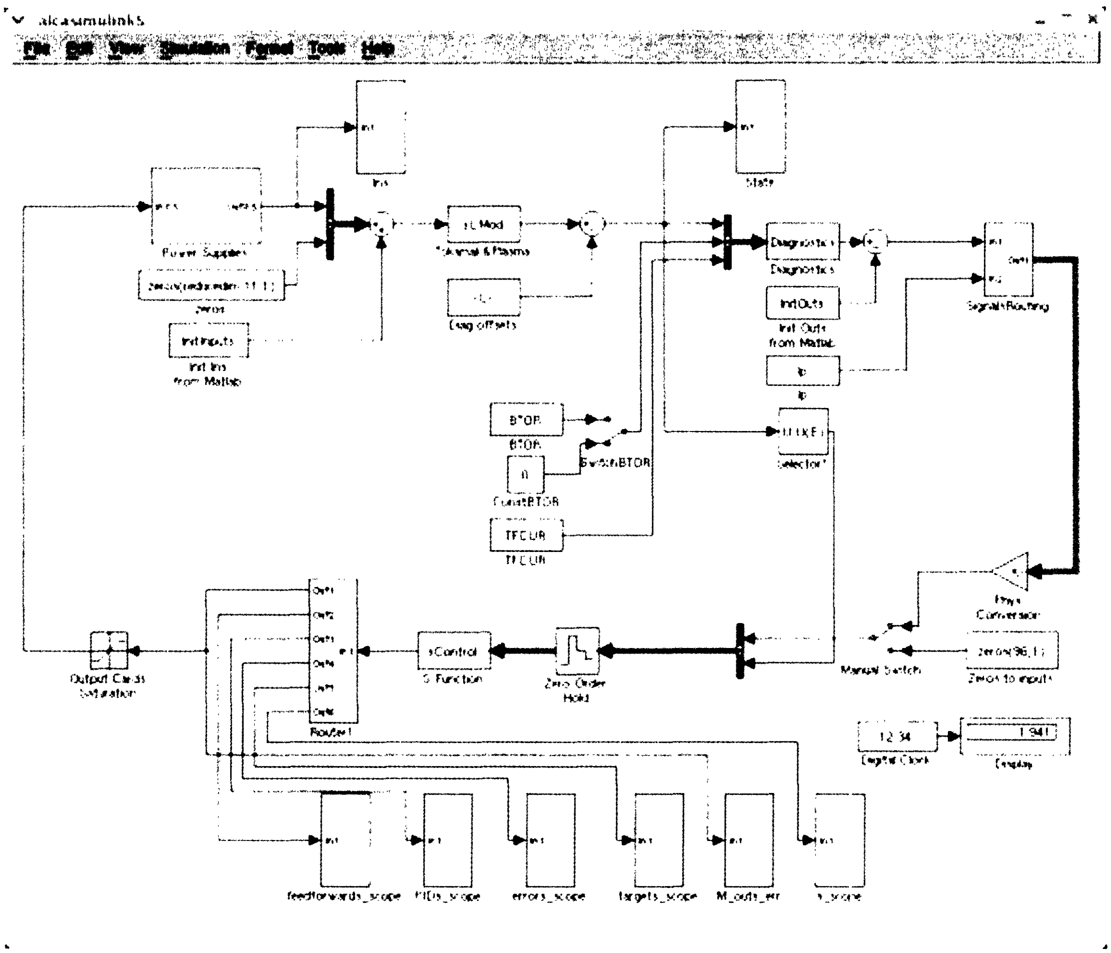


Figure 6-5: The Simulink block diagram. The power supplies subsytem contains the models of the power supplies. In the simplest case, they are single pole systems, with saturation blocks to limit their output voltages. The blocks implementing the tokamak, the diagnostics and the control system are Matlab sFunctions

Simulink simulations, because it does not average the field index over the full plasma current extent. Therefore the coefficients of the linearized vertical equilibrium equation are computed as an average over six filaments conveniently placed in the cross-section of the plasma. The position of the filaments can be changed with the parameter *Plasma Elongation* in the control main panel. This parameter expresses the ratio between the height and the width of an ideal rectangular box containing the plasma elements. The width of the box is actually fixed at half the plasma radius, while the height is changed from the user interface. The default current profile is parabolic squared:

$$J(r) = J_0 \left(1 - \frac{r^2}{(\rho a)^2}\right)^2$$

where a is the radius of the plasma and ρ is an effective parameter used to tune the distribution of the current over the six filaments. ρ is changed from the user interface with the entry *Current Distr Radius.* The relative weights of the filaments w_j are normalized such that they sum to 1. The linearized vertical equilibrium equation is then:

$$\sum_{i=1}^{numcoils} \sum_{j=1}^{numfilaments} \left(\frac{B_{rij}}{I_i}\delta I_i + \frac{\partial B_{rij}}{\partial z_p}\delta z_p + \frac{\partial B_{rij}}{\partial r_p}\delta r_p\right) w_j = 0$$

We can rewrite this equation to show how the coefficients of the linear problem are averaged over the six filaments:

$$\sum_{i=1}^{numcoils} \left(\sum_{j=1}^{numfil} w_j \frac{B_{rij}}{I_i}\right) \delta I_i + \left(\sum_{j=1}^{numfil} w_j \frac{\partial B_{rij}}{\partial z_p}\right) \delta z_p + \left(\sum_{j=1}^{numfil} w_j \frac{\partial B_{rij}}{\partial r_p}\right) \delta r_p = 0$$

The output vector $\vec{Y}$ of the tokamak sFunction is the state:

$$\vec{Y} = \vec{X}$$

and contains all the currents in the active and passive coils and the information about the plasma, that is its current, its radial and vertical position and its minor radius. The information on the position of the plasma is used by the diagnostics sFunction to update the coefficients which express the coupling of the plasma current with the magnetic diagnostics. In order to avoid useless computation and speed up the simulation, the matrix $\overleftrightarrow{A}$ and the diagnostics matrix are updated only at time intervals during which a sensible displacement of the plasma occurs. An indicative value is given by the decay times of the currents in the walls, usually a few ms. The update frequency of the matrices is set with the entry *Update Matrices* in

ControlMainPanel. The time saving is significant, especially in the case of models with a dense placement of the coils, where the matrix $\overleftrightarrow{A}$ is close to singularity and the time steps of the simulation are necessarily small.

The diagnostics matrix is calculated by the scripts *Flux_ Full.m*, *Flux_ Partial.m* and *BP_ Coils.m*, before the Simulink simulation begins. These scripts are run when the button *Load Diagnostics* is pressed. The diagnostics sFunction multiplies the input vector of currents by the diagnostics matrix and outputs the full fluxes, the partial fluxes, the magnetic field at the locations of the poloidal pick-up coils and the currents in the active coils. Other information, which is used by DPCS but is not included in our model, is made available through special blocks which read from the Matlab workspace. For example the toroidal field BTOR in a particular shot is read from the tree and written in the Matlab workspace in an appropriate format, then it is interpolated during the simulation and fed-through the diagnostics block to the input of DPCS. The control algorithm implemented in *sCMod.m* is a simple MIMO PID controller and is a functional equivalent of the IDL code controlling the real machine. The core of the code is reported here for reference:

```
i = floor((t-StartTime)/T)+1;
for j=1:16
  a(j) = A_matrix(j).signals.values(i,:)*u;
  p(j) = (References10.signals.values(i,j)*10 + ReferencesIP.signals.values(i,j)*...
    u(56))*P_outs(j).signals.values(i);
end
y = p + a;
y1 = (y - y_last)./Td1;
for j=1:16
  y2(j) = (y2_last(j) + T*y(j)*Is.signals.values(i,j))*int_masks(j).signals.values(i);
end
y_last = y;
y2_last = y2;
z = y2 + Ds.signals.values(i,:).*y1 + Ps.signals.values(i,:).*y;
```

```
for j=1:10
  w(j) = M_matrix(j).signals.values(i,:)*z';
  v(j) = V_outs(j).signals.values(i);
end
sys = [(v + w),v,z,y,p,a];
```

Note that only the output signals controlling the power supplies are used in the simulations (M_outs 1 through 9 and M_out 11). The real system DPCS has currently 32 output channels, controlling the power supplies and additional actuators. The Simulink model also comprises single pole models of the power supplies, with saturation blocks to limit their output voltages, and a simple implementation of the fizzle detector, which senses the presence of the plasma and forces the currents in the active coils to zero, if no plasma has been formed in a particular time window. The model used for the OH1 power supply is illustrated in figure 6-6.

The Simulink block diagram can be updated with the usual tools provided with Simulink, that is the blocks in the Simulink block library and the blocks reserved for user defined functions, whether Matlab scripts or more complex sFunctions. Similarly, the simulation of the model uses the Simulink engines and optimization tools and we won't spend time on this point. The workflow already described in the case of a Matlab open loop simulation does not differ significantly from the case of a Simulink simulation, but the quantities needed for the Simulink simulation must be read from the database of the real experiment and must be formatted and written in the Matlab workspace, before the simulation begins. The buttons in the sub-panel *Simulink simulation* connect with the *MDSplus* database, which stores the information of the real experiment, and load the relevant matrices, waveforms and parameters. In particular, *Load As* reads the observer matrices, *Load Ms* reads the controller matrices, *Load Ps*, *Load Ds* and *Load Is* read the PID gains, *Load P_ outs* and *Load V_ outs* read the reference and feed-forward waveforms and *Load PS parameters* reads information on the power supplies and their operation during the shot. The DPCS cycle time and the start and stop times are input in the corresponding fields, and the waveforms and matrices are resampled on the DPCS timebase with the button *Build Waveforms*. This operation is equivalent to what happens in the IDL procedure *dpcs_ init* of the DPCS code. All the data and data structures are finally formatted to work with Simulink and are written in the Matlab workspace using *Init Model Param*. For

