Simulations of Multiple Spacecraft Maneuvering with MATLAB/Simulink and Satellite Tool Kit

McCamish, Shawn B.; Ciarcià, Marco; Romano, Marcello

McCamish, Shawn B., Marco Ciarcià, and Marcello Romano. "Simulations of Multiple Spacecraft Maneuvering with MATLAB/Simulink and Satellite Tool Kit." Journal of Aerospace Information Systems 10.7 (2013): 348-358.
http://hdl.handle.net/10945/50881

JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 10, No. 7, July 2013

Simulations of Multiple Spacecraft Maneuvering with MATLAB/Simulink and Satellite Tool Kit

Shawn B. McCamish,* Marco Ciarcià,† and Marcello Romano‡
Naval Postgraduate School, Monterey, California 93940

DOI: 10.2514/1.35328

A software interface between the MATLAB/Simulink environment and the Satellite Tool Kit Environment is introduced. This research is based on the need for validating model performance and visualizing simultaneous multiple-spacecraft proximity maneuvers for emerging missions. It is common for spacecraft systems to be modeled with MATLAB and Simulink. Furthermore, the software package Satellite Tool Kit is often used for animating and evaluating spacecraft maneuvers. In this research, a MATLAB/Satellite Tool Kit interface was developed to propagate six-degree-of-freedom spacecraft models, compared against Satellite-Tool-Kit-generated ephemeris, and animated for analysis. MATLAB script with necessary formatting is used for Satellite Tool Kit initialization and animation. The MATLAB/Satellite Tool Kit simulation interface allows variations in number, shape, and dimensions of spacecraft. Additionally, numerous model and simulation parameters can be selected and synchronized between MATLAB and Satellite Tool Kit. Furthermore, either predetermined, or randomly distributed, initial spacecraft positions and orientations are permitted by the interface. The paper gives enough details to allow the interested readers to adapt to their needs and further develop the proposed software interface.

I. Introduction

MODEL validation and simulation visualization are critical aspects of engineering analysis and performance evaluation for spacecraft control algorithms [1]. An effective test scenario simulates the environment in which the control algorithm is expected to operate. For spacecraft control system design, it is common for engineers to use The MathWorks, Inc. products MATLAB and Simulink [2,3] for dynamics and kinematics modeling. These models are often independently developed and coded by separate researchers; therefore, modeling discrepancies can be difficult to troubleshoot. This research suggests a method of spacecraft model validation by comparison with the Satellite Tool Kit (STK) spacecraft analysis software [4]. STK, developed by Analytical Graphics Inc., can be used as an orbital propagator for both simulation and emulation of the desired spacecraft models. In addition to spacecraft model validation, STK can be used for detailed animation of spacecraft simulations.

In our research, a multiple-spacecraft six-degree-of-freedom (6-DOF) simulation was developed for the purpose of control algorithm development during close-proximity operations. The simulator incorporates a 6-DOF MATLAB/Simulink numerical model with three-dimensional (3-D) STK visualization. Both the model validation and 3-D visualization are intended to support engineering evaluation during control algorithm development.

Since STK is not as commonly used in engineering fields as MATLAB, the developed simulation interface between the two software programs is of high potential interest to many. In this research, MATLAB-developed spacecraft models were compared and validated based on the STK standard. This paper serves an introduction to the capabilities of a MATLAB/STK simulation interface and gives enough details to allow the readers to adapt and further develop it for their purposes [1]. As a sample case application of the developed software interface, we present the cross validation of a multiple-spacecraft dynamic model and of a novel control algorithm. The paper is organized as follows. Section II introduces the newly developed MATLAB/STK simulation interface, Sec. III reports the sample application of the developed interface for spacecraft modeling verification, and Sec. IV introduces the use of the software interface for multiple-spacecraft model animation.

II. MATLAB/STK Simulation Interface

The developed MATLAB/STK interface takes advantage of STK's standard connection protocol to initialize STK from MATLAB. STK developers have enabled their code to be executed by MATLAB via an application program interface called AgConnect [4]. The connection allows for properly formatted commands to be sent via a transmission control protocol/internet protocol (TCP/IP) port. By exploiting this feature, formatted data, such as ephemeris and attitude data, can be passed between applications. For instance, the interface allows all spacecraft physical characteristics and simulation parameters to be defined in MATLAB and related to STK. Also, STK data can be passed to MATLAB for analysis. The general functions and information flow between MATLAB and STK are shown in Fig. 1.

A. Overview of Spacecraft Dynamic Modeling in MATLAB/Simulink

Initially, generalized spacecraft characteristics were used to model the spacecraft in MATLAB/Simulink. All multiple-spacecraft simulation parameters are assigned in MATLAB and implemented by a single Simulink model. Spacecraft states are concatenated into vectors and passed through the Simulink model. A high-level Simulink spacecraft model is shown in Fig. 2.

Presented as Paper 2007-6805 at the AIAA Modeling and Simulation Technologies Conference and Exhibit, Hilton Head, SC, 20–23 August 2007; received 5 May 2008; revision received 10 December 2012; accepted for publication 23 January 2013; published online 26 July 2013.

*Ph.D. Candidate, ECE Department, 700 Dyer Road; Major, U.S. Air Force; sbmccami@nps.edu.
†Research Associate, Spacecraft Robotics Laboratory, Mechanical and Aerospace Engineering Department; mciarcia@nps.edu.
‡Associate Professor and Director, Spacecraft Robotics Laboratory, Mechanical and Aerospace Engineering Department and Space Systems Academic Group; mromano@nps.edu. Associate Fellow AIAA.

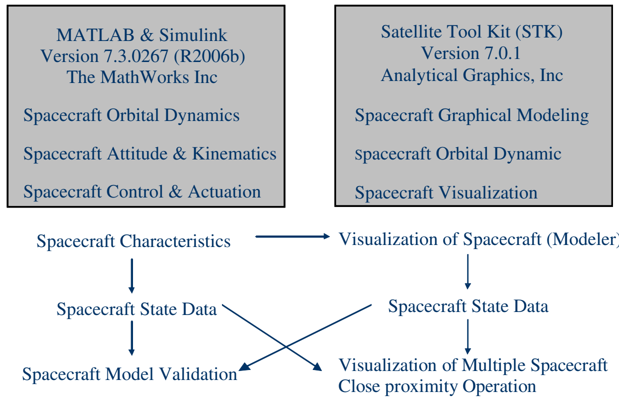

Fig. 1 MATLAB/STK simulation interface overview.

