

APPENDIX A

SOFTWARE

INFORMATION

The Feedback Thermal Control Experiment and Process Simulator are analog systems so a data acquisition board and Matlab's data acquisition toolbox are used to collect data and plot it. The four ECP systems use two versions of special control software: ECP Executive (Torsion and Industrial Trainer) and ECP User-MV Executive (CMG and MagLev). The Feedback Fluid Process Control Experiment software provides a user-friendly graphical user interface. The software guides the student through practical experiments and is fairly self-explanatory.

Matlab Data Acquisition

The Thermal Control and Process Simulator experiments use an analog control system. This means that the feedback loop consists of analog electronic components (e.g. resistors, capacitors, op-amps, etc.). In order to collect data and plot it in Matlab, the data acquisition toolbox and a data acquisition board must be used. Here is an example Matlab code that collects data from two channels of the DAS08 data acquisition board used in the lab:

```
% Data acquisition from the PCI DAS-08

% creates DAQ board object in Matlab
ai = analoginput('cbi', 1);
% configs Matlab for fixed ground reference point (singleended),
% not referenced to any voltage, like a volt meter (differential)
set(ai, 'InputType', 'SingleEnded');
% sample rate at 1000Hz
set(ai, 'SampleRate', 1000);
% 1000 samples per trigger
% 1 second at 1000Hz
set(ai, 'SamplesPerTrigger', 1000);
% sets channel 0 for data acq
chan = addchannel(ai, 0);
% activate board
start(ai);
% acquires data for period set above,
% places in two 1000 number long vectors
[data,time] = getdata(ai);
% deactivate board
stop(ai);
% quick plot
plot(time,data);
% deletes board object, ai no longer usable
delete(ai)
% clears PC memory of ai
clear ai
```

Matlab has extensive on-line help for those students who need assistance getting started. A shortcut to Matlab is on the desktop. Use the Matlab editor to input the file above (lines starting with % are comments). Save the file as `das08in.m`. From the Matlab prompt, type

```
>> das08in
```

and the program will capture and plot the data. You can adjust the sample rate and number of samples as needed. You may also use the Matlab file `softscope.m` to collect data and display on the PC screen.

ECP Executive Software

The ECP Executive program is the user's interface to the system. It is a menu driven / window environment that is intuitively familiar and quickly learned - see Figure 1.10. This software runs on an IBM PC or compatible computer and communicates with ECP's digital signal processor (DSP) based real-time controller. Its primary functions are supporting the downloading of various control algorithm parameters (gains), specifying command trajectories, selecting data to be acquired, and specifying how data should be plotted. In addition, various utility functions ranging from saving the current configuration of the Executive to specifying analog outputs on the optional auxiliary DAC's are included as menu items.

Background Screen

The *Background Screen*, shown in Figure A.1, remains in the background during system operation including times when other menus and dialog boxes are active. It contains the main menu and a display of real-time data, system status, and an *Abort Control* button to immediately discontinue control effort in the case of an emergency.

Main Menu Options

The *Main menu* is displayed at the top of the screen and has the following choices: File, Setup, Command, Data, Plotting, and Utility.

The File menu contains the following pull-down options: Load Settings, Save Settings, About, and Exit. The Load Settings dialog box allows the user to load a previously saved configuration file into the Executive. A configuration file is any file with a ".`cfg`" extension which has been previously saved by the user using Save Settings. The Save Settings option allows the user to save the current control algorithm, trajectory, data gathering and plotting parameters for future retrieval via the Load Settings option.

The Setup menu contains the following pull-down options: Control Algorithm, User Units, Communications. Setup Control Algorithm allows the entry of various control structures and control parameter values to the real-time controller – see Figure A.2.