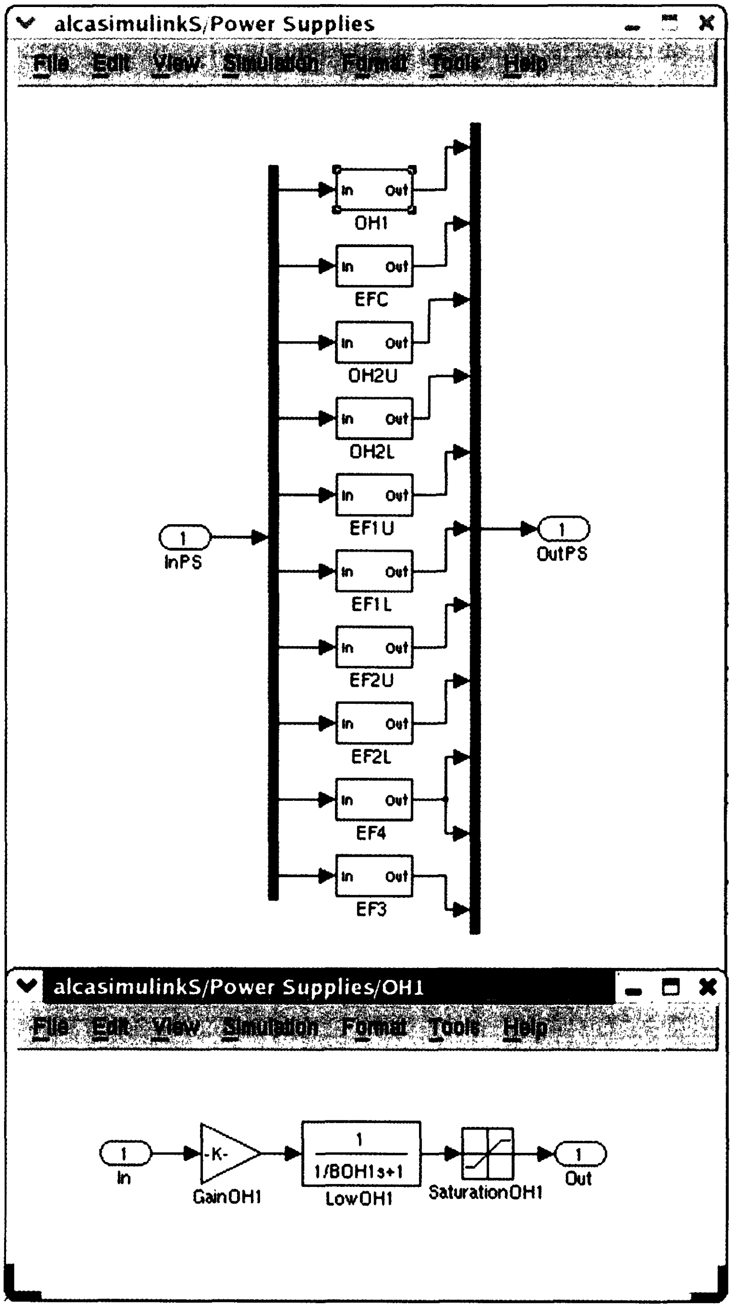

alcasimulinkS/Power Supplies
File Edit View Simulation Format Tools Help
1
InPS
In Out
OH1
EFC
OH2U
OH2L
EF1U
EF1L
EF2U
EF2L
EF4
EF3
1
OutPS
alcasimulinkS/Power Supplies/OH1
File Edit View Simulation Format Tools Help
1
In
-K-
GainOH1
1
1/BOH1s+1
LowOH1
SaturationOH1
1
Out

Figure 6-6: The subsystem of the power supplies in the Simulink model and a detail of the implementation of the OH1 supply

improving the efficiency of the code, the variables are global and are initially defined in the script *LaunchScript.m.*

When a fizzle discharge is simulated, the plasma turn-on time is chosen greater than the stop time. Otherwise, at the turn-on time the plasma is introduced in the model. The breakdown and the initial phase of the current rise is particularly difficult to model. We overcome the problem by introducing a fixed coil to represent the plasma at the beginning of the current rise. The coil is not allowed to move, but its current can vary as a consequence of the inductive drive from the active coils. Once the initial phase is over (usually after $10ms$, corresponding to a breakdown current of $100kA$), we let the plasma go and control it. The fields *Turn on plasma* and *Free plasma* in the user interface allow to set the corresponding times. The buttons *Inputs Errors* and *A_ out Errors* plot the errors between the real signals and the simulated ones at the input of DPCS and at the output of the observer matrix.

Chapter 7

Alcasim simulation results

7.1 Matlab open loop simulations

In this chapter we discuss the simulation results from a physical viewpoint. First, we discuss some static data, that is we compare the matrix of the mutual and self-inductances of the active coils calculated by Alcasim with the matrix stored in the tree and used for calculations on Alcator C-Mod. Secondly, we analyze a static equilibrium where the currents are copied from a real shot, to verify that our simulator outputs the right value for the plasma current. In a following section, we simulate the evolution of perturbed equilibria of the plasma in cases where the number of coils is small and the geometry of the problem is simple.

7.1.1 Static Analysis

The values of the mutual and self-inductances of the active coils of Alcator C-Mod are reported in the following table The source is the full model of the vessel described in [18]. The values for OH1, OH2U, OH2L, EF1U, EF1L, EF2U, EF2L, EF3, EF4U, EF4L and EFC are reported in order. Note that two EF3 coils are connected in series and two EFC coils are connected in anti-series. Note also that the EF4 coils are connected in parallel in the machine.

$$M_{rect}=\begin{pmatrix}
0.00580 & 0.000933 & 0.000929 & 0.00169 & 0.00168 & 0.00169 & 0.00168 & 0.00337 & 0.00146 & 0.00146 & -7.57E-7 \\
0.000933 & 0.000569 & 3.68E-5 & 0.00102 & 9.71E-5 & 0.000655 & 0.000195 & 0.000787 & 0.000407 & 0.000268 & -5.79E-5 \\
0.000929 & 3.68E-5 & 0.000569 & 9.71E-5 & 0.00102 & 0.000195 & 0.000655 & 0.000787 & 0.000268 & 0.000407 & 5.79E-5 \\
0.00169 & 0.00102 & 9.71E-5 & 0.0112 & 0.000300 & 0.00417 & 0.000702 & 0.00388 & 0.00219 & 0.00117 & -0.000362 \\
0.00168 & 9.71E-5 & 0.00102 & 0.000300 & 0.0112 & 0.000702 & 0.00417 & 0.00388 & 0.00117 & 0.00219 & 0.000362 \\
0.00169 & 0.000655 & 0.000195 & 0.00417 & 0.000702 & 0.0246 & 0.00188 & 0.0150 & 0.00810 & 0.00370 & -0.00132 \\
0.00168 & 0.000195 & 0.000655 & 0.000702 & 0.00417 & 0.00188 & 0.0246 & 0.0150 & 0.00370 & 0.00810 & 0.00132 \\
0.00337 & 0.000787 & 0.000787 & 0.00388 & 0.00388 & 0.0150 & 0.0150 & 0.0508 & 0.0155 & 0.0155 & -1.16E-10 \\
0.00146 & 0.000407 & 0.000268 & 0.00219 & 0.00117 & 0.00810 & 0.00370 & 0.0155 & 0.0494 & 0.100 & -0.000429 \\
0.00146 & 0.000268 & 0.000407 & 0.00117 & 0.00219 & 0.00370 & 0.00810 & 0.0155 & 0.100 & 0.0494 & 0.000429 \\
-7.57E-7 & -5.79E-5 & 5.79E-5 & -0.000362 & 0.000362 & -0.00132 & 0.00132 & -1.16E-10 & -0.000429 & 0.000429 & 0.000749
\end{pmatrix} H$$

The values calculated by Alcasim, using the circular cross-section formulas and the necessary series and anti-series connections, are:

$$M_{Alcasim}-\begin{pmatrix}
0.00505 & 0.000813 & 0.000813 & 0.00139 & 0.00139 & 0.00141 & 0.00141 & 0.00300 & 0.00130 & 0.00130 & 0 \\
0.000813 & 0.000498 & 4.21E-5 & 0.000909 & 9.46E-005 & 0.000594 & 0.000186 & 0.000772 & 0.000396 & 0.000264 & -5.41E-5 \\
0.000813 & 4.21E-5 & 0.000498 & 9.46E-005 & 0.000909 & 0.000186 & 0.000594 & 0.000772 & 0.000264 & 0.000396 & 5.41E-5 \\
0.00139 & 0.000909 & 9.46E-005 & 0.00906 & 0.000255 & 0.00355 & 0.000592 & 0.00347 & 0.00197 & 0.00104 & -0.000320 \\
0.00139 & 9.46E-005 & 0.000909 & 0.000255 & 0.00906 & 0.000592 & 0.00355 & 0.00347 & 0.00104 & 0.00196 & 0.000320 \\
0.00141 & 0.000594 & 0.000186 & 0.00355 & 0.000592 & 0.0194 & 0.00157 & 0.0132 & 0.00722 & 0.00325 & -0.00115 \\
0.00141 & 0.000186 & 0.000594 & 0.000592 & 0.00355 & 0.00157 & 0.0194 & 0.0132 & 0.00325 & 0.00722 & 0.0011 \\
0.00300 & 0.000772 & 0.000772 & 0.00347 & 0.00347 & 0.0132 & 0.0132 & 0.0471 & 0.0143 & 0.0143 & 0 \\
0.00130 & 0.000396 & 0.000264 & 0.00197 & 0.00104 & 0.00722 & 0.00325 & 0.0143 & 0.0432 & 0.00918 & -0.000410 \\
0.00130 & 0.000264 & 0.000396 & 0.00104 & 0.00196 & 0.00325 & 0.00722 & 0.0143 & 0.00918 & 0.0432 & 0.000410 \\
0 & -5.41E-5 & 5.41E-5 & -0.000320 & 0.000320 & -0.00115 & 0.0011 & 0 & -0.000410 & 0.000410 & 0.000647
\end{pmatrix} H$$

The error is always within 10%, with bigger errors for the self-inductances of single coils. This result doesn't come unexpected, because the actual shape of the coils is more influential in these calculations. The resistances of the coils are:

$$\mathrm{R}_{Alcator}=\begin{pmatrix} 0.00301 & 0.000659 & 0.000659 & 0.0108 & 0.0108 & 0.0203 & 0.0203 & 0.0276 & 0.0348 & 0.0348 & 0.00256 \end{pmatrix}\Omega$$

The resistances calculated by Alcasim, using the copper resistivity at liquid nitrogen $0.4 * 10^{-6}\ \Omega * cm$, and a correction factor 0.64177 for the portion of the cross section occupied by the insulation, are:

$$\mathrm{R}_{Alcasim}=\begin{pmatrix} 0.00372 & 0.000733 & 0.000733 & 0.0108 & 0.0108 & 0.0182 & 0.0182 & 0.0366 & 0.0342 & 0.0342 & 0.00222 \end{pmatrix}\Omega$$

The agreement is very good, but coarser for the EF3 and OH coils. In any case, the coils resistances can be tuned individually from the user interface. To verify the static equilibrium solver, we applied typical values of the currents in the active coils: $OH1 : -20kA$, $OH2 : -10kA$, $EF2 : +2kA$, $EF3 : -9kA$, $EF4 : -2.5kA$. We obtained the reasonable plasma equilibrium current of $1.3MA$, when the plasma major radius was 62 cm, the minor radius was 21 cm and $\beta_\theta = 0.1$, $l_i = 1$.

7.1.2 Transient Analysis

In this section we present the results from the simulations of perturbed plasma equilibria where the plasma evolves from an equilibrium to a new equilibrium or to a disruption. Note that all the simulations presented in this section were run with very small currents in the coils and the plasma, for the purpose of assessing the physical meaningfulness of the results obtained from the massive and massless simulators, rather than simulating realistic operating conditions. Both stable and unstable field curvatures are investigated and resistive and ideal instability regions are identified in the latter case.

Vertically Stable Plasma, $n_c = 0.40$

To simulate this case, only the EF3 coils are present and they are independent. We use the copper resistivity at room temperature $1.6 * 10^{-6}\ \Omega * cm$. The EF3 coils resistance is 0.0470Ω and the corresponding time constant is $0.436s$. The field curvature is positive, providing vertical stability, but the plasma can become radially unstable, especially in the region near the coils. We study three different cases. In the first two, we apply the same small step perturbation to both coils in order to produce a compression or expansion of the plasma. The step amplitude is about 10% of the static DC voltage. In the third case, we apply an asymmetric perturbation, that is a 5% amplitude step is applied to each coil, but with opposite signs.