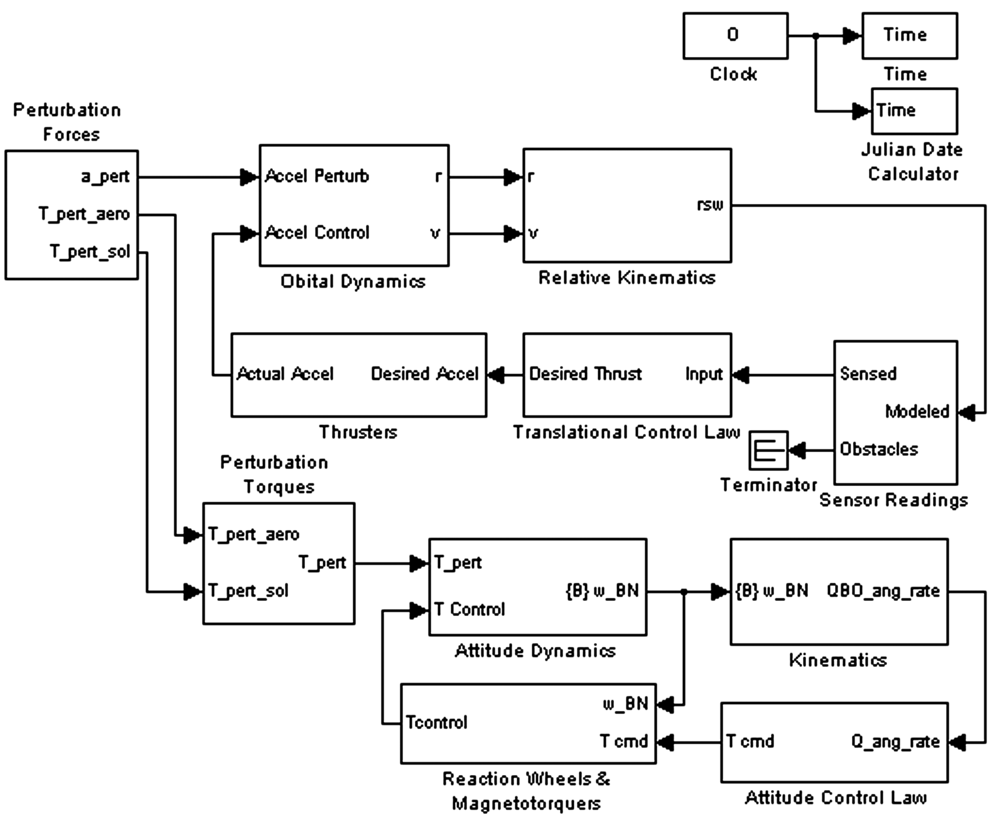

Fig. 2 High-level Simulink model for multiple-spacecraft simulation.

Several parameters must be assigned before multiple-spacecraft simulations can be executed. These parameters can include the number of simulations, initial time, selection of the acting perturbations, number of spacecraft and obstacles, selection of control algorithms, and duration of simulation. For each simulation, a reference start time must be selected. In this research, the Universal Time (UT1) in the format of [year, month, day, hour, minute, second] was used.

The inclusion of perturbations is critical to the fidelity of the 6-DOF spacecraft model. These perturbations forces and torques modify the spacecraft's simple two-body orbital propagation. Variations in the Earth's shape and mass ($J2$–$J4$ coefficients), atmospheric drag on the spacecraft, third-body (sun and moon) forces, and solar-radiation pressure result in disturbances in a spacecraft's orbit [5] and attitude dynamics (gravity gradient, and aero- and solar torque).

Once the number of chase spacecraft is selected, their initial positions must be assigned. These positions can be provided in classical orbital elements, Earth-centered inertial (ECI), or target spacecraft's *RSW* reference frame, as described in Fig. 3. Sample values of simulation conditions are summarized in Table 1. Similar to chase spacecraft, stationary obstacles can also be included in the simulation. The control algorithm development is discussed more in detail in [6,7].

For detailed 3-D simulations, each spacecraft's physical characteristics must be further defined. The spacecraft size, shape, and mass are variable for each simulation. The center of mass of the spacecraft is assumed to be located at the geometric center, and the moments of inertia are calculated based on the spacecraft's shape [8]. Each spacecraft is assumed to have six thrusters (for translational control) aligned along the positive and negative primary body axes of the spacecraft. For the attitude control system, we consider three orthogonal reaction wheels aligned along the primary spacecraft body axis. Selected spacecraft model characteristics for a sample spacecraft are listed in Table 2.

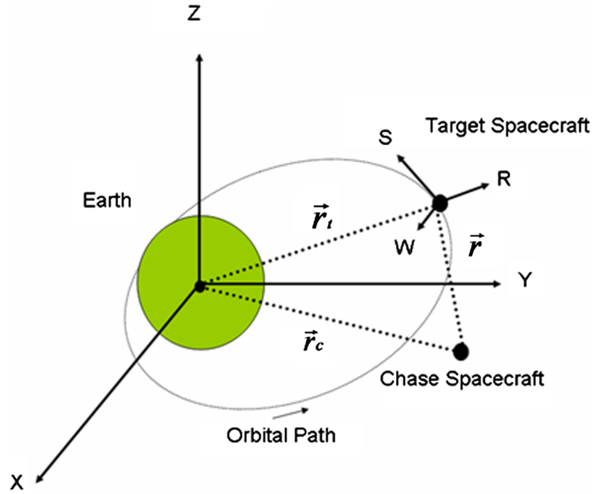

Fig. 3 Relative reference frame.

B. Overview of Satellite Tool Kit

The STK core modules allow for analytical and numerical orbital propagation of multiple spacecraft. An additional module, called AgConnect, allows STK to be used in conjunction with MATLAB [4]. By transferring information between MATLAB and STK, MATLAB-constructed simulation dynamics and kinematics can be verified and visualized. The STK/Connect module provides the means for STK to communicate with applications, such as MATLAB, through the use of a TCP/IP socket. Additionally, the MATLAB/STK interface may use the STK/Communications module to emulate signal propagation delays, frequency and bandwidth limitations, and bit error rates for each sensor and spacecraft [9]. Typical STK users select scenario parameters via the STK graphical user interface (GUI) interface. This allows primary objects,

Table 1 Spacecraft maneuver parameters

	Parameter	Value
Target spacecraft	Min altitude	300 km
	Max altitude	2000 km
Chase spacecraft	Number	1–12
Chase spacecraft	R axis	1.0–1000 m
Initial position	S axis	1.0–1000 m
	W axis	1.0–1000 m
Chase spacecraft	R axis	0 m/s
Initial velocity	S axis	0 m/s
	W axis	0 m/s

Table 2 Sample spacecraft characteristics

	Parameter	Value
Physical properties	Length/width	1.0 m
	Height	1.0 m
	Mass	100 kg
	Moment of inertia X	16.67–50 kg · m^2
	Moment of inertia Y	16.67–50 kg · m^2
	Moment of inertia Z	16.67–50 kg · m^2
Thrusters	Number of thrusters	1–6
	Propellant type	Hydrazine
	Max thrust per axis	1.0 N
	Specific impulse	100–200 s
	Steady-state impulse	200 s
	Min pulse duration	0.05 s
	Min Δv	0.0005 m/s
	Max total Δv	30–120 m/s
	Propellant mass	3% of total
Attitude actuators	Reaction wheels (RWs)	3
	RW max torque	0.055 N · m
	RW max angular momentum	4–12 N · m · s
	Initial RW angular rate	0 RPM[a]
	RW spin axis inertia	0.1426 kg · m^2
	Magnetotorquers	3
	Max dipole moment	100 A · m^2
Docking tolerances	Max axial	±2.0 mm
	Max lateral	±2.0 mm
	Max angular	±0.1 deg

[a]RPM denotes revolutions per minute.

such as spacecraft, to be loaded with desired constraints. Additionally, sensors can be defined and attached to the satellites. The GUI has dropdown menus with layers to define basic characteristic: two-dimensional (2-D) graphic animation, 3-D graphic animation, and related constraints.

