SV1simc Manual

Concurrent Dynamics International

May 2014

Objectives

- Part I:
  - Build a model file (SV1simc.txt) to simulate a satellite with 4 reaction wheels and 8 jets
  - Build simplot1.m to view sim results
- Part II:
  - Build control.dll to interact with the main program xsv01.exe during run time to simulate the dynamics and control of the SV1simc vehicle.
  - Xsv01.exe Examples:
    - use reaction wheels to maintain LVLH attitude with jets idled
    - use reaction jets and wheels to maintain LVLH attitude

License Restrictions

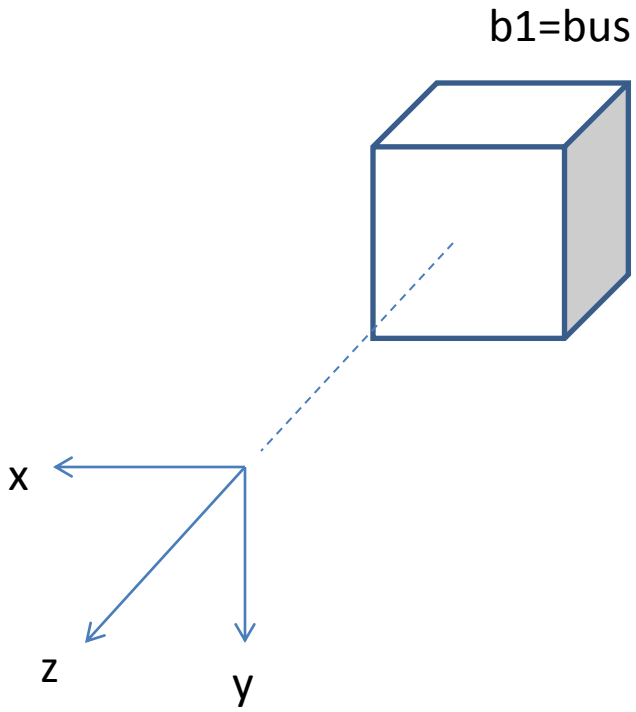
License type	Buildx.exe	Xsv01.dll
Enterprise	none	none
Project	Must stay with the object count specified by license gflag <=12	Runs with model_files with license specified object count gflag <=12

- Project license permits satellite simulations of satellites with a specified object count in {bodies, wheels, forces} and in a unique configuration. No restrictions are placed on the mass property of bodies and wheels, and force placement and parameters or initial conditions.
- Project license permits gravity model with gflag <=12 (see page 35)

Part I Buildx Topics

- Physical model
- Buildx tasks
- Key files
- Main Menu
- Model Menus
- Model Items
- Body Menu
- B1 page
- Body Actuation Signals
- Wheel menu
- Force menu
- Gravity Menu
- Dynamics Input
- Dynamics Output
- Plot Data
- Simplot
- Save Model
- Exit Buildx
- Q & A

SV1simc Model



reaction wheels axes on b1:

w1=[1 -1 1]/sqrt(3)
w2=[1 1 1]/sqrt(3)
w3=[-1 1 1]/sqrt(3)
w4=[-1 -1 1]/sqrt(3)

jet forces on b1:

f1.pos=[-3 3 0] , f1.vec=[1 0 0]
f2.pos=[-3 -3 0] , f2.vec=[1 0 0]
f3.pos=[-3 0 3] , f3.vec=[1 0 0]
f4.pos=[-3 0 -3] , f4.vec=[1 0 0]
f5.pos=[0 3 3] , f5.vec=[0 -1 0]
f6.pos=[0 3 -3] , f6.vec=[0 -1 0]
f7.pos=[0 0 0] , f7.vec=[0 0 0]
f8.pos=[0 0 0] , f8.vec=[0 0 0]

On the Simulation

- We build a model file in Part I to simulate the dynamics and control of a vehicle in orbital environment. The model file defines the mass property of the vehicle and reaction wheels. It also defines jet forces, vehicle's initial orbit motion and the dynamics input and output signals for the control system.

- Part II presents the procedure to generate the control.dll required by xsv01.exe for the simulation. Examples are then presented to run xsv01.exe with control.dll to simulate the control/dynamics as specified by sv1simc.txt

Buildx Tasks

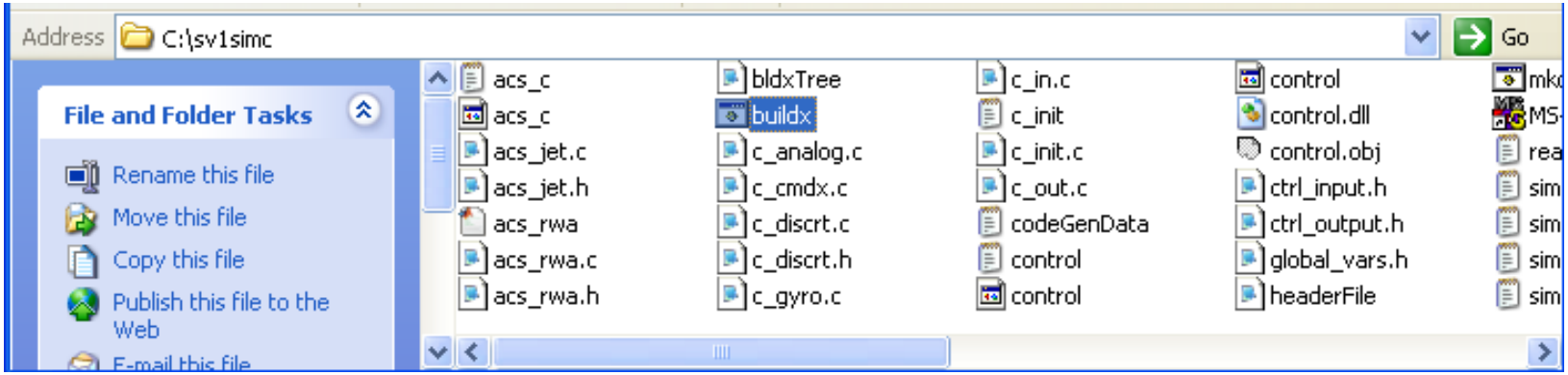
- Build a model file to define a satellite simulation:
  - {mass property,model connectivity, degree of freedom,xsv01.dll input/output,plot data,gravity model,…etc}
- Build control.cpp template for SV1simc control system
- Edit & browse the attributes of satellite components
  - {bodies,wheels,forces,markers,gravity,initial state,…}
- Build a Matlab plot script, simplot1.m, to view sim results

Key Files in c:\SV1simc

- Buildx.bat='..\Buildx'
- Input files router: siminput.txt
- Working files: sim1files.txt
- Simplot text: i.e. simplot1.txt
- License: simlicense.txt

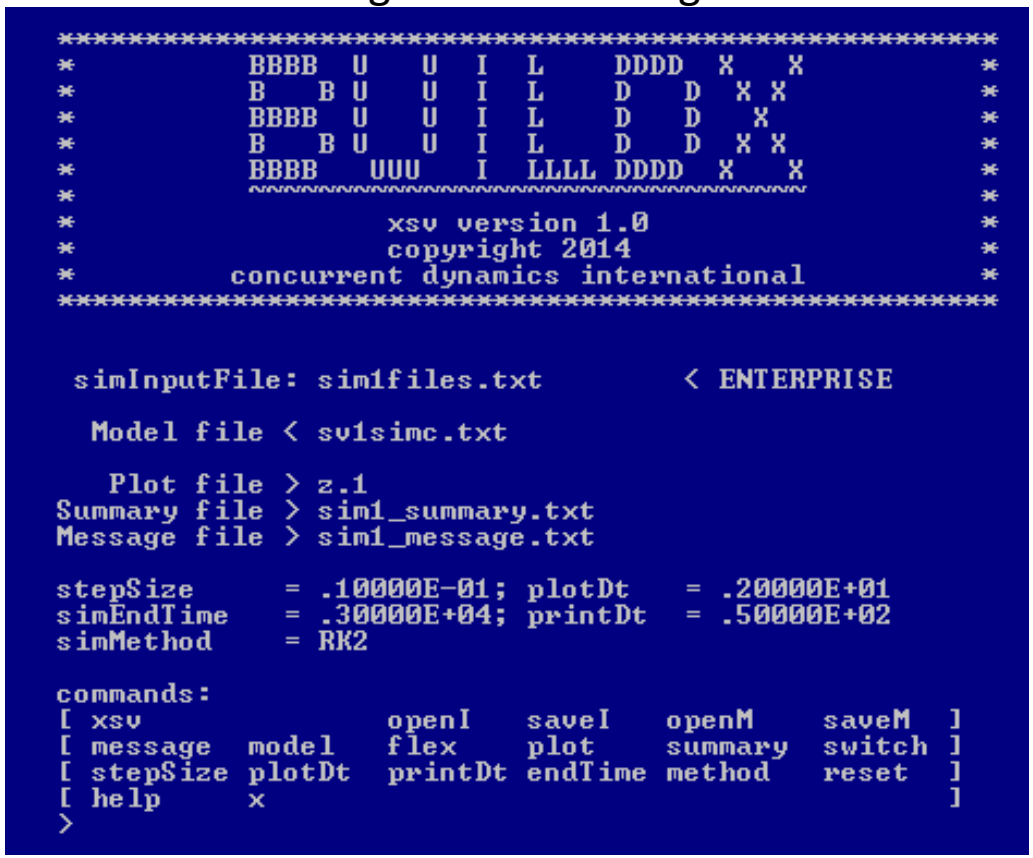
Starting Buildx.exe

- Click c:\SV1simc\Buildx.bat to start Buildx.exe and see the Main Menu



Main Menu

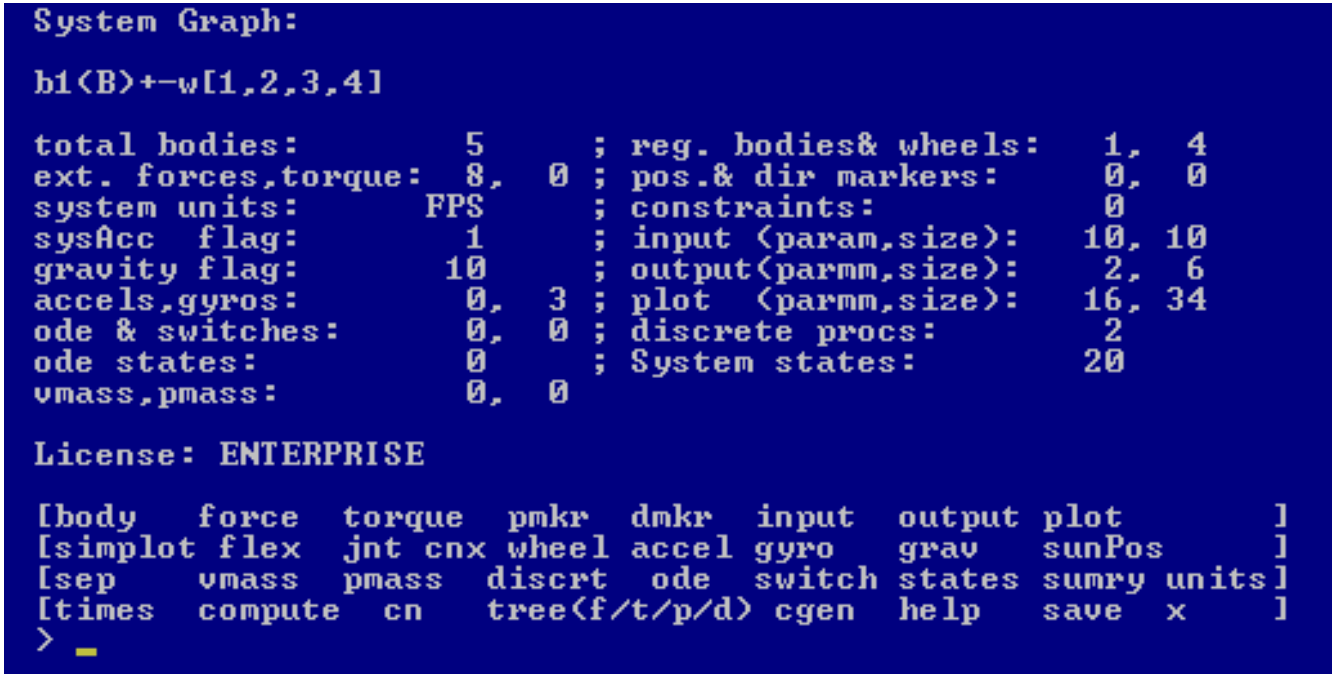
- Shows all working files and timing data defined in the simInputFile



