Arraysimc Manual

Concurrent Dynamics International

October 2014

Objectives

- Part I:
  - Build a model file (Arraysimc.txt) to simulate a satellite with 3 reaction wheels, 2 arrays(ea. 5 parts) and 6 jets
  - Panels of each array are deployed from a stowed condition in a coordinated manner
  - Show use of ‘jnt’ command to implement simple joint position or rate command

- Part II:
  - Build control.dll to interact with the main program xsv01.exe during run time to simulate the dynamics and control of the Arraysimc vehicle with a coordinated array deployment at start
  - Xsv01.exe Examples:
    - use reaction wheels to maintain LVLH attitude with jets idled
    - use reaction jets and wheels to maintain LVLH attitude

Part I Topics

- Physical Model
- Buildx Tasks
- Key Files
- Main Menu
- Model Menus
- Body Menu
- All b_1 Data
- Body Actuation Signals
- Wheel Menu
- Force Menu
- Dynamics Input
- Dynamics Output
- Plot Data
- Gravity/Orbit Menu
- Lock & Gear Constraints
- JNT Menu
- Save Model Data
- Simplot
- Exit Buildx
- Q & A

License Restrictions

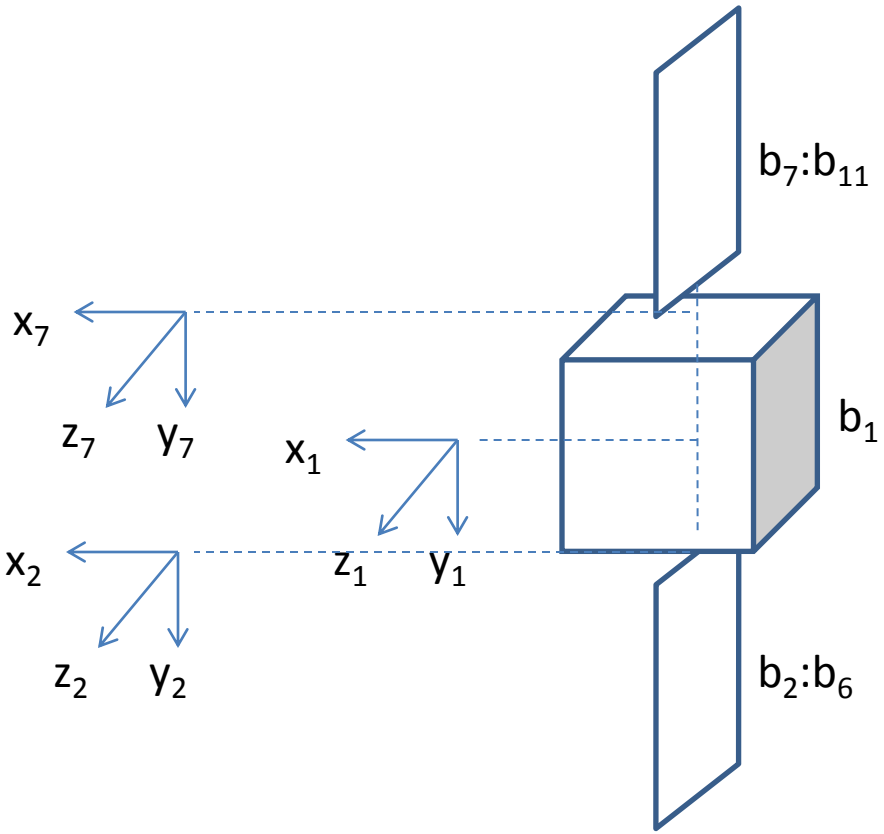
License type	Buildx.exe	Xsv01.dll
Enterprise	none	none
Project	Must stay with the object count specified by license gflag <=12	Runs with model_files with license specified object count gflag <=12

- 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 36)

How to Use This Manual

- This Arraysimc manual is for Enterprise license users where no restrictions are placed on the object counts {body, wheels, forces} in creating models.
- Arraysimc.txt is a seed model_file for Enterprise license users to create other models such as arrayed satellites with one array, or arrayed satellite with additional appendages.
- CDI has many seed models to expedite the development of more complex spacecrafts, i.e. those that use CMG's for ACS, tethered satellite
- This manual is applicable to Project license users whose model object count is {11, 4, 8} as Arraysimc.

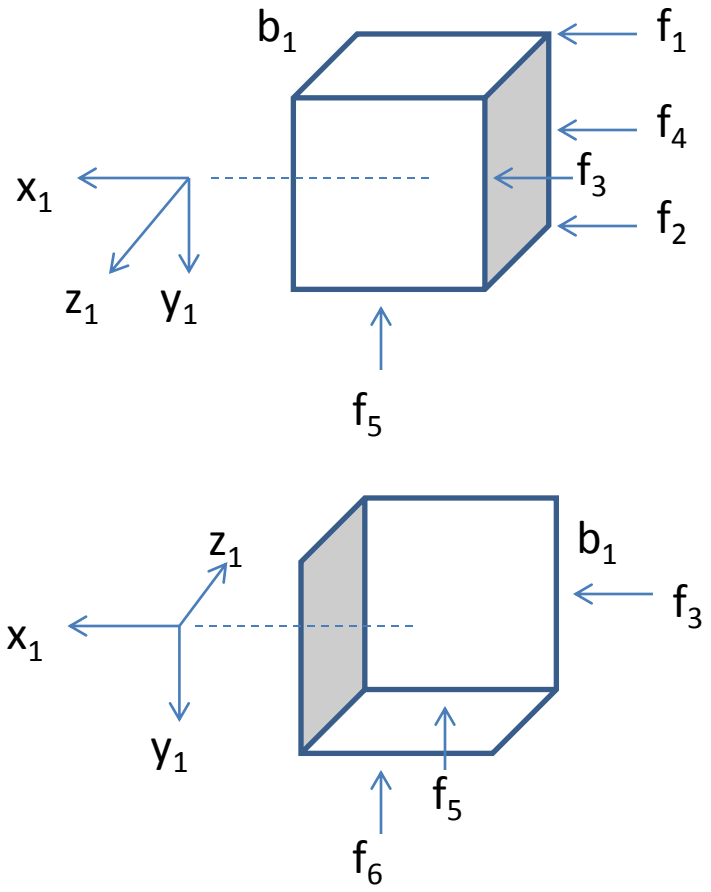
ArraySimc Model



b_1=bus,
b_2: b_6 = array1
b_7:b_{11}= array2

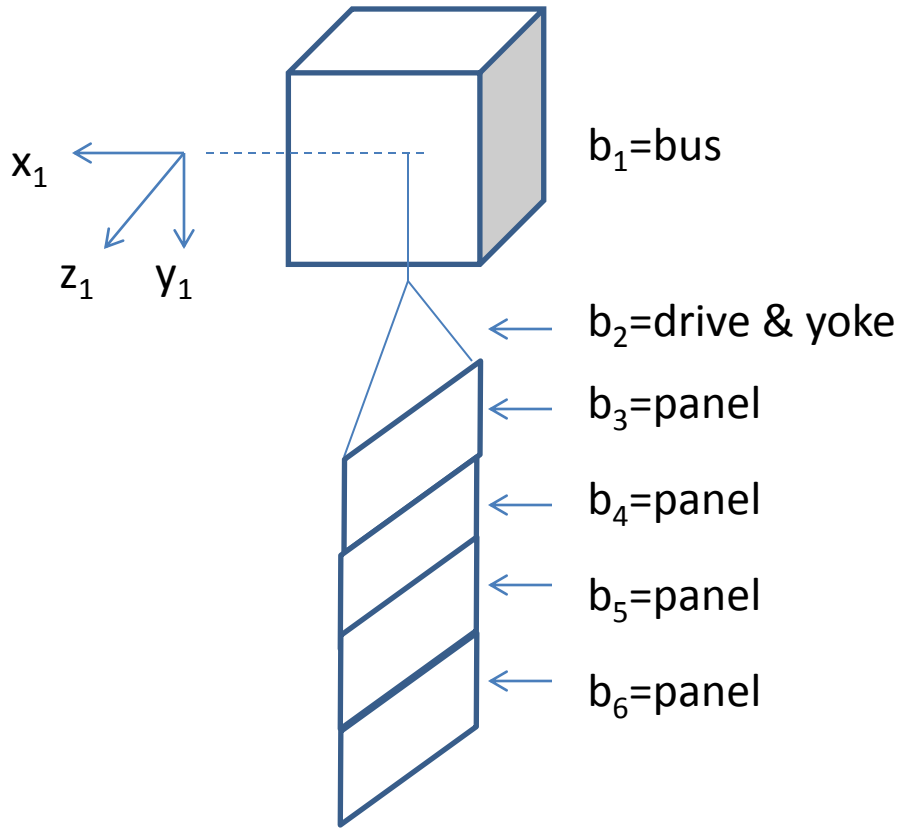
reaction wheels axes on b_1 :
w_1=[1 -1 1] /sqrt(3)
w_2=[1 1 1] /sqrt(3)
w_3=[-1 1 1] /sqrt(3)
w_4=[-1 -1 1] /sqrt(3)

Jet Locations on b_1



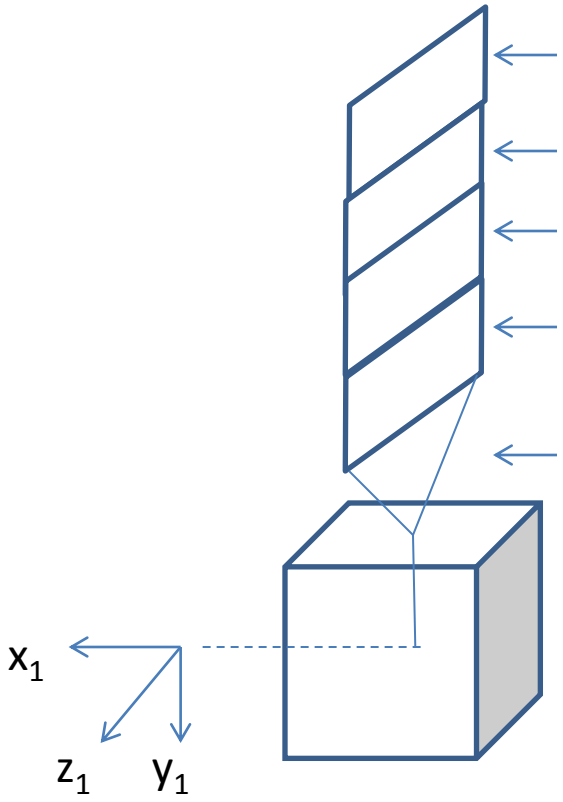
fpos:			fvec:		
f_1 :[-3	-3	0]	f_1 :[.2	0	0]
f_2 :[-3	3	0]	f_2 :[.2	0	0]
f_3 :[-3	0	3]	f_3 :[.2	0	0]
f_4 :[-3	0	-3]	f_4 :[.2	0	0]
f_5 :[0	3	3]	f_5 :[0	-.2	0]
f_6 :[0	3	-3]	f_6 :[0	-.2	0]

Array 1



dvec:			svec:		
b_2:[0	3	0]	b_2 :[0	1.5	0]
b_3:[0	3	0]	b_3 :[0	1.5	0]
b_4:[0	3	0]	b_4:[0	1.5	0]
b_5:[0	3	0]	b_5 :[0	1.5	0]
b_6:[0	3	0]	b_6 :[0	1.5	0]

Array 2



dvec:	svec:
b_7: [0 -3 0]	b_7 : [0 -1.5 0]
b_8: [0 -3 0]	b_8 : [0 -1.5 0]
b_9: [0 -3 0]	b_9: [0 -1.5 0]
b_{10}:[0 -3 0]	b_{10}:[0 -1.5 0]
b_{11}:[0 -3 0]	b_{11}:[0 -1.5 0]

On the Simulation

- We are building a model file in Part I to support a simulation of an arrayed vehicle in an orbital environment. The array panels are to deploy in a coordinated manner. The vehicle attitude shall be controlled to follow LVLH frame. The model file defines the mass property of bodies and wheels, specifies forces parameters and constraints, defines gravity model and the dynamics input/output signals required by the control system.