1. *Plasma Compression*

The plasma evolution in this case is plotted in figures 7-1 and 7-2. As we can see, the plasma radius decreases in order to conserve the toroidal flux though the plasma. The plasma major radius decreases, the vertical position remains substantially unchanged, apart from a small numerical noise. The plasma current slightly increases. The massive simulation shows oscillations due to numerical noise and convergence problems, given the small mass of the plasma. The results presented here were obtained using $\Delta t = 1ms$, $n_{p0} = 5 * 10^{24}$.

2. *Plasma Expansion*

The plasma evolution in this case is plotted in figures 7-3, 7-4. As we can see, the plasma becomes radially unstable, as it moves towards the EF3 coils (at $RPlasma = 81cm$ the curvature index is $n_c \simeq 1.5$). Interestingly, the massive simulation evolves regularly, while the massless simulation has discontinuities when the plasma is approaching the wall. This happens because the passive currents cannot support the evolution of the plasma anymore, but the massless simulator is still forcing the plasma position to a zero of the radial field. Conversely, in the massive simulator the plasma has inertia and cannot jump instantaneously to a new position, thus there are not discontinuities and the results are physically meaningful. The plasma major radius increases during the radial expansion, the vertical position remains substantially unchanged, apart from a small numerical noise. The plasma current slightly decreases. The results presented here were obtained using $\Delta t = 1ms$, $n_{p0} = 1 * 10^{24}$.

3. *Plasma asymmetric perturbation*

In this case a step of 5% of the DC value is applied to the coils with opposite signs. The results are plotted in figures 7-5, 7-6 and are obtained with $\Delta t = 1ms$ and $n_{p0} = 5 * 10^{24}$. Notably, the plasma remains vertically stable in this case.

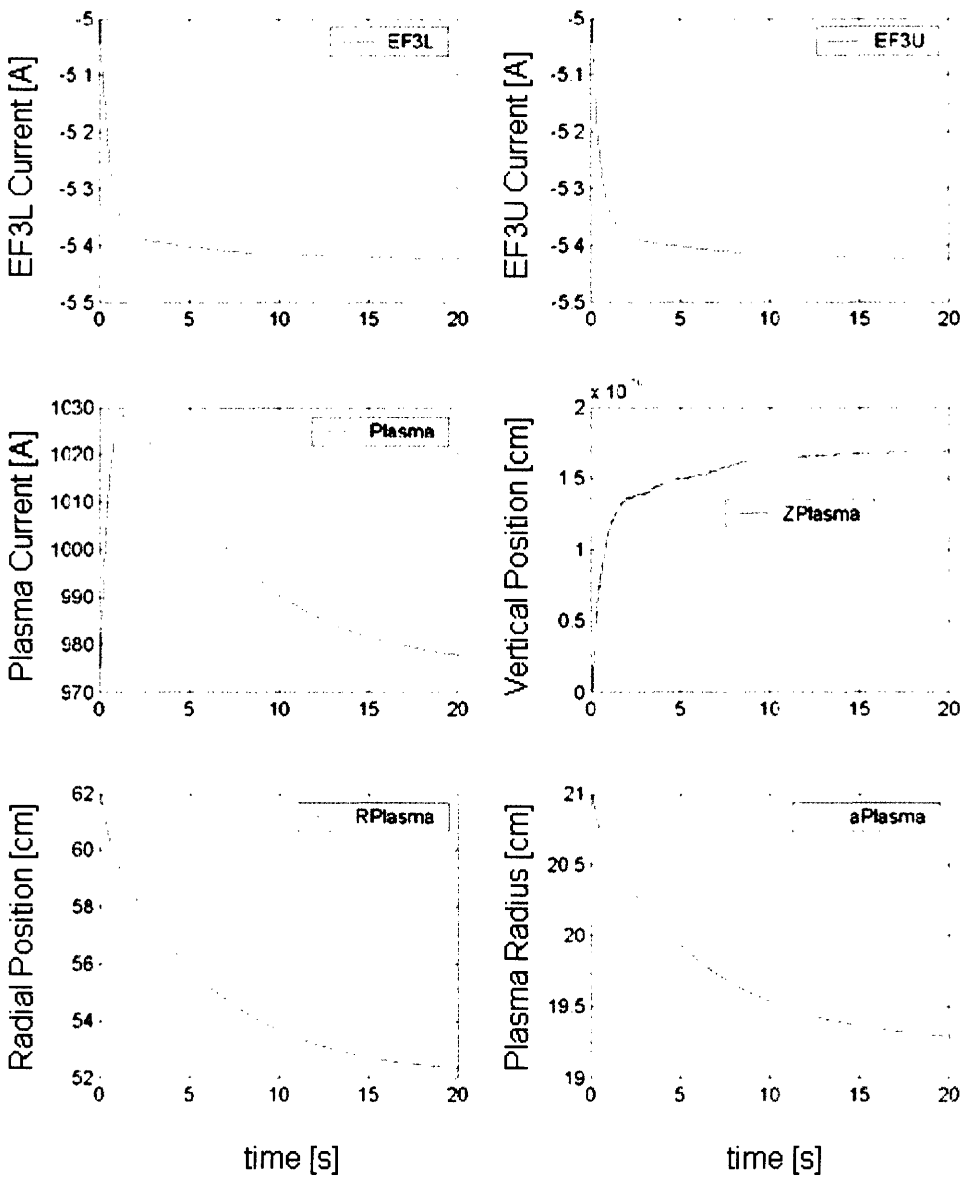


Figure 7-1: Compression evolution of a perturbed equilibrium for a massless simulation

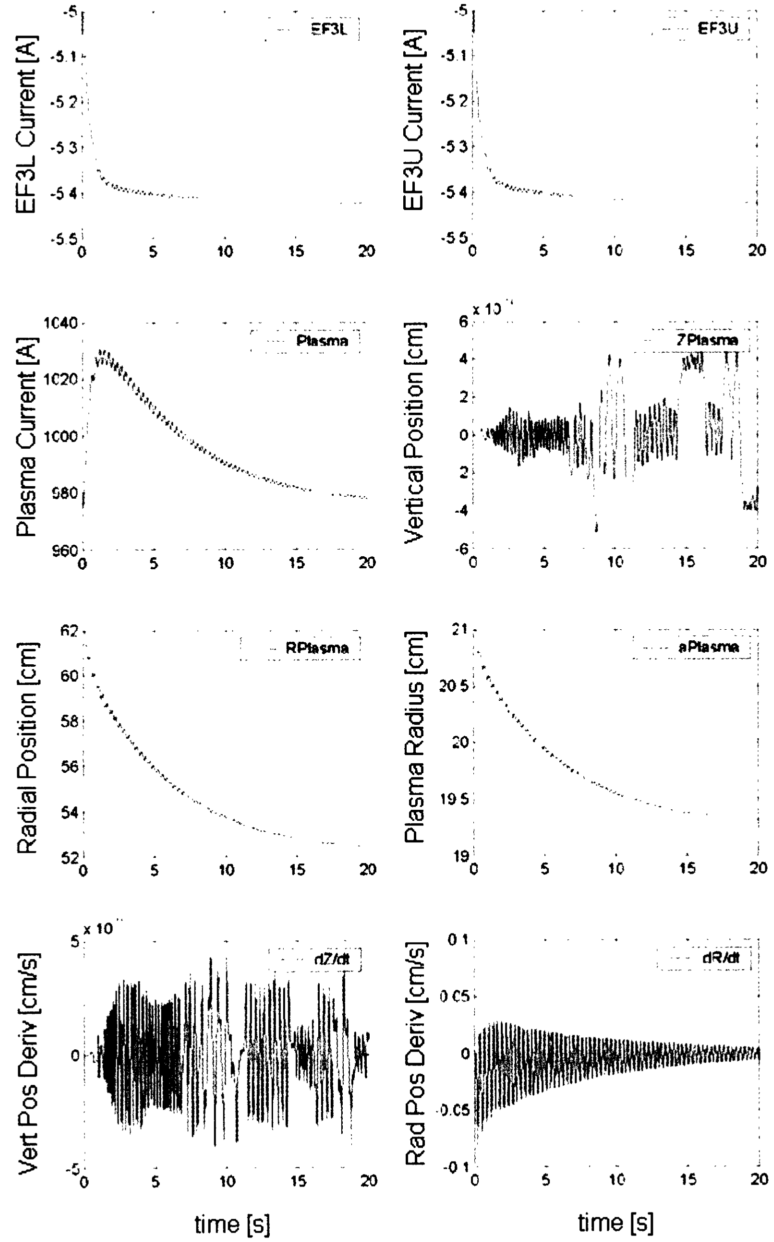


Figure 7-2: Compression evolution of a perturbed equilibrium for a massive simulation. The small oscillations on the traces of the plasma current and the position of the plasma are numerical artifacts and can be removed by increasing the mass of the plasma in the simulations

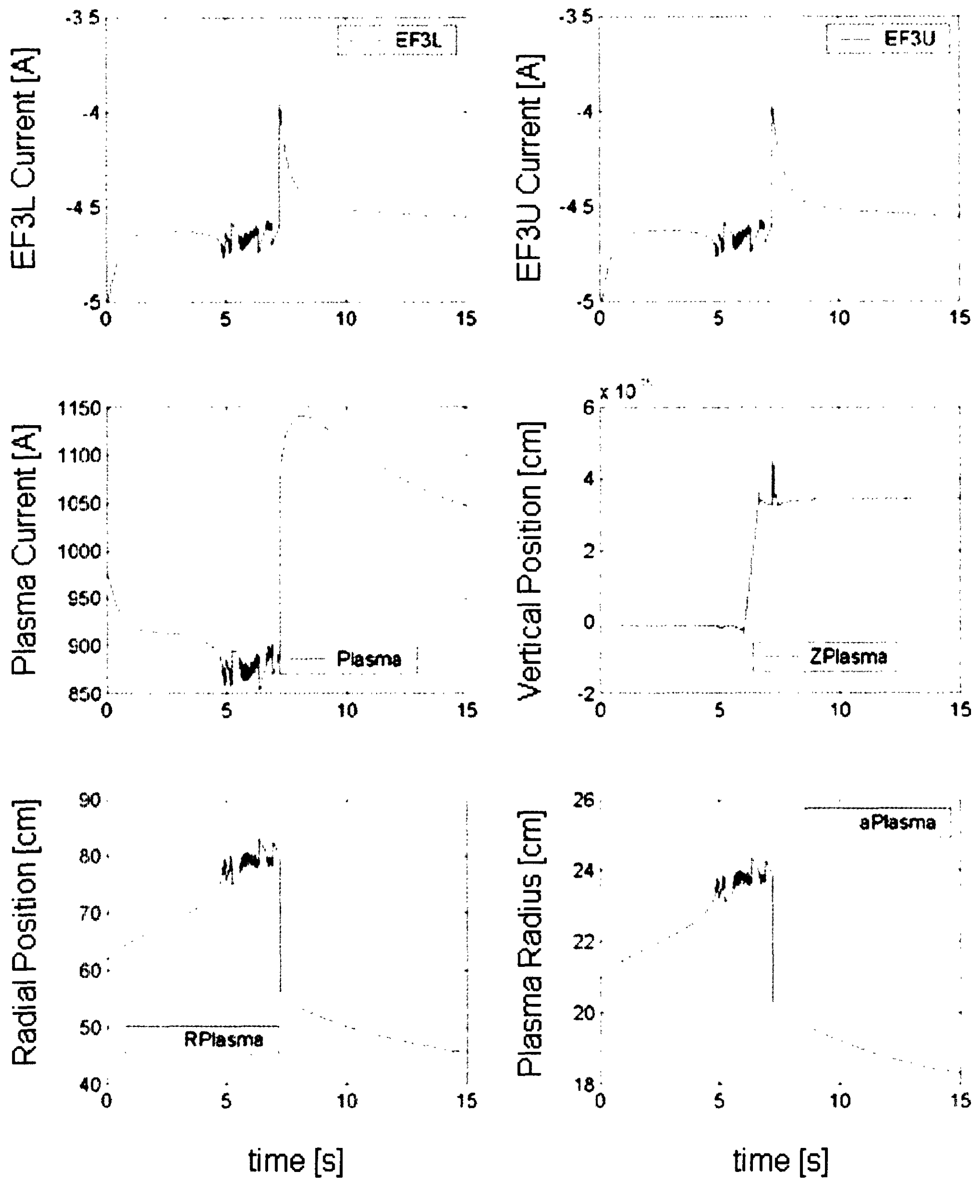


Figure 7-3: Expansion evolution of a perturbed equilibrium for a massless simulation