- Type 'xsv' to go to Model Menus

Model Menus

- See model parts size or to choose menus to edit/browse



- SV1simc has 1 regular body, 4 wheels and 8 forces

Sv1simc Model Items

Defined Items	Menu	Parameters/data
1 body	Body	Mass, inr, svec,dvec, dcm0, axis,ang,wrel,parent,type
4 wheels	Wheel	Winr, axis, wspd, parent, type
8 forces	Force	Fvec,fpos, parent,type
Plant input	Input	Xv01.dll input data list
Plant output	Output	Xsv01.dll output data list
Plot data	Plot	Xsv01.dll plot data list
Orbit	Gravity	Vehicle's orbit position and velocity

- Attributes of the defined items can be seen by going to the Menus specified above. Use the Model Menu commands to do that.
- Part I gives the instructions on how to define the above parameters and data
- If already familiar with Part I, go to Part II on page 50

Model Menus Commands

- Type 'body’ to go to Body Menu
- Type 'force’ to go to Force Menu
- Type 'grav’ to go to Gravity Menu
- Type 'pmkr’ to go to Position Marker Menu
- Type 'dmkr’ to go to Direction Marker Menu
- Type 'input’ to go to Input Menu
- Type 'output’ to go to Output Menu
- Type 'plot’ to go to Plot Menu
- Type 'simplot’ to go to SimPlot Menu
- Type 'time’ to go to Timing Menu
- Type 'help’ to get definition on data and commands
- Type 'x’ to exit menu

Body Menu

- See body summay, type ‘body’ from Model Menu
- SV1simc has 1 body, i.e. b1

```
                          ~ Body Menu ~
   no. of bodies :  1
   sysh_eci      :       2.094       10.966       -1.262
   sysh_b1       :       3.491       10.472       -2.094
   syscm         :        .000          .000          .000

   dvec =          .000          .000          .000
   svec =          .000          .000          .000
   hpos =          .000          .000          .000
   rpos =          .000          .000          .000
   wrel =          .200          .500         -.100
   inr  =   1000.000    1200.000    1200.000
                .0000         .0000         .0000

   idx name     pa  u  fl vm tp ax -- angle - -- mass --
=>  1 b1         0 FPS  0  -  B  x         .000     100.000

   [ sel  edit  idx   name  par  dvec  svec  type  whl  units]
   [ axis  ang  dpos  wrel dvel  inr   mass  hpos  hvel rpos  ]
   [ add  addF  cnx   jnt  rem   dmkr  pmkr  gvec  rvel  w    ]
   [ brch  cn   copy  up   down  dolc  save  help  zero  x    ]
   > _
```

Body Menu Data & Commands

- Sysheci — system angular momentum in ECI frame
- Syshb1 — sysheci in b1 frame
- Syscm — system cm in b1 frame
- Dvec-inr — position, rate and inertia of the body pointed by '=>' arrow
- Column data
- Pa — parent body index
- U — units of measure {FPS=English, MKS=Metric}
- Tp — joint type {a-h}
- Ax — local axis for 1 dof joints {x, y or z}
- Ang — initial joint angle (deg)
- Mass — body mass

- Body Menu Commands:
- Type 'add<j>’ to add bodies to bj : i.e. ‘add1’ to add bodies to b1
- Type 'rem<j>’ to remove bj from list
- Type 'name<j>’ to edit bj.name
- Type 'par<j>’ to edit bj.parent
- Type 'type<j>’ to edit motion type, bj.type: {a…h}
- Type 'axis<j>’ to edit bj.axis: {x, y or z}
- Type 'ang<j>’ to edit initial inboard bj.ang for 1 dof rotational joints
- Type 'wrel<j>’ to edit bj.angular_rate
- Type 'mass<j>’ to edit bj.mass
- Type 'svec<j>’ to edit bj.svec
- Type 'dvec<j>’ to edit bj.dvec
- Type 'inr<j>’ to edit bj.inr
- Type 'edit<j>’ to see all data on bj
- Type 'help’ to get definition on data and commands
- Type 'x’ to exit menu

3 Ways to Edit Body Data

- Direct edit: type the following commands for immediate edit
  - name<j>, type<j> axis<j>, ang<j>, mass<j>
  - units<j>
  - i.e. ‘name1’ to change b1 name

- Parameter menu edit: type any of the following parameters and go to named menu and then edit
  - {dvec, svec, inr, wrel, dpos, dvel, … }

- Selected body edit: Type 'edit<j>’ to edit all parameters of bj

Inertia Menu

- Need define moi of each body bj about the bj.cm
- Type 'inr’ from Body Menu to see a summary of moi data

```
   idx name    -     ixx    - -     iyy    - -     izz    -
               ---- ixy --- ---- ixz --- ---- iyz ---
 =>  1 b1       .100000E+04   .120000E+04   .120000E+04
                .0000000E+00  .0000000E+00  .0000000E+00
```

- Type 'inr<j>’ to edit bj.inr
- Type 'x’ to exit menu

Dvec Menu

- Need define dvec(j), bj.hinge.position in bj.parent frame, for each j
- Type 'dvec’ from Body Menu and see a dvec summary

```
   idx name        u ------------ ----dvec---- ------------
 =>  1 b1        FPS   .000000E+00   .000000E+00   .000000E+00
```

- note: by design b1.dvec=0
- Type 'dvec<j>’ to edit bj.dvec
- Type 'x’ to exit menu

Svec Menu

- Need define svec(j), bj.position.cm in bj.local frame, for each j
- Type 'svec' from Body Menu and see an svec summary

```
   idx name        u ------------ ----svec---- ------------
 =>  1 b1        FPS   .000000E+00   .000000E+00   .000000E+00
```

- note: b1.cm here is collocated with b1.hinge point, but can be nonzero
- Type 'svec<j>' to edit bj.svec
- Type 'x' to exit menu

Pos Summary

- See where bj.cm is in b1 frame for all j
- Type 'rpos’ in Body Menu to see a summary of bj.cm

```
   idx name   ---------- ---rpos--- ----------
 =>   1 b1          .000        .000        .000
```

- note: b1.cm here is collocated with b1.hinge point but need not be so

All b1 Data

- See all data on b1, type ‘edit1’ from Body Menu

```
idx   name      par gflg   u   tp ax          ang          mass
  1 b1            0   10 FPS   b  x           .000   .10000E+03

> inertia(inr)= .10000E+04   .12000E+04   .12000E+04
                .00000E+00   .00000E+00   .00000E+00
> dVec         =       .000         .000         .000
  hngVec       =       .000         .000         .000
> sVec         =       .000         .000         .000
  rVec         =       .000         .000         .000
> dcm0         =   1.000000     .000000     .000000
                   .000000    1.000000     .000000
                   .000000     .000000    1.000000
  Euler seq    =123
  Euler angs   =       .000         .000         .000

  b2i matrix   =   -.000002     .000000   -1.000000
                   .422618     .906308    -.000001
                   .906308    -.422618    -.000002

> wRel         =      .2000        .5000       -.1000
> dPos         =       .000         .000         .000
> dVel         =       .000         .000         .000
  force on b   =       .000         .000         .000
  torque on b  =       .000         .000         .000
```

Bj Page Info

- 1st block: all attributes of bj, {idx, … torque on b}

- 2nd block: commands to change attributes in block1 or to go to another Body Menu: {idx, axis,… x}
  - idx<j>: goes to another bj page
  - dcm0: edit relative dcm of bj

    for j=1, this is the b1 attitude wrt LVLH frame
  - svec: edit the bj cm position in bj coord, etc…
  - help: see data and command definitions
  - x: exit this page

Body Actuation Signals

- Bj Inboard force or torque actuates that body and impacts the motion of the rest of the system. Accelerations can be specified for joints with prescribed motion
- The Dynamics Input signals for bj depends on bj.type. They are processed as follows:

type	Size	Input	processing
A	1	Htqax,j	bj.torque(axis)=Htqaxj
B	3	Htq,j	bj.torque=Htqj
C	1	Wraccax,j	bj.wracc(axis)=Wraccaxj
D	3	Wracc,j	bj.wracc=wraccj
E	1	Frcax,j	bj.force(axis)=frcaxj
F	3	Frc,j	bj.force=frcj
G	1	Hraccax,j	bj.hraccax=hraccaxj
H	3	Hracc,j	bj.hracc=hraccj

Wheel Menu

- See wheel summary, type ‘whl’ from Model Menu
- SV1simc has 4 wheels all mounted on b1 (see pa)

```
                        ~ Wheel Menu ~
   nof wheels      :   4
   Sysh_eci        :         1.52          1.54          -.43
   SysCm           :         .000          .000          .000

   w_inr           :       .10000
   w_spd(rpm)      :         .000
   w_mom           :         .000

   az_axis         :   2
   axis(1) az(d)   :   45.000000
   axis(1) el(d)   :  -35.264390

   units           : FPS

   idx name    pa t ---------- ---axis--- ---------- --winr- -w(rpm)-
=>   1 whl1     1 A     .577350    -.577350     .577350    .1000         .0
     2 whl2     1 A     .577350     .577350     .577350    .1000         .0
     3 whl3     1 A    -.577350     .577350     .577350    .1000         .0
     4 whl4     1 A    -.577350    -.577350     .577350    .1000         .0

   [ sel  name  units  par  type  inr   wspd  axis  azel azAxis long ]
   [ show order  idx   add  rem   copy  save  help   x                ]
   >
```

Wheel Menu Data & Commands

- Data:
- whlj.parent= 1, for all j, means all wheels are on b1
- whlj.type=A means whlj.input is scalar wheel torque
- whlj.axis= [x y z] , whlj spin axis in parent frame
- whlj.speed= initial whlj speed
- whlj.inr=whlj spin axis inertia