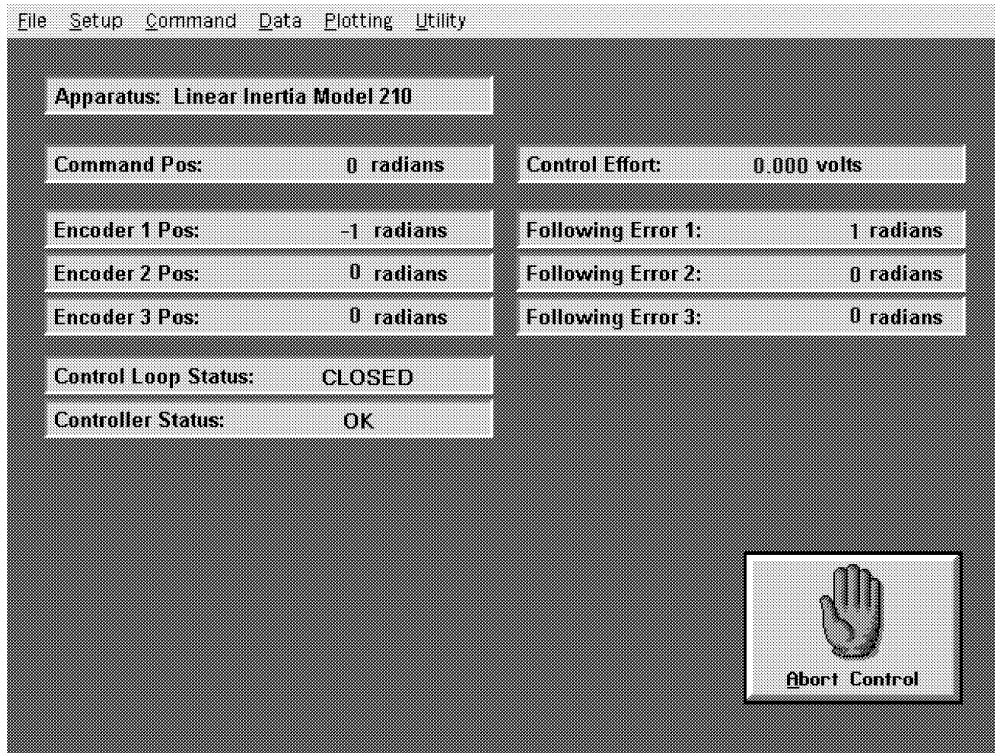


Figure A.1: The Background Screen

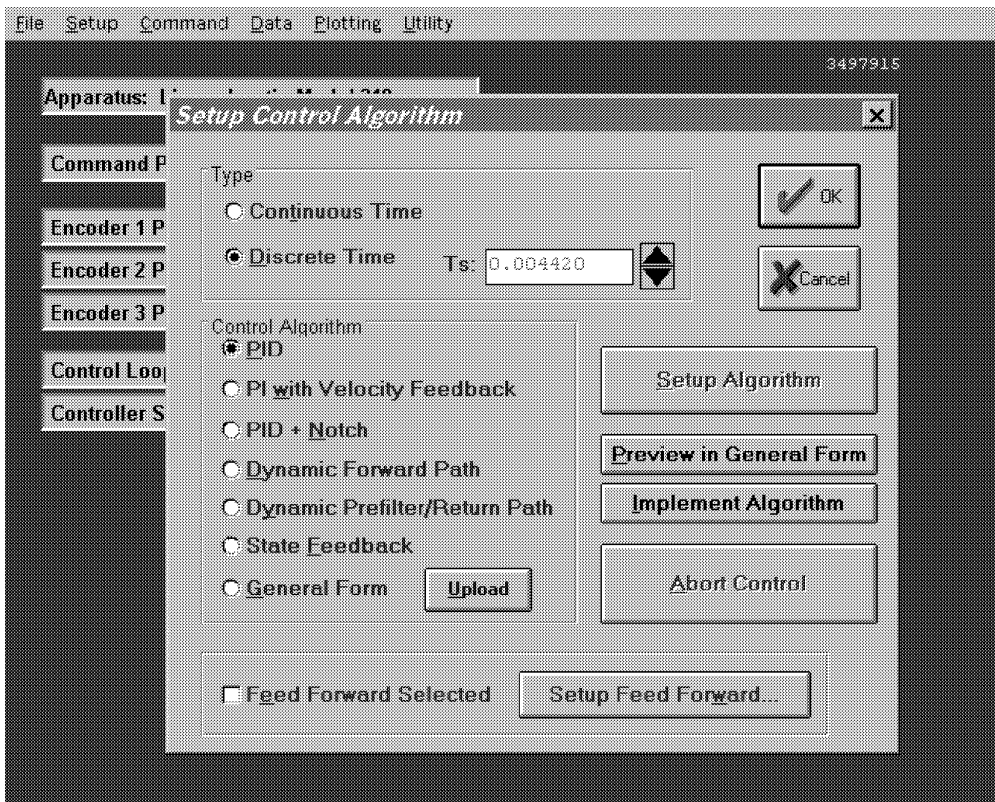


Figure A.2: Setup Control Algorithm Dialog Box

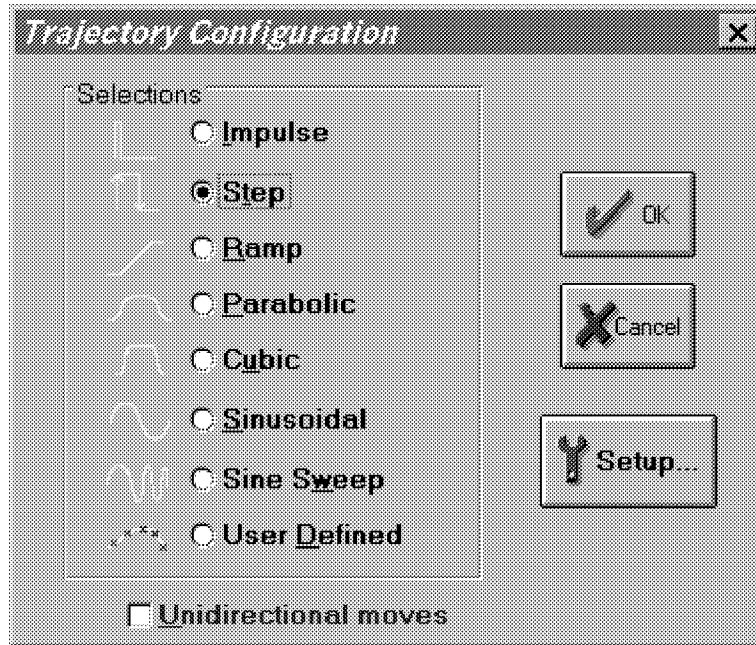


Figure A.3: The Trajectory Configuration Dialog Box

The Command menu contains the following pull-down options: Trajectory, Disturbance, and Execute. The Trajectory Configuration dialog box (see Figure A.3) provides a selection of trajectories through which the apparatus can be maneuvered. The Disturbance Configuration dialog box provides a selection of disturbance torque profiles for the disturbance motor. The Execute dialog box is entered after a trajectory is selected. Here the user commands the system to execute the current specified trajectory and may also choose viscous friction and an output disturbance (system option). After selecting disturbance and data gathering options, the user normally selects Run. The real-time controller will begin execution of the specified trajectory. Once finished, and provided the Sample Data box was checked, the data will be uploaded from the DSP board into the Executive (PC memory) for plotting, saving and exporting.

The Data menu contains the following pull-down options: Setup Data Acquisition, Upload Data, and Export Raw Data. Setup Data Acquisition allows the user to select one or more of data items to be collected at a chosen multiple of the servo loop closure sampling period while running any of the trajectories mentioned above. Selecting Upload Data allows any previously gathered data to be uploaded into the Executive. This feature is useful when one wishes to switch and compare between plotting previously saved raw data and the currently gathered data. The Export Raw Data function allows the user to save the currently acquired data in a text file in a format suitable for reviewing, editing, or exporting to other engineering/scientific packages such as Matlab[®].¹ The first line is a