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

Buildx Tasks

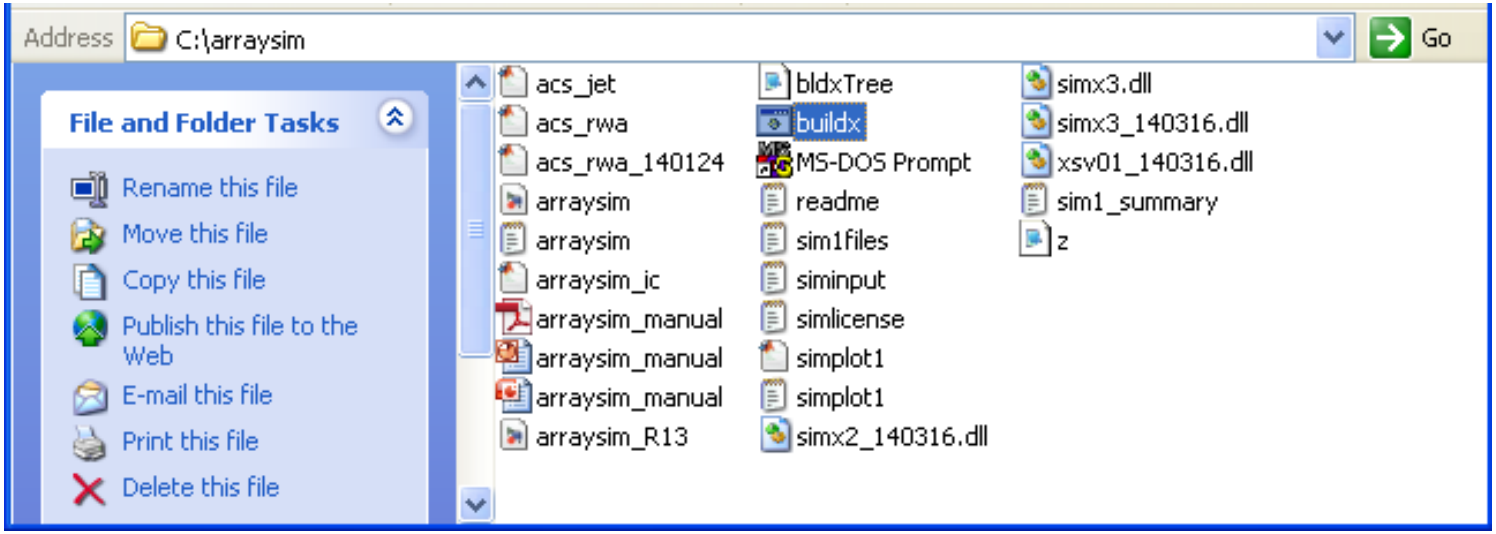
- Create Arraysimc.txt to represent Arraysimc
- Set up Arraysimc.txt to control array drive joints in position or in rate
- Define gear constraints to have a coordinated array deployment
- Define lock constraints for end of deployment
- Define simplot1.m

Key Files

- Buildx.bat='..\Buildx\'
- Input file router: siminput.txt
- Working files: sim1files.txt
- Model_file: Arraysimc.txt
- Simplot file: simplot1.txt
- License: simlicense.txt

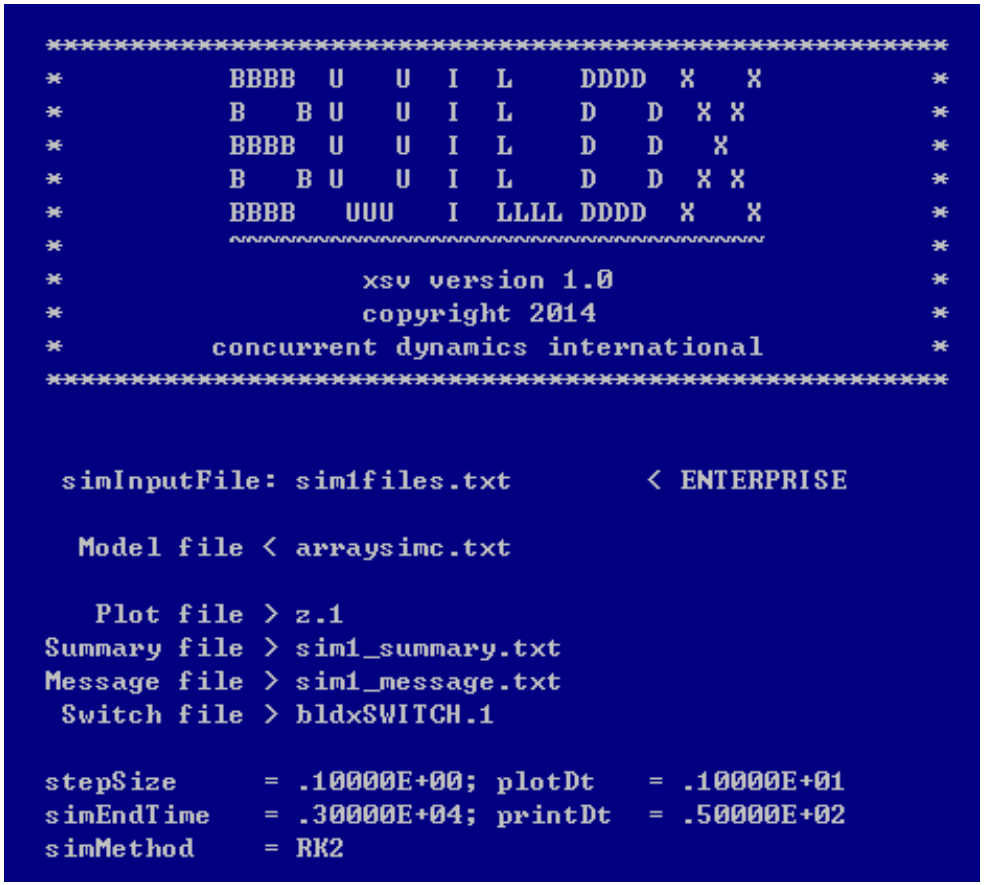
Start Buildx.exe

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



Start Buildx.exe

- See all files defined in simInputFile
- Type ‘xsv’ to go to Model Menus



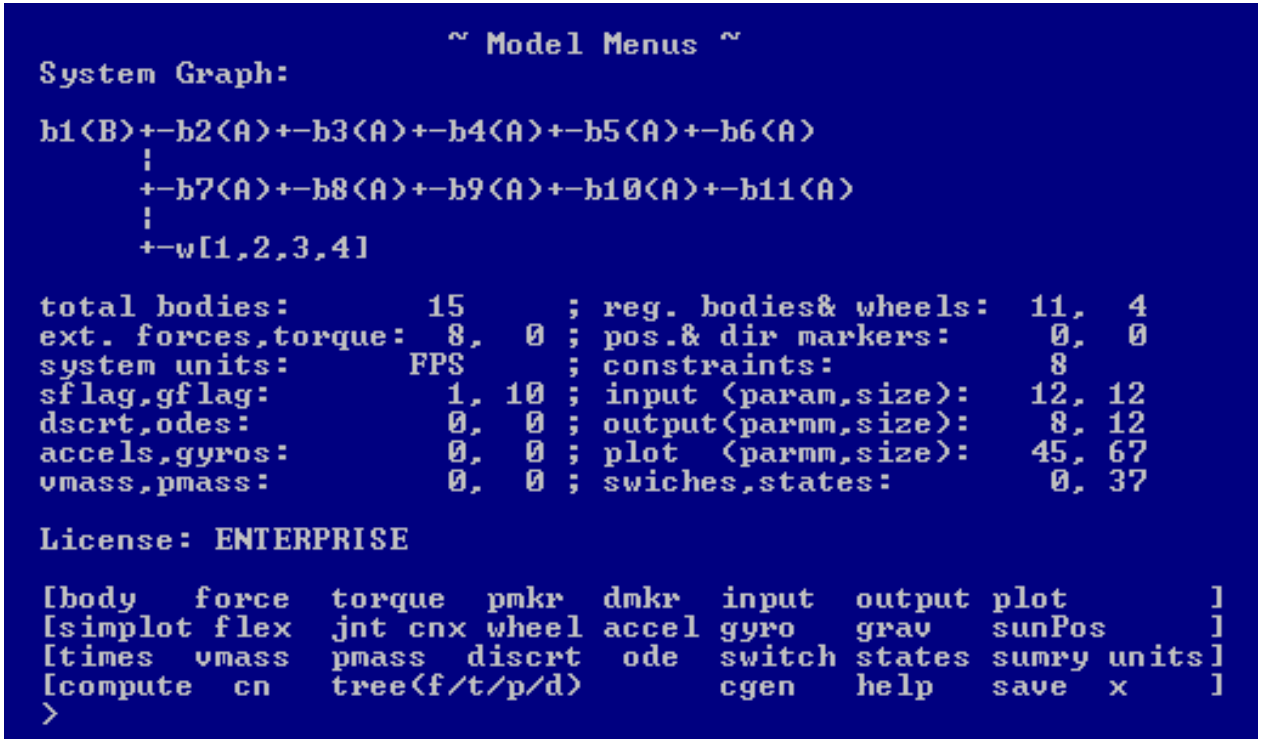
Arraysimc Model Items

Defined Items	Menu	Parameters/data
11 body	Body	Mass, inr, svec,dvec, dcm0, axis,ang,wrel,jnt, parent,type
4 wheels	Wheel	Winr, axis, wspd, parent, type
8 forces	Force	Fvec,fpos, parent,type
8 constraints	CN	2 lock, 6 gear constraints
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 61

Model Menus

- See model parts size and go to menus to edit/browse



- Arraysimc has 11 regular bodies, 4 wheels and 8 jets

Body Menu

- See body summary, type ‘body’ from Model Menus
- Arraysimc has 11 bodies

```
  idx name     pa  u   fl vm tp ax -- angle - ----mass-----
=>  1 b1        0 FPS  0  -  B  x        .000  .100000E+03
    2 drive1    1 FPS  0  -  A  y        .000  .100000E+01
    3 yoke1     2 FPS  0  -  A  z     -90.000  .100000E+01
    4 pnl1a     3 FPS  0  -  A  z     180.000  .300000E+01
    5 pnl1b     4 FPS  0  -  A  z    -180.000  .300000E+01
    6 pnl1c     5 FPS  0  -  A  z     180.000  .300000E+01
    7 drive2    1 FPS  0  -  A  y        .000  .100000E+01
    8 yoke2     7 FPS  0  -  A  z      90.000  .100000E+01
    9 pnl2a     8 FPS  0  -  A  z    -180.000  .300000E+01
   10 pnl2b     9 FPS  0  -  A  z     180.000  .300000E+01
   11 pnl2c    10 FPS  0  -  A  z    -180.000  .300000E+01

  [ 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   ]
  >
```

- Joint angles (3:11) above put the arrays in stowed configuration
- Joint angles (3:11)= 0 is the deployed configuration

- Type ‘add<j>’ to add bodies to b_j : i.e. ‘add1’ to add bodies to b_1
- Type ‘rem<j>’ to remove b_j from list
- Type ‘name<j>’ to edit b_j .name
- Type ‘par<j>’ to edit b_j .parent
- Type ‘type<j>’ to edit motion type, b_j .type: {a…h}
- Type ‘axis<j>’ to edit b_j .axis: {x, y or z}
- Type ‘ang<j>’ to edit initial inboard b_j .ang for 1 dof rotational joints
- Type ‘wrel<j>’ to edit b_j .angular_rate
- Type ‘mass<j>’ to edit bj.mass
- Type ‘svec<j>’ to edit b_j .svec
- Type ‘dvec<j>’ to edit b_j .dvec
- Type ‘inr<j>’ to edit b_j .inr
- Type ‘edit<j>’ to see all data on b_j

- Type ‘help’ to get definition on data and commands
- Type ‘x’ to exit menu

Inertia Menu

- Need define moi of each body b_j about the b_j.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
		.000000E+00	.000000E+00	.000000E+00
2	drive1	.100000E+01	.100000E+01	.100000E+01
		.000000E+00	.000000E+00	.000000E+00
3	yoke1	.100000E+01	.100000E+01	.100000E+01
		.000000E+00	.000000E+00	.000000E+00
4	pnl1a	.100000E+02	.500000E+01	.100000E+02
		.000000E+00	.000000E+00	.000000E+00
5	pnl1b	.100000E+02	.500000E+01	.100000E+02
		.000000E+00	.000000E+00	.000000E+00
6	pnl1c	.100000E+02	.500000E+01	.100000E+02
		.000000E+00	.000000E+00	.000000E+00
7	drive2	.100000E+01	.100000E+01	.100000E+01
		.000000E+00	.000000E+00	.000000E+00
8	yoke2	.100000E+01	.100000E+01	.100000E+01
		.000000E+00	.000000E+00	.000000E+00
9	pnl2a	.100000E+02	.500000E+01	.100000E+02
		.000000E+00	.000000E+00	.000000E+00
10	pnl2b	.100000E+02	.500000E+01	.100000E+02
		.000000E+00	.000000E+00	.000000E+00
11	pnl2c	.100000E+02	.500000E+01	.100000E+02
		.000000E+00	.000000E+00	.000000E+00

- Type ‘inr<j>’ to edit b_j.inr
- Type ‘x’ to exit menu

Dvec Menu

- Need define dvec(j), b_j.hinge.position in b_j.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
	2	drive1	FPS	.000000E+00	.300000E+01	.000000E+00
	3	yoke1	FPS	.000000E+00	.300000E+01	.000000E+00
	4	pnl1a	FPS	.000000E+00	.300000E+01	.000000E+00
	5	pnl1b	FPS	.000000E+00	.300000E+01	.000000E+00
	6	pnl1c	FPS	.000000E+00	.300000E+01	.000000E+00
	7	drive2	FPS	.000000E+00	-.300000E+01	.000000E+00
	8	yoke2	FPS	.000000E+00	-.300000E+01	.000000E+00
	9	pnl2a	FPS	.000000E+00	-.300000E+01	.000000E+00
	10	pnl2b	FPS	.000000E+00	-.300000E+01	.000000E+00
	11	pnl2c	FPS	.000000E+00	-.300000E+01	.000000E+00

- Note: by design b_1.dvec=0
- Type ‘dvec<j>’ to edit b_j.dvec
- Type ‘x’ to exit menu

Svec Menu

- Need define svec(j), b_j.position.cm in b_j.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
	2	drive1	FPS	.000000E+00	.150000E+01	.000000E+00
	3	yoke1	FPS	.000000E+00	.150000E+01	.000000E+00
	4	pnl1a	FPS	.000000E+00	.150000E+01	.000000E+00
	5	pnl1b	FPS	.000000E+00	.150000E+01	.000000E+00
	6	pnl1c	FPS	.000000E+00	.150000E+01	.000000E+00
	7	drive2	FPS	.000000E+00	-.150000E+01	.000000E+00
	8	yoke2	FPS	.000000E+00	-.150000E+01	.000000E+00
	9	pnl2a	FPS	.000000E+00	-.150000E+01	.000000E+00
	10	pnl2b	FPS	.000000E+00	-.150000E+01	.000000E+00
	11	pnl2c	FPS	.000000E+00	-.150000E+01	.000000E+00

- Note: b_1.cm here is collocated with b_1.hinge point, but can be nonzero
- Type ‘svec<j>’ to edit b_j.svec
- Type ‘x’ to exit menu

Pos Summary

- See where b_j.cm is in b_1 frame for all j
- Type ‘rpos’ in body menu to see a summary of b_j.cm

	idx	name	----------	---rpos---	----------
=>	1	b1	.000	.000	.000
	2	drive1	.000	4.500	.000
	3	yoke1	1.500	6.000	.000
	4	pnl1a	1.500	6.000	.000
	5	pnl1b	1.500	6.000	.000
	6	pnl1c	1.500	6.000	.000
	7	drive2	.000	-4.500	.000
	8	yoke2	1.500	-6.000	.000
	9	pnl2a	1.500	-6.000	.000
	10	pnl2b	1.500	-6.000	.000
	11	pnl2c	1.500	-6.000	.000

- Note: b_1.cm here is collocated with b_1.hinge point but need not be so

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 b_1 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 b_j

All b_1 Data

- See all data on b_1, 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         =    .694221     -.060737     -.717194
                     .087156      .996195      .000000
                     .714465     -.062507      .696873
   Euler seq    =123
   Euler angs   =       .000      -45.823         5.000

   b2i matrix   =   -.000002      .000000    -1.000000
                     .996195      .087156     -.000002
                     .087156     -.996195      .000000

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

B_j Page Info

- 1st block: all attributes of b_j, {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 b_j page
  - dcm0: edit relative dcm of b_j

    for j=1, this is the b_1 attitude wrt LVLH frame
  - svec: edit the b_j cm position in b_j coord, etc…
  - help: see data and command definitions
  - x: exit this page

Body Actuation Signals

- B_j 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 b_j are processed based on b_j.type as follows.