- Commands:
- Type 'add<j>' to add wheels to bj
- Type 'rem<j>' to remove whlj
- Type 'name<j>' to edit whlj.name
- Type 'type<j>' to edit whlj.type={A or C}
- Type 'axis<j>' to edit whlj.axis
- Type 'wspd<j>' to edit whlj.wspd
- Type 'help' to get definitions of data and commands
- Type 'x' to exit menu

Wheel Control Signals

- Dynamics Input signals for whlj: whltqj, whlaccj
- Run time input processing of wheel control signals:

whlj.type	Input signal	size	Sim action
A	whltqj	1	whlj.tq= whltqj
C	whlaccj	1	whlj.acc= whlaccj

Force Menu

- See force summary, type ‘force’ from Model Menus
- SV1simc has 8 jet forces
- fj.parent=1 for all j, means they all impinge b1
- fj.type=1 for all j means that the xsv01.dll input for them are 1/0 signals

	idx	name	p	t	c	---fmag---	----fx----	----fy----	----fz----
=>	1	f1	1	1	1	1.000	1.000	.000	.000
	2	f2	1	1	1	1.000	1.000	.000	.000
	3	f3	1	1	1	1.000	1.000	.000	.000
	4	f4	1	1	1	1.000	1.000	.000	.000
	5	f5	1	1	1	1.000	.000	-1.000	.000
	6	f6	1	1	1	1.000	.000	-1.000	.000
	7	f7	1	1	1	.000	.000	.000	.000
	8	f8	1	1	1	.000	.000	.000	.000

- note: f7,f8 are not given force values because they are not needed by acs_jet.m of SV1simc1

- Type ‘pos’ to see the position of defined forces in b1 coord as next

	idx	name	p	t	c	---fmag---	---posx---	---posy---	---posz---
=>	1	f1	1	1	1	1.000	-3.000	-3.000	.000
	2	f2	1	1	1	1.000	-3.000	3.000	.000
	3	f3	1	1	1	1.000	-3.000	.000	3.000
	4	f4	1	1	1	1.000	-3.000	.000	-3.000
	5	f5	1	1	1	1.000	.000	3.000	3.000
	6	f6	1	1	1	1.000	.000	3.000	-3.000
	7	f7	1	1	1	.000	.000	.000	.000
	8	f8	1	1	1	.000	.000	.000	.000

- Type ‘rxf’ to see the torque of defined forces in b1 coord

	idx	name	---tqmag--	---tqx----	---tqy----	---tqz----
=>	1	f1	3.000	.000	.000	3.000
	2	f2	3.000	.000	.000	-3.000
	3	f3	3.000	.000	3.000	.000
	4	f4	3.000	.000	-3.000	.000
	5	f5	3.000	3.000	.000	.000
	6	f6	3.000	-3.000	.000	.000
	7	f7	.000	.000	.000	.000
	8	f8	.000	.000	.000	.000

Force Menu Data & Commands

- Data:
- fj.parent= 1, for all j, means all jets are on b1 (bus)
- fj.type=1 means fj.input is an on/off signal
- fj.fvec= [x y z] ,directional vector of fj
- fj.fmag= magnitude of fj when activated
- fj.fpos=[x y z], impact position of fj in parent frame

- Commands:
- Type 'add<j>' to add external forces to bj
- Type 'rem<j>' to remove fj
- Type 'name<j>' to edit fj.name
- Type 'type<j>' to edit fj.type={1,2 or 3}
- Type 'fvec<j>' to edit fj.fvec
- Type 'fpos<j>' to edit fj.fpos

- Type 'par<j>' to edit fj.parent
- Type 'pos' to show all force impact positions in b1 frame
- Type 'rxf' to show torque of all forces about f_ref in b1 frame
- Type 'help' to get definitions of data and commands
- Type 'x' to exit menu

- Dynamics Input signal for fj is xfj

Run time input processing of xfj:

fj.type	xfj	Size	Sim Action
1	1=on, 0=off	1	fj.force= xfj*fj.fvec
2	Scalar magnitude of fj.force	1	fj.force= xfj*fj.unitv
3	3 x 1 force vector in b1 frame	3	fj.force= xfj

Gravity/Orbit Menu

- Need to specify the vehicle orbit
- Type 'grav' from Model Menu to enter Gravity Menu
- SV1simc orbit :
  - 35 deg inclined circular orbit
  - True anomaly= 90 deg
  - orbit period is 100 minutes
  - gflag= 10, meaning spherical earth gravity model
  - epoch: 12.23.2009 (see sunpos menu) relates to LST

Gravity Menu

```
> units      (U)=  FPS
> syspos        =    37130491.887          27.245          58.427
> sysvel        =            -.034        8232.778       17655.249

> refpos        =    37130491.887          27.245          58.427
> refvel        =            -.034        8232.778       17655.249

  gravity:
> gx gy gz      =  -10.2101349554    -.0000074919    -.0000160664
  mu            =  .140764418E+17

  ephemeris:
> semimajor (U)=    37167659.547
> ecc           =      .001000
> incl    (deg)=    65.000000
> rasc    (deg)=      .000000
> argp    (deg)=      .000000
> t_anom  (deg)=      .000099
  e_anom  (deg)=      .000099
  m_anom  (deg)=      .000099
  m_motion(d/s)=    .03000000

> LST_ang (deg)=  268.291360
> LST(h:m:s)    = 17:53: 9.9

> period  (min)=         200.000; revs/day=   7.200
> period  (sec)=       12000.000
  range      (U)=    37130491.887
  equ. radius  =    20925646.325; J2=  .108263E-02
  prg.altitude =    16204845.562; apg.altitude=    16279180.881
  we  (d/s,r/s)=        .00417807        .00007292

> sysacc flag   =  1
> gravity flag  = 10
> atd option    = LVLH attitude unchanged
```

Gravity Model Selections

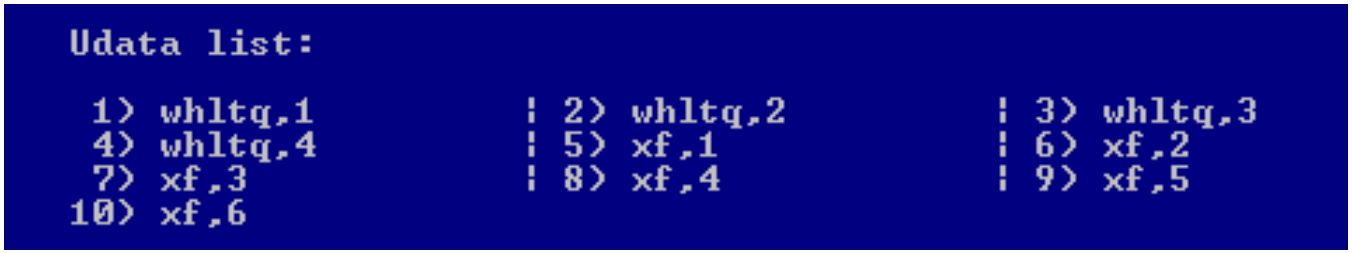
- Gravity flag: gflag
  - 0 g is fixed as given by [gx gy gz] (flat earth)
  - 10 g at bj.cm is defined by syspos (spherical earth)
  - 11 g at bj.cm is defined by bj.pos.eci
  - 12 same as #11 plus gravity gradient torque on bj
  - 20 g at bj.cm with J2 is defined by syspos (oblate earth)
  - 21 g at bj.cm with J2 is defined by bj.pos.eci
  - 22 same as #21 plus gravity gradient torque on bj
  - 30 g at bj.cm with J2, J3 and J4 is defined by syspos (oblate earth)
  - 31 g at bj.cm with J2, J3 and J4 is defined by bj.pos.eci
  - 32 same as #31 plus gravity gradient torque on bj

Gravity Menu Commands

- Type 'spos’ to edit orbit position in eci coordinates
- Type 'svel’ to edit total velocity in eci coordinates
- Type 'grav’ to edit gravitational acceleration, [gx, gy, gz]
- Type 'semi’ to edit semimajor axis
- Type 'ecc’ to edit eccentricity, … and so forth
- Type 'perm’ to edit orbit period in minutes
- Type 'sflag’ to run simulation in prescribed(spos,svel) mode or in force determined(spos,svel) mode
- Type 'gflag’ to select the gravity model for the simulation
- Type 'help’ to get definitions of data and commands
- Type 'x’ to exit menu
- Buildx automatically updates all orbit parameters when one of them is altered, i.e. changing ‘perm’ results in a new (spos, svel, ephemeris) …etc.

Dynamics Input

- Need input data to xsv01.dll to actuate the vehicle dynamics during run time
- Type ‘input’ from Model Menus to open the input menu to select data
- Use ‘newlist’ to get a suggested list for the current model
- Use ‘add’ and ‘rem’ command to modify current input list (udata)
- SV1simc input list is as follows:



- whltq, 1:4= four rwa torque
- xf,1:6 = six jet on/off signals

Input Menu Commands

- Type 'add’ to add new variables to the end of udata list
- Type 'add<j>’ to insert new variables at udata(j)
- Type 'rem’ to remove a group of variables
- Type 'rem<j>’ to remove udata(j)
- Type 'chg<j>’ to change udata(j)
- Type 'len’ to see ordinal position of udata and their length
- Type 'x’ to exit udata menu

- A variable selection menu appears on commands {add, chg}
- Type 'sel<j>’ to select var(j) to add or chg
- Type 'x’ to return to udata menu

- Procedure to define input signals for SV1simc:
    1. Type ‘add’ from Input Menu and see an Input Selection Menu
    2. Type ‘sel17’ for ‘whltq’ variable
    3. Reply the prompt with ‘1, 4’ to select htqax1:4
    4. Type ‘sel22’ for ‘xf’variable
    5. Reply the prompt with ‘1, 6’ to select xf1:6
    6. Type ‘x’ to return to Input Menu
    7. Type ‘x’ to exit Input Menu

Dynamics Output

- Need output motion signals from the dynamics engine to drive control system during run time
- Type 'output' from Model Menus to open the output menu to select signals
- Use 'newlist' to get a suggested list for the current model
- Use 'add' and 'rem' command to modify current output list (ydata)

- wrel, 1 = bus angular rate in b1 frame
- b2osml, 1= small angle b1.attitude_err wrt LVLH frame