¹The bracketed rows end in semicolons so that the entire file may be read as an array in Matlab by running it as a script once the header is stripped i.e. the script should be: <array name>= [exported data file]. Variable values over time are the columns of this array; the rows are the variable value set at successive sample numbers.

text header labeling the columns followed by bracketed rows of data items gathered. The user may choose the file name with a default extension of ".text" (e.g. lqrstep.txt). The first column in the file is sample number, the next is time, and the remaining ones are the acquired variable values. Any text editor may be used to view and/or edit this file.

The Plotting menu contains the following pull-down options: Setup Plot, Plot Data, Axis Scaling, Print Plot, Load Plot Data, Save Plot Data, Real Time Plotting, and Close Window. The Setup Plot dialog box allows up to four acquired data items to be plotted simultaneously – two items using the left vertical axis and two using the right vertical axis units. Plot Data generates a plot of the selected items. Axis Scaling provides for scaling or “zooming” of the horizontal and vertical axes for closer data inspection – both visually and for printing. The Print Data option provides for printing a hard copy of the selected plot on the current PC system printer. The Load Plot Data dialog box enables the user to bring into the Executive previously saved ".plt" plot files. The Save Plot Data dialog box enables the user to save the data gathered by the controller for later plotting via Load Plot Data. The Setup Real Time Plotting dialog box enables the user to view data in real time as it is being generated by the system. Thus the data is seen in an oscilloscope-like fashion.