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

Wheel Menu

- See wheel summary, type ‘whl’ from Model Menus
- ArraySim has 4 wheels all mounted on b_1 (see pa=1)

```
                        ~ Wheel Menu ~
  nof wheels     :   4
  sysh_eci       :  .359908E+01   .331808E+01 -.199014E+01
  syscm          :  .245902E+00   .000000E+00   .000000E+00

  w_inr          :  .100000E+00
  w_spd(d/s)     :  .000000E+00
  w_mom          :  .000000E+00

  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   .5773503  -.5773503    .5773503    .1000        .0
    2 whl2     1 A   .5773503   .5773503    .5773503    .1000        .0
    3 whl3     1 A  -.5773503   .5773503    .5773503    .1000        .0
    4 whl4     1 A  -.5773503  -.5773503    .5773503    .1000        .0

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

- whl_j.parent= 1, for all j, means all wheels are on b_1
- whl_j.type=A means whl_j.input is scalar wheel torque
- whl_j.axis= [x y z] , whl_j spin axis in parent frame
- whl_j.speed= initial whl_j speed
- whl_j.inr=whl_j spin axis inertia

- Type ‘add<j>’ to add wheels to bj
- Type ‘rem<j>’ to remove whl_j
- Type ‘name<j>’ to edit whl_j.name
- Type ‘type<j>’ to edit whl_j.type={A or C}
- Type ‘axis<j>’ to edit whl_j.axis
- Type ‘wspd<j>’ to edit whl_j.wspd

- Type ‘help’ to get definitions of data and commands
- Type ‘x’ to exit menu

Wheel Control Signals

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

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

Force Menu

- See force summary, type ‘force’ from Model Menus
- ArraySim has 8 jet forces
- f_j.parent=1 for all j, means they all impinge b_1
- f_j.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	.200	.200	.000	.000
	2	f2	1	1	1	.200	.200	.000	.000
	3	f3	1	1	1	.200	.200	.000	.000
	4	f4	1	1	1	.200	.200	.000	.000
	5	f5	1	1	1	.200	.000	-.200	.000
	6	f6	1	1	1	.200	.000	-.200	.000
	7	f7	1	1	1	.000	.000	.000	.000
	8	f8	1	1	1	.000	.000	.000	.000

- Note: f_7,f_8 are not given force values because they are not needed by ACS_jet.m of ArraySimc

- Type ‘pos’ to see the position of defined forces in b_1 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 b_1 coord

	idx	name	---tqmag--	---tqx----	---tqy----	---tqz----
=>	1	f1	.600	.000	.000	.600
	2	f2	.600	.000	.000	-.600
	3	f3	.600	.000	.600	.000
	4	f4	.600	.000	-.600	.000
	5	f5	.600	.600	.000	.000
	6	f6	.600	-.600	.000	.000
	7	f7	.000	.000	.000	.000
	8	f8	.000	.000	.000	.000

Force Menu Data & Commands

- Data:
- f_j.parent= 1, for all j, means all jets are on b_1 (bus)
- f_j.type=1 means f_j.input is an on/off signal
- f_j.fvec= [x y z] ,directional vector of f_j
- f_j.fmag= magnitude of f_j when activated
- f_j.fpos=[x y z], impact position of f_j in parent frame

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

- Type 'par<j>' to edit fj.parent
- Type 'pos' to show all force impact positions in b_1 frame
- Type 'rxf' to show torque of all forces about f_ref in b_1 frame
- Type 'help' to get definitions of data and commands
- Type 'x' to exit menu

- Dynamics Input signal for f_j is xfj:
- Run time input processing of xfj:

f_j.type	xfj	size	Sim action
1	1=on, 0=off	1	f_j.force= set value or zero
2	Scalar magnitude of fj	1	f_j.force= xfj*f_j.unitv
3	3 x 1 force vector in b1 frame	3	f_j.force= xfj

Gravity/Orbit Menu

- Need to specify the vehicle orbit
- Type ‘grav’ from Model Menus to enter Gravity Menu
- ArraySim orbit :
  - 65 deg inclined circular orbit
  - True anomaly= 0 deg
  - orbit period is 200 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 b_j.cm is defined by syspos (spherical earth)
  - 11 g at b_j.cm is defined by b_j.pos.eci
  - 12 same as #11 plus gravity gradient torque on b_j
  - 20 g at b_j.cm with J2 is defined by syspos (oblate earth)
  - 21 g at b_j.cm with J2 is defined by b_j.pos.eci
  - 22 same as #21 plus gravity gradient torque on b_j
  - 30 g at b_j.cm with J2, J3 and J4 is defined by syspos (oblate earth)
  - 31 g at b_j.cm with J2, J3 and J4 is defined by b_j.pos.eci
  - 32 same as #31 plus gravity gradient torque on b_j

- 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.

Constraint Menu

- See constraint summary, type ‘cn’ from Model Menus or Body Menu

	idx	type	ic	ln	ov	b1	b2	f1	f2	d1	d2	d3	p1	p2
=>	1	LOCK	0	1	0	3	0	0	0	0	0	0	0	0
	2	LOCK	0	1	0	8	0	0	0	0	0	0	0	0
	3	GEAR	1	1	0	3	4	0	0	0	0	0	0	0
	4	GEAR	1	1	0	4	5	0	0	0	0	0	0	0
	5	GEAR	1	1	0	5	6	0	0	0	0	0	0	0
	6	GEAR	1	1	0	8	9	0	0	0	0	0	0	0
	7	GEAR	1	1	0	9	10	0	0	0	0	0	0	0
	8	GEAR	1	1	0	10	11	0	0	0	0	0	0	0

- ArraySim needs the following:
  a. Lock constraints for b_3 and b_8 for end of array deployment
  b. Gear constraints for b_4:b_6 to constrain angular motion between joint pairs (3,4),(4,5),(5,6)
  c. Gear constraints for b_8:b_{11} to constrain angular motion between joint pairs (8,9),(9,10),(10,11)

- Cn Data:
- cn(j).name= constraint name
- cn(j).ic= 1 means enable cnj at t=0, 0 means otherwise
- cn(j).ln=number of cn equations for cn(j)
- cn(j).(b1, b2)= two bodies involved in cn(j)
- cn(j).(f1, f2)= two forces involved in cn(j)
- cn(j).(d1, d2,d3)= unit vectors involved in cn(j)
- cn(j).(p1, p2)= position markers involved in cn(j)

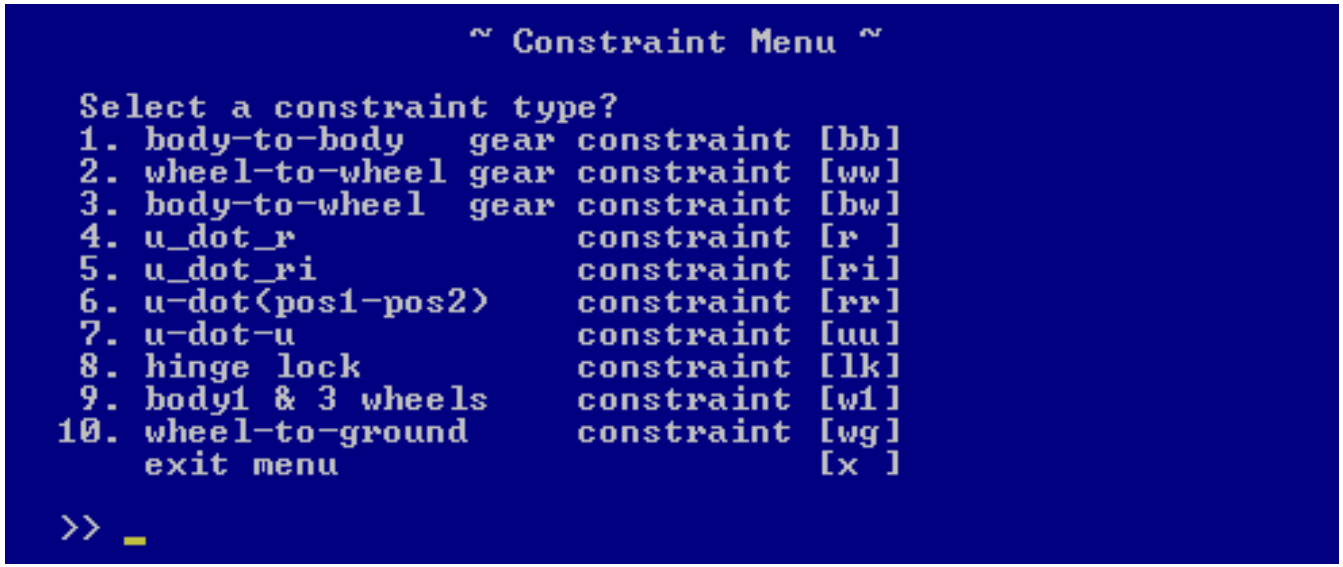
- Cn Commnads:
- Type ‘add’ to add a constraint
- Type ‘rem<j>’ to remove cnj(j)
- Type ‘edit<j>’ to edit cn(j)
- Type ‘body<j>’ to edit cn(j).(b1,b2)
- Type ‘pmkr<j>’ to edit cn(j).(p1,p2)
- Type ‘frc<j>’ to edit cn(j).(f1,f2)

- Type ‘dmkr<j>’ to edit cn(j).(d1,d2,d3)
- Type ‘ic<j>’ to toggle cn(j).ic between 0 and 1
- Type ‘help’ to get definitions of data and commands
- Type ‘x’ to exit menu

- Dynamics Input signal for cn(j)= cnj
- Input processing of cnj:

Cnj	size	Sim action
1	1	Enables the constraint represented by cn(j)
0	1	Disable cn(j)

- See constraint selection menu, type ‘add’ from Constraint Menu



- Type ‘bb’ to select body-to-body gear constraint
- Type ‘ww’ to select wheel-to-wheel gear constraint
- ans so forth
- Type ‘lk’ to select a hinge lock constraint
- Type ‘x’ to exit this menu

Lock Constraint

- Constraint(j) is controlled by the dynamics input signal cnj
- Constraint(j) is enabled when cnj=1
- If cnj.type=Lock & cnj=1, then cnj.body.joint is locked
- In this case, cnj.body=3 and 8
-
- To define cnj.lock for b3 and b8:
    1. Type ‘add’ from Constraint Menu
    2. Select ‘LK’ for constraint
    3. complete ‘LK’ dialog by replying 3 to obtain a lock constraint on body 3
    4. repeat 1-3 for body 8

Gear Constraint

- Constraint(j) is controlled by the dynamics input signal cnj
- Constraint(j) is enabled when cnj=1
- If cnj.type=BB & cnj=1, then cnj.body1.angle and cnj.body2.angle are constrained to satisfy
- k1*body1.angle+ k2*body2.angle= 0

- To define cnj.type=BB for b_3 and b_4:
  1. Type 'add' from Constraint Menu
  2. Select 'BB' for constraint
  3. Complete 'BB' dialog by replying 3 and 4 for body1 and body2
  4. Reply with 2 and 1 for k1 and k2, since ang3 need to travel from 90 to 0 while ang4 needs to move from -180 to 0
- Repeat steps 1-4 for other pairs listed on page 38 with k1,k2=[1, 1]
- Type gains to see all k1, k2

- See constraint control gains, type ‘gains’

	idx	type	ic	k1	k2	pgain	vgain
=>	1	LOCK	0	.00000E+00	.00000E+00	.40000E+01	.40000E+01
	2	LOCK	0	.00000E+00	.00000E+00	.40000E+01	.40000E+01
	3	GEAR	1	.20000E+01	.10000E+01	.40000E+01	.40000E+01
	4	GEAR	1	.10000E+01	.10000E+01	.40000E+01	.40000E+01
	5	GEAR	1	.10000E+01	.10000E+01	.40000E+01	.40000E+01
	6	GEAR	1	.20000E+01	.10000E+01	.40000E+01	.40000E+01
	7	GEAR	1	.10000E+01	.10000E+01	.40000E+01	.40000E+01
	8	GEAR	1	.10000E+01	.10000E+01	.40000E+01	.40000E+01

JNT Menu

- Sim Engine can generate hinge torque for simple joint position or rate control for specified joints
- Implements the following bj.torque:

  htqaxj= kp*(cmd_ang – angj)+kv*(cmd_rate – wrelaxj)+ preload

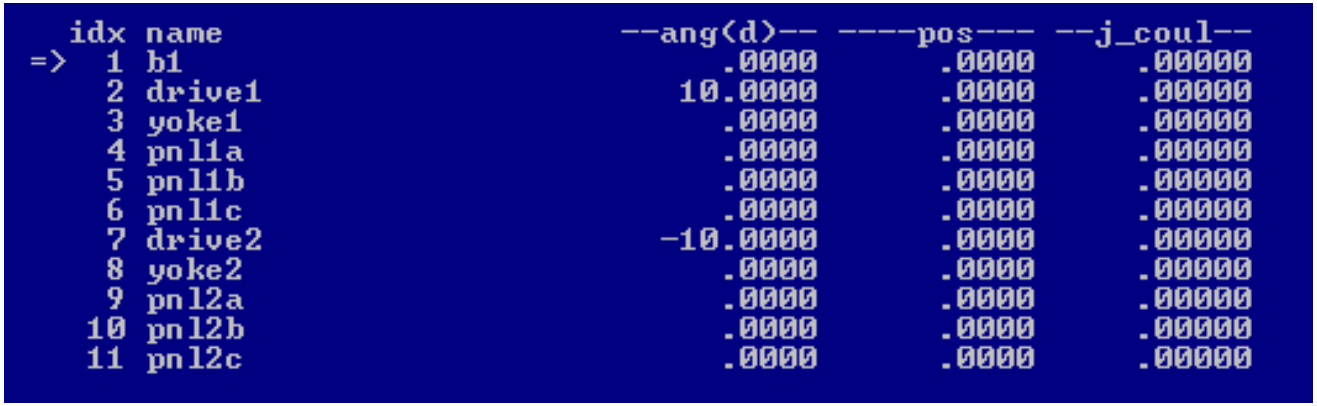
- See joint control or JNT summary, type ‘jnt’ from Model Menus or Body Menu

	idx	name	tp	ax	mode	----kp----	----kv----	---pLoad--
=>	1	b1	B	x	0 0 0	.0000	.0000	.0000
	2	drive1	A	y	1 0 0	.3200	6.8000	.0000
	3	yoke1	A	z	1 0 0	.0000	50.0000	.5000
	4	pnl1a	A	z	0 0 0	.0000	.0000	.0000
	5	pnl1b	A	z	0 0 0	.0000	.0000	.0000
	6	pnl1c	A	z	0 0 0	.0000	.0000	.0000
	7	drive2	A	y	1 0 0	.3200	6.8000	.0000
	8	yoke2	A	z	1 0 0	.0000	50.0000	-.5000
	9	pnl2a	A	z	0 0 0	.0000	.0000	.0000
	10	pnl2b	A	z	0 0 0	.0000	.0000	.0000
	11	pnl2c	A	z	0 0 0	.0000	.0000	.0000

- Arraysimc needs JNT for the following tasks:
  1. Position or rate control for array drives: b_2, b_7
  2. Highly damped pushoff force on b_3 and b_8 to deploy arrays

- bj.jnt.mode = (a,b,c)
  - a=1 means position control, a=2 means rate control
  - b=0 ignore for now
  - c=0 ignore for now
- bj.jnt.(kp, kv) = joint torque position and rate control constants
- bj.jnt.(preload)= joint torque preload
- bj.jnt.j_coul = joint Coulomb force/torque

- Type ‘modej’ to edit bj.jnt.mode
- Type ‘pgainj’ to edit bj.jnt.kp, and so forth
- Type ‘show’ to see commands and j_coul summary



	idx	name	--ang(d)--	----pos---	--j_coul--
=>	1	b1	.0000	.0000	.00000
	2	drive1	10.0000	.0000	.00000
	3	yoke1	.0000	.0000	.00000
	4	pnl1a	.0000	.0000	.00000
	5	pnl1b	.0000	.0000	.00000
	6	pnl1c	.0000	.0000	.00000
	7	drive2	-10.0000	.0000	.00000
	8	yoke2	.0000	.0000	.00000
	9	pnl2a	.0000	.0000	.00000
	10	pnl2b	.0000	.0000	.00000
	11	pnl2c	.0000	.0000	.00000

JNT Position/Rate Control

- For Position control
- set b_j.jnt.mode=[1 0 0]
- set b_j.jnt.ang= cmd angle
- set b_j.jnt.preload= 0
- set b_j.jnt.pgain= kp (provided by analysis)
- set b_j.jnt.vgain= kv (provided by analysis)

- For Rate Control
- set b_j.jnt.mode=[2 0 0]
- set b_j.jnt.ang= cmd rate
- set b_j.jnt.preload= 0
- set b_j.jnt.pgain= 0
- set b_j.jnt.vgain= kv (provided by analysis)

Highly Damped Torque

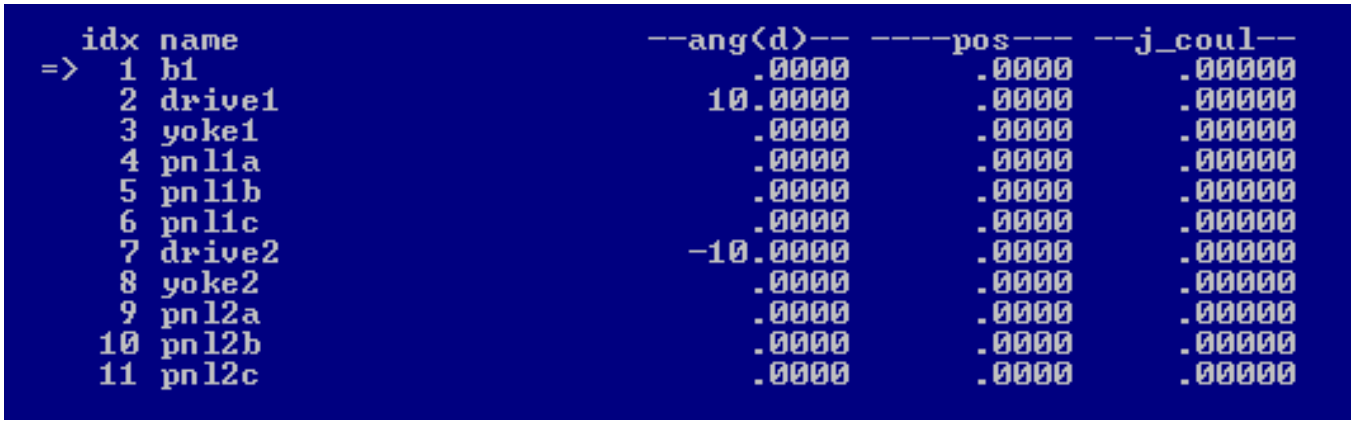
- Have a highly damped torque at bj.hinge to initiate array deployement with the following hinge torque for j=3 and 8:

  bj.htqax= preload -kv*bj.rate

- for b_3.joint:
- set b_3.jnt.mode=[1 0 0]
- set b_3.jnt.preload= .5
- set b_3.jnt.pgain= 0.
- set b_3.jnt.vgain= 50

- for b_8.joint:
- set b_8.jnt.mode=[1 0 0]
- set b_8.jnt.preload= -.5
- set b_8.jnt.pgain= 0.
- set b_8.jnt.vgain= 50

- See ang/rate and j_coul commands summary, type ‘show’



idx	name	--ang(d)--	----pos---	--j_coul--
=> 1	b1	.0000	.0000	.00000
2	drive1	10.0000	.0000	.00000
3	yoke1	.0000	.0000	.00000
4	pnl1a	.0000	.0000	.00000
5	pnl1b	.0000	.0000	.00000
6	pnl1c	.0000	.0000	.00000
7	drive2	-10.0000	.0000	.00000
8	yoke2	.0000	.0000	.00000
9	pnl2a	.0000	.0000	.00000
10	pnl2b	.0000	.0000	.00000
11	pnl2c	.0000	.0000	.00000

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)
- Arraysimc input list is as follows:

```
Udata list:

 1) cn,1          ¦ 2) cn,2          ¦ 3) whltq,1
 4) whltq,2       ¦ 5) whltq,3       ¦ 6) whltq,4
 7) xf,1          ¦ 8) xf,2          ¦ 9) xf,3
10) xf,4          ¦11) xf,5          ¦12) xf,6
```

- cn, 1:2 = hinge locking constraint enable signals
- whltq, 1:4= four rwa torque
- xf,1:6 = six jet on/off signals

- 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

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)

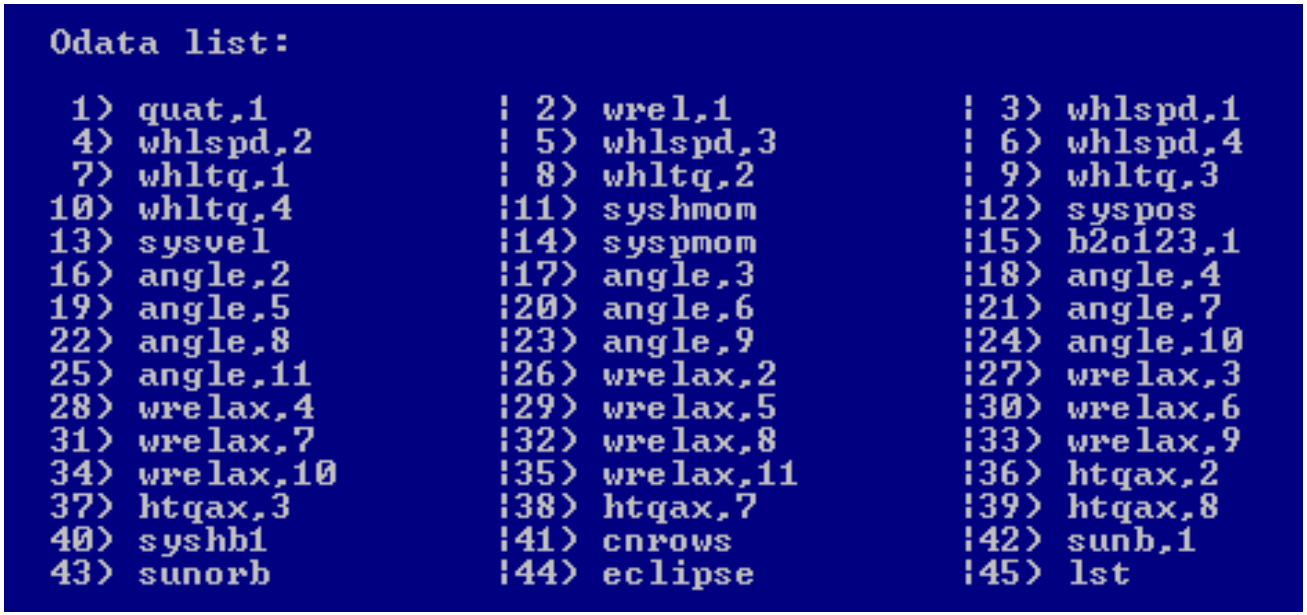
- angle,3 & 8 = array locking joint angles
- w, 1 = bus angular rate in b1 frame
- b2osml, 1 = small angle b1.attitude_err wrt LVLH frame
- whlspd, 1:4 = rwa speed

- 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

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)
- Arraysimc plot list is as follows:



- Plot data groups:
  - Bus: quat1, wrel1, syshb1, sunb1, b2o1231
  - System: syspos, sysvel, sysacc, syshmom, sunorb,cnrows, eclipse,lst
  - Wheels: whlspd1:4, whltq1:4
  - Arrays: angle2:11, wrelax2:11, htqax2:11

- 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

Simplot

- Need to construct simplot1.m to view sim results in Matlab, then type ‘simplot’ from Model Menus (page 11) or Plot Menu (page 54)
- All plot data were selected in Plot Menu
- A. steps from simPlot Menu:
  1. Type 'add’ to add figures and respond with ‘3’ to create 3 figures
  2. Type 'title1’ to set figure (1) title: i.e. reply with ‘system’
  3. Type 'vars1’ to define variables to be plotted in fig 1. this opens the plot variables page
- B. steps from vars menu:
  1. Type 'addv9’ and reply with 5to add 5 variables starting with the variable(9), syshmom
  2. Type 'addv2’ and reply with 1 to add ‘wrel1’ to the variable list
  3. Type 'addp’ and respond with ‘1,6’ to create six 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.2, A.3 and all B steps with proper indexing to define other figures for sat3w2a
- 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 11) 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 arraysim.txt
2. save model data to another file
3. Cancel save
Select 1:3? 1

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

- On hitting return, current model data would have been saved to Arraysimc.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 57