Output Menu Commands

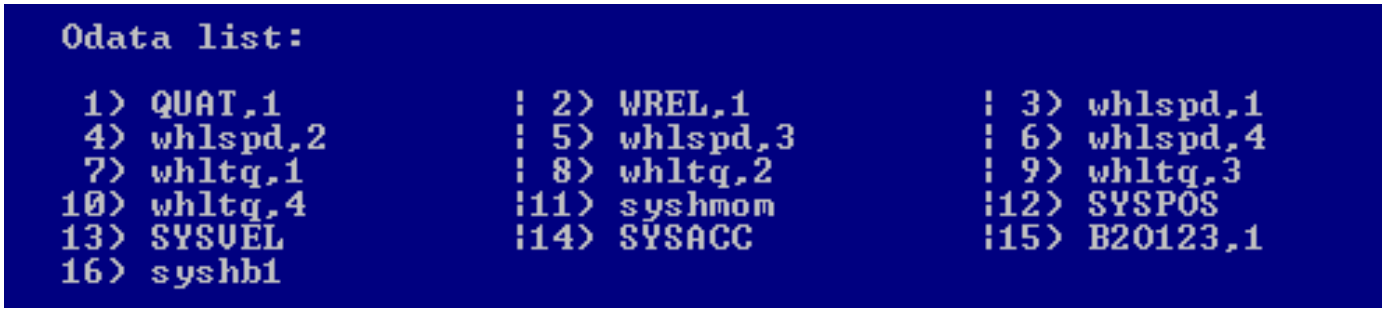
- Type 'add’ to add new variables to the end of ydata list
- Type 'add<j>’ to insert new variables at ydata(j)
- Type 'rem’ to remove a group of variables
- Type 'rem<j>’ to remove ydata(j)
- Type 'chg<j>’ to change ydata(j)
- Type 'len’ to see ordinal position of udata and their length
- Type 'x’ to exit ydata menu

- A variable selection menu appears on commands {add, chg}
- Type 'sel<j>’ to select var(j) to add or chg
- Type 'x’ to return to ydata menu

- Procedure to define output signals for SV1simc:
    1. Type ‘add’ from Output Menu and see an Output Selection Menu
    2. Type ‘sel65’ for ‘wrel’ variable
    3. Reply with ‘1, 1’ to select wrel1
    4. Type ‘sel3’ for ‘b2osml’ variable
    5. Reply with ‘1, 1’ to select b2osml1
    6. Type ‘x’ to return to Input Menu
    7. Type ‘x’ to exit Input Menu

Plot Data

- Sample selected data from the dynamics engine to plotfile during run time
- Type 'plot' from Model Menus to open the Plot Menu to select plot data
- Use 'newlist' to get a suggested list for the current model
- Use 'add' and 'rem' command to modify current plot data list (odata)
- SV1simc plot list is as follows:



- quat, 1= b1.attitude_quaternion in eci frame
- wrel, 1= bus angular rate in b1 frame … etc.
- syshb1= system angular momentum in b1 frame

Plot Menu Commands

- Type 'add’ to add new variables to the end of odata list
- Type 'add<j>’ to insert new variables at odata(j)
- Type 'rem’ to remove a group of variables
- Type 'rem<j>’ to remove odata(j)
- Type 'chg<j>’ to change odata(j)
- Type 'len’ to see ordinal position of udata and their length
- Type 'x’ to exit odata menu

- a variable selection menu appears on commands {add, chg}
- Type 'sel<j>’ to select var(j) to add or chg
- Type 'x’ to return to odata menu

- Procedure to define Plot Data:
    1. Similar to those given for Input and Output data selection
    2. quat, 1= b1.attitude_quaternion in eci frame
    3. wrel, 1= b1.angular rate … etc.
    4. syshb1= system angular momentum in b1 frame

Simplot

- Need to construct simplot1.m to view sim results in Matlab, then type ‘simplot’ from Model Menus page (page 12) or Plot Menu (page 43)
- All plot data were selected in Plot Menu
- A. steps from simPlot Menu:
  1. Type 'add’ to add figures and respond with ‘4’ to create 4 figures
  2. Type 'title1’ to set figure (1) title: i.e. reply with ‘system’
  3. for ‘title2’ and ‘title3’ commands, respond with ‘bus’, ‘wspd’ and ‘wtq’ respectively
  4. Type 'vars1’ to define variables to be plotted in fig 1. this opens the plot variables page
- B. steps from vars menu:
  1. Type 'addv11’ and reply with 4 to add 4 variables starting with the variable(11), syshmom
  2. Type 'addv16’ and reply with 1 to add ‘syshb1’ to the variable list
  3. Type 'addp’ and respond with ‘1,5’ to create five subplots for figure(1)
  4. Type 'format’ and respond with ‘3,2’ to plot 6 subplots in 3 rows and 2 columns format
  5. Type 'x’ to go back to simPlot Menu
- Repeat steps A.3 and all B steps with proper indexing to define subplots of other figures for SV1simc (i.e. ‘bus’, ‘wspd’ and ‘wtq’)
- Final steps from simPlot Menu:
  1. Type 'save’ and reply with ‘simplot1.txt’ to save simplot data to simplot1.txt
  2. Type 'make’ to create simplot1.m, see completion message
  3. Type 'x’ to exit simPlot Menu. Simplot1.m is ready.

Save Model Data

- Need to save current data to a model file
- Type 'save' from Model Menus (page 12) and complete the save dialog as follows

```
[body   force  torque  pmkr  dmkr  input  output plot        ]
[simplot flex  jnt cnx wheel accel gyro   grav   sunPos     ]
[times  vmass  pmass  discrt  ode  switch states sumry units]
[compute  cn   tree(f/t/p/d)       cgen   help   save  x     ]
> save

Commands:
1. save model data to sv1sim.txt
2. save model data to another file
3. Cancel save
Select 1:3? 1

model data has been saved to sv1sim.txt
```

- On hitting return, current model data would have been saved to SV1simc.txt
- Try option 2

Exit Buildx

- 3 ways to exit Buildx:
  - go to Model Menus or Main Menu and type ‘q’ <return>
  - go to Main Menu and type ‘x’
  - Click the ‘x’ on top right corner of the Buildx window

- note: Buildx.exe does not save model data automatically. see save procedure on page 47

Q & A

- Can one add and delete bodies, wheels and forces?
  - yes if you have enterprise license, and no if you have a project license
- Can jet forces placement and alignment be specified like in real vehicles?
  - Yes, one must in every case match the jet controller with the torque characteristics of the jets
- Are all Project licenses strictly for object count {1, 4, 8}?
  - no, for example a Project license for a vehicle with 4 CMG's has an object count of {5, 4, 8} and a unique parent-child relation between bodies, wheels and forces
- How to setup SV1simc as a 3 wheel system under a Project license?
  - go to xsv.wheel menu and set whl(4).type=3, whl(4).wspd=0
  - edit whlj.axis, whlj.inr, whlj.wspd for j=1:3 as necessary
- How to setup satsim with < 8 jets under a Project license?
  - as an example, let's disable jet(7:8):

    go to xsv.forcemenu and set frc(7:8).type=1 and not include these in xsv.input list (udata)

- How can one see all the available input, output and plot variables when choosing them from udata, ydata and odata menus?
  - all available variable list is shown when one types ‘add’ command from the menu
  - Type 'defj’ to get the definition of variable(j) in that list
  - Type 'selj’ from the add menu to select variable(j) to the list

- How does one change the plot data sample period?
  - Type 'plotdt’ from the Main Menu or from the times menu to do that

- Why are there ‘dt’ and other time specification in the times menu?
  - those time specifications are not used for the Simulink applications, they are for the Fortran and C implementation of xsv01 engine

Part II Topics

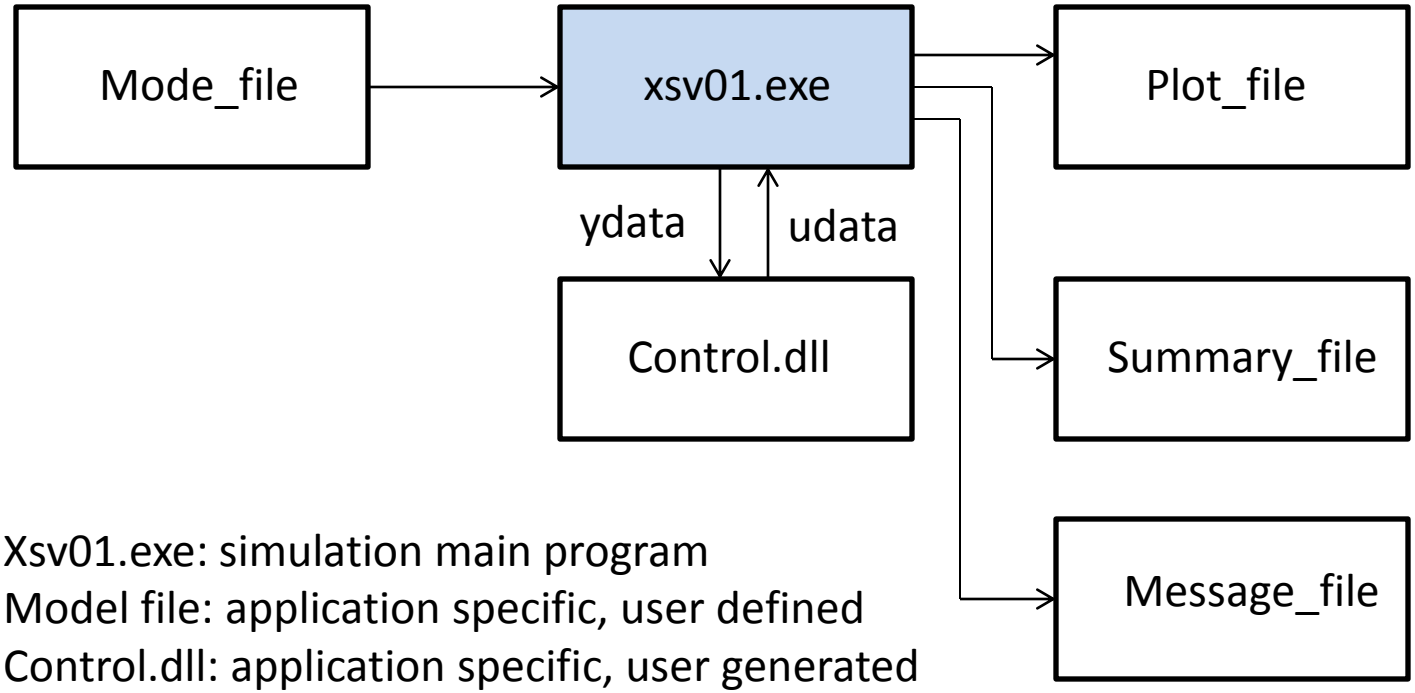
- Xsv01.exe
- How Control.dll Works
- User_code.c
- Vehdata.h
- Utilc.lib subroutines
- Acsc.lib subroutines
- Control System
- Example1
- Example2
- Adjustable Simulation Parameters
- Exercises
- Simulation Notes
- Summary

XSV01.exe

- Xsv01.exe is the main program. It comes with the SV1simc package. Its functions are to:
  - Read data from model file to initialize the simulation database
  - Integrate numerically the motion equations required by the model file while passing motion signals to and receiving actuation signals from the control system represented by control.dll.

- Control.dll is the application specific control system that user provides to cause the vehicle motion respond in a desired manner.

Xsv01.exe Dataflow



Xsv01.exe: simulation main program
Model file: application specific, user defined
Control.dll: application specific, user generated

- Model_file: defines all parameters of the vehicle for the simulation
- Summary_file: records all the model parameters and initial condition of an xsv01.exe run
- Plot_file: a file of sampled plot data recorded during a simulation
- Message_file: messages from xsv01.exe during a run

> Part I has already described how to build the model file for sv1simc.