ECP User User-MV Software

The ECP User-MV Software is more flexible than the ECP Executive software. The main difference is in the Setup Control Algorithm. Rather than using predefined controllers, the ECP User-MV software allows the user to write control algorithms, compile them, and implement them via the DSP based controller. Figure 1.13 shows the control algorithm dialog box. The *Sampling Period* field allows the user to change the servo period " T_s " in multiples of 0.000884 seconds (e.g. 0.000884, 0.001768 etc.). The *minimum* sampling period is 0.000884 seconds (1.1 KHz). Note that if the user servo algorithm is long and/or complex the code execution may run longer than the sampling period. In such a case a *Servo Time Limit Exceeded* condition will occur which will cause the control loop to be automatically opened by the Real-time Controller. The user may then either *increase* the sampling time or edit the user servo algorithm to *reduce* execution time. In general, the combination of high sampling frequency, complex control laws and sinusoidal or sine sweep trajectories (which also require significant real-time processing) may cause the *Servo Time Limit Exceeded* condition. The User Code view box displays the latest user servo algorithm edited or loaded from the disk. You cannot edit the algorithm via the view box. You may however browse it via the arrow keys. Edit Algorithm opens up the ECPUSR Editor where servo algorithms may be created by the user. Algorithms may also be created using any text editor if saved with a “.ALG” extension.

