Proceedings of The ASME 2017 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2017
August 6-9, 2017, Cleveland, Ohio, USA

DETC2017-67936

UPDATE: ADVANCEMENT OF CONTACT DYNAMICS MODELING FOR HUMAN SPACEFLIGHT SIMULATION APPLICATIONS

Thomas A