> Part II begins next

Part II Introduction

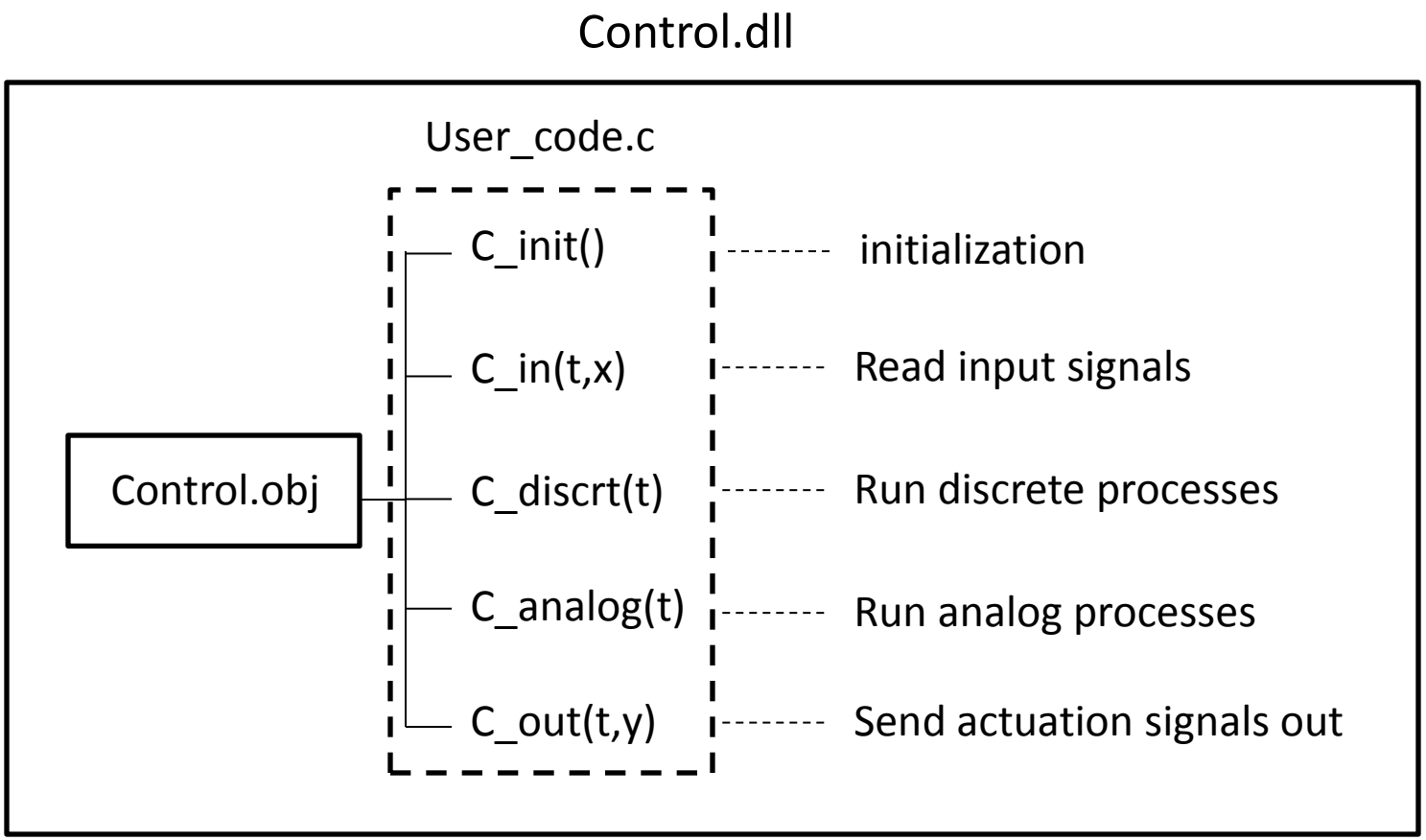
- The following charts show how to build a trial control.dll that has the framework of the control system needed for the application.
- Two parts need to be built: control subroutines and header file
- At a minimum, we would use Buildx.Codegen Menu to construct 5 key control subroutines and their header files.
- Any discrete/analog process in the control system counts as an additional control subroutine that Codegen must account for.
- Special control subroutine names cause Codegen to insert practical details into those routines
- Constants and utility routines inserted by Codegen in the trial control.dll need be replaced with those based on analysis and application specific algorithms later.
- A functional control.dll should be obtained after a few iterations of running it with xsv01.exe and correcting errors.
- From that point on, user can expand the size and details of the control subroutines to satisfy their simulation needs.

How Control.dll Works

- It calls five subroutines: c_init, c_in, c_discrt, c_analog, c_out
- The purpose and frequency of call to these subroutines are shown on the following pages.
- The header file used by these subroutines are defined in the vehdata.h
- Files in the package that help in the construction of control.dll are:

Files in the package	Description
Control.obj	Object code of control.c
Utilc.lib	Matrix-vector math utility routines
Utilc.h	Utilc subroutine call prototypes
Acs_c.lib	Attitude control system subroutines
Acs_c.h	Acs_c subroutine call prototypes

Primary Control.dll Subroutines



Primary C-Routines

subroutine	Call frequency	Function
C_init	Once at start of simulation	Initialize control related parameters
C_in*	At top of every derivative calculation	Read the motion signals from the plant dynamics
C_discrt	At top of every derivative calculation	Execute all discrete events at discrete times
C_analog	At every derivative calculation	Execute all analog computations for signal processing and control
C_out*	At end of every derivative calculation	Send actuation signals back to the plant dynamics

Building Control.dll

- A Short description of how to build control.dll:

1. Go through a 'Codegen' procedure given next using Buildx.exe to generate a user_code.c.tmp and the related header file vehdata.h.tmp. The former is a template of the five subroutines called by control.dll
2. User would add details to user_code.c.tmp and vehdata.h.tmp as required by the application.
3. At the end of that process change user_code.c.tmp to user_code.c
4. Likewise, change vehdata.h.tmp to vehdata.h
5. Compile control.dll with the following CL command at DOS prompt
   - 'CL control.obj user_code.c util.lib acsc.lib /LD'

Codegen Procedure

- Purpose: generate user_code.c.tmp, vehdata.h.tmp
- Given: vehicle model file from Part I

Preprocessing:

1. Go to Buildx >xsv > Model Menus
2. Use ‘discrt’ command to define discrete/analog processes : name and sample period (as necessary); period=0 => analog process
3. Use ‘ode’ command to define application specific ‘ordinary differential equations’: name, and state size (as necessary)
4. Use ‘switch’ command to define switches for events: switch name and simple switch function (as necessary)
5. Type ‘cgen’ to open the Codegen Menu

SV1simc Discrete Processes

- For SV1simc, we defined 2 processes: acs_rwa(discrt), acs_jet(discrt)
- This definition is done by using ‘add’, ‘name’ and ’period’ commands

```
                    ~ Discrete Process Menu ~

  Nof discrete process:  2

  Idx  Name           Period     Start_t

=>  1 acs_rwa      .1000E+00   .0000E+00
    2 acs_jet      .1000E+01   .0000E+00

  [ sel  add  rem  copy  idx  name order period  start x ]
   >
```

- Need to save the new model data to sv1simc.txt
- Next, Go to Codegen Menu by typing ‘cgen’ from Model Menus
- Note: acs_rwa and acs_jet are two special discrete process names, because they cause Codegen to insert functional details in the generated code

ACS_C.lib Subroutines

- These are subroutines written for the examples, by that they need be replaced when configuration or mass property change from those given.

Subroutine	purpose	Examples
Get_acscmd	Compute ACS command signal to null attitude error	Sv1simc, arraysimc, cmg4simc
Get_6jettimes	Compute 6 jet on/off times to null attitude error command	Sv1simc, arraysimc
Get_hmgr6jettimes	Compute 6 jet on/off times to null system angular momentum	cmg4simc
Get_cmgtq	Compute CMG input torque to null attitude error	cmg4simc

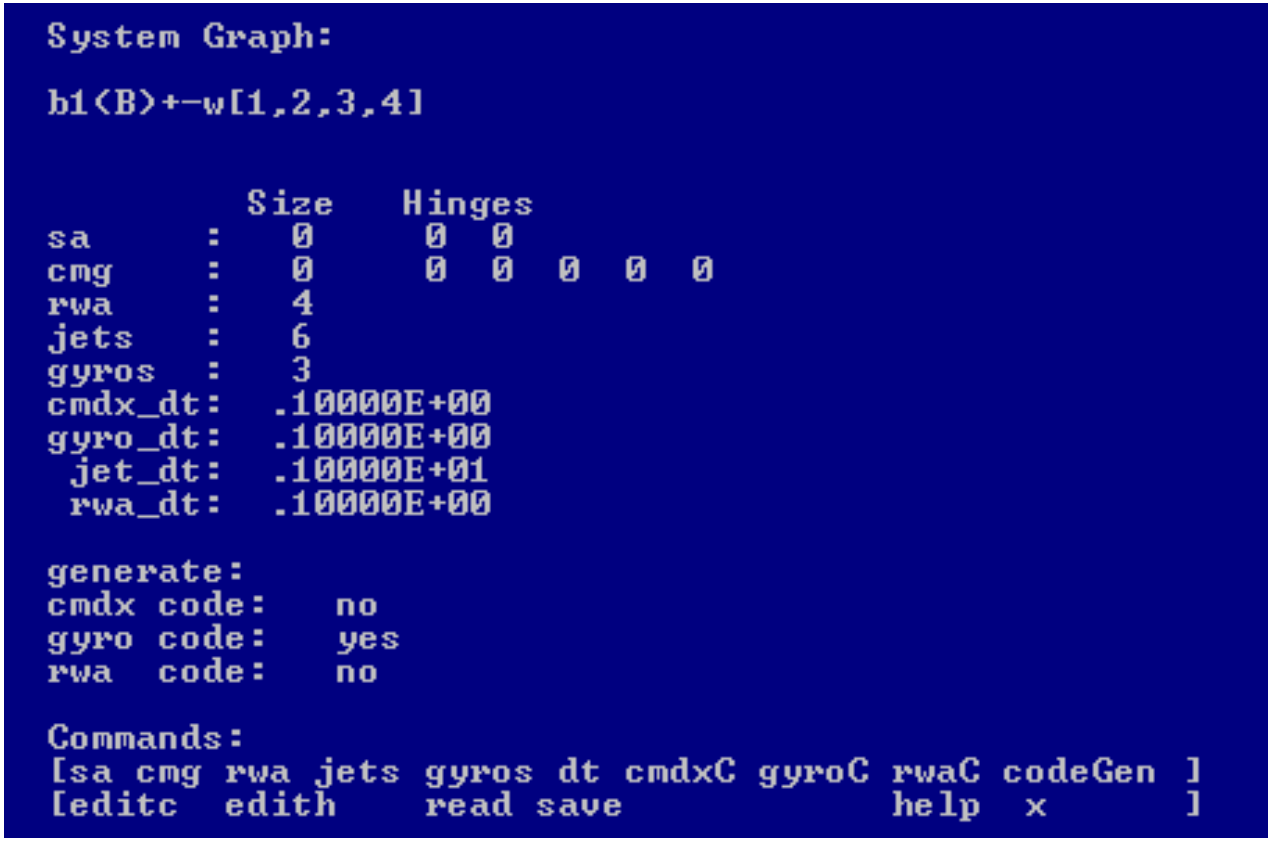
Special Names

- These special subroutine names when given to discrete processes cause functional codes to be generated for those .c.tmp routines