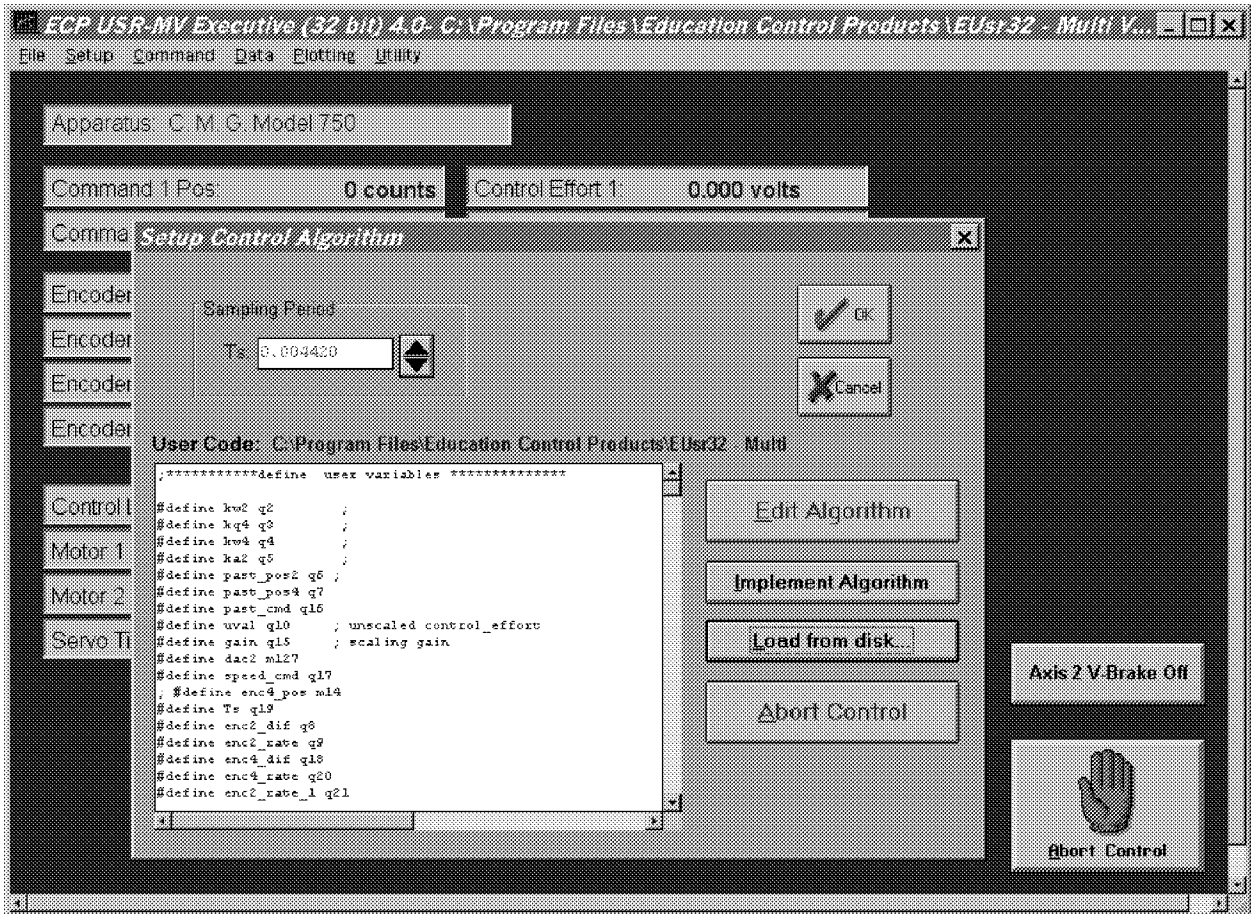


Figure A.4: Setup Control Algorithm Dialog Box

Structure of User Written Control Algorithm

Any user written control algorithm code is made up of three distinct sections:

- the definition segment
- the variable initialization segment
- the servo loop or real-time execution segment

When the ECPUSR program down loads the algorithm to the Real-time Controller, it uses the definition segment to assign internal q-variables ($q_1..q_{100}$) to the user variables defined in the definition segment. The variable initialization segment is to be used to assign values to the servo gains and/or coefficients that either remain constant or must be assigned some initial value prior to running the servo loop code. The servo loop code segment starts with a "begin" statement and ends with an "end" statement. All the legitimate assignment and condition statements between these two statements will be executed every Sample Period provided that the execution time of the code does not exceed the Sample Period. (If this occurs the " Servo Time **Limit Exceeded** " condition will be shown on the background screen and the loop will be opened up. The user may then

reduce the complexity of the algorithm between the "begin" and the "end" statements. Alternatively, if appropriate, the Sample Period may be increased.)

Definition Segment

There are 100 *general variables* `q1` to `q100` which may be used by the user for gains, controller coefficients, and controller variables. These variables are used internally by the Real-time Controller. They are stored and manipulated as 48-bit floating point numbers. For users' convenience, the `#define` statement may be used to assign to the `q`-variables text labels appropriate for particular servo algorithms. For example:

```
#define gain_1 q2 ;assigns to the variable q2 the name gain_1
```

or

```
#define past_pos1 q6 ;assigns to the variable q6 the name past_pos1
```

Note that all the text beyond the comment delimiter ";" are ignored by the Real-time Controller and may be used for annotation by the user. Also the special variables `q10`, `q11`, `q12` and `q13` may be acquired via the Data menu along with other standard collectable data. This feature allows the users to inspect critical internal variables of their specific control algorithms in addition to the command and sensor feedback positions and control effort(s).

In addition to the 100 general variables, there are eight *global variables* as follows:

`cmd1_pos`

`cmd2_pos`

`enc1_pos`

`enc2_pos`

`enc3_pos`

`enc4_pos`

`control_effort1`

`control_effort2`

The first six of these are predefined to contain the instantaneous commanded positions and actual encoder positions 1 to 4 respectively. The value assigned to the `control_effort` global variables by the user algorithm will be used as the control effort (i.e. outputs to the DAC's → servo amplifiers → motors) for that particular servo cycle - see example below.

Important Note: The above global variable names must not be used as general variable names by users in any definition statement.

Initialization Code Segment

In this segment the user may predefine the algorithm constants (gains and controller coefficients) and initial values of the algorithm's variables. For example:

```
gain_1=0.78 ;assigns to gain_1 the value of 0.78
gain_1=0.78*3/gain_3 ;requires gain_3 to be previously defined
past_pos1=0.0 ; initialize the past position cell
```

Note that the above three examples assume that the `#define` statement was used in the Definition Code segment to relate one general variable `qi` ($i=1\dots 100$) to the text variables such as `gain_1`. Also, all initialization and constant variable assignments should be done outside the servo loop code segment to maximize the servo loop execution speed.

Important Note: If the subsequent user specified servo loop algorithm controls rotor speed (i.e. `control_effort1` is an output) and the rotor speed has been initialized (see Initialize Rotor Speed, Section 2.1.6.4), the initialization code must include the statement "`m136=0`". (Alternatively, the Definition Segment could state "`#define rotorspeedreg m136`", and subsequently the initialization code would state `rotorspeedreg=0.`) This disables the background routine that controls the rotor under Initialize Rotor Speed. If this routine is not disabled, it will take precedence over the user's servo loop routine and the rotor speed will remain constant.