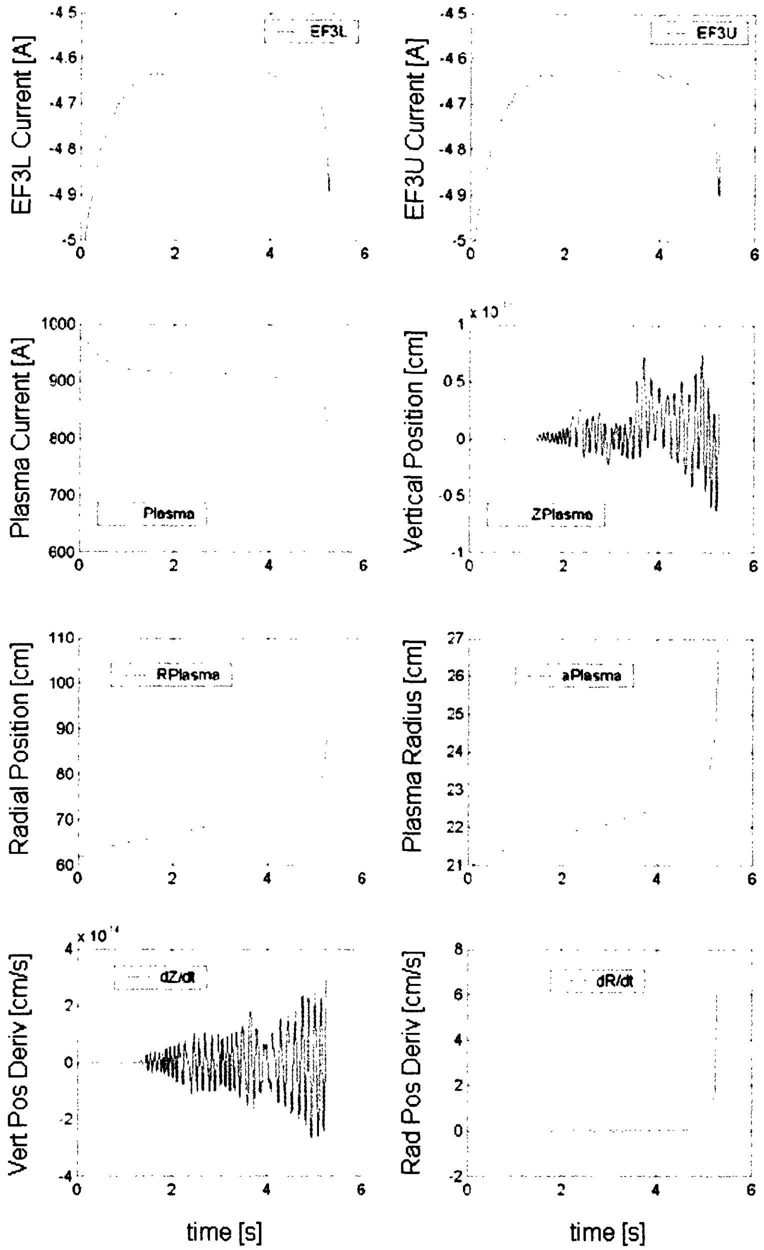


Figure 7-4: Expansion evolution of a perturbed equilibrium for a massive simulation. The small oscillations on the traces of the plasma current and the position of the plasma are numerical artifacts and can be removed by increasing the mass of the plasma in the simulations