Process name	Purpose	Acs_c.lib routines invoked
acs_rwa	Generate RWA torque for ACS	Get_acscmd
acs_cmg	Generate CMG torque for ACS	Get_acscmd Get_cmgtq
acs_jet	Compute jet on/off times for ACS	Get_acscmd Get_6jettimes
hmgr_jet	Compute jet on/off times to null system angular momentum, syshb1	Get_hmtr6jettimes

Codegen Menu

- Data on this page needs be edited to assist in code generation



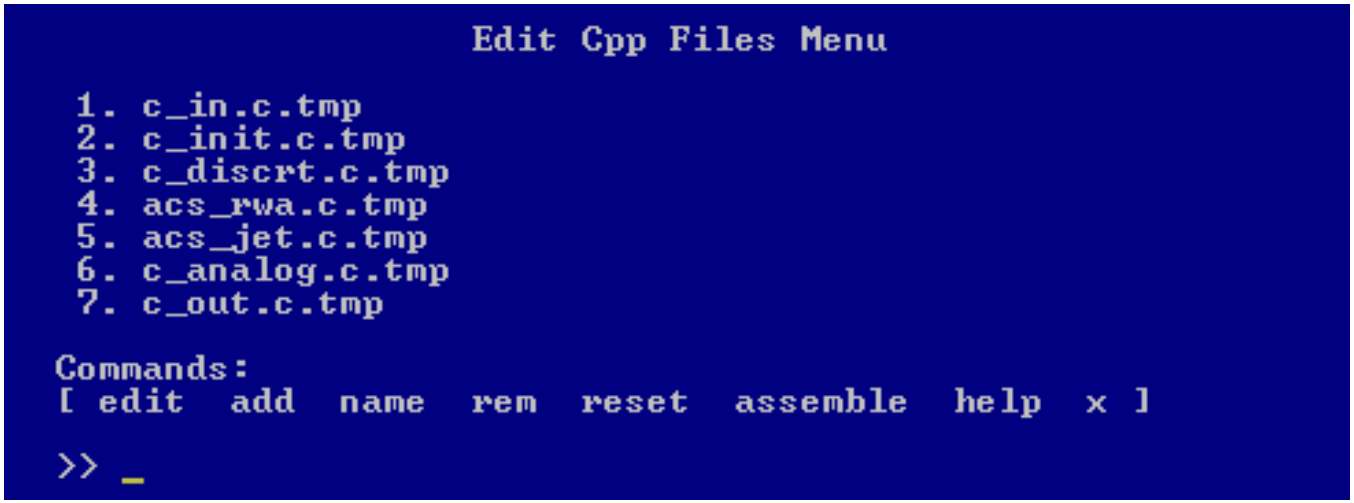
- Type ‘sa’ to define number of arrays (max=2) and the driver hinges (bodies)
- Type ‘cmg’ to define number of CMG’s and the rotor platform bodies
- Type 'rwa’ to define number of reaction wheels
- Type ‘jets’ to define number of jets
- Type ‘gyros’ to define number of gyros
- Type ‘dt’ to define the sample period of 4 discrete processes
- Type ‘codegen’ to generate components of user_code.c.tmp and vehdata.h.tmp
- Type ‘gyroc’ to toggle ‘generate gyro code’ flag
- Type ‘rwac’ to toggle ‘generate rwa code’ flag
- Type ‘cmdxc’ to toggle ‘generate cmdx code’ flag
- Type ‘editc’ to view and edit all *.c.tmp codes
- Type ‘edith’ to view and edit all *.h.tmp codes
- Type ‘save’ to save codegen data to ‘codegendata.txt’
- Type ‘read’ to read codegen data from ‘codegendata.txt’
- Type ‘help’ to get definition of menu data and commands
- Type 'x’ to exit menu

- Let's concentrate on the two discrete processes that was defined for sv1sim1c.txt
- Next, type 'codegen' to generate the .c.tmp and .h.tmp files needed to build user_code.c.tmp and vehdata.h.tmp

```
> codegen
created ctrl_input.h.tmp
created ctrl_output.h.tmp
created global_vars.h.tmp
created c_in.c.tmp
created c_init.c.tmp
created c_discrt.h.tmp
created c_discrt.c.tmp
created acs_rwa.h.tmp
created acs_jet.h.tmp
pgain & vgain for rwa control? 10,200
start and end times of asc_rwa? 2000 0
created acs_rwa.c.tmp
start and end times of asc_jet? 0 2000
created acs_jet.c.tmp
created c_gyro.c.tmp
created c_analog.c.tmp
created c_out.c.tmp
```

- You will be prompted for some gain and timing information in this dialog. Supply as appropriate.
- End time less than Start time means execute the code for 't > Start time'
- Start time= End time= 0 means ignore time window code request

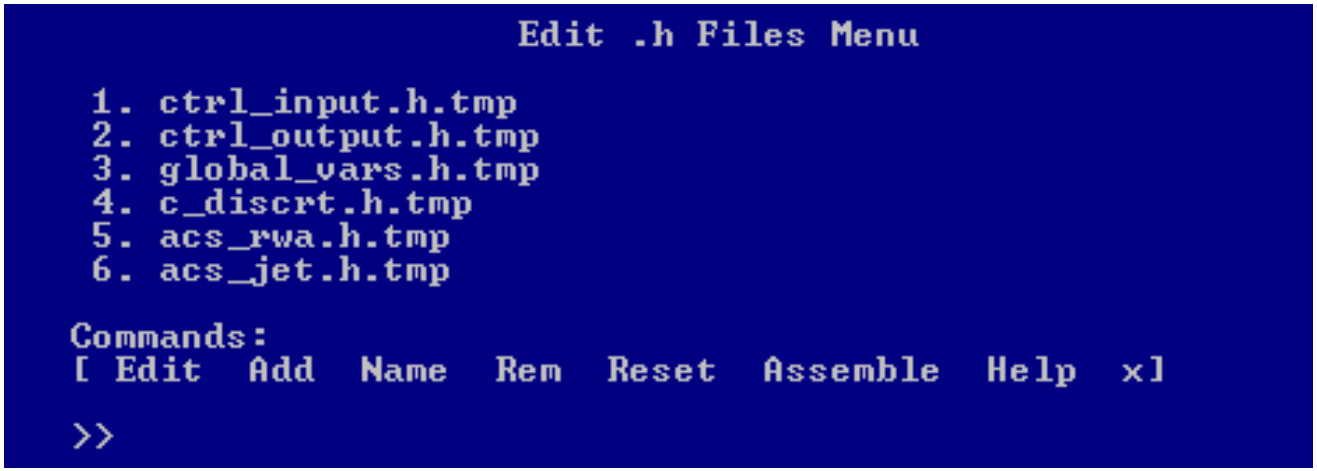
- Next, type ‘editc’ to view user_code.c.tmp components and to assemble user_code.c.tmp



1. One can view and edit each one of the *.c.tmp code using the ‘editj’ command where j is the index preceeding the displayed code.
2. Type ‘assemble’ to assemble all displayed *.c.tmp subroutines into user_code.c.tmp. The latter becomes item 8 in the above list.
3. Type ‘edit8’ in this case to view user_code.c.tmp

- Just edit user_code.c.tmp from here (i.e. ‘edit8’) until it is acceptably complete
- Type ‘Accept’ to copy user_code.c.tmp to user_code.c, and exit (see Appendix A)

- Type ‘edith’ to view the generated vehdata.h.tmp components and to assemble vehdata.h.tmp



1. One can view and edit each one of the *.h.tmp code using the ‘editj’ command where j is the index preceeding the displayed code.
2. Type ‘assemble’ to assemble all displayed *.h.tmp subroutines into vehdata.h.tmp. The latter becomes item 7 in the above list.
3. Type ‘edit7’ in this case to view vehdata.h.tmp

- Just edit vehdata.h.tmp from here on (i.e. ‘edit7’) until it is acceptably complete
- Type ‘Accept’ to copy vehdata.h.tmp to vehdata.h, and exit (see Appendix B)

Compile Control.dll

- After obtaining user_code.c and vehdata.h files, exit buildx.exe
- Type ‘mkcontrolc.bat’ to compile control.dll
- Fix any compilation error that appear in user_code.c or in vehdata.h
- The xsv01.exe simulation is now ready to run

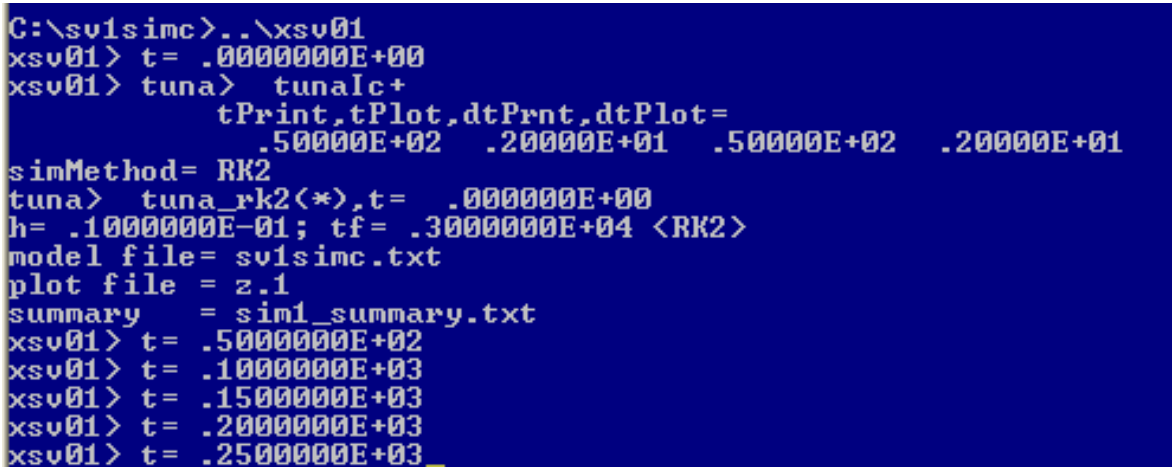
Example1

- This example shows the LVLH control with jet for t < 2000 and with rwa for t > =2000.
- The attitude command signal is generated from true b1 angular rate (w1) and xsv01.exe provided LVLH attitude error (b2osml1)

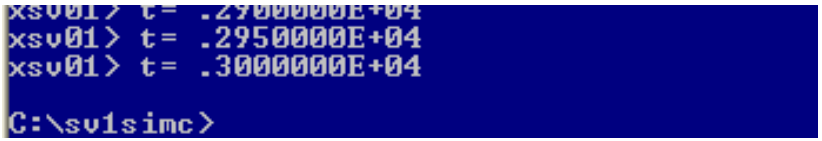
- Buildx procedure:
  - start Buildx and go to body menu
  - set 'wrel1' to [.1, .2, .3]degrees
  - from gravity menu, set [ecc,incl,period]=[0.,35 deg, 100 min]
  - save model data to SV1simc.txt from Model Menus

Run xsv01.exe

- Click c:\SV1simc\xsv01.bat and see a running display as follows.



...