This becomes an issue in routines such as MIMO controllers where some initial momentum bias is required to for gyroscopic torque to be effective, but motor speed control is required to provide reactive torque. The user may wish to Initialize Rotor Speed, then take over control of the rotor torque with his/her algorithm.

Servo Loop Segment

This segment starts with a "`begin`" statement and terminates with an "`end`" statement. All the legitimate assignment and condition statements between these two statements will be executed every *Sample Period* provided that the execution of the code does not exceed the *Sample Period*.

An Example

Consider the following user-written control algorithm program:

```
***** Definition code segment*****
#define kpf q1      ;define kpf as general variable q1
#define k1 q2      ;define k1 as general variable q2 and so on
#define k2 q3
#define k3 q4
#define k4 q5
#define past_pos1 q6
#define past_pos2 q7
#define dead_band q8
```

```

;***** Initialization code segment *****
past_pos1=0      ;initialize algorithm variables
past_pos2=0
kpf=0.93        ;initialize constant gains etc.
k1=0.78
k2=3.14
k3=0.156
k4=7.58
dead_band=100   ;the size of dead band is set at 100 counts

;***** Servo Loop Code Segment *****
begin
    if ((abs(enc1_pos) !> dead_band)
        k1=k1+0.5
    else
        k1=0.78
    endif
    control_effort=kpf*cmd_pos-k1*enc1_pos-k3*enc2_pos-k3*(enc1_pos-
past_pos1)-k4*(enc2_pos-past_pos2)
    past_pos1=enc1_pos
    past_pos2=enc2_pos
end

```

This is a simple state feedback algorithm with a conditional gain change based on the size of Encoder 1 position. First the required general variables are defined. Next, their values are initialized. And finally between the "begin" and the "end" statement the servo loop code is written which is intended to run every *Sample Period*. In addition, the "if" and the "else" statements are used to change the value of the k1 gain according to the *absolute* ("abs") value of Encoder 1 instantaneous position.

Language Syntax For Real-time Algorithms

Constants

Constants are numerical values not subject to change. They are treated internally as 48-bit floating point numbers (32-bit mantissa, 12-bit exponent) by the Real-time Controller. They must be entered in decimal format as the following examples suggest:

```

1234
3
03      ; (leading zero OK)
-27.656

```

```
0.001
.001      ; (leading zero not required)
```

Variables

There are 100 general variables `q1` to `q100` which may be used by the user for gains, controller coefficients, controllers variables and program flow flags. Examples:

```
q1=10.05      ; (assign to the variable q1 the values of 10.05)
q2=q1*0.05    ; (assign to the variable q2 the value of q1*0.05)
```

Note that when the define statement is used in the Definition Code segment to give names to the appropriate `q` variable then the above two examples may be written as:

```
#define gain_1 q1
#define gain_2 q2
.
.
.
gain_1=10.05
gain_2=gain_1*0.05
```

Arithmetic Operators

The four standard arithmetic operators are: `+`, `-`, `*`, `/`. The standard algebraic precedence rules apply: multiply and divide are executed before add and subtract, operations of equal precedence are executed from left to right, and operations inside parentheses are executed first.

There is an additional `%` modulo operator, which produces the resulting remainder when the value in front of the operator is divided by the value after the operator. This operator is particularly useful for dealing with roll over condition of command or actual positions.

Functions

Functions perform mathematical operations on constants or expressions to yield new values. The general format is:

```
{function name} ({expression})
```

The available functions are `SIN`, `COS`, `TAN`, `ASIN`, `ACOS`, `ATAN`, `SQRT`, `LN`, `EXP`, `ABS`, and `INT`.

Note: All trigonometric functions are evaluated in units of radians (not degrees).

SIN	This is the standard trigonometric sine function
------------	--

Syntax	sin ({expression})
Domain	All real numbers
Domain Units	radians
Range	-1.0 to 1.0
Range units	none
Possible Errors	N/A
COS	This is the standard trigonometric cosine function
Syntax	cos ({expression})
Domain	All real numbers
Domain Units	radians
Range	-1.0 to 1.0
Range units	none
Possible Errors	N/A
TAN	This is the standard trigonometric tangent function
Syntax	tan ({expression})
Domain	All real numbers except +/- pi/2, 3pi/2, ...
Domain Units	radians
Range	-1.0 to 1.0
Range units	none
Possible Errors	divide by zero on illegal domain. (may return max. value.
ASIN	This is the inverse sine (arc sine) function with its range reduce to +/- pi/2
Syntax	asin ({expression})
Domain	-1.0 to 1.0
Domain Units	none
Range	-pi/2 to pi/2
Range units	radians
Possible Errors	illegal domain
ACOS	This is the inverse cosine (arc-cosine) function with its range reduced to 0 to pi.
Syntax	acos ({expression})
Domain	-1.0 to 1.0