Several additional STK-supported modules may be useful for spacecraft proximity maneuver simulations. These modules supplement the core STK visualization environment and assist in the evaluation of mission sequences. In particular, the Astrogator module can be used to support detailed maneuver analysis and operations [10]. Similarly, the Advanced Close Approach Tool (ACAT) module may be useful in collision-avoidance maneuver planning. This STK module allows for the additional situational awareness of any spacecraft or object in the referenced database. The ACAT module enables proximity indications and visual cues for variable close-approach calculations.

C. MATLAB/STK Interface

The MATLAB/STK interface allows overall simulation parameters and spacecraft physical characteristics to be defined in MATLAB and related to STK via the mexConnect MATLAB commands or formatted native STK commands. This interface technique takes advantage of STK's standard connection protocol for initializing STK from MATLAB. Once it is initialized, native STK commands can be called from MATLAB. In addition, formatted ephemeris and attitude files can be passed from MATLAB to STK. Similarly, data can be retrieved from STK by using the `stkReport` command. Getting ephemeris from STK and passing it to the MATLAB engine allows for comparison of independently generated STK and MATLAB spacecraft propagation. Evaluating the results of this comparison allows for validation of a developed MATLAB model based on the STK's High-Precision Orbital Propagator (HPOP). The satellite of interest must be initialized and assigned with HPOP before the propagation parameters and perturbations can be tailored via HPOP commands.

The MATLAB/STK simulation interface code was written in MATLAB (.m file format) with some MATLAB/Simulink files nested within it. These subfiles serve as modular components of the MATLAB/STK interface, allowing easier simulation variation and modification between users.

For the standard MATLAB/STK connection, both MATLAB and STK applications should be loaded and launched. Once launched, STK can be initialized from MATLAB by using the commands `stkInit` or `agiInit`. Information on the path setting established can be found by using `agiGetConfig`. The next step in the connection processes is to open a socket by using `stkOpen`. This configures the path and assigns a socket variable and a MATLAB variable, called `STKError`, for reference. Via the established MATLAB/STK connection, STK can be commanded by using either a select number of mexConnect commands or native STK commands. The command paths for these two methods are slightly different. The mexConnect commands are a limited subset of core MATLAB/STK interface commands that can be found by exploring the MATLAB help menu. These mexConnect commands are all prefixed with `stk`, such as the `stkInit` commands mentioned previously. Since the mexConnect commands are limited, there is a general command that allows native STK commands to be executed from MATLAB. The general execution of STK commands via MATLAB can be conducted by using `stkExec(ConID, 'Command Path Parameter')`. A full listing of native STK commands that can be implemented in this fashion are listed in the STK help menu under the path <Automate/Extend/Integrate>, <Command Listings>, <Alphabetical Listing>. The first step is usually to open a new STK scenario and ensure that any previous scenarios are closed. The initial MATLAB/STK interface code is listed in MATLAB code excerpt 1 (Fig. 4). The `conID` variable serves as a numeric representation of the established connection, and it will be used often in `stkExec` commands. The STK scenario name was determined by a MATLAB variable, such as `CONST.simnum`, which can be selected by the user to represent the number of simulations or scenarios desired. The same name was then used to establish the new STK scenario. The `stkNewObj` command path is establishing the new scenario at the highest-level path level. All subsequent STK objects, such as spacecraft, will be assigned under this current scenario.

The selected simulation initial date and time must be properly formatted and passed to STK. For STK applications, the date and time must be rearranged and passed in the format of [21 Jul 2007 12:30:00.0]. If the MATLAB `datestr` command is used to determine the date and time, then a method of reformatting and passing the date and time to STK is listed in MATLAB code excerpt 2 (Fig. 5).

In the `newpara` variable, the second-to-last number is the animation time step in seconds, and the last number is the highest speed or refresh rate in seconds. The scale of these rates will influence the simulation by changing the animation of the scenario. These values can also be assigned as variables in the `stkConnect` command, where the variable `scen_nam` is used.

Multiple-spacecraft simulations require several spacecraft to be created. The sample code in MATLAB code excerpt 3 (Fig. 6) shows a spacecraft, called target, being created. The visualization option (VO) command is used to call a general spacecraft graphical model that is available to STK. The spacecraft graphical model can be selected from a default list, usually located at `C:\Program Files\AGI\STK 7\STKData\VO\Models\Space`, or a custom user-generated spacecraft graphical model. The development of simple spacecraft graphical models will be discussed in more detail in Sec. IV.

III. Spacecraft Model Verification

High fidelity 6-DOF spacecraft model dynamics can be verified by comparison of MATLAB and Simulink propagation with a custom STK propagator. The STK HPOP was used to ensure spacecraft propagation conditions matched all model variations. Details on numerical integration applied to orbital propagation are provided in [7]. The modeled space environment perturbations, including variations in the Earth's shape and

```
stkInit                          %initializes the STK/MATLAB Interface
remMachine = stkDefaultHost;
delete(get(0,'children'));       %clear any open charts within MATLAB
conID=stkOpen(remMachine);       %Open the Connect to STK
%%Check to see if a scenario is open
scen_open=stkValidScen;
if scen_open == 1
rtn = questdlg('Close the current STK Scenario?');
if ~strcmp(rtn,'Yes')
stkClose(conID)
return
else
stkUnload('/*')
end
end
scen_nam=['Multi_Spacecraft',num2str(CONST.simnum)];
stkNewObj('/','Scenario',scen_nam);
```

Fig. 4 MATLAB code excerpt 1: initializing STK scenario via MATLAB.

```
para_now=datestr(now);
now=datestr(now);
para_now(3)=' ';
para_now(7)=' ';
PARA.TIME=clock;     %sets formatted time for MATLAB
stkSetTimePeriod(para_now,para_now,'GREGUTC');
stkSetEpoch(para_now,'GREGUTC');
stkSyncEpoch;
newpara=strcat('SetValues_','"',para_now, '"',' 0.2 0.1');
newpara(10)=' ';
rtn=stkConnect(conID,'Animate',['Scenario/',scen_nam], newpara);
rtn=stkConnect(conID,'Animate',['Scenario/',scen_nam],'Reset');
```