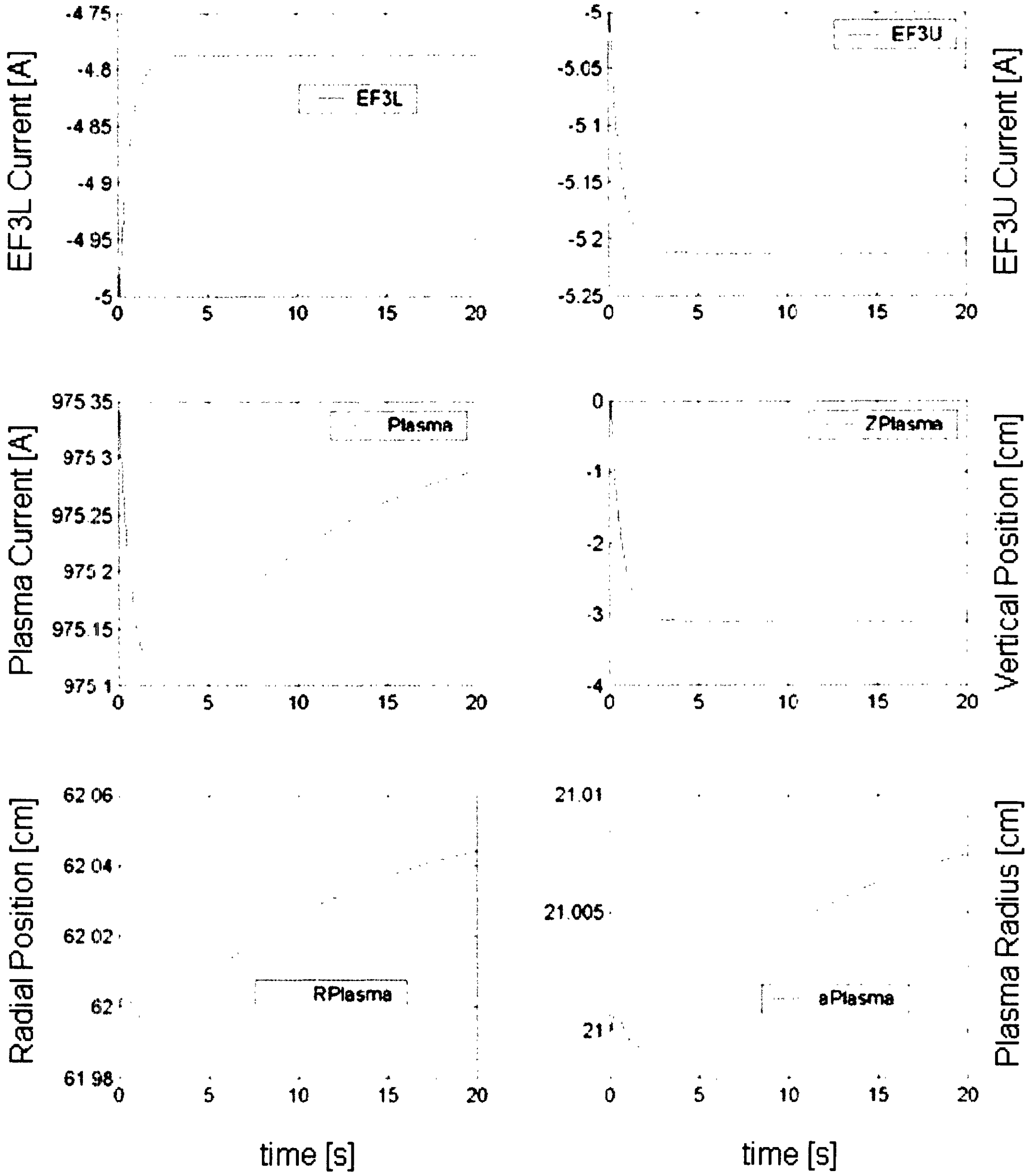


Figure 7-5: Asymmetric perturbation, massless simulation

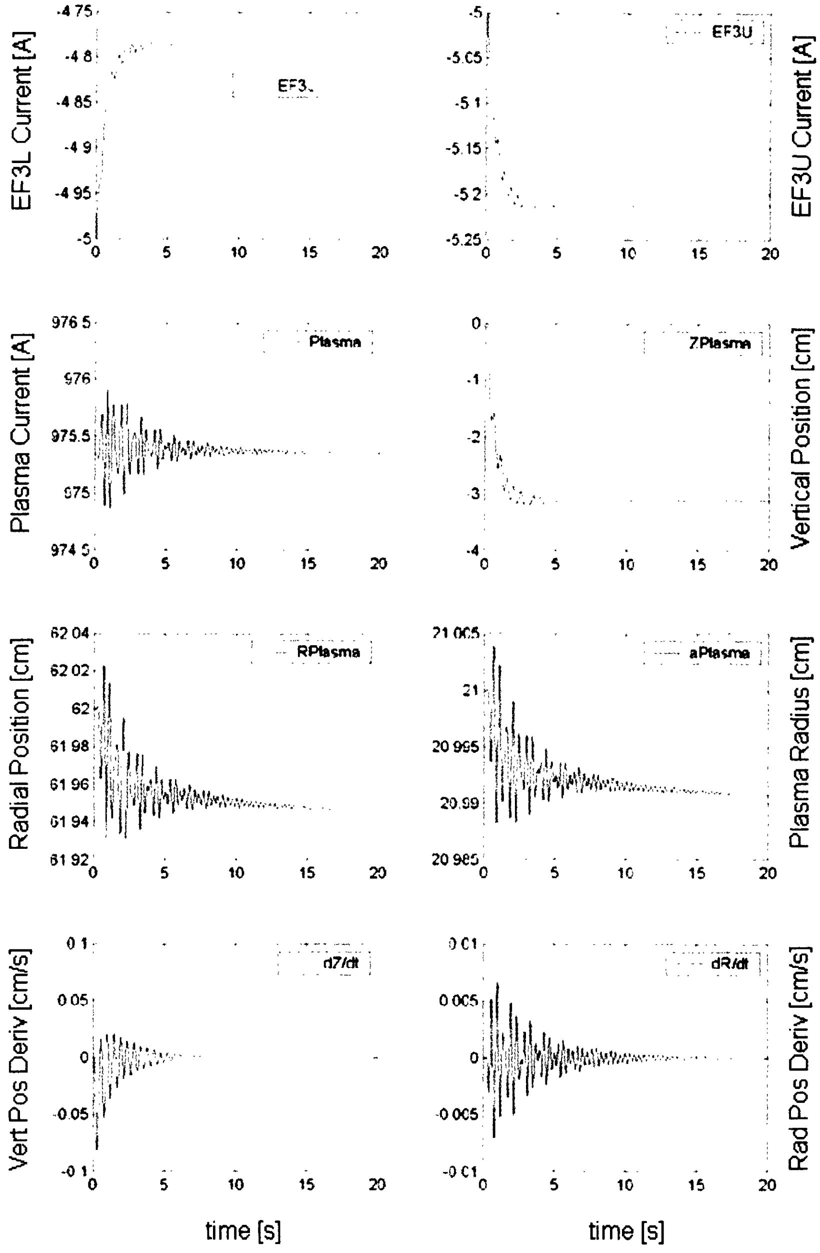


Figure 7-6: Asymmetric perturbation, massive simulation

Vertically Unstable Plasma, $n_c = -1.4$

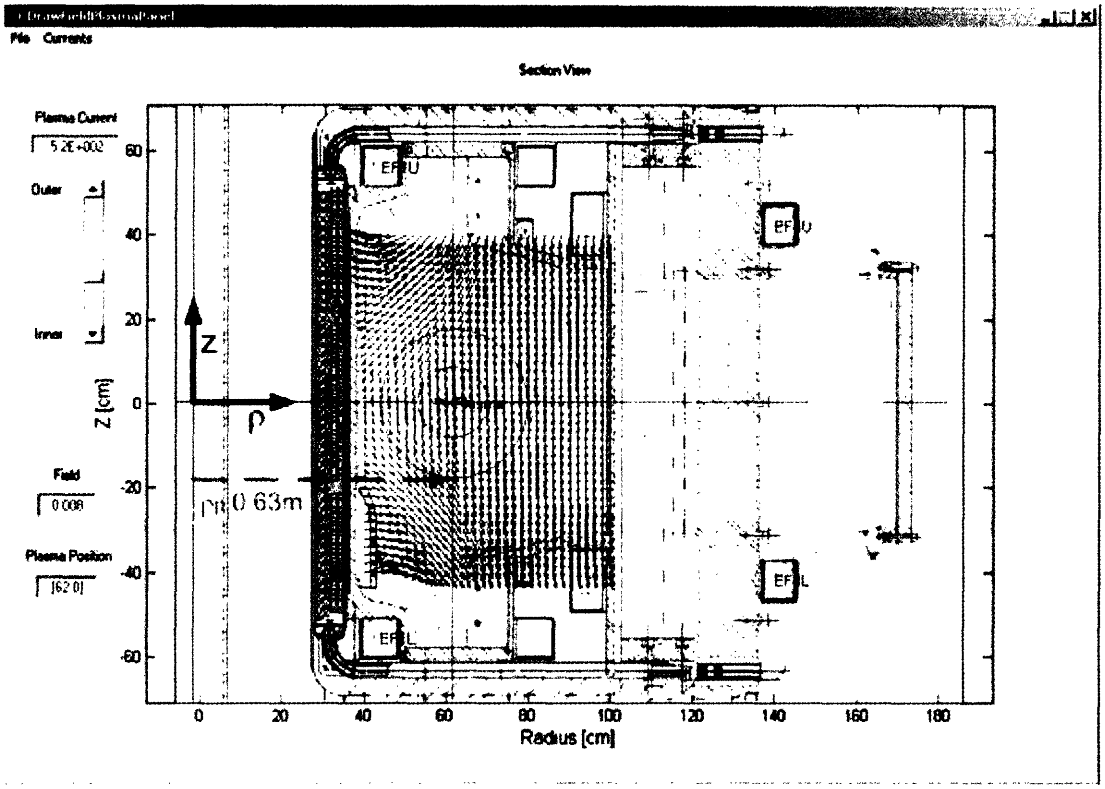
When the field curvature is negative, our model of the plasma becomes vertically unstable. In fact, if slightly removed from the equilibrium position, the plasma will experience a $\overrightarrow{J} \times \overrightarrow{B}$ force in the same direction of the initial displacement, instead of a restoring force. There are two possibilities: the plasma is ideally unstable, that is only the plasma natural inertia slows down the run-away of the plasma, and in this case the typical times are the Alfven times, that is hundreds of microseconds. In the second possibility, there are enough passive conductors around the plasma to induce currents and magnetic fields which oppose the movement of the plasma. Unfortunately, these fields decay with the currents which generate them, that is with the currents' time scale. In this case the plasma is resistively unstable. A resistively unstable plasma is always in a position of instantaneous equilibrium. This is the reason why the massless simulation works correctly only for resistively unstable plasmas.

1. *Ideally Unstable Plasma*

The model of the system and the initial equilibrium are illustrated in figure 7-7. For the active coils we use the copper resistivity at room temperature $1.6 * 10^{-6}\ \Omega * cm$. The EF1 coils resistance is 0.0432Ω and the time constant is $0.210s$. The EF4 coils resistance is 0.137Ω and the time constant is $0.316s$. All the coils are independent. In the case of an ideally unstable plasma the massless simulation is unreliable. Figure 7-8 shows the results from the massive simulation when an unstable plasma is slightly perturbed (the amplitude of the perturbation is +1% of the DC value and is applied to the upper EF1 coil). The exponential evolution is slowed down by the coupling with the coils.

2. *Resistively Unstable Plasma*

The model of the vacuum vessel used to demonstrate the regime of resistive instability of the plasma is illustrated in figure 5-4. The vacuum vessel is modeled with a set of independent coils. The default resistivity is that of stainless steel $7.4 * 10^{-5}\ \Omega * cm$. For the active coils we use the copper resistivity at room temperature $1.6 * 10^{-6}\ \Omega * cm$. All the coils are independent. In our simulations we compare the results of the massive and massless simulations, in order to identify those regions where the plasma becomes ideally unstable. The time step must be



Plasma Current 5.2E+002

Field 0.008

Plasma Position 162.0

Figure 7-7: An ideally unstable plasma equilibrium

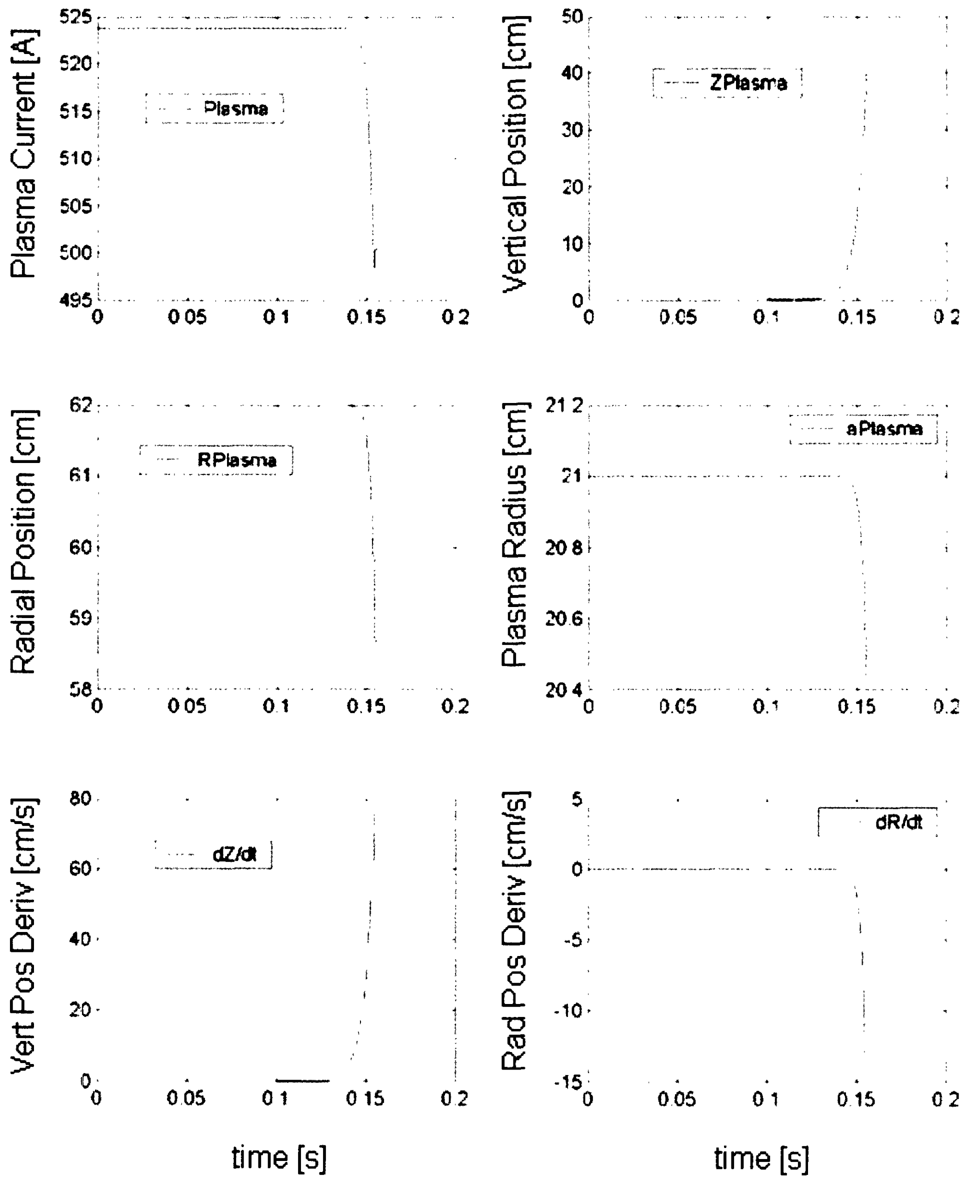


Figure 7-8: Massive simulation in the case of an ideally unstable plasma

chosen appropriately, as a function of the resistivity of the passive elements and the mass of the plasma, in order to obtain reliable results in a useful time.

The set of coils used and the initial equilibrium are illustrated in figure 5-4. The EF1 currents are $5A$ and the EF4 currents are $-3.5A$. The unstable plasma is slightly perturbed at time $0.1s$. The amplitude of the perturbation is $+1\%$ of the DC value and is applied to the upper EF1 coil. In the simulations we have always chosen the time step such that the results agree with the case when a smaller time step is used. We are thus free from errors resulting from an inappropriate choice of the time resolution.

Figure 7-9 shows the results from the massless simulation. The vacuum vessel resistivity is that of stainless steel and the time step is 100 μs. While this time step is usually too accurate for resistively unstable plasmas, and a larger time step would produce faster simulations, it is necessary to reproduce the evolution of ideally unstable plasmas.

The massless simulation agrees with the massive simulation of figure 7-10 until the plasma becomes ideally unstable, as it approaches the wall. Figure 7-10 was obtained with $\Delta t = 50$ μs and an initial plasma density $n_{p0} = 10^{20}/m^3$. The evolution of the plasma can be slowed down by increasing the density of the plasma or decreasing the resistivity of the walls.