Domain Units	none
Range	0 to pi
Range units	radians
Possible Errors	illegal domain
atan	This is the inverse tangent function (arc-tangent).
Syntax	atan ({expression})
Domain	all reals
Domain Units	none
Range	-pi/2 to pi/2
Range units	radians
Possible Errors	N/A
LN	This is the natural logarithm function (log base e)
Syntax	ln ({expression})
Domain	All positive numbers
Domain Units	none
Range	all real
Range units	none
Possible Errors	illegal domain
EXP	This is the exponentiation function (e^x). Note: to implement the y^x function, use $e^{x \ln(y)}$ instead. A sample expression would be exp (q1 * ln (q2)) to implement $q2^{q1}$.
Syntax	exp ({expression})
Domain	All real numbers
Domain Units	none
Range	all positive reals
Range units	none
Possible Errors	N/A
SQRT	This is the square root function
Syntax	sqrt ({expression})
Domain	All non-negative real numbers

Domain Units	free
Range	All non-negative real numbers
Range units	free
Possible Errors	illegal domain
ABS	This is the absolute value function
Syntax	<code>abs({expression})</code>
Domain	All real numbers
Domain Units	free
Range	all non-negative reals
Range units	free
Possible Errors	N/A
INT	This is a truncation function which returns the greatest integer less than or equal to the argument, e.g. (<code>int(2.5) =2</code> , <code>int(-2.5)=-3</code>)
Syntax	<code>int({expression})</code>
Domain	All real numbers
Domain Units	free
Range	integers
Range units	free
Possible Errors	none

Expressions

An expression is a mathematical construct consisting of constants, variables and functions connected by operators. Expressions can be used to assign a value to a variable or as a part of condition (see below). A constant may be used as an expression, so if the syntax calls for `{expression}`, a constant may be used as well as a more complicated expression. Examples of expressions:

```
#define variable_1 q1
#define variable_2 q2
.
.
.
512
variable_1
variable_1 -variable_2
```

```
1000*cos(variable_1*variable2)
abs(cmd_pos)
```

Note that the define statements should be used to define the user variables in terms of q variables. In addition, these variables should be initialized.

Variable Value Assignment Statement

This type of statement calculates and assigns a value to a variable. Example:

```
control_effort= kp*(cmd_pos-enc1_pos)
```

Comparators

A comparator evaluates the relationship between two values (constants or expressions). It is used to determine the truth of a condition in Servo Loop Code segment via the `--IF` statement (see below). The valid comparators are:

```
=      (equal to)
!=     (not equal to)
>      (greater than)
!>    (not greater than; less than or equal to)
<      (less than)
!<    (not less than, greater than or equal to)
```

Note: the comparators `<=` and `>=` are not valid. The comparators `!>` and `!<`, respectively, should be used in their place.

Conditional Statement

In the Servo Loop Code segment between the `begin` and the `end` statements, the `if` statement may be for conditional execution of parts of the control algorithm. The format is as follows:

```
if ( {condition})
    valid expression
    .
    .
endif
```

Also the `else` statement may be used as follows:

```

if ( {condition})
    valid expression
    .
    .
else
    valid expression
    .
    .
endif

```

If the {condition} is true, with no statement following on the line of the `if` statement, the Real-time Controller will execute all subsequent statements on the following lines down to the next `endif` or `else` statements.

A {condition} is made up of two expressions compared via the comparators described above. For Example:

```

if (control_effort>16000)
    control_effort=16000
endif

```

or,

```

If (enc2_pos>1.1*deadband)
    kp=1.0
else
    kp=1.2
endif

```

The {condition} statement may be in a **compound** form using the `and` and the `or` operators as follows:

```

If (enc2_pos>1.1*deadband and enc1_pos<enc2_pos)
    kp=1.0
else
    kp=1.2
endif

```

Note that the condition in the `if` statement line must be surrounded by parenthesis. Also avoid nesting more than three layers of `if` statements.