Fig. 5 MATLAB code excerpt 2: date and time formatting.

```
stkNewObj('*/','Satellite','Target');
stkExec(conID,'VO */Satellite/Target Model Filename "s/c model"');
stkExec(conID, ['SetMass */Satellite/Target Value ',num2str(Mass)]);
stkExec(conID, ['SetMass */Satellite/Target Matrix ',num2str(Inertia)]);
stkExec(conID, ['SetAttitude */Satellite/Target Profile InertFix Quat',…
   num2str(Quaternion0),' "CentralBody/Earth J2000"']);
stkSetPropCart('*/Satellite/Target','HPOP','J2000',tstart,…
   tstopdummy,stepsize,orbitepoch,STATE.ri(1:3)',STATE.vi(1:3)');
```

Fig. 6 MATLAB code excerpt 3: spacecraft object creation.

```
stkSetPropCart('*/Satellite/Target','HPOP','J2000',tstart,tstop,stepsize,…
  orbitepoch,STATE.ri(1:3)',STATE.vi(1:3)');
```

Fig. 7 MATLAB code excerpt 4: spacecraft propagation.

mass ($J2$–$J4$ coefficients), atmospheric drag on the spacecraft, third-body (sun and moon) forces, and solar-radiation pressure, were sequentially evaluated. Table 3 reports the average differences, in terms of position and velocity, of several low-Earth-orbit orbital propagations of 10 spacecraft at various initial conditions. Each orbit was propagated for 30 min and 1 min durations with fixed 1 s step sizes [11].

The HPOP command allows for assignment of both a numerical integration method, such as the fourth-order Runge–Kutta, and an Earth gravitational model (EGM), such as EGM96 [2,3,5]. The HPOP command is of the general form `stkExec(ConID, 'HPOP Path Parameter')`. Once the HPOP is assigned to a spacecraft, the perturbation forces can be further defined. The desired perturbations can be related to the enabling condition on the Simulink model so that the various perturbations can be independently evaluated.

Selection of the perturbation parameter, `CONST.PERT.choice`, activates the desired perturbation code. The coefficients can be synchronized with any particular user-defined model by modifying the .grv file selected. As previously discussed, the variable can be passed into the STK command. For instance, the coefficient of drag, represented by `Cd`, was incorporated into the `Drag` and `SolarRad` command settings.

Once the HPOP settings are completed, the actual propagation can be computed. The initial stop time, `tstopdummy`, is replaced with the final desired propagation time, `tstop`. The spacecraft propagation command is listed in MATLAB code excerpt 5 (Fig. 7), where the propagation parameters of `stepsize, orbitepoch, STATE.ri`, and `STATE.vi` are the same as in the initial propagation command.

The data from STK propagation can be passed to the MATLAB workspace. This will allow any desired postprocessing and evaluation of the data. STK spacecraft state data can be passed in several formats. For this research, the ECI position and velocity were desired. The sample code for passing spacecraft ephemeris is listed in MATLAB code excerpt 5 (Fig. 8), with the variables, prefixed with STATE.STK, arbitrarily assigned. The STK ephemeris data are in column format, with each row representing a numerical integration step. Although not the focus of this research, the user can also pass spacecraft attitude data back to the MATLAB workspace as shown in MATLAB code excerpt box 6 (Fig. 9).

IV. Spacecraft Model Animation

Multiple spacecraft maneuvers can be simulated via MATLAB and animated via STK. The MATLAB initiates an STK TCP/IP connection to send commands over a specified port. The STK scenario is developed by executing STK commands. Spacecraft dimensional models for STK

Table 3 Average differences of MATLAB and STK propagation (1 s fixed step size integration)

Perturbation model	Average difference	30 min	1 min
Simple two body	Position	8.237×10^{-10} m	3.391×10^{-10} m
	Velocity	6.890×10^{-13} m/s	6.082×10^{-13} m/s
$J2$ perturbation	Position	4.650×10^{-1} m	1.102×10^{-3} m
	Velocity	6.908×10^{-4} m/s	5.346×10^{-5} m/s
$J4$ perturbation	Position	8.286×10^{-1} m	1.452×10^{-3} m
	Velocity	1.256×10^{-3} m/s	7.183×10^{-5} m/s
Aerodynamic drag	Position	2.904×10^{-9} m/s	6.291×10^{-10} m/s
	Velocity	2.904×10^{-9} m/s	6.291×10^{-10} m/s
Solar drag	Position	2.075×10^{-3} m	6.656×10^{-8} m
	Velocity	4.267×10^{-6} m/s	3.277×10^{-9} m/s
Third-body effects	Position	5.271×10^{-2} m	6.330×10^{-5} m
	Velocity	1.065×10^{-4} m/s	3.777×10^{-6} m/s
Total perturbation	Position	1.360 m	3.739×10^{-3} m
	Velocity	2.047×10^{-3} m/s	3.357×10^{-4} m/s

```
[stkData,stkName]=stkReport('*/Satellite/Target','J2000 ECI Position
  Velocity');
STATE.STK.time=stkFindData(stkData{1},'Time');
STATE.STK.r(:,1)=stkFindData(stkData{1},'x');
STATE.STK.r(:,2)=stkFindData(stkData{1},'y');
STATE.STK.r(:,3)=stkFindData(stkData{1},'z');
STATE.STK.v(:,1)=stkFindData(stkData{1},'vx');
STATE.STK.v(:,2)=stkFindData(stkData{1},'vy');
STATE.STK.v(:,3)=stkFindData(stkData{1},'vz');
STATE.STK.ECI=[STATE.STK.r, STATE.STK.v];
```

Fig. 8 MATLAB code excerpt 5: STK spacecraft ephemeris data.

```
[stkData,stkNames]=stkReport('*/Satellite/Target','Attitude Quaternions');
STATE.STK.Qbn(:,iqi+1)=stkFindData(stkData{1},'q1');
STATE.STK.Qbn(:,iqi+2)=stkFindData(stkData{1},'q2');
STATE.STK.Qbn(:,iqi+3)=stkFindData(stkData{1},'q3');
STATE.STK.Qbn(:,iqi+4)=stkFindData(stkData{1},'q4');
```

Fig. 9 MATLAB code excerpt 6: STK spacecraft attitude data.

visualization are prewritten in STK Modeler. Once the model is loaded and the scenario is established, data can be passed between STK and MATLAB. For this research, visualization included basic cubic, spherical, and cylindrical spacecraft graphical models. A snapshot of a sample 3-D animation of multiple spacecraft is shown in Fig. 10. Three cubic chase spacecraft are shown converging toward a common target spacecraft, with a spherical obstacle in the background on the right.

A. STK Spacecraft Model

For standard STK model assignment, a STK .mdl file must be available. This file is assigned to the desired STK satellite object using the command listed in MATLAB code excerpt 7 (Fig. 11), where `isi` is a numerical variable that was used to distinguish multiple uses of the same basic model file. This is useful to model multiple spacecraft with the same basic properties. A sample spacecraft model filename may be `mdl_cube1.mdl`. This would assign the STK satellite object named targe, with the model spacecraft model described by `mdl_cube1.mdl`. The VO command can be used to call a general spacecraft model, which is available in STK. The spacecraft model can be selected from a default list, located at `C:\Program Files\AGI\STK 7\STKData\VO\Models\Space`, or a custom user-generated spacecraft graphical model.