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 {11, 4, 8}?
  - no, for example a CMG4sim Project license has an object count of {5, 4, 8} and a unique parent-child relation between bodies, wheels and forces
- How to setup Arraysimc 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

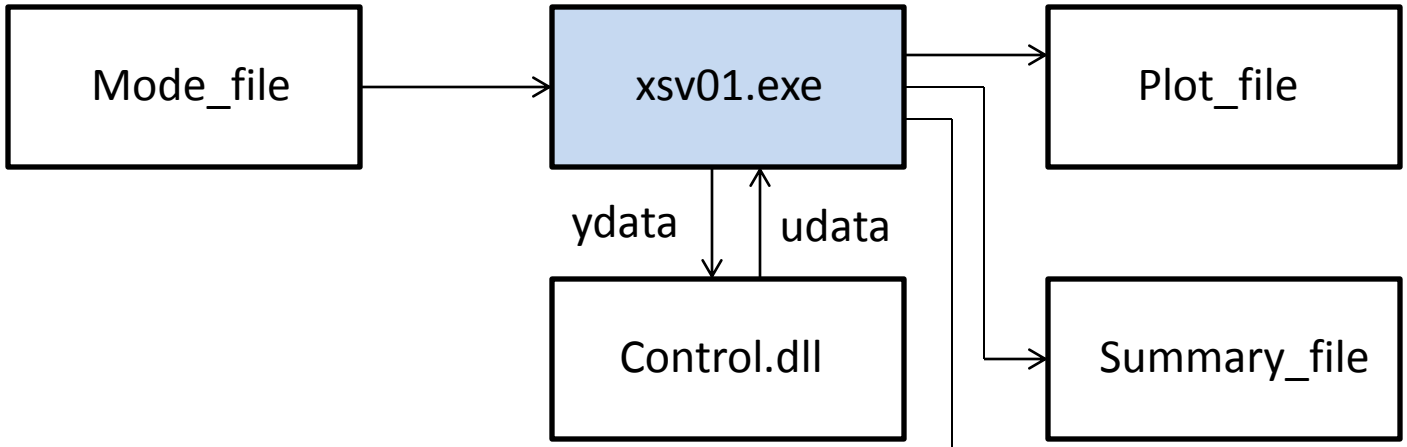
- Xsv01.exe
- How Control.dll Works
- User_code.c
- Vehdata.h
- Utilc.lib subroutines
- Acsc.lib subroutines
- Control System
- Example1
- Example2
- Adjustable Simulation Parameters
- Exercises
- Simulation Notes
- Summary

XSV01.exe

- Xsv01.exe is the main program that comes with the Arraysimc 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 Arraysimc.

> Part II begins next

Part II Introduction

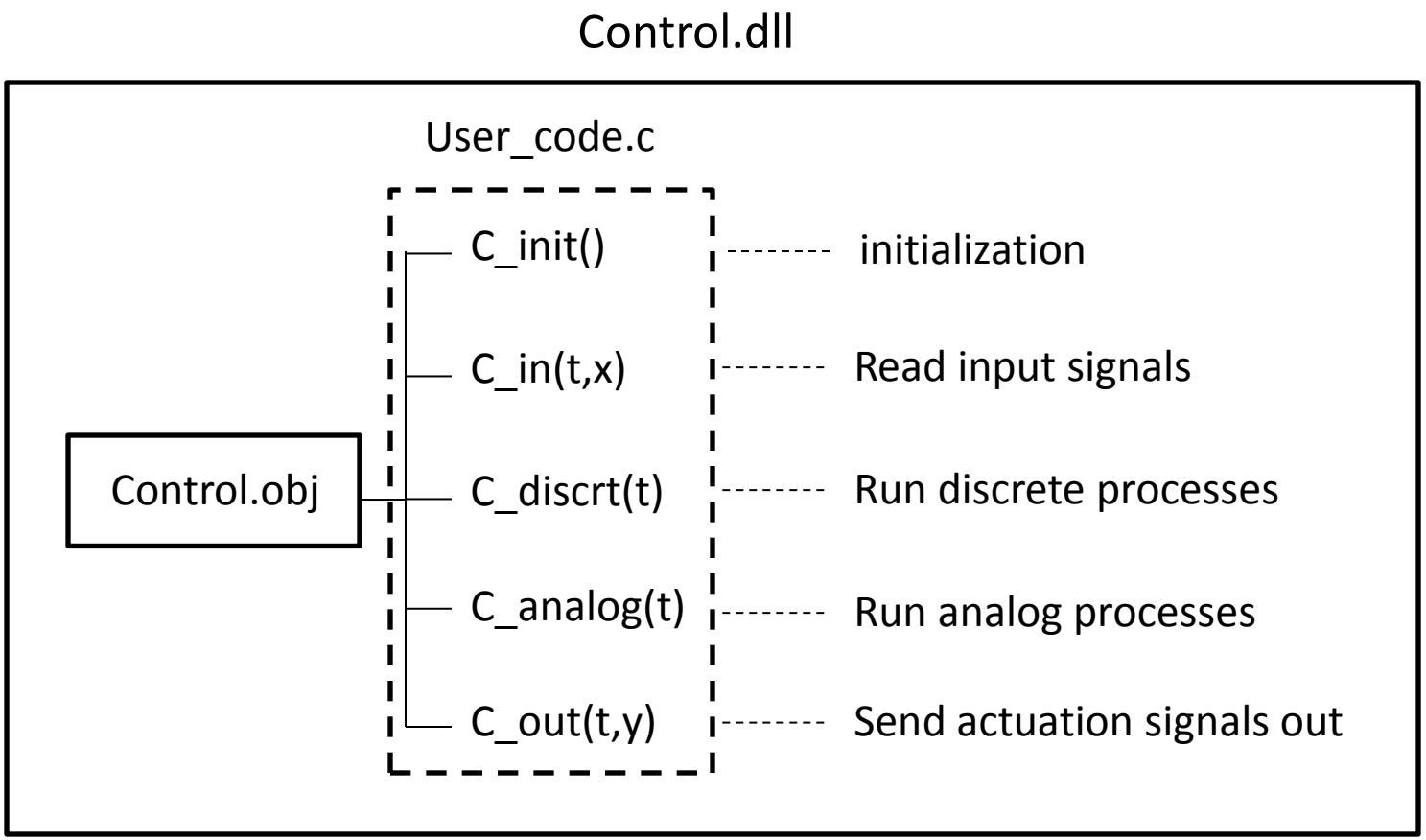
- The following charts show how to build a trial control.dll that has the framework of the control system needed for the application.
- Two parts need to be built: control subroutines and header file
- At a minimum, we would use Buildx.Codegen Menu to construct 5 key control subroutines and their header 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

Arraysimc Discrete Processes

- For Arraysimc, 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 Arraysimc.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	Arraysimc, arraysimc, cmg4simc
Get_6jettimes	Compute 6 jet on/off times to null attitude error command	Arraysimc, 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

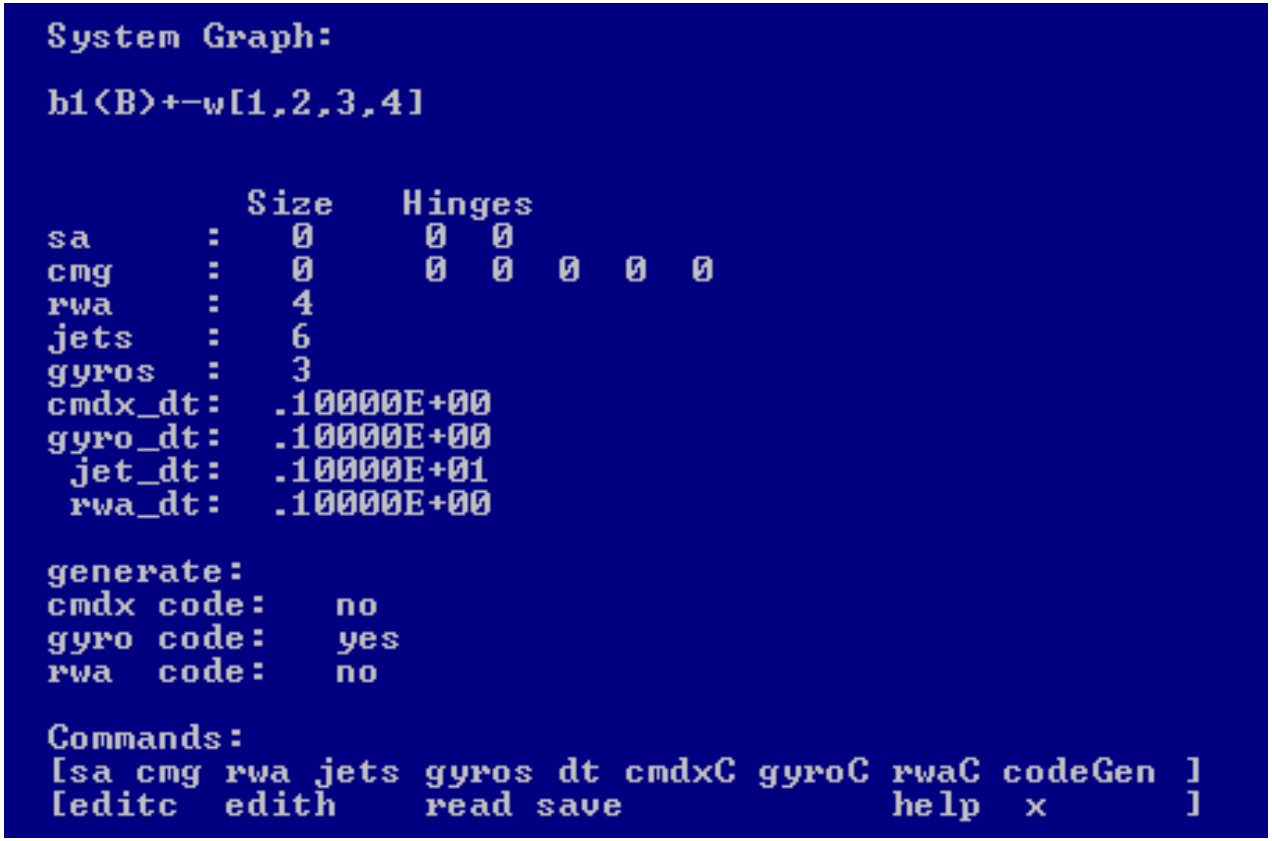
Special Names

- 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



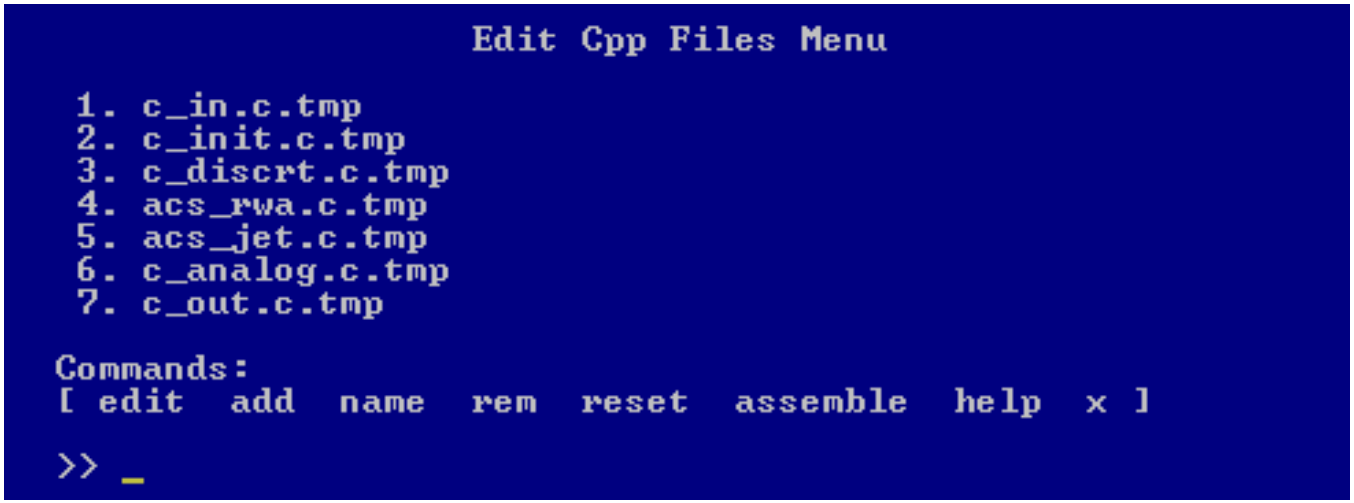
- Type ‘sa’ to define number of arrays (max=2) and the driver hinges (bodies)
- Type ‘cmg’ to define number of CMG’s and the rotor platform bodies
- Type 'rwa’ to define number of reaction wheels
- Type ‘jets’ to define number of jets
- Type ‘gyros’ to define number of gyros
- Type ‘dt’ to define the 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

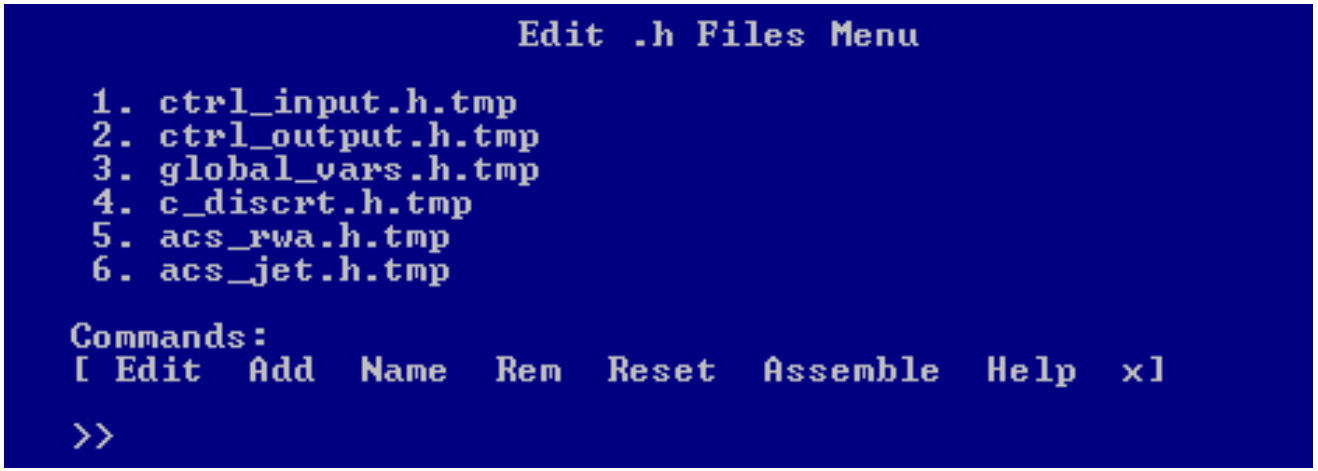
- Next, type ‘editc’ to view user_code.c.tmp components and to assemble user_code.c.tmp



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