7.1.3 Matlab simulation speed

The speed of the open loop simulator has been tested with a model of the machine comprising 72 independent coils. At each iteration, the simulator must invert a 76×76 matrix (it's 77×77 in the case of the massive simulation), for evaluating the increments to the various physical quantities, and must compose the matrix and the vector for the next iteration. In these operations, the contribution of each coil to the magnetic field and the magnetic field gradient must be evaluated, thus the computation of four elliptic integrals is needed inside a nested loop whose length equals the number of coils. The evaluation of the elliptic integrals inside this loop makes much of the time of the computation. In order to optimize the execution, we've copied the code which executes the elliptic integrals inside the code of the simulator in order to avoid the call of an external routine and the usual check for the input arguments.

We have been able to reduce the total time per iteration to $42.5ms$ in the case of the massless simulator and $34.0ms$ in the case of the massive simulator. The iteration time of the massless

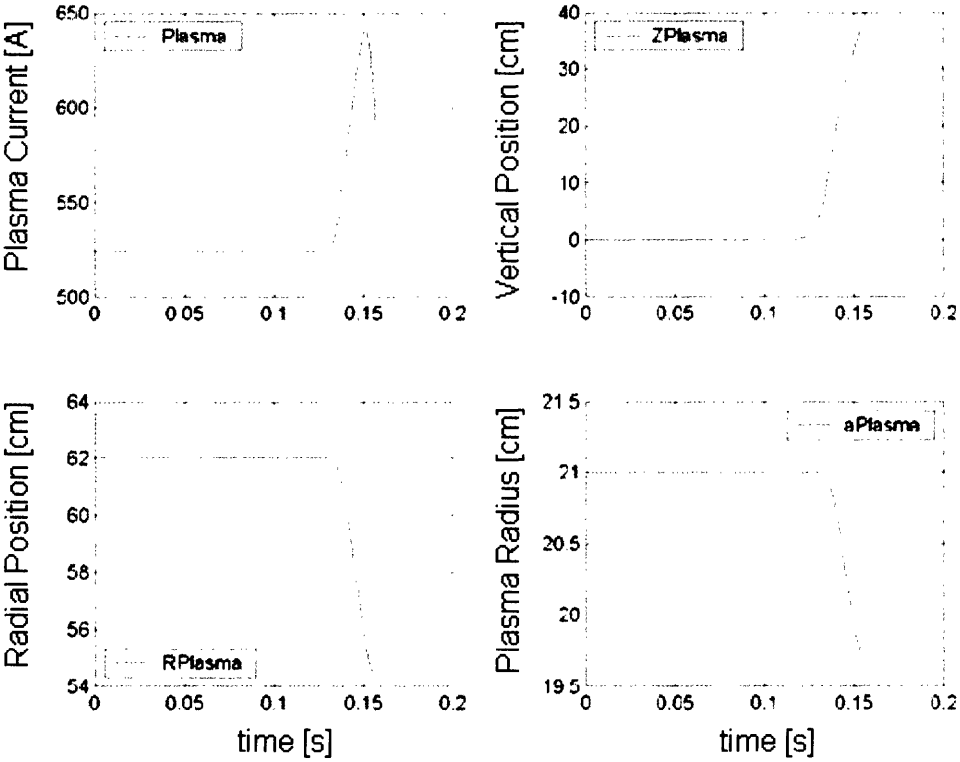


Figure 7-9: Results from the massless simulation of a resistively unstable plasma. $\Delta t = 100\ \mu s$ for this simulation

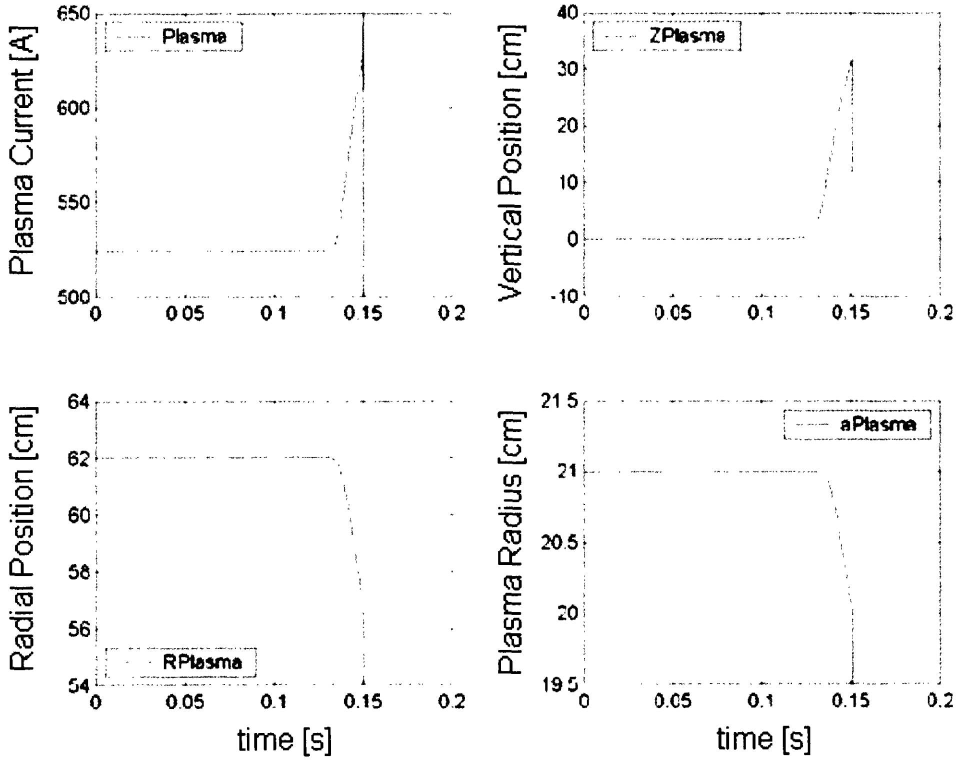


Figure 7-10: Results from the massive simulation of a resistively unstable plasma. $\Delta t = 50\ \mu s$ and the plasma initial density $n_{p0} = 10^{20}$ for this simulation

simulator is larger because the evaluation of the field gradients requires the computation of additional elliptic integrals in the proximity of the plasma position. These iteration times sum up in a total simulation time of a few minutes for a plasma discharge during a few seconds, when the time step is $1ms$.

7.2 Simulink closed loop simulations

The Simulink model is particularly interesting, because it comprises the various subsystems and allows the user to test new control algorithms. The preliminary tests were done with the model of the machine drawn in figure 5-2. This model is quite rich and comprises 188 coils. The EF3 coils are connected in series, the EFC coils are in anti-series and the EF4 coils are in parallel. The setup of full-flux, partial-flux and poloidal field coils is loaded from the *magnetics* tree of the shot being simulated. Similarly, the parameters describing the operation of the power supplies (activation times, switching times, open/close and crow-bar resistances, etc.) are loaded from the *engineering* tree of the shot and the control waveforms and configuration parameters are read from the *hybrid* tree of the shot.

First, some fizzle shots were simulated, that is shots when the plasma failed to form. This was done in order to test the combination of the machine model and the diagnostics, without the additional complication of the plasma. The fizzle shots were also used as reference cases during the debugging of the simulator, which was particularly cumbersome. It required the fragmentation of the system in smaller sub-systems, such as the tokamak and plasma, the diagnostics, the control system, etc., and the independent test of each of them. The simulated currents in the active coils agreed within 5% with the real values. Similar agreement was obtained with the signals of the full-flux loops and the partial-flux loops. The signals of the poloidal coils showed larger discrepancies, as large as 20%, but the calculations of relevant observables based on these signals, such as the plasma radius and vertical position, were following the real signals within a few percent. The discrepancies of the poloidal field signals are due to the model of the machine which is not perfectly identical to the real tokamak. For example, the tokamak is not up-down symmetric, while the model used in the simulations is.

Secondly, the plasma was introduced in the simulations. The breakdown and the initial phase of the current rise is particularly difficult to model, because the plasma current is very small and its position hardly detectable. The problem was overcome by introducing a fixed coil to represent the plasma at the beginning of the current rise. The coil is not allowed to move, but its current varies as a consequence of the inductive drive from the active coils. Once the initial phase is over (usually after $10ms$, corresponding to a breakdown current of $100kA$), the plasma is freed and the loop controls its position. Many different shots were tested, with elongations in the range $1.5 \div 1.8$ and various vertical and radial control targets, and the nominal values were used for the simulation setup parameters, i.e. for the plasma elongation, the plasma resistivity, the resistivity of the active and passive coils in the model, the configuration of the diagnostics and the gains and bandwidths of the power supplies[20]. One example is in figure 7-11, which refers to shot 1050804011. The real discharge produced a plasma with $1MA$ peak current and $k = 1.7$ elongation. The ratio of the decay index and the single mode critical index for this plasma was $xnnc = 0.8$[1]. The values entered in the user interface were *Plasma Elongation* = 1.7, *Current Distr Radius* = 10 (i.e. the equivalent radius of the parabolic squared distribution of current is ten times the minor radius a. This is almost equivalent to considering a uniform distribution of the current over the six filaments which represent the plasma in the vertical equilibrium. See also section 6.2) and *Plasma Resistivity* = 3 (i.e. *Plasma Resistivity* = $4.8E - 6\ \Omega \cdot cm$). The active coils resistivity is the liquid nitrogen copper resistivity with additional tuning (±10%) to better match the outputs of the power supplies. The matrices are updated every $0.5ms$ and the DPCS cycle is $100\mu s$. Figures 7-12, 7-13, 7-14 compare the real and simulated error signals, power supplies outputs and coil currents. The solid red lines show the simulations while the dashed black lines are the experimental data. The similarity of the signals representing the power supplies outputs is especially useful, since the simulator can become a tool for studying the saturation of the power supplies during the ramp-up of high performance plasmas.

Figure 7-15 refers to shot 1050706014 for which $xnnc = 1.2$ in the flattop of the discharge. The parameters controlling the six elements of the plasma are *Plasma Elongation* = 1.8 and *Current Distr Radius* ×10. The plasma proved to be too unstable with these settings and

[1] This quantity is evaluated by EFIT and is an estimator of the vertical instability of a plasma. The limit for Alcator plasmas is $nxxc = 1.2$.