1. STK Graphical Model Development

Three simple and distinctive spacecraft models were developed in this research. They are based on simple spherical, cylindrical, or cubic spacecraft designs. Once again, the `fprintf` command serves as our primary formatting tool. In this example, the cubic shape serves as the primary body component with supplemental docking ports and thrusters components. The sample cubic spacecraft model code is presented in

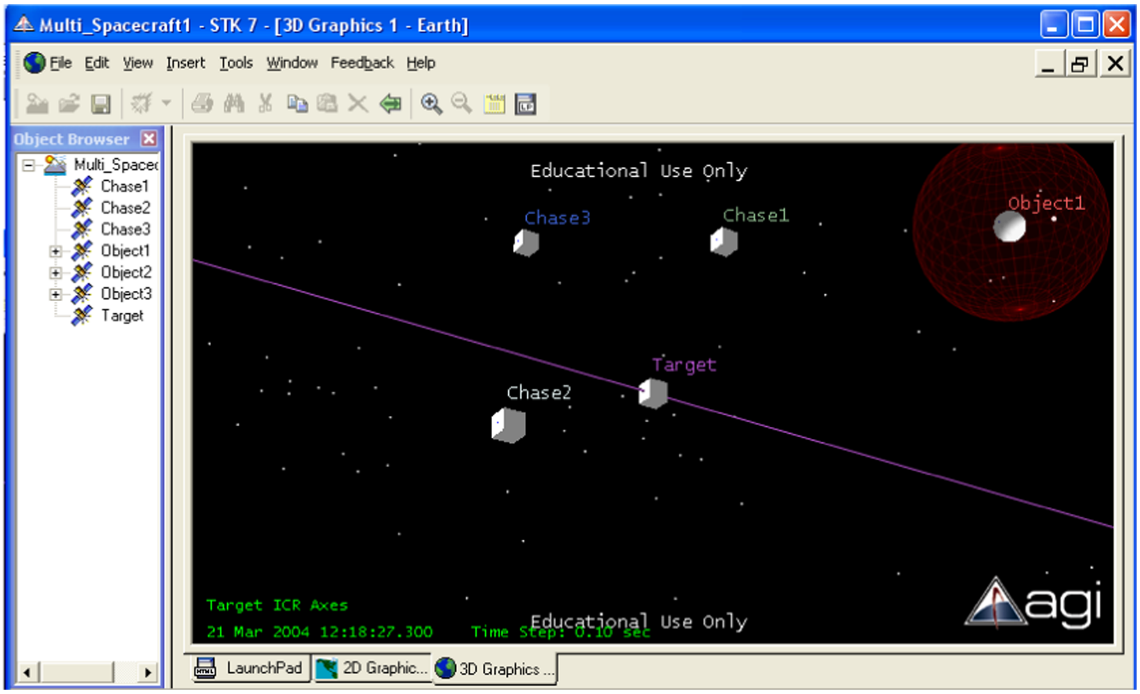

Fig. 10 Sample 3-D STK view animation frame.

```
stkExec(conID,['VO */Satellite/Target Model Filename
  "C:\Userfiles\STK\Scenario\mdl_cube',num2str(isi),'.mdl"']);
```

Fig. 11 MATLAB code excerpt 7: spacecraft model assignment.

MATLAB code excerpt 8 (Fig. 12). The variable L represents user-defined dimensional parameters that can be implemented in the model. In this code, the face component was defined first. Next, this component was translated and rotated as necessary to determine all sides of the cubic spacecraft. Additional components (cone, cylinder, etc.) can also be defined. More detailed components can be generated by combining additional subcomponents.

Once the primary spacecraft components, including the cubic faces, docking ports, and thrusters, are defined, they can be assembled. Each of the six cubic components will make the basic spacecraft shape. Next, a thruster with flame articulation will be centered on each face of the cubic spacecraft. Finally, the desired number of docking ports will be added based on the total number of spacecraft.

2. *STK Three-Dimensional Visualization Options*

Several graphics and VO parameters can be tailored for clear visualization of multiple-spacecraft maneuvers. The primary scaling of the model view is accomplished by selecting the following VO command in MATLAB code excerpt 9 (Fig. 13). `Ratio` is a scaling factor which determines the spacecraft dimensional appearance during the animation, which is chosen to be equal to one in our case. The label and graphic features of the spacecraft can be modified, as in MATLAB code excerpt 10 (Fig. 14).

The orbital path is typically shown for spacecraft propagation. However, for multiple spacecraft, these projections can result in visual confusion. These orbital pass projections can be modified as in MATLAB code excerpt 11 (Fig. 15). In addition to labeling the spacecraft, inclusion of a body axis may be useful for attitude reference. A 3-D spacecraft body reference frame can be added to a model, as in MATLAB code excerpt 12 (Fig. 16). Besides straightforward labeling and graphics modifications, additional sensors may be desired. These sensors may represent actual ranging and communication devices, or useful visual projections for engineering analysis. Each sensor must be uniquely named. In this research, a conical ranging and docking sensor was added to each chase spacecraft. The ranging sensor represents the local sensor view from the chase spacecraft to the target spacecraft, or position. The simple conic range sensor is assigned in MATLAB code excerpt 13 (Fig. 17). First, the sensor is positioned and defined on a spacecraft with an exclusive name. In this example, `Chase` is an alphanumeric label unique to each chase


```
filename=['C:\Userfiles\STK\Scenario\mdl_cube',num2str(isi),'.mdl'];
mdl_cube=fopen(filename,'wt');
mdl_cube=fopen(filename,'wt');
fprintf(mdl_cube,'Component face\n'); %Define front component
fprintf(mdl_cube,'\t Polygon\n');
fprintf(mdl_cube,'\t\t Specularity\t 0.2\n');
fprintf(mdl_cube,'\t\t Shininess\t 25.6\n');
fprintf(mdl_cube,'\t\t Translucency\t 0.025\n');
fprintf(mdl_cube,'\t\t FaceColor burlywood\n');
fprintf(mdl_cube,'\t\t BackfaceCullable\t No\n');
fprintf(mdl_cube,'\t\t Translucency 0.025\n');
fprintf(mdl_cube,'\t\t NumVerts\t 4\n');
fprintf(mdl_cube,'\t\t Data\n');
fprintf(mdl_cube,['\t\t -',num2str(L/2),'\t ',num2str(L/2),'\t  ',
  num2str(L/2), '\n']);
fprintf(mdl_cube,['\t\t ',num2str(L/2),'\t ',num2str(L/2),'\t -',
  num2str(L/2),'\n']);
fprintf(mdl_cube,['\t\t ',num2str(L/2),'\t ',num2str(L/2),'\t ',
  num2str(L/2),'\n']);
fprintf(mdl_cube,['\t\t -',num2str(L/2),'\t ',num2str(L/2),'\t ',
  num2str(L/2),'\n']);
fprintf(mdl_cube,'\t EndPolygon\n');
fprintf(mdl_cube,'EndComponent\n');
fprintf(mdl_cube,'Component left\n'); %Define left component
fprintf(mdl_cube,'\t Rotate 0 0 90\n');
fprintf(mdl_cube,'\t Refer\n');
fprintf(mdl_cube,'\t Component face\n');
fprintf(mdl_cube,'\t EndRefer\n');
fprintf(mdl_cube,'EndComponent\n');
fprintf(mdl_cube,'Component top\n'); %Define top component
..
fprintf(mdl_cube,'Component back\n'); %Define back component
..
fprintf(mdl_cube,'Component right\n'); %Define right component
..
fprintf(mdl_cube,'Component bottom\n'); %Define bottom component
..
```

Fig. 12 MATLAB code excerpt 8: sample cubic spacecraft model.

```
stkExec(conID, 'VO */Satellite/Target ScaleModel Ratio');
```

Fig. 13 MATLAB code excerpt 9: spacecraft model size scaling.

```
stkExec(conID,'VO */Satellite/Target LabelOffsetInPixels Off');
stkExec(conID,['VO */Satellite/Target LabelXYZ 0.0 ',num2str(L),' 0.0']);
stkExec(conID,'Graphics */Satellite/Target Basic Color cyan LineWidth 4.2');
```

Fig. 14 MATLAB code excerpt 10: spacecraft model graphic features.

```
stkExec(conID,'VO */Satellite/Target Pass OrbitLead None');
stkExec(conID,'VO */Satellite/Target Pass OrbitTrail None');
```

Fig. 15 MATLAB code excerpt 11: spacecraft orbital path projections.

```
stkExec(conID, 'VO */Satellite/Target SetVectorGeometry Data
FixVectorAxesScaleToModel On Scale 0.01');
stkExec(conID,'VO */Satellite/Target SetVectorGeometry Modify
"Satellite/Target Body Axes" Show On ShowLabel On Color Gold ArrowType 3D');
```

Fig. 16 MATLAB code excerpt 12: spacecraft model 3-D body reference frame.

spacecraft. Next, the pointing direction of the sensor is determined. The range sensor continually points toward the target spacecraft and adjusts its projection based on the relative distance from the target, represented by `rsw_norm`. This modification of the projection may not be necessary for most sensor applications. Finally, the color of the sensor projection can be modified as desired. Similarly, a docking sensor may be added to the spacecraft graphical model, as listed in MATLAB code excerpt 14 (Fig. 18).

The docking sensor is located at the same position, but it has a much wider cone of 45 deg. The docking sensor points in a fixed direction from the spacecraft body axis, with a limited projection of half of the spacecraft's length. Next, the sensor projection color is selected from a matrix, called `dockc`. These simple sensors are useful in visually representing ranges and fields of view (FOVs) as spacecraft are animated through their dynamics and kinematics. A sample cubic spacecraft with labels and sensor projections is shown in Fig. 19. The three-body axes are labeled "Body *X*", "Body *Y*", and "Body *Z*". The projection on the right side of the spacecraft is the docking sensor projection, and the small line extending to the bottom left is the range sensor. The white protrusions on the top and right are thruster firings.

One additional sensor projection was useful in visualizing spacecraft exclusion regions during close-proximity operations. A boundary sensor projection was placed around regions that spacecraft were intended to avoid. These areas were referred to as obstacles and labeled with `Obstcl`. Spherical projections encompassing these regions can be generated using a conical sensor defined with a 360 deg FOV, such as listed in MATLAB code excerpt 15 (Fig. 20). The spherical sensor projections were made translucent in order to not obscure the view of maneuvering spacecraft. A sample spherical boundary sensor projection about a spherical object is shown in Fig. 21. The sensor projection is the globular grid with the chase


```
stkExec(conID,['Location */Satellite/',Chase,'/Sensor/Range',Chase,' Fixed
Cartesian ',num2str(L/2+0.001),' 0 0']);
stkExec(conID,['Define */Satellite/',Chase,'/Sensor/Range',Chase,'
SimpleCone 0.25']);
stkExec(conID,['Point */Satellite/',Chase,'/Sensor/Range',Chase,' Targeted
Tracking Satellite/Target Hold']);
stkExec(conID,['VO */Satellite/',Chase,'/Sensor/Range',Chase,' Projection
SpaceProjection ',num2str(rsw_norm(1,isi+1)-L)]);
stkExec(conID,['Graphics */Satellite/',Chase,'/Sensor/Range',Chase,'
SetColor cyan']);
```

Fig. 17 MATLAB code excerpt 13: simple conic range sensor for spacecraft model.

```
stkExec(conID,['Location */Satellite/',Chase,'/Sensor/Dock',Chase,' Fixed
Cartesian ',num2str(L/2+0.001),' 0 0']);
stkExec(conID,['Define */Satellite/',Chase,'/Sensor/Dock',Chase,' SimpleCone
45.0']);
stkExec(conID,['Point */Satellite/',Chase,'/Sensor/Dock',Chase,' Fixed YPR
321 0 90 0']);
stkExec(conID,['VO */Satellite/',Chase,'/Sensor/Dock',Chase,' Projection
SpaceProjection ',num2str(L/2)]);
stkExec(conID,['Graphics */Satellite/',Chase,'/Sensor/Dock',Chase,' SetColor
',dockc(isi-1,:)]);
```

Fig. 18 MATLAB code excerpt 14: simple conic docking sensor for spacecraft model.

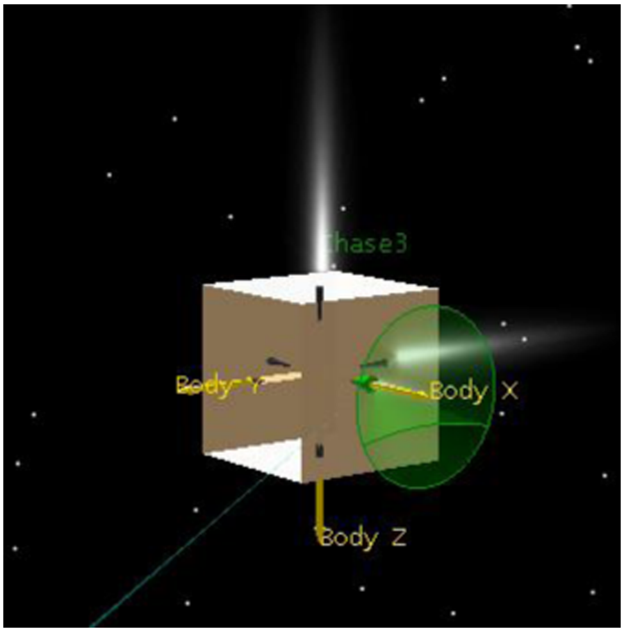

Fig. 19 Sample cubic spacecraft graphical model.

spacecraft in the upper right. These sensors allowed for robust engineering evaluation of the collision-avoidance function of spacecraft control algorithms [6,7,12,13]. From these basic examples, the user can develop a vast array of representative sensors. Sensor projection ranges and colors can be varied based on logical condition. For instance, ranging sensors may be programmed to modify their color as the distance to a target varies. This may allow for quick visual analysis of multiple-spacecraft maneuver performance.

B. Formatting MATLAB Data for STK Files

The visualization and animation of the MATLAB/Simulink data with the desired spacecraft model is a useful tool for engineering analysis and evaluation. In particular, it was vital in the development of a multiple-spacecraft control algorithm during simultaneous close-proximity operations [6,7]. The3-D representation for spacecraft during collision-avoidance maneuvers enabled effective troubleshooting. For instance, the undesired clipping of the corners of chase spacecraft while converging to the target spacecraft during docking maneuvers was easily shown in STK animation. Due to this evaluation, the control algorithm collision-avoidance logic was modified and its performance robustness was improved. Additionally, visualization of the simultaneous multiple-spacecraft maneuvers in STK allows for logical point-of-view changes and animation rate changes.

The animation of the spacecraft can be based on the data generated by STK or the MATLAB/Simulink model. STK-generated data are already in the proper format. However, MATLAB/Simulink data must be properly formatted in order to be used by STK. STK allows for several specific data file formats, such as STK ephemeris and attitude files. These files can be properly written by executing short MATLAB scripts using `fprintf` commands. A sample STK ephemeris file can be created, such as in MATLAB code excerpt 16 (Fig. 22). The STK ephemeris file name and path, in the first line, are arbitrary, as long as it has the proper `.e` suffix. Although, it is useful to save all related STK scenario files in the same folder. The first few fprintf lines are used to establish the desired reference frame and interpolation for the data. The variables `STATE.time` and `para_now` are used to synchronize the data points with the time constraints, as discussed in Sec. I.C.2. The data are formatted into seven columns, represented by time, position vector, and velocity vector. Each row represents an iteration, or step size, of the data. Similarly, a STK quaternion attitude file can be created, such as in MATLAB code excerpt 17 (Fig. 23). The STK attitude file name and path are arbitrary, as long as it has the proper `.a` suffix. This sample STK attitude file is formatted into five columns with time, the three-element quaternion vector, and the quaternion scalar (rotational) term.

C. Sample Visualization

STK scenarios can be made into video files that can offer useful representation of complex situations and maneuvers. For instance, a sample visualization of a docking maneuver between seven cubic spacecraft is shown in Fig. 24. The docking maneuver consists of a target spacecraft at the center of the screen with six chase spacecraft converging from different initial positions. The six chase spacecraft are labeled Chase1–Chase6,

```
stkExec(conID,['Define */Satellite/',Obstcl,'/Sensor/Bound',Obstcl,' Conical
0 180 0 360']);
stkExec(conID,['SetConstraint */Satellite/',Obstcl,'/Sensor/Bound',Obstcl,'
Range Max ',num2str(OBS.DoL)]);
stkExec(conID,['Location */Satellite/',Obstcl,'/Sensor/Bound',Obstcl,' Fixed
Cartesian 0 0 0']);
stkExec(conID,['VO */Satellite/',Obstcl,'/Sensor/Bound',Obstcl,'  Projection
SpaceProjection ',num2str(OBS.DoL)]);
stkExec(conID,['Graphics */Satellite/',Obstcl,'/Sensor/Bound',Obstcl,'
SetColor red']);
stkExec(conID,['VO */Satellite/',Obstcl,'/Sensor/Bound',Obstcl,'
Translucency 95']);
stkExec(conID,['VO */Satellite/',Obstcl,'/Sensor/Bound',Obstcl,'
TranslucentLines On']);
```

Fig. 20 MATLAB code excerpt 15: obstacle region of influence projection.

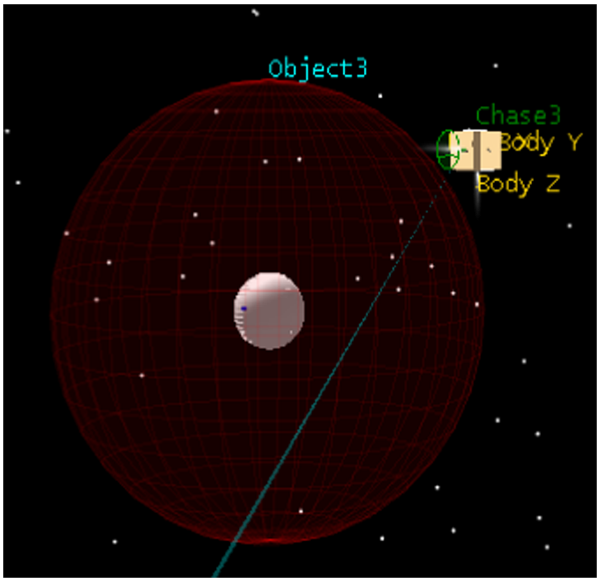

Fig. 21 Sample spherical boundary sensor.

```
filename='C:\Userfiles\STK\Scenario\ephemeris_t.e';
ephemeris=fopen(filename,'w');
fprintf(ephemeris,'stk.v.4.3\n\n');
fprintf(ephemeris,'BEGIN Ephemeris\n\n');
fprintf(ephemeris,'NumberOfEphemerisPoints\t %6.0f\n',length(STATE.time));
fprintf(ephemeris,'ScenarioEpoch\t %s\n',num2str(para_now));
fprintf(ephemeris,'InterpolationMethod\t Lagrange\n');
fprintf(ephemeris,'InterpolationOrder\t 5\n');
fprintf(ephemeris,'CentralBody\t Earth\n');
fprintf(ephemeris,'CoordinateSystem\t J2000\n');
fprintf(ephemeris,'EphemerisTimePosVel\n\n');
data=[STATE.time(:,1),r(:,ici+1:ici+3),v(:,ici+1:ici+3)]';
fprintf(ephemeris,'%16.14e\t %16.14e\t %16.14e\t %16.14e\t %16.14e\t
%16.14e\t %16.14e\n',data);
fprintf(ephemeris,'\n END Ephemeris\n');
fclose(ephemeris);
```

Fig. 22 MATLAB code excerpt 16: formating MATLAB/Simulink ephemeris data for STK.

```
filename2='C:\Userfiles\STK\Scenario\attitude_t.a';
attitude=fopen(filename2,'w');
fprintf(attitude,'stk.v.5.0\n\n');
fprintf(attitude,'BEGIN Attitude\n\n');
fprintf(attitude,'NumberOfAttitudePoints\t %6.0f\n',length(STATE.time));
fprintf(attitude,'ScenarioEpoch\t %s\n',num2str(para_now));
fprintf(attitude,'BlockingFactor\t 20\n');
fprintf(attitude,'InterpolationOrder\t 5\n');
fprintf(attitude,'CentralBody\t Earth\n');
fprintf(attitude,'CoordinateAxes\t J2000\n');
fprintf(attitude,'AttitudeTimeQuaternions\n\n');
data=[STATE.time(:,1),Qbn(:,iqi+1:iqi+4)]';
fprintf(attitude,'%16.14e\t %16.14e\t %16.14e\t %16.14e\t %16.14e\n',data);
fprintf(attitude,'\n END Attitude\n');
fclose(attitude);
```

Fig. 23 MATLAB code excerpt 17: formating MATLAB/Simulink attitude data for STK.

respectively. Chase spacecraft 1 through 3 have obstacles placed along their primary maneuver paths. The three obstacles are labeled Object1–Object3, respectively. The obstacles are 2.0 m spheres with globular mesh region of influence projections.

The solid line through the center of the screen is the targets spacecraft's orbital path. The lighter lines are range sensors from each chase spacecraft toward the target spacecraft. The Earth's horizon can be seen at the bottom of the figure.

This sample animation was developed to the collision-avoidance and close-proximity maneuver capabilities of a novel new control algorithm described in detail in [6,7]. This sample animation has been made available online.

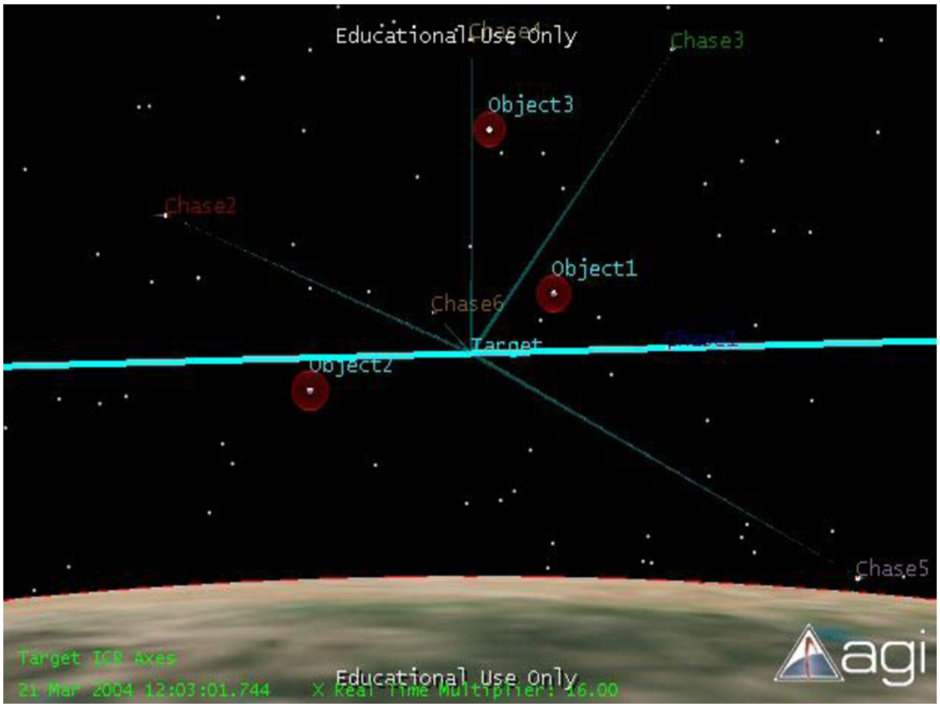

Fig. 24 Sample visualization of multiple-spacecraft maneuver.

V. Conclusions

A MATLAB/ Satellite Tool Kit (STK) simulation interface was developed for spacecraft model validation and visualization. By using the proposed interface, the spacecraft characteristics and parameters, together with simulation results, can be interchanged between MATLAB/ Simulink and STK. In particular, using STK via MATLAB is a versatile and effective method of animating six-degree-of-freedom spacecraft models. The resulting STK animation of spacecraft propagations is useful for engineering evaluation and result presentations.

The proposed interface was used during the development of a multiple-spacecraft control algorithm for close-proximity operations. Model validation gave confidence in the performance results, and visualization allowed for straightforward evaluation. The animation allowed for immediate identification of undesirable performance, such that modifications and improvements could be made to the control algorithm.

Acknowledgments

This research has been supported in part by the U.S. Department of Defense. S. B. McCamish gratefully acknowledges the support from both faculty and students at the Naval Postgraduate School.

References

[1] McCamish, S. B., and Romano, M., "Simulations of Relative Multiple-Spacecraft Dynamics and Control by MATLAB–Simulink and Satellite Tool Kit," *AIAA Modeling and Simulation Technologies Conference*, AIAA Paper 2007-6805, Aug. 2007.
[2] MATLAB version 7.3, The MathWorks Inc., Natick, Massachusetts, 2006.
[3] Simulink version 6.5, The MathWorks Inc., Natick, Massachusetts, 2006.
[4] Satellite Tool Kit, Software Package, Analytical Graphics Inc., Exton, PA, 2006.
[5] Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, 2nd ed., Microcosm Press, El Segundo, CA, 2001.
[6] McCamish, S. B., Romano, M., and Yun, X., "Autonomous Distributed Control Algorithm for Multiple Spacecraft in Close Proximity Operations," *AIAA Guidance, Navigation and Control Conference*, AIAA Paper 2007-6857, Aug. 2007.
[7] McCamish, S. B., Romano, M., and Yun, X., "Autonomous Distributed LQR/APF Control Algorithm for Multiple Small Spacecraft during Simultaneous Close Proximity Operations," *21st Annual AIAA/USU Conference on Small Satellites*, SSC07-XIII-3, Logan, UT, Aug. 2007.
[8] Larson, W.J., and Wertz, J.R., *Space Mission Analysis and Design*, 3rd ed., Microcosm Press, El Segundo, CA, 2004.
[9] Endres, S. M., "Simulation and Emulation of the Space Networking Environment," M.S. Thesis, Dept. of Electrical Engineering and Computer Science, Case Western Reserve Univ., Cleveland, OH, Jan. 2005.
[10] Carrico, T., Langster, T., Carrico, J., Vallado, D., Loucks, M., and Alfano, S., "Proximity Operations for Space Situational Awareness," *Advanced Maui Optical and Space Surveillance Technologies Conference*, Wailea Maui, HI, Sept. 2006, pp. 1–11.
[11] McCamish, S. B., "Distributed Autonomous Control of Multiple Spacecraft During Close Proximity Operations," Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, Dec. 2007.
[12] McCamish, S. B., Romano, M., and Yun, X., "Autonomous Distributed Control of Simultaneous Multiple Spacecraft Proximity Maneuvers," *IEEE Transaction on Automation Science and Engineering*, Vol. 7, No. 3, 2010, pp. 630–643. doi:10.1109/TASE.2009.2039010
[13] McCamish, S. B., Romano, M., Nolet, S., Edwards, C. M., and Miller, D. W., "Flight Testing of Multiple Spacecraft Control on SPHERES During Close Proximity Operations," *Journal of Spacecraft and Rockets*, Vol. 46, No. 6, 2009, pp. 1202–1213. doi:10.2514/1.43563

G. Brat
Associate Editor