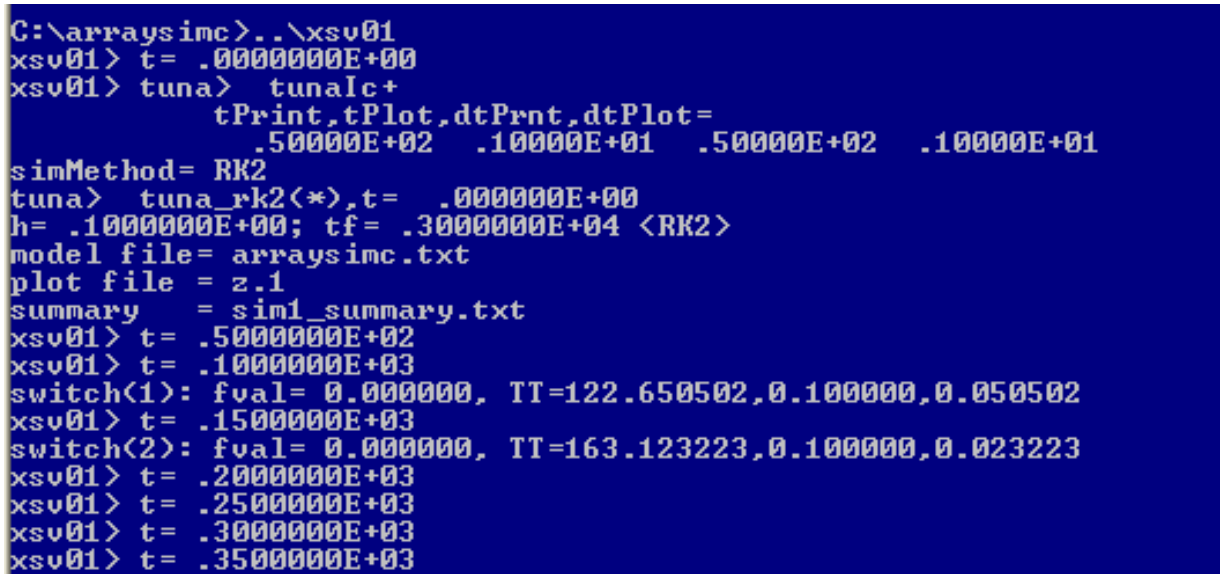
- This example shows
  - Arrays are deployed until all joint angles are 0 , at t={122.6, 163.2} sec
  - Array drive are position controlled to b2.ang=10, b7.ang=-10
  - the LVLH control with jets followed by rwa given a large initial vehicle angular momentum

- Buildx procedure:
  - go to xsv.body menu and set b1.wrel to [.2, 1.0, -.5]
  - Set (b2,b3,b7,b8).jnt.mode=[1 0 0]
  - Set (b2,b7).jnt.ang=[10,-10], (b3,b8).jnt.ang=[0, 0]
  - Set (b2,b7).(kp, kv)=[32, 6.8], (b3,b8).(kp,kv)=[0, 50]
  - Set (b2,b7).preload=0, (b3,b8).preload= 0.5
  - Set orbit.(ecc, incl, period)=(0, 10 deg, 100 min)

- C_init:
  - set worb= [0, 2*pi/6000 , 0] ; orb period= 100 min

Run XSV01.exe

- Click c:\Arraysimc\xsv01.bat and see a running time display



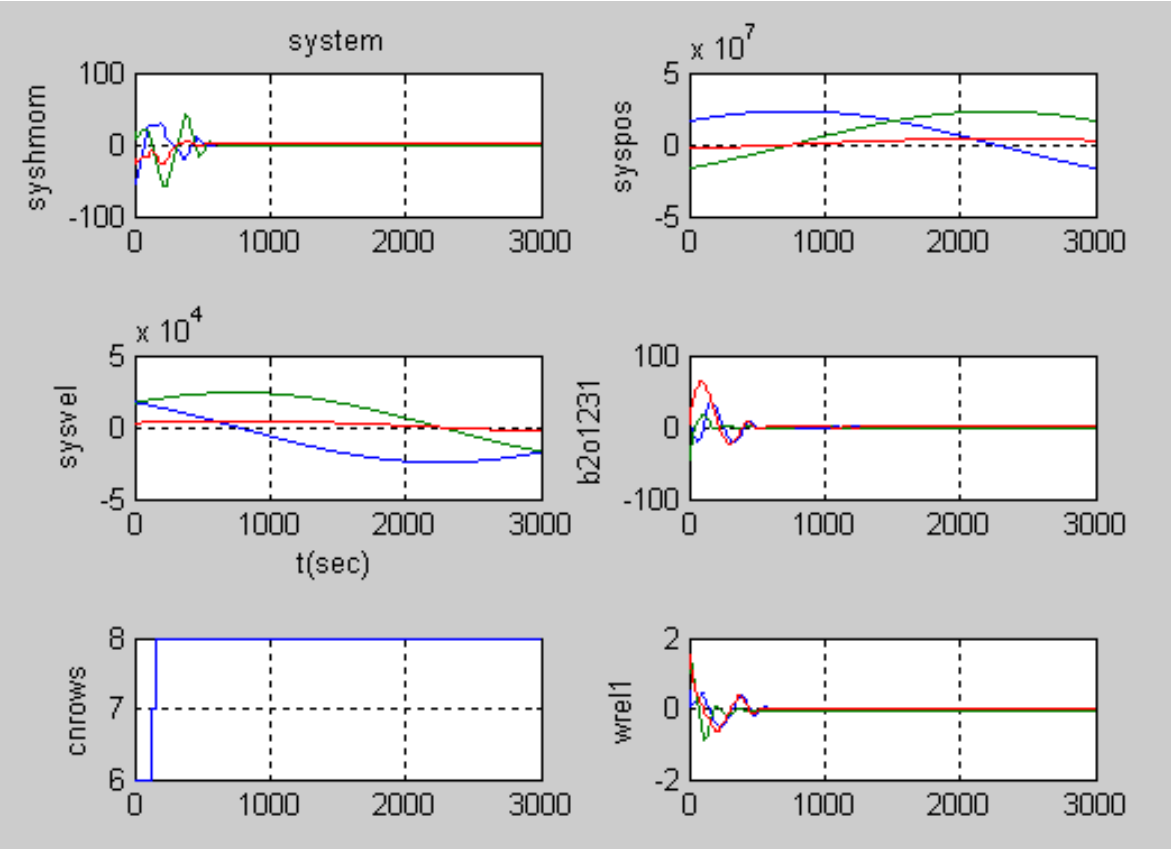
...

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 56)

Fig.1 System Motion-1



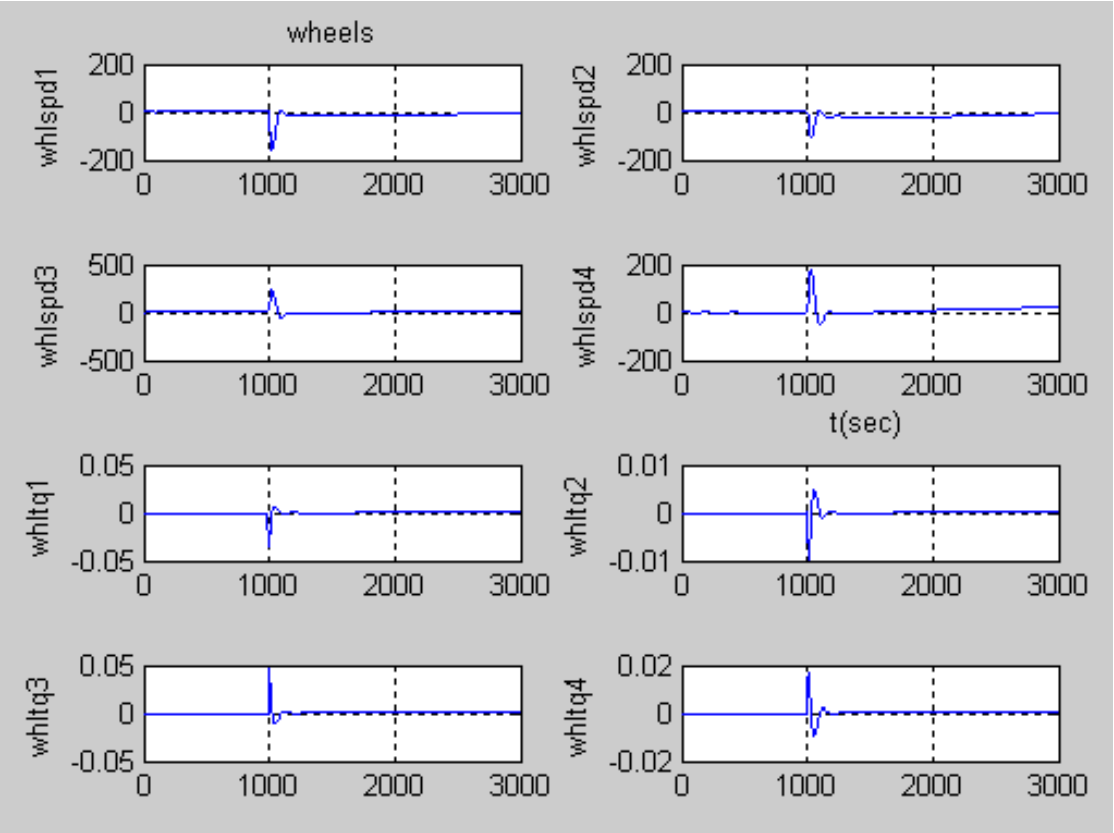
- orbit:
  incl= 10 deg
  circular
  per=100 min
  gravity: gflag=10
  => spherical earth grav

- see syshmom, constant for t >1000

- ACS:
  b1.rpy=b2o1231=0
  b1.wrel= orbit rate

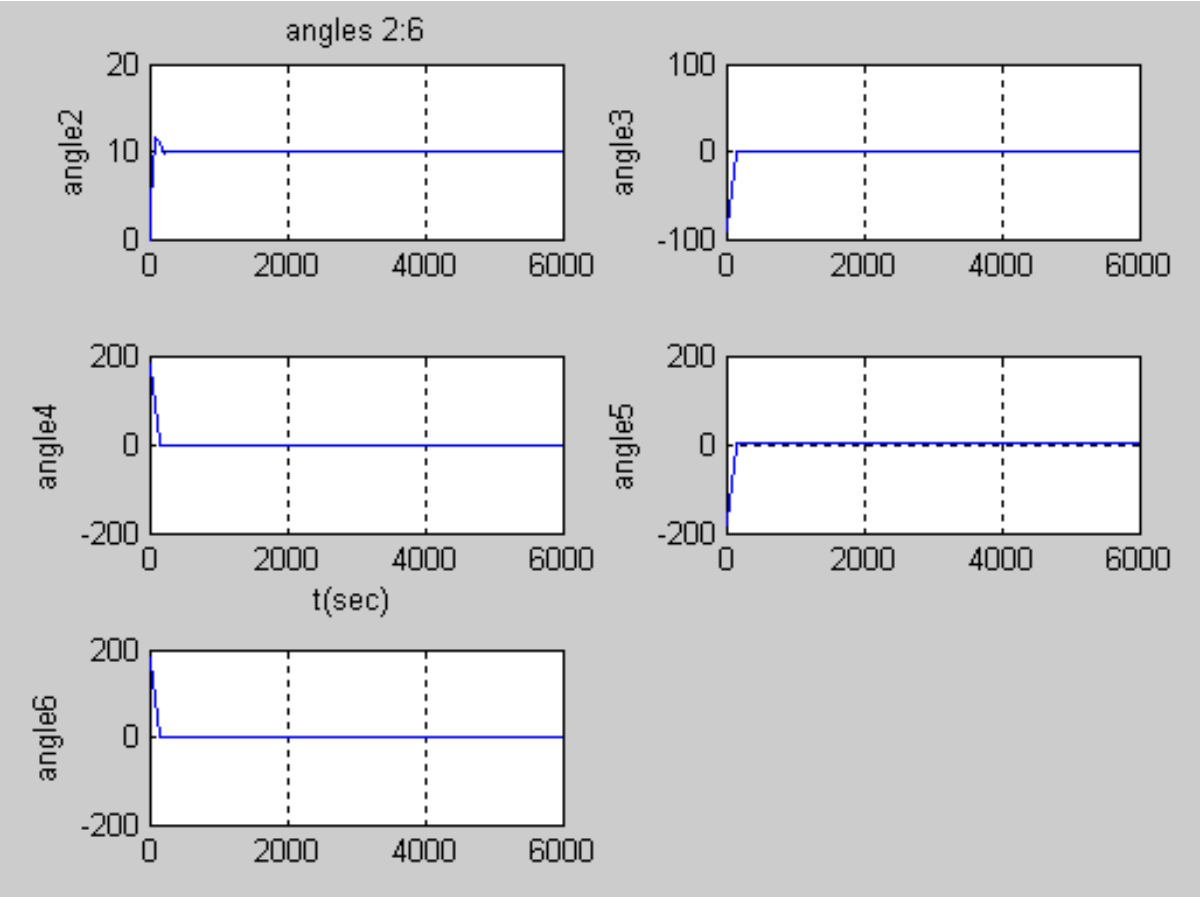
- hinge lock constraints on at t={122.6, 163.1} sec

Fig.2 Wheels Motion-1



- Wheel speed and wtq amplitudes are small due to small syshb1

Fig.3 Array1 Motion-1

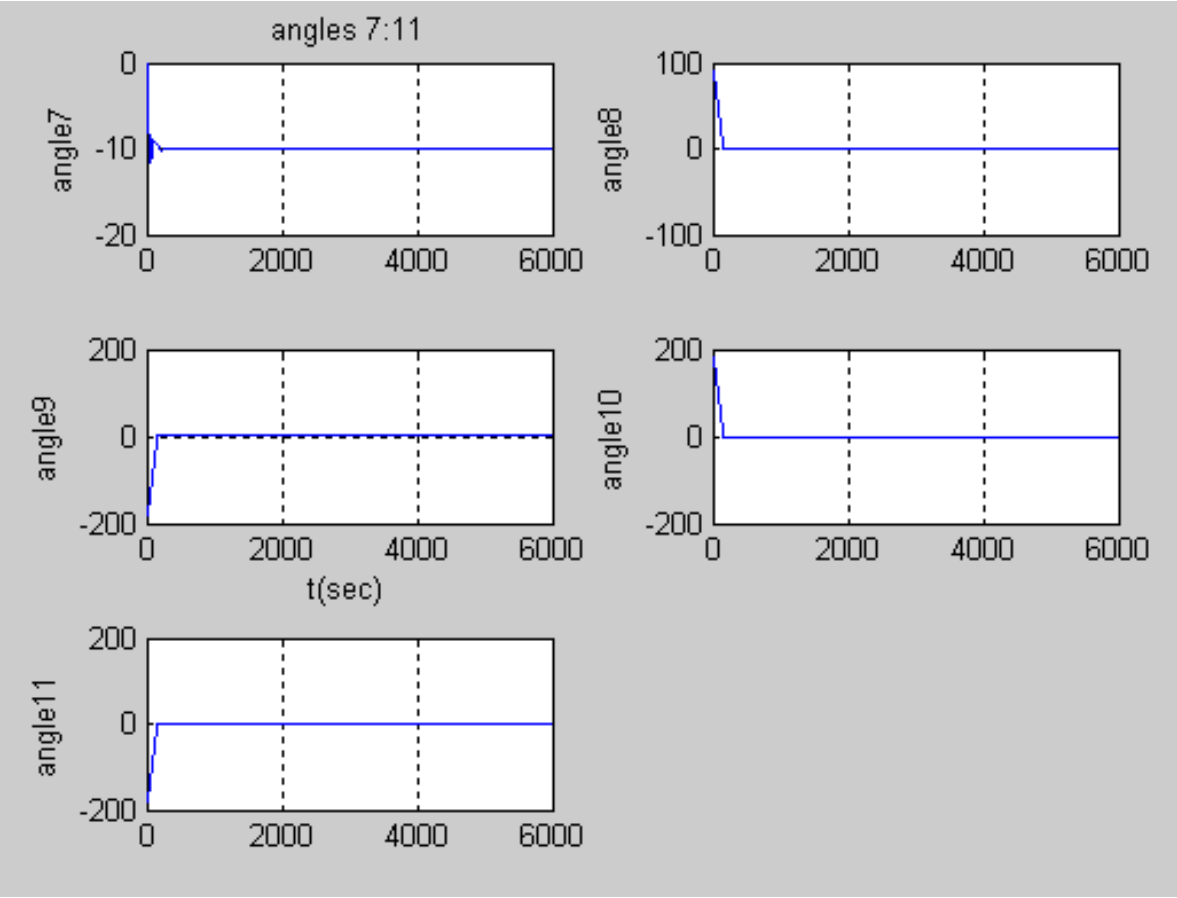


- array 1control:
- ss drive1 angle(ang2)= 10 deg

- array1 (ang 3:6) satisfy gear constraints cn3:5

- array1 (ang 3:6) locks at t=165 sec

Fig.4 Array2 Motion-1

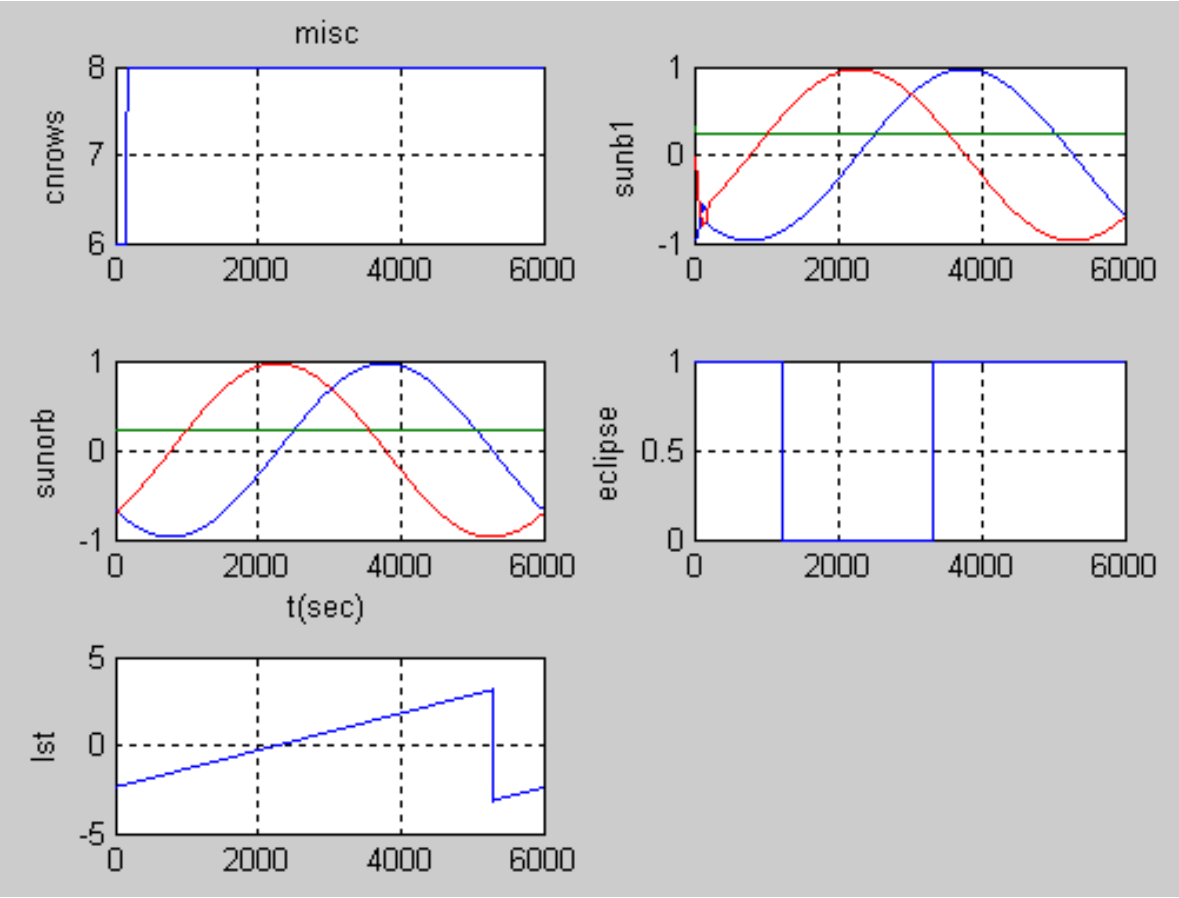


- array 2control:
- ss drive2 angle(ang7)= -10 deg

- array2 (ang 8:11) satisfy gear constraints cn6:8

- array2 (ang 8:11) locks at t=165 sec

Fig.5 Other Signals-1



- other
- cnrows= nof cn's enabled
- sunb1= sun in b1 coord
- sunorb= sun in LVLH coord
- eclipse= 1 means in earth shadow
- lst= local satellite time (rad)

Comment-1

- This example shows that the vehicle inertial angular momentum is constant and stayed near zero due to the initial ACS_jet LVLH control
- The two arrays were deployed and locked properly as by the ‘lock’ and ‘gear’ constraints
- The array drive were position controlled to 10 deg for b2 and -10 deg for b7 per jnt settings