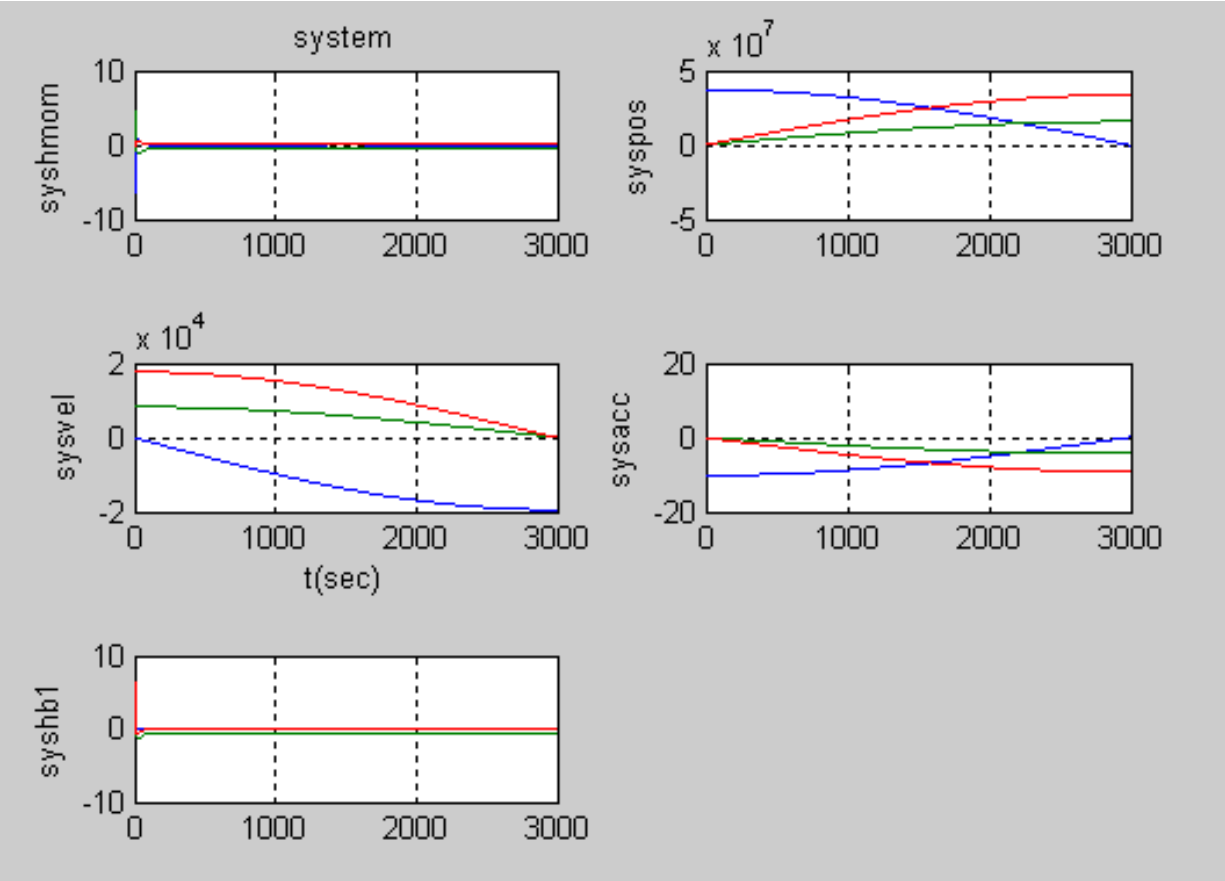


View Sim Results

- Type 'load z.1’ from Matlab window to read in sim result

- Type 'simplot1(z)’ to view result
  - simplot1.m is a script that was constructed using Buildx.exe (see page 46)

Fig.1 System Motion

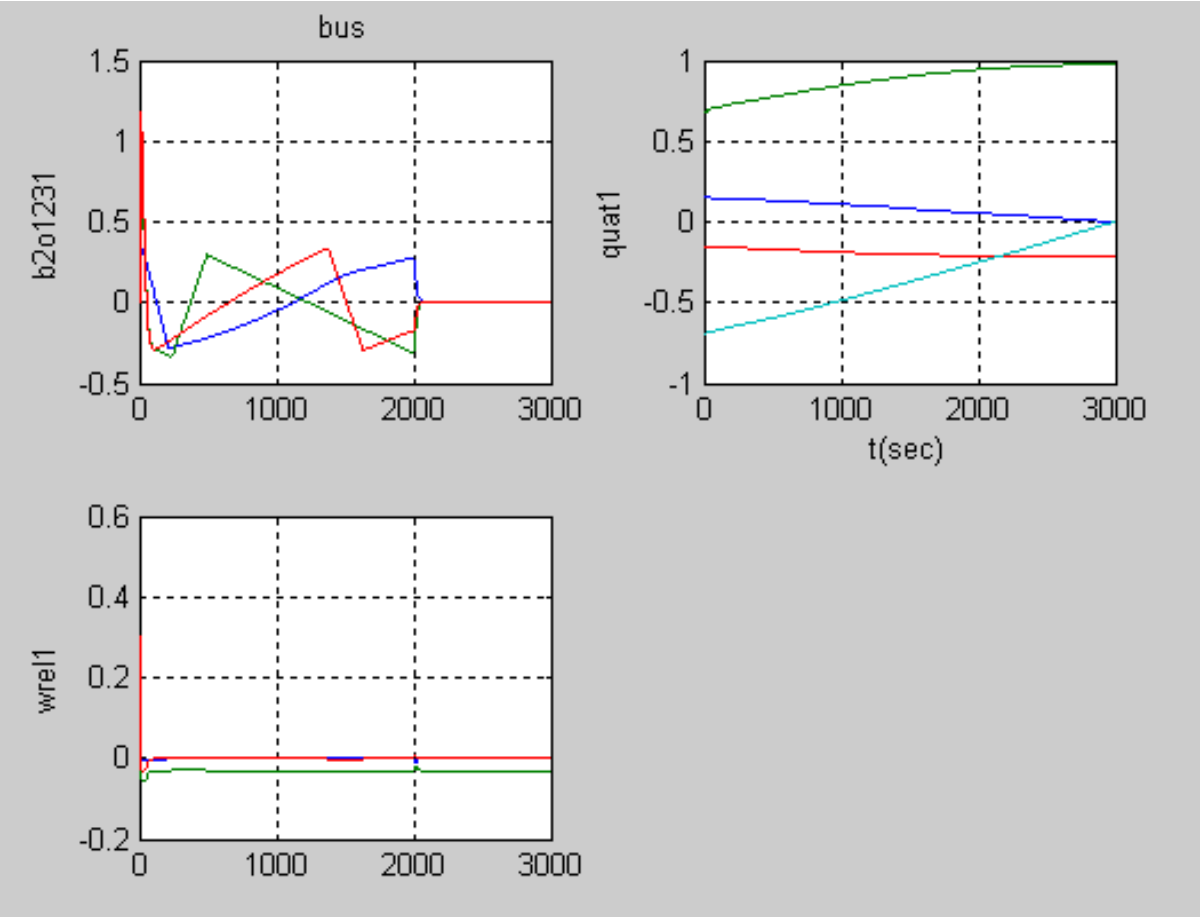


- orbit:
  35 deg incl
  ecc=.001
  6000 sec orbit period
  gravity: gflag=10

- s.s. syshmom:
  constant system angular momentum in eci coord
  hmag ~ 0.6 ft-lb-s

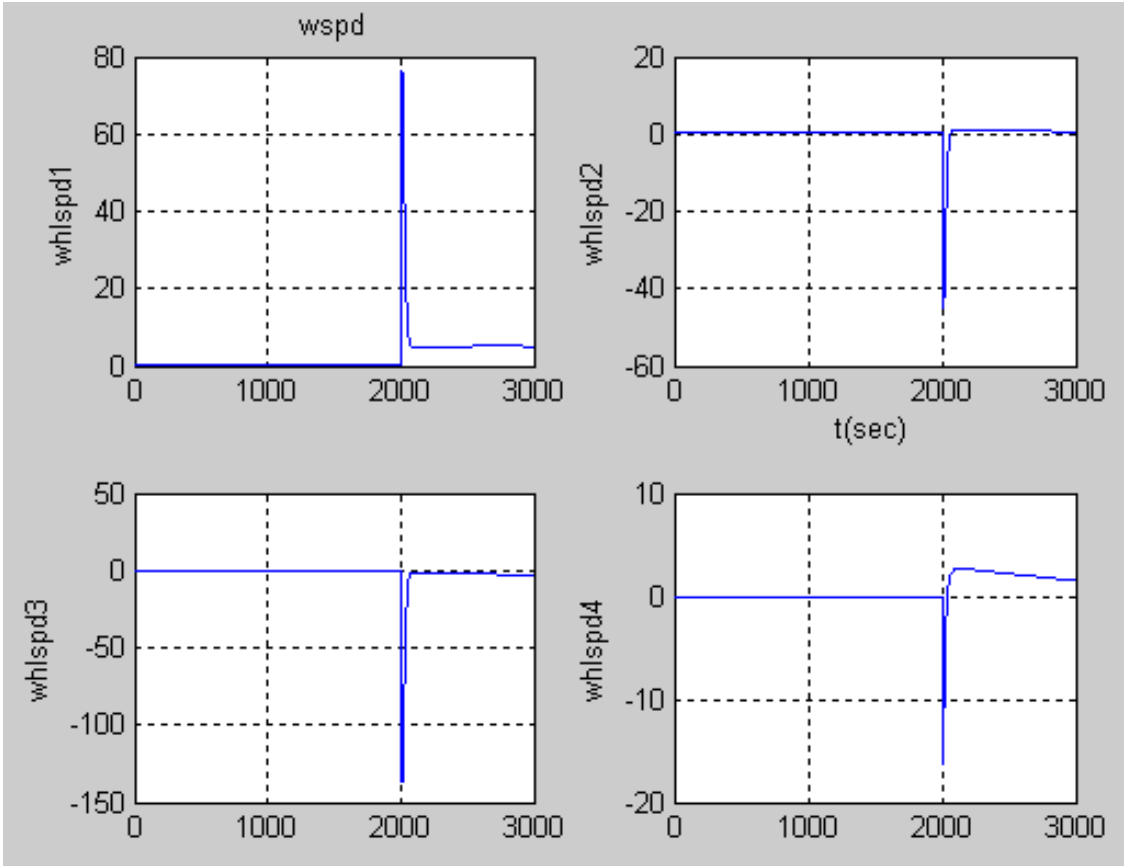
- syshb1:
  syshmom in b1 coord

Fig.2 B1 Motion



- acs:
  b1 attitude is LVLH
  b201231=rpy ~ 0
  for t > 2000
- ss b1 angular rate ~
  [0 –orbrate 0] d/s
- sat3w orbrate= .03 d/s

Fig.3 Wheel Speed



- wheel speeds are low due to jet LVLH ACS for t < 2000 that kept syshb1 to near zero (Fig. 1)

Comment

- This example showed that the vehicle inertial angular momentum is reduced from 7.75 ft-lb-s to near zero (0.6) shortly after enabling jets at start. The momentum remained so for the rest of the simulation
- The LVLH attitude error is under 0.3 deg for t=[0:2000] with jet ACS and stayed near zero thereafter because of RWA ACS
- The steady state body rate is the orbit rate as controlled by the jets and the RWA
- The wheel speed is low because of the size of the system angular momentum.