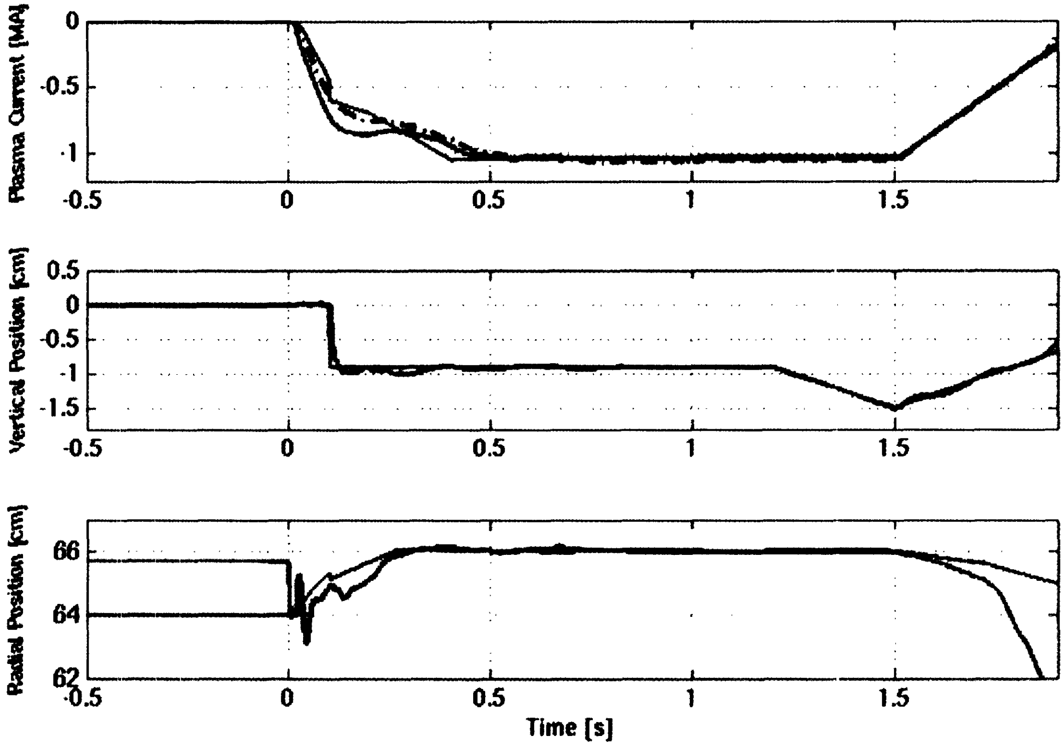


Figure 7-11: Simulation of the plasma discharge for shot 1050804011. The red solid traces are the simulated signals, while the black dashed trace in the first panel is the real current measured by the diagnostics. The blue traces are the target waveforms of the Plasma Control System (PCS)

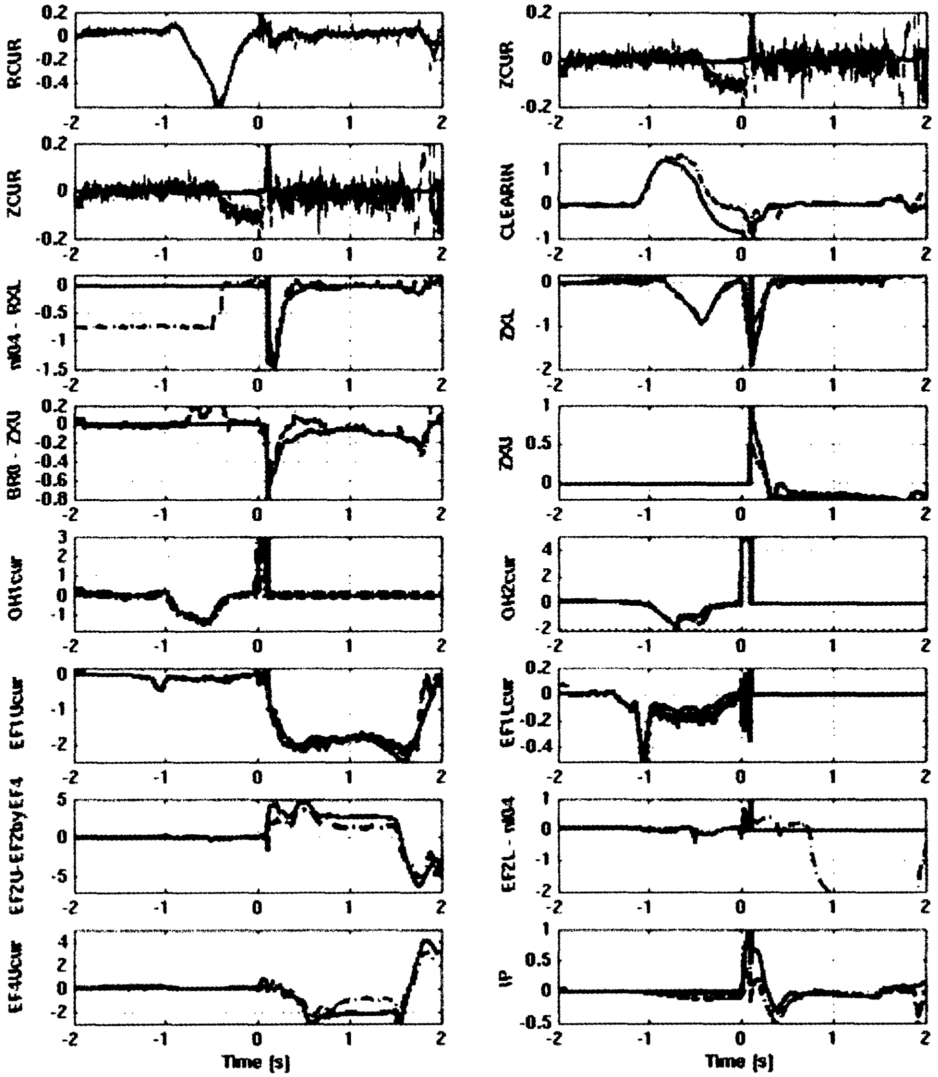


Figure 7-12: Comparison between the real and simulated feedback error signals in the case of shot 1050804011. The red solid traces are the simulated signals. The agreement is fairly good, in spite of our simple model of the plasma. We are not controlling density, thus the large discrepancy in the corresponding signals

couldn't be controlled by the simulator, but it was controlled in the real shot. However, shot 1050706003 from the same day had similar programming and disrupted with a vertical instability at about the time shown in the simulation, at $nxxc = 1.23$ and $k = 1.83$.

The worst time performance was obtained with the Simulink simulation of a 4 second plasma discharge, whose critical elongation required the update of the matrices every $0.5ms$: the simulation took approximately 20 minutes on a P4 $3GHz$ single-processor computer. Further work is needed to optimize the code and improve the time performance. However, models with a dense placement of coils suffer from an intrinsic limitation, because the matrix $\overleftrightarrow{A}$ is close to singularity and the time steps of the simulations are necessarily small.

These successful simulations came after a long debugging process only close to the deadline for thesis submission, which is why there are no more examples included here. Ongoing work will assess the reliability of the simulator and the possibility of using it as an effective tool for predicting features of C-Mod plasma discharges.

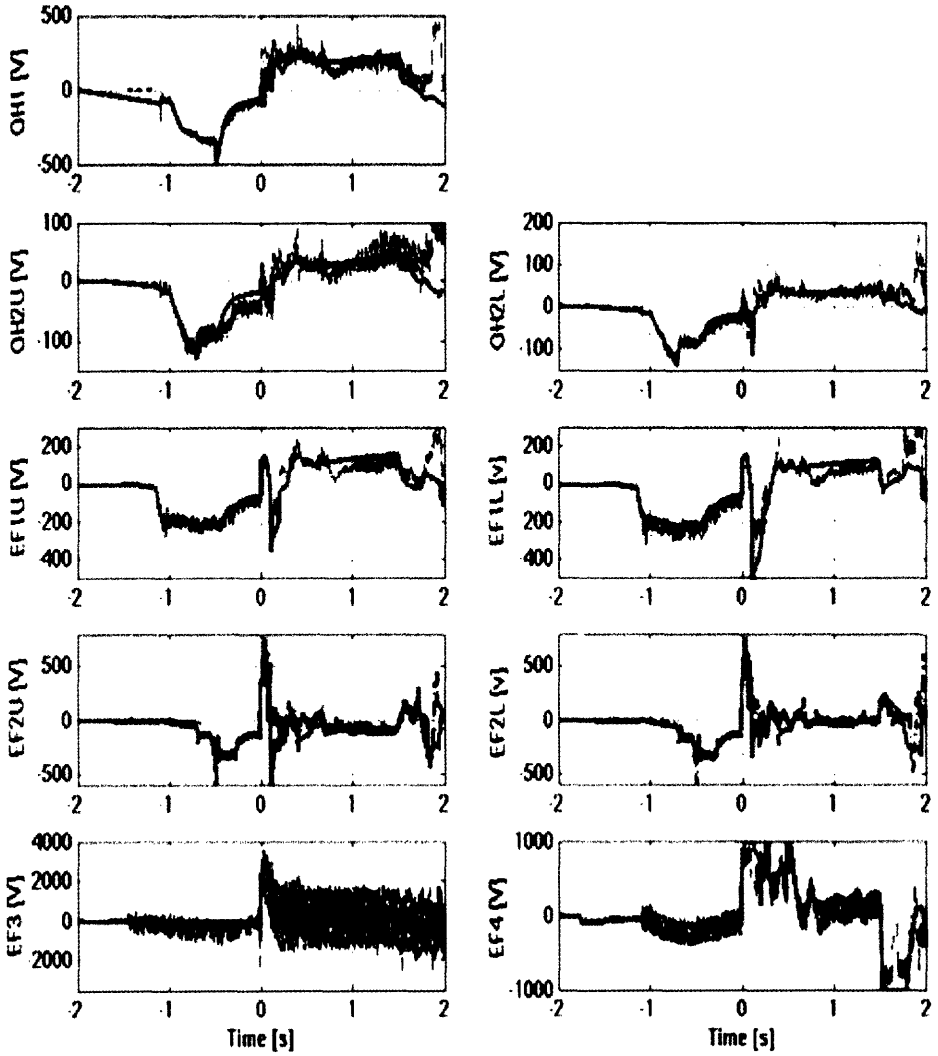


Figure 7-13: Comparison between the real and simulated output voltages from the power supplies in the case of shot 1050804011. The red solid traces are the simulated signals. Notice the small discrepancy at 0.8s in the EF1 input voltages. About this time the plasma went into H-mode. The overall good agreement demonstrates the adequacy of our simple model of the plasma