- The wheel speed amplitude is low due to the small system angular momentum brought about by the ACS jet activities early on
- Generally the vehicle response is well behaved as specified

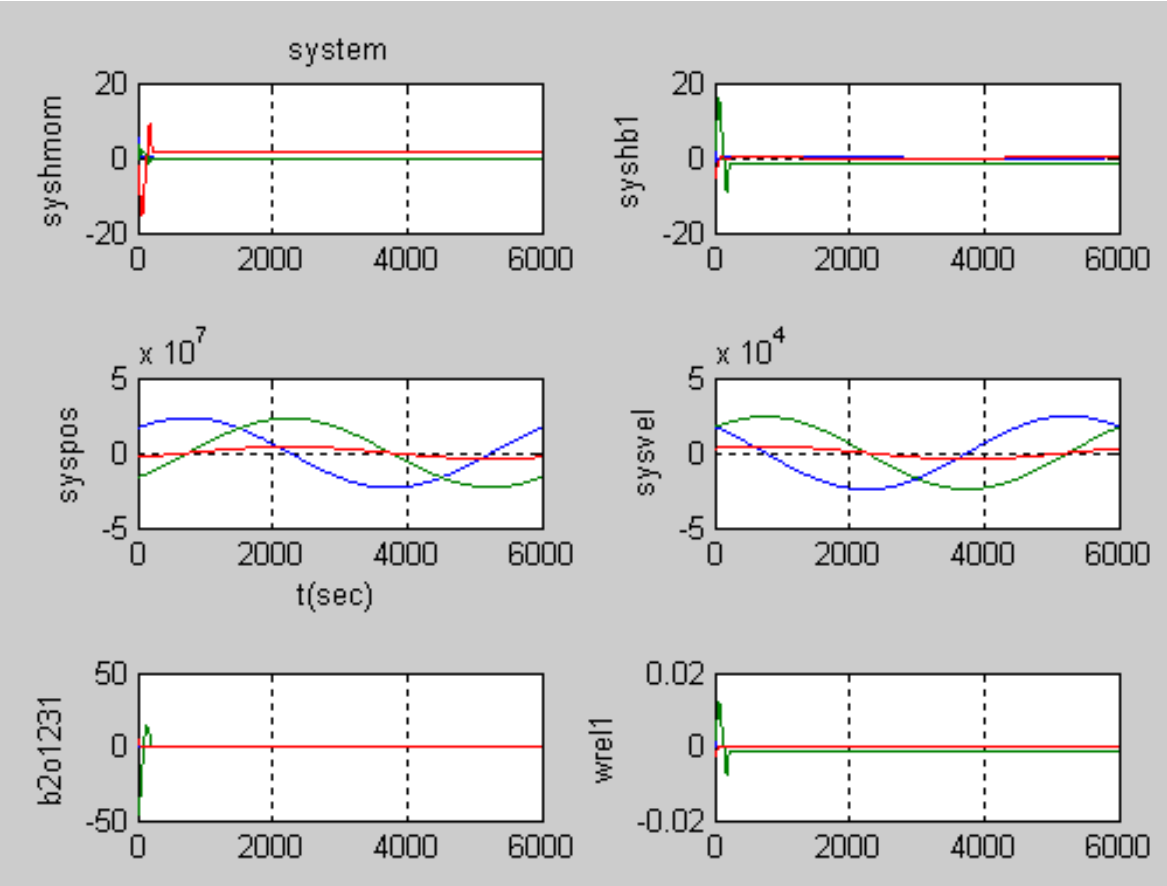
Example2

- This example shows
  - Arrays are deployed until all joint angles are 0
  - Array drive are rate controlled to b2.ang= 0.1 d/s, b7.ang=-0.1 d/s
  - the LVLH control with jets followed by rwa given a large initial vehicle angular momentum

- Buildx procedure:
  - go to xsv.body menu and set b1.wrel to [.2, 1.0, -.5]
  - Set (b2,,b7).jnt.mode=[2 0 0], (b3,,b8).jnt.mode=[1 0 0]
  - Set (b2,b7).jnt.ang=[0.1,-0.1], (b3,b8).jnt.ang=[0, 0]
  - Set (b2,b7).(kp, kv)=[0, 6.8], (b3,b8).(kp,kv)=[0, 50]
  - Set (b2,b7).preload=0, (b3,b8).preload= 0.5
  - Set orbit.(ecc, incl, period)= (0, 10 deg, 100 min)

- Arraysimc_ic.m:
  - set worb= [0, 2*pi/6000 , 0] ; per=100 min

Fig.6 System Motion-2

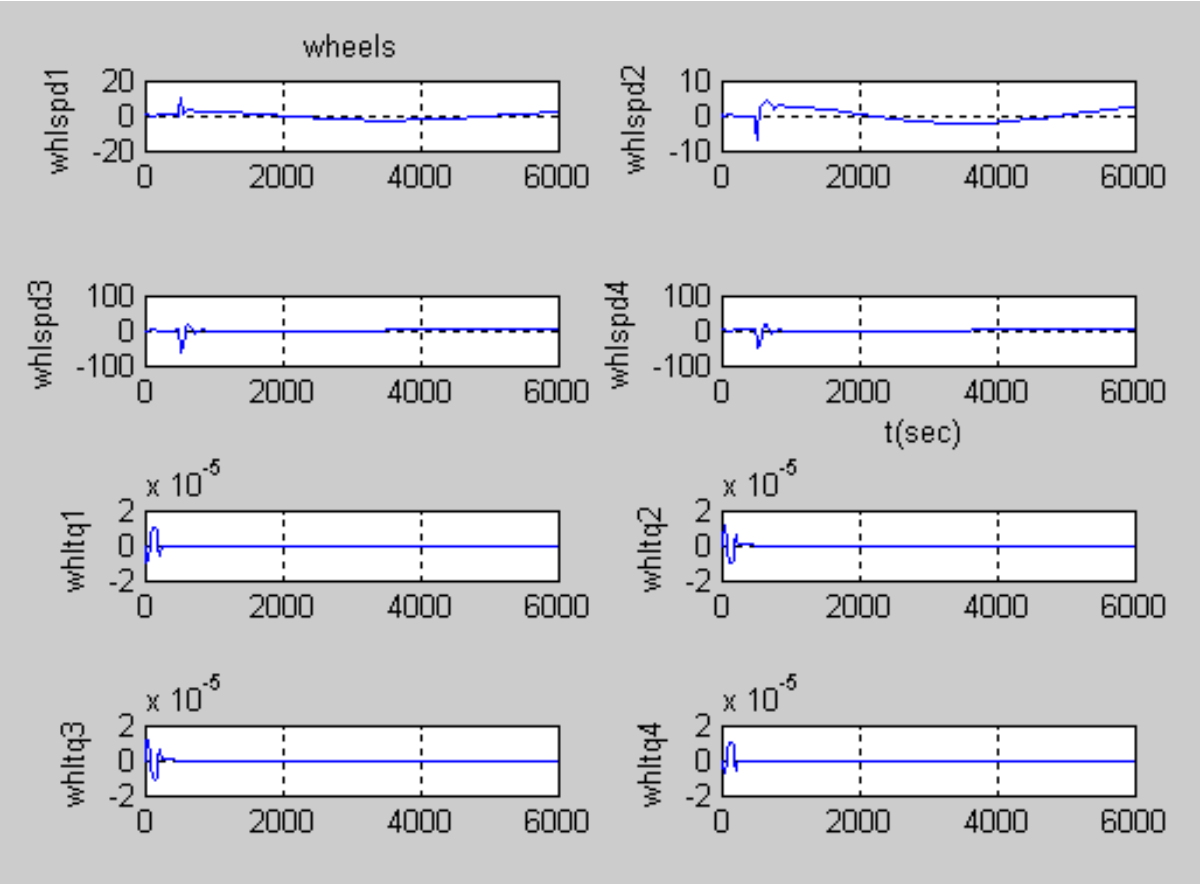


- orbit:
  incl= 10 deg
  circular
  per=100 min
  gravity: gflag=10
  => spherical earth grav

- see syshmom, constant past t=500

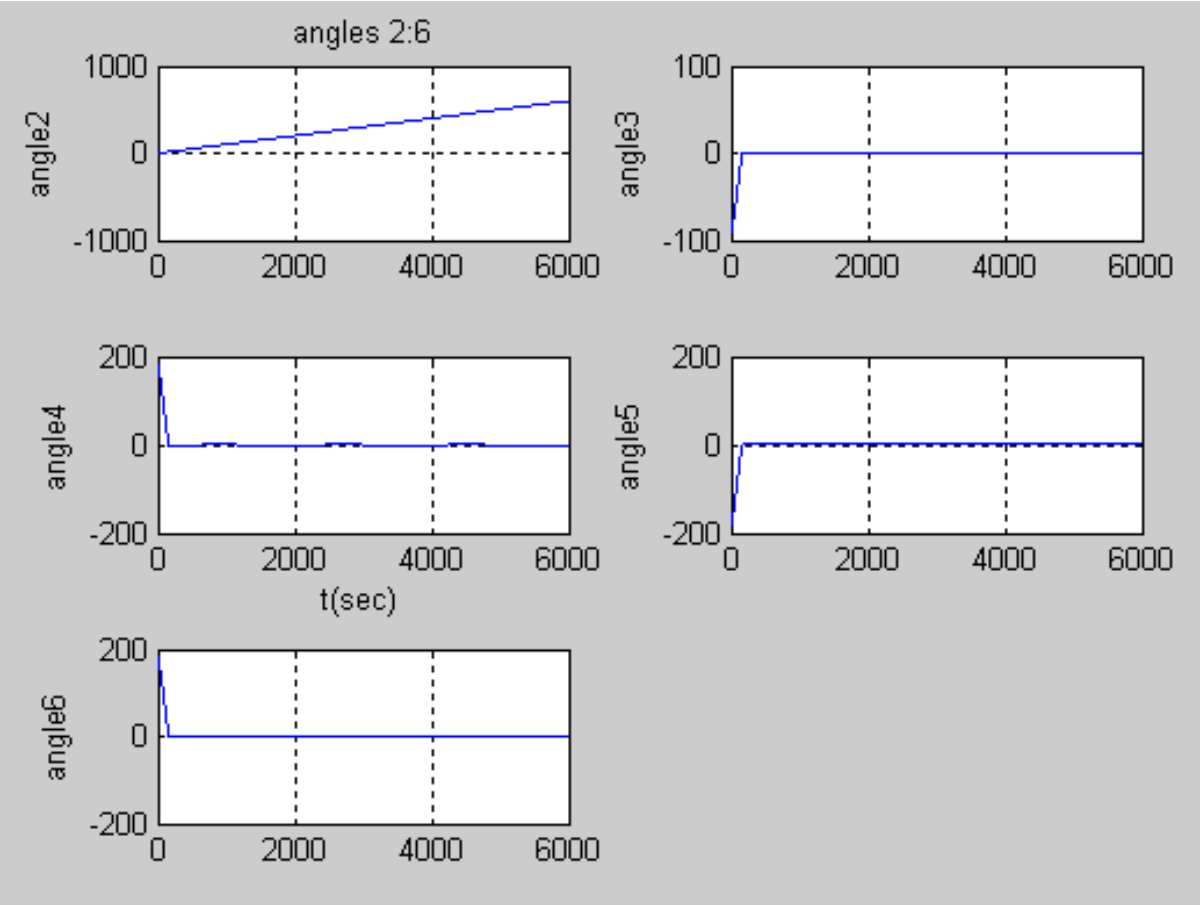
- ACS:
  b1.rpy=b2o1231=0
  b1.wrel= orbit rate

Fig.7 Wheels Motion-2



- Wheel speed and wtq amplitudes are small due to small syshb1

Fig.8 Array1 Motion-2

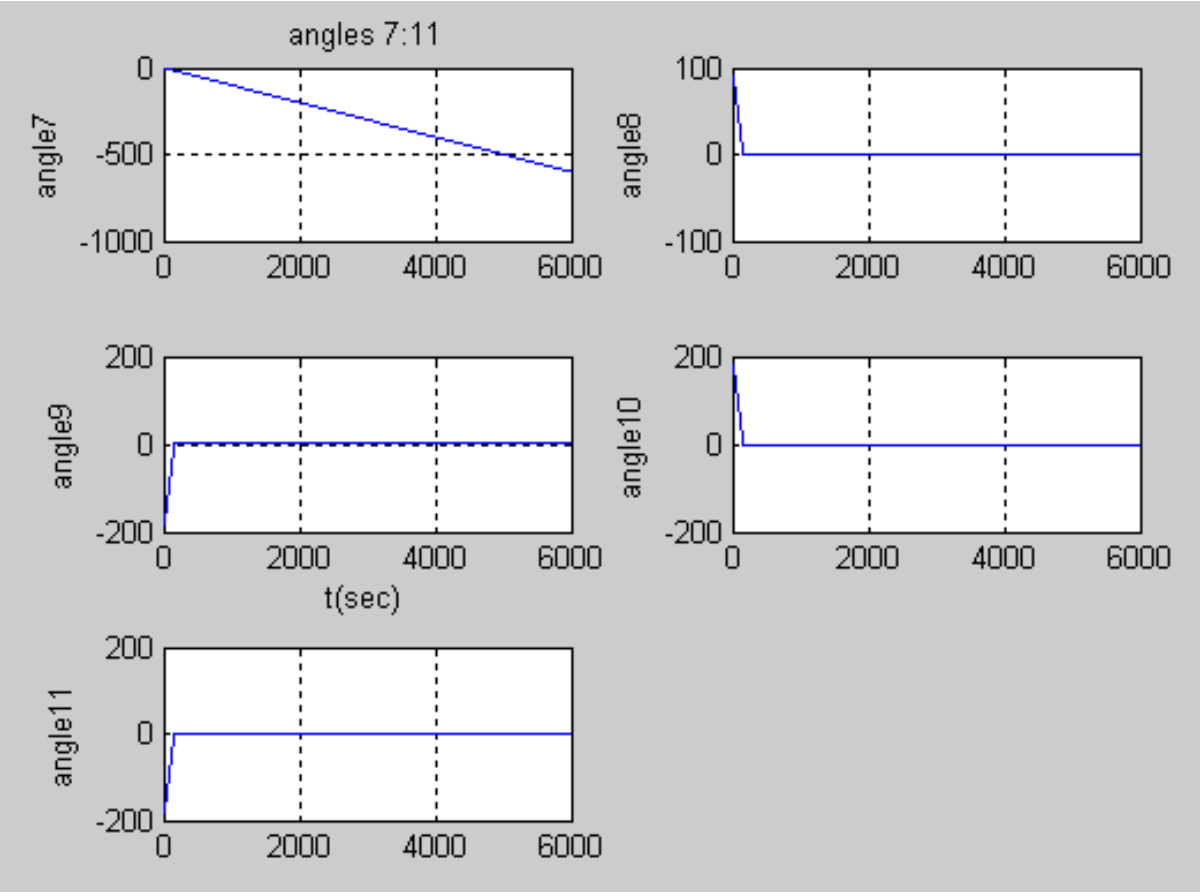


- array 1control:
- ss drive1 b2.wrelax= 0.1 d/s

- array1 (ang 3:6) satisfy gear constraints cn3:5

- array1 (ang 3:6) locks at t=165 sec

Fig.9 Array2 Motion-1

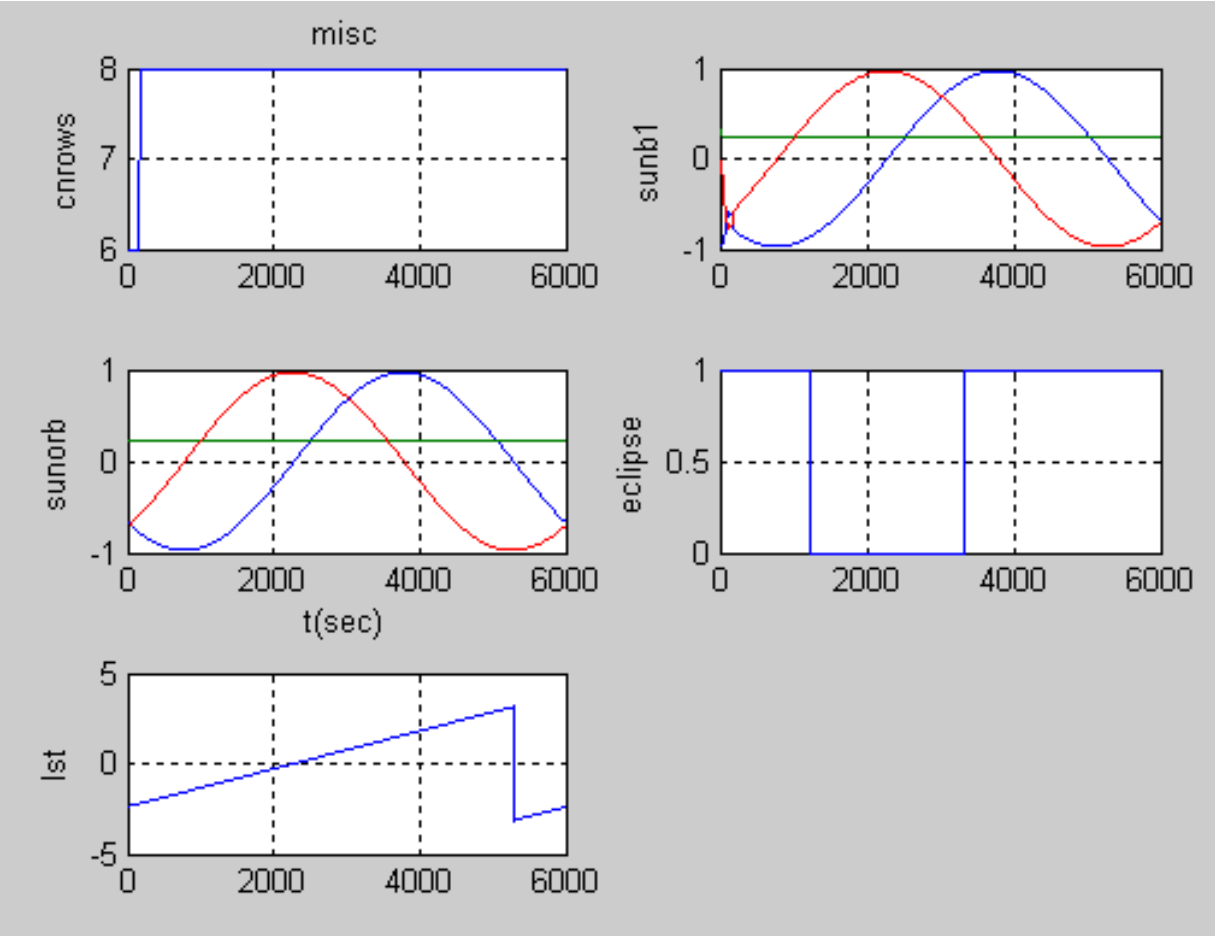


- array 2control:
- ss drive2 b7.wrelax= -0.1 d/s

- array2 (ang 8:11) satisfy gear constraints cn6:8

- array2 (ang 8:11) locks at t=165 sec

Fig.10 Other Signals-2



- cnrows= nof cn's enabled
- sunb1= sun in b1 coord
- sunorb= sun in LVLH coord
- eclipse= 1 means in earth shadow
- lst= local satellite time (rad)

Comment-2

- This example shows that the vehicle inertial angular momentum is constant and stayed near zero due to the initial ACS_jet LVLH control
- The two arrays were deployed and locked properly as by the ‘lock’ and ‘gear’ constraints
- The array drive were rate controlled to 0.1 d/s for b2 and -0.1 d/s for b7 per JNT settings