Example2

- Model file for this example is identical to the one used in Example1.
- The change is to replace the true rate ‘w1’ with gyro derived ‘gyr_rate’ in computing the rate error in acs_rwa.c and acs_jet.c subroutines in user_code.c.

- Procedure:

1. Make the above changes in user_code.c
2. Type ‘mkcontrolc.bat’ to compile control.dll
3. Type ‘xsv01.exe’ to run the simulation
4. View sim results in Matlab using simplot1.m

- Comment:
  - The three charts, System Motion, B1 Motion, and Wheel Speed response, from this example are nearly identical to those obtained from Example1

Adjustable Sim Parameters

- Dt: simulation integration step size
- plotDt: plot data sample period
- printDt: time display sample period
- T: simulation period
- Method: Integration method, RK2 or RK4

- Edit procedure:

1. Go to Main menu
2. Use ‘stepsize’, ‘dtplot’, ‘printdt’, ‘endtime’, ‘method’ commands to change the
3. sim parameters
4. Type ‘savel’ to save changes to siminputfile
5. Reply with ‘sim1files.txt’
6. exit buildx

Exercises

actions	parameters	reference
change mass property	mass, inr, svec, dvec	pages:15-20
change initial condition	ang,wrel,wrelax, dcm0	pages:15-17
change orbit	ephemeris, orbit period	pages:33-36
add /remove bodies*		pages:15-17
add /remove wheels*		pages:25-27
add/remove forces*		pages:29-32
modify input (udata)		pages:37-39
modify output (ydata)		pages:40-42
modify plot (odata)		pages:43-45
modify simplot1.m		page:46

* Not for project licenses

actions	changes	control system config
design your own rwa controller	•may need new ydata •may need less than 4 wheels	•likely identical to SV1simc •adjust i/o mux/dmux
design your own jet controller	•may need new ydata •may need nof jets other than 8	•likely identical to SV1simc •adjust i/o mux/dmux

Simulation Notes

Subject	SV1simc	comments
gforces	comment applies to SV1simc	gravity forces are auto-computed by sim engine for all bj in system
jet forces	force locations and vectors were selected for SV1simc to cause each jet torque to align with a particular b1.xyz axes to get a simple jet controller	generally jets are not placed ideally as in SV1simc because of other design considerations. as such, associated jet controller can be complex
wheels	SV1simc chose a 4 corner pyramid configuration	pyramid base to height ratio can vary and the center axis of pyramid need not be along any particular b1.xyz axis
geometry	SV1simc is a gyrostat and b1.cm is same as system.cm regardless of how b1.svec is defined	position and orientation of body parts are defined by their dvec, svec ,dcm0 and joint coordinates

Subject	SV1simc	comments
b1.dcm0	comment applies here also	Dcm0 of b1 is the initial LVLH attitude of the vehicle
mass, inertia	comment does not apply to SV1simc since it has only one regular body	mass and inertia can be set to zero for non-terminal bodies to represent ideal massless joints
sunb1,nadirb1	sun vector and nadir vector in b1 frame were not used in SV1simc control system	these two ydata are available for for sun sensor and earth sensor modeling
b2osml	this small angle LVLH attitude error of b1 computed by the sim engine is the rpy signal in the examples meaning (roll,pitch, yaw) angles.	this ydata signal is available as a functional convenience to ACS analysis. A more realistic rpy signal needs be built from sunb1 and nadirb1
syshb1	this is the system angular momentum in b1 frame, not used in SV1simc	this ydata signal is available as a functional convenience to momentum mgmt design

Summary

- Two examples given here show that xsv01.exe with a properly constructed control.dll can simulate the dynamics of a vehicle with jet and RWA ACS in maintaining LVLH attitude.

- The value of SV1simc is that its mass property can be changed to fit the vehicle of interest . It requires that a C/C++ compiler be installed on the PC to compile control.dll. With the Buildx editor, one can define a variety of Dynamics Input/Output signals to design and test his application specific control system. More importantly, it can be very effective in the design of any single body vehicle, its operations and control system.

Appendix A
User_code.c

```
// This user_code.c is generated by Buildx.exe
//
// These subroutines manage the data in:
// vehdata.h  variables & constants used by
//                 control.dll
// utilc.h        have the prototypes of matrix-vector
//                 subroutines used in the generated code
#include <stdio.h>
#include <math.h>
#include "vehdata.h"
#include "utilc.h"

// Control Input Mapping: x->local data
 void c_in(double *x,double *T){

   equal( w1,     &x[0],3  ) ;
   equal( b2osml1,&x[3],3  ) ;
   equal( gyr_ang,&x[6],3  ) ;
   t= T[0] ;
}
```

```
// initialization procedure.....
void c_init(){
  worb[0]= 0.;
  worb[1]= -.523599E-03; //(rps),temporary
  worb[2]= 0.;
}

// c_discrt procedure.....
void c_discrt(double *T){
  int jet; // jet index

  discrete(acs_rwa, T, &acs_rwa_t) ;
  discrete(acs_jet, T, &acs_jet_t) ;
  discrete( c_gyro, T, &gyr_t   ) ;
  for (jet=0; jet <  6; jet++ ){
      xf[jet]=0;
   if( T[0] > jet_on[jet] & T[0] <= jet_off[jet] )
      xf[jet]=1;
  }
}
```

```
void acs_rwa( double *ts ){
 //This procedure is called at time= *ts
  double G[12]={ .433013E+00, .433013E+00,-.433013E+00,-.433013E+00,
                -.433013E+00, .433013E+00, .433013E+00,-.433013E+00,
                 .433013E+00, .433013E+00, .433013E+00, .433013E+00};
  double cmd[3];
  double gain[2]={ .100000E+02, .200000E+03}; //temporary

  double sc=-1 ;

  if(*ts >   .200000E+04 ){
  subv( worb, gyr_rate, tempv );
  get_acscmd( gain, b2osml1, tempv, cmd); //see acsc.h
  mtxmv( G, cmd,  wtq, 4, 3);//wtq=G*cmd

  mtxsv(  &sc, wtq,  wtq, 4   );//wtq=-wtq
  }

 *ts= *ts + acs_rwa_dt ;
}
```

```
void acs_jet( double *ts ){
 //This procedure is called at time= *ts
  double cmd[3];
  double gain[2]={10,200}; //temporary
  double hjet=  .300000E+01; //jet impulse/sec, temporary

  if(*ts <=  .200000E+04){
  subv( worb, gyr_rate, tempv );
  get_acscmd( gain, b2osml1, tempv, cmd); //see acsc.h
  get_6jettimes(ts, cmd, hjet, jet_on, jet_off ); //*TEMP CODE*
  }

 *ts= *ts + acs_jet_dt ;
}

// c_gyro procedure.....
void c_gyro(double *ts){
   double tempv[3];

   subx( gyr_ang, gyr_ang_prev, tempv, 3) ;
   mtxsv( &gyr_freq,  tempv, gyr_rate, 3) ;
   equal( gyr_ang_prev, gyr_ang,      3) ;
   *ts= *ts + gyr_dt ;
}
```

```
//  Control analog procedure...
 void c_analog(double *T){

}


//  Control Output Mapping: u->dx/dt
 void c_out(double *T,double *u){

   equal( &u[0],wtq,4   ) ;
   equal( &u[4],xf,6   ) ;
}
```

Appendix B
Vehdata.h

```
/* control data: Udata */
double whltq1; /*  wheel( 1) torque*/
double whltq2; /*  wheel( 2) torque*/
double whltq3; /*  wheel( 3) torque*/
double whltq4; /*  wheel( 4) torque*/
double xf1; /*  external force on body( 1)*/
double xf2; /*  external force on body( 2)*/
double xf3; /*  external force on body( 3)*/
double xf4; /*  external force on body( 4)*/
double xf5; /*  external force on body( 5)*/
double xf6; /*  external force on body( 6)*/
double jet_on[100]      ; /* jet start time       */
double jet_off[100]     ; /* jet end time         */
double xf[20]           ; /* jet [1/0] array      */
double worb[3]          ; /* orb rate in b1 frame */
double wtq[5]           ; /* wheel torque(max 5)  */
double cmgtq[5]         ; /* cmg.input.tq(max 5)  */
double cmg_ang[5]       ; /* cmg.input.ang(max 5) */
double cmg_rate[5]      ; /* cmg.input.rate(max 5)*/
int    current_jet      ; /* jet being processed  */
```

```
/* control data: Ydata */
double w1[3]; /*  Body( 1) ANG RATE IN Body(1) FRAME*/
double b2osml1[3]; /*  Ref body to orbit attitude SMALL(XYZ) angles*/
double gyr_ang_x[3]; /*  GYRO cumulative angles*/

/* Global variables & constants */
double tempv[3],tempw[3];
double pi=    3.14159265358979 ;
double twopi= 6.28318530717959 ;
double d2r=   0.01745329251994 ;
double r2d=  57.29577951308232 ;
double t ;

void turn_jet_off( double * ) ;

 // c_gyro: output delta angle per gyro per acs_dt
void c_gyro(double *);
double gyr_dt  =  .10000E+00   ;
double gyr_freq=  .10000E+02   ;
double gyr_ang[3]      ;
double gyr_ang_prev[3] ;
double gyr_rate[3]     ;
double gyr_t   =  0.       ;
```

- void acs_rwa(double *) ;
- double acs_rwa_data[10];
- double acs_rwa_t= .000000E+00;
- double acs_rwa_dt= .100000E+00;
-
- void acs_jet(double *) ;
- double acs_jet_data[10];
- double acs_jet_t= .000000E+00;
- double acs_jet_dt= .100000E+01;
-

Appendix C
Utilc.h

- void addmtv(double *, double *, double *, double *);
- void addmv(double *, double *, double *, double *);
- void addsv (double *, double *, double *, double *);
- void addsx (double *, double *, double *, double *, int);
- void addv(double *, double *, double *);
- void addx(double *, double *, double *, int);
- void cross(double *, double *, double *);
- double cswitch (int, double *, double *, double *);
- void dc2q(double *, double *);
- void dcsmall(double *, double *);
- void discrete(void (*)(double *), double *, double *);
- double dot(double *, double *);
- double dotx(double *, double *, int n);
- void equal(double *, double *, int);

- void mtxsv(double *, double *, double *, int);
- void mtxmtv(double *, double *, double *, int, int);
- void mtxmv(double *, double *, double *, int, int);
- void multmm(double *, double *, double *);
- void multmtv(double *, double *, double *);
- void multmv(double *, double *, double *);
- void multsv(double *, double *, double *);
- void null(double *, int);
- void qdot(double *, double *, double *);
- void qinv(double *, double *);
- void qiqmult(double *, double *, double *);
- void qmult(double *, double *, double *);
- void q2dc(double *, double *);
- void submtv(double *, double *, double *, double *);
- void submv(double *, double *, double *, double *);
- void subv(double *, double *, double *);
- double unitvec(double *, double *);
- void xyzrot (int *, double *, double *, double *);