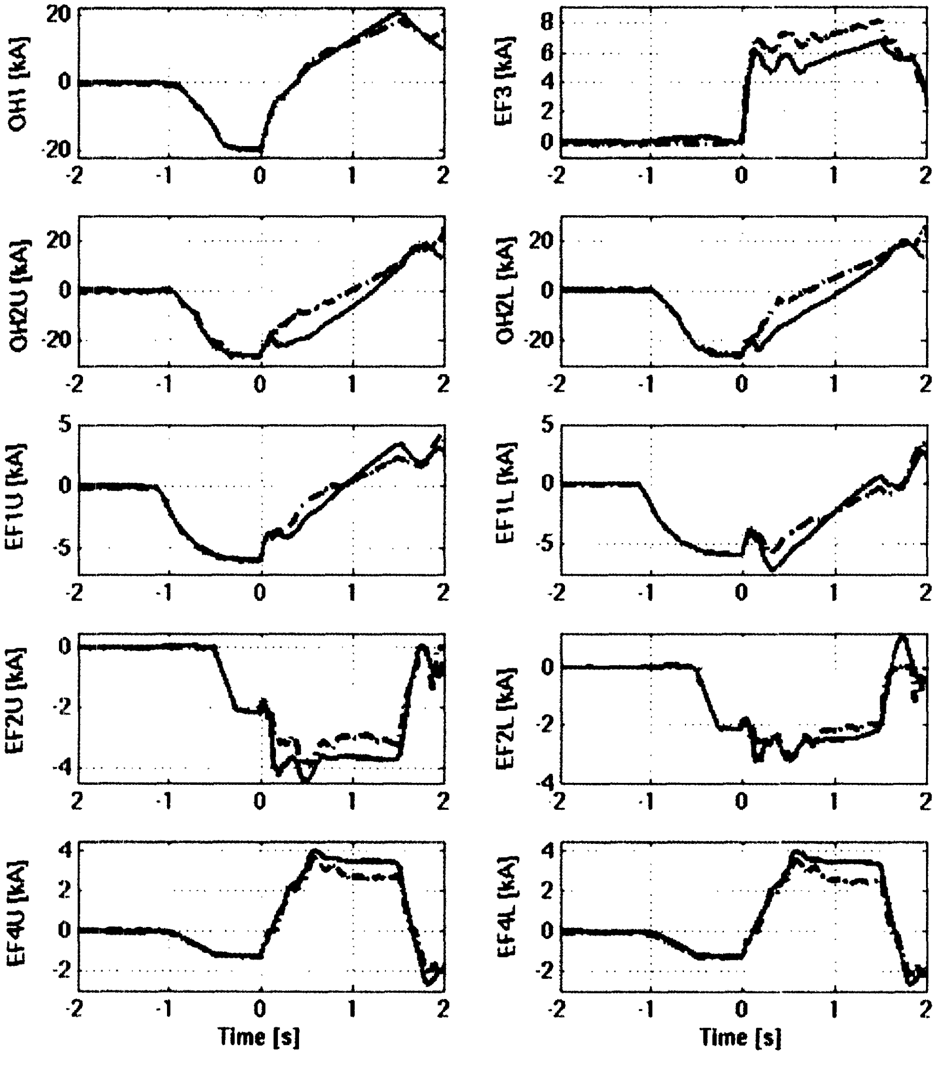


Figure 7-14: Comparison between the real and simulated coils currents in the case of shot 1050804011. The red solid traces are the simulated signals. The discrepancies between the real and the simulated signals, particularly evident in the EF2, EF3 and EF4 currents, are realistically due to our approximate model of the plasma shape and resistivity

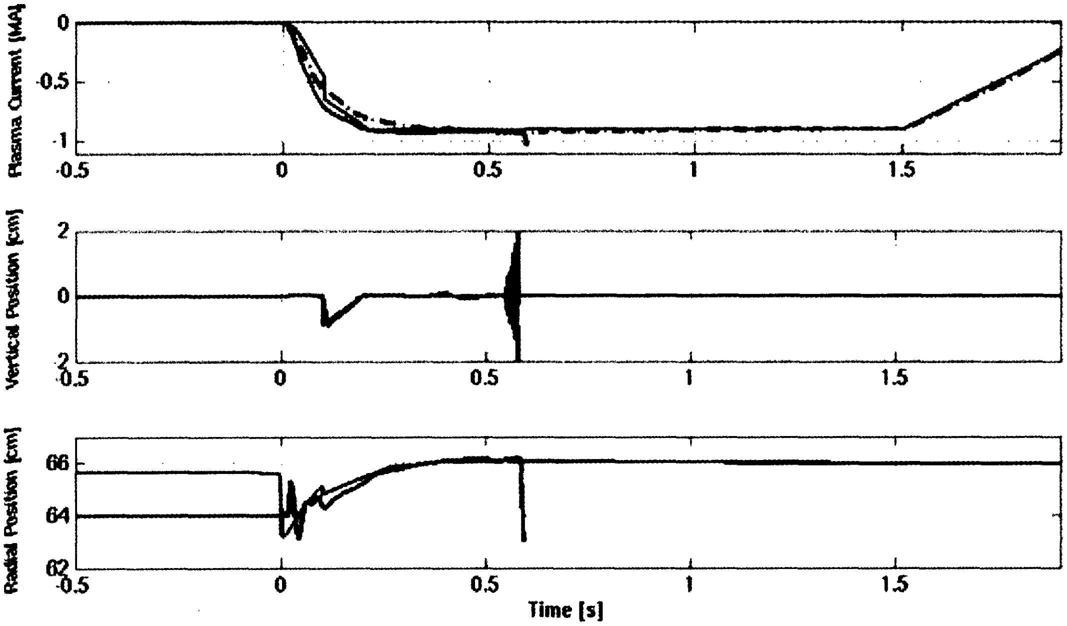


Figure 7-15: Simulation of the plasma discharge for shot 1050706014. The red solid traces are the simulated signals, while the black dashed trace in the first panel is the real current measured by the diagnostics. The blue traces are the PCS target waveforms. The plasma in this simulation disrupts, while the real shot didn't. However, shot 1050706003 from the same day had a similar programming and disrupted with a vertical instability at about the time shown in the simulation

Chapter 8

Conclusions

The first part of this thesis discusses the digital plasma control system (DPCS) of Alcator C-Mod. It currently implements the same PID control algorithm as the previous system Hybrid. DPCS was carefully debugged during the 2003-2004 experimental campaign and came on-line at the start of the 2004-2005 campaign. The system has performed reliably during the 2004-2005 campaign. In fact, some advanced and adaptive features have already been implemented in the digital system, for example the compensation of the input offsets.

The digital system is more accurate than the previous Hybrid, as it doesn't have problems of analog offsets, drifts and leakages. It is more robust and easier to maintain and upgrade. In addition, the flexibility of the digital technology and the power of the high level programming in IDL allow to easily run new experiments concerning with the control of the plasma. The IDL code already includes calls to customizable routines to generalize the basic PID control algorithm. As explained in section 3.2, additional processing can be done on the inputs, the observers, the PID outputs and the controllers in order to control particular quantities or apply advanced algorithms. The latter can be adaptive, in the sense that the results at each stage of the calculations may depend on the recent trajectory of the plasma. In other words, each stage of the calculations may include a filter with memory of the recent past.

In the future we will exploit these new features. One possible application concerns with the saturation of the power supplies when high current plasmas are run. An adaptive filter can look at the recent trajectory of the power supplies outputs and the target and feed-forward waveforms can be corrected in real time if the power supplies are running to their rails.

The second part of the thesis introduces the Matlab-Simulink simulator Alcasim. This code can be used to draw and simulate different machine designs and makes large use of graphical user interfaces to facilitate the design process and the setup of the simulation parameters. Alcasim can simulate both open-loop and closed-loop discharges. For the closed loop simulations, we developed a model in Simulink. This turns to be particularly effective, as the block-diagram language of Simulink allows the user to easily upgrade the model with the simple substitution of individual sub-systems. In the case of closed loop simulations, Alcasim loads all the information concerning the diagnostics, the power supplies and the control parameters from the database of the real experiments. A functional equivalent of the DPCS code is run in the closed loop simulations. With Alcasim we were able to simulate the entire cycle of C-Mod plasma discharges. The comparison of the simulated and the real data show consistent agreement.

Future work will aim at interfacing the IDL real time code with the Simulink environment: the IDL real time code could invoke a modified version of the procedure llAcquire, which would read an output from the Simulink-Matlab model of the power supplies, tokamak and diagnostics, and write an input for the next cycle of the Simulink simulation. The real time code of DPCS would then be completely embedded in the simulator.

Alcasim will also help to test new control algorithms, which will address specific problems occurring with high performance plasmas, for example the saturation of the power supplies.

Bibliography

[1] I.H. Hutchinson, S.F. Horne, G. Tinios, S.M. Wolfe, R.S. Granetz, *Plasma shape control: a general approach and its application to Alcator C-Mod.* Fusion Technology 30, 137 (1996)

[2] D.A. Humphreys, I.H. Hutchinson, *Axisymmetric Magnetic Control Design in Tokamaks using Perturbed Equilibrium Plasma Response Modeling.* Fusion Technology 23, 167, (1993)

[3] S. Horne, M. Greenwald, I. Hutchinson, S. Wolfe, G. Tinios, T. Fredian, J. Stillerman, *Performance of the C-Mod Shape Control System.* IEEE 15th Symposium on Fusion Engineering, 242-245 (1994)

[4] M. Ariola, A. Pironti, A. Portone, *A Framework for the Design of a Plasma Current and Shape Controller in Next Generation tokamaks.* Fusion Technology 36, 263 (1999)

[5] M.A. Firestone, J.W. Morrow-Jones, T.K. Mau, *Comprehensive Feedback Control of a tokamak Fusion Reactor.* Fusion Technology 32, 390 (1997)

[6] L. Scibile, B. Kouvaritakis, *Application of an Adaptive Algorithm to the Control of the Plasma Vertical Position.* Fusion Technology 36, 139 (1999)

[7] J.R. Ferron, M.L. Walker, L.L. Lao, H.E. St. John, D.A. Humphreys, J.A. Leuer, *Real Time Equilibrium Reconstruction for tokamak Discharge Control.* Nuclear Fusion 38 (7), 1055 (1998)

[8] M. Ariola, G. Ambrosino, J.B. Lister, A. Pironti, F. Villone, P. Vyas, *A Modern Plasma Controller Tested on the TCV tokamak.* Fusion Technology 36, 126 (1999)

[9] M. Ariola, G. Ambrosino, A. Pironti, J.B. Lister, P. Vyas, *Design and Experimental Testing of a Robust Multivariable Controller on a tokamak.* IEEE Transactions on Control Systems Technology 10 (5), 646 (2002), and the references therein

[10] www.mdsplus.org

[11] J.B. Lister, F. Hofmann, J.-M. Moret, at al., *The control of Tokamak Configuration Variable plasmas.* Fusion Technology 32, 321 (1997)

[12] J.B. Lister, M.J. Dutch, P.G. Milne, R.W. Means, *A High Performance Digital Control System for TCV.* IEEE Transactions on Nuclear Science, 45 (4), 2044, 1998

[13] IDL Research Systems Inc., www.rsinc.com

[14] J.A. Stillerman, M. Ferrara, T.W. Fredian, S.M. Wolfe, *Digital real time plasma control system for Alcator C-Mod.* IAEA 5th Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research (2005). To be published on Fusion Engineering and Design.

[15] J.A. Leuer, M.L. Walker, D.A. Humphreys, J.R. Ferron, A. Nerem, B.G. Penaflor, *Development of a closed loop simulator for poloidal field control in DIII-D.* IEEE 18th Symposium on Fusion Engineering (1999)

[16] J.A. Leuer, R.D. Deranian, J.R. Ferron, D.A. Humphreys, R.D. Johnson, B.G. Penaflor, M.L. Walker, A.S. Welander, R.R. Khayrutdinov, V. Dokouka, D.H. Edgell, C.-M. Fransson, *DIII-D plasma control simulation environment.* IEEE 20th Symposium on Fusion Engineering (2003)

[17] R.D. Deranian, J.R. Ferron, D.A. Humphreys, et al., *Integrated Plasma Control in Next-Generation Devices usinh DIII-D Modeling and Simulation Approaches.* Fusion Science and Technology 47, 768-773 (2005)

[18] G. Tinios, S.F. Horne, I.H. Hutchinson, S.M. Wolfe, *Model Reduction for Axisymmetric Tokamak Control.* Fusion Technology 24, 355 (1993)

[19] G. Tinios, S.F. Horne, I.H. Hutchinson, S.M. Wolfe, *Comparison of Models to Experiment for the Purposes of Axisymmetric Control in Alcator C-Mod.* Fusion Technology 30, 201 (1996)

[20] G. Tinios, *Axisymmetric Control in Alcator C-Mod.* PFC/RR-94-7, Massachusetts Institute of Technology (1995)