- The wheel speed amplitude is low due to the small system angular momentum brought about by the ACS jet activities early on
- Generally the vehicle response is nearly identical to that from Example1 as specified

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 sim parameters above
3. Type ‘savel’ to save changes to siminputfile
4. Reply with ‘sim1files.txt’
5. exit buildx

exercises

actions	parameters	reference
change mass property	mass, inr, svec, dvec	pages:17-23
change initial condition	ang,wrel,wrelax, dcm0	pages:17,18
change orbit	ephemeris, orbit period	pages:34-37
add /remove bodies*		pages:17-18
add /remove wheels*		pages:27-29
add/remove forces*		pages:30-33
modify input (udata)		pages:50-51
modify output (ydata)		pages:52-53
modify plot (odata)		pages:54-55
modify simplot1.m		page:56

* 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 Arraysimc •adjust i/o mux/dmux
design your own jet controller	•may need new ydata •may need nof jets other than 8	•likely identical to Arraysimc •adjust i/o mux/dmux

Simulation Notes

Subject	Arraysimc	comments
gforces	comment applies to Arraysimc	gravity forces are auto-computed by sim engine for all bj in system
jet forces	force locations and vectors were selected for Arraysimc 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 Arraysimc because of other design considerations. as such, associated jet controller can be complex
wheels	Arraysimc 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	Arraysimc 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	Arraysimc	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 Arraysimc 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 Arraysimc 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 Arraysimc	this ydata signal is available as a functional convenience to momentum mgmt design

Summary

- Two examples given here show that xsv01.exe can simulate multibodied system that require constraints for array deployment.

- The value of Arraysimc is that its mass property can be varied to fit the vehicle of interest . One is not constrained to add bodies to build just arrays. They could be other appendages. 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 arrayed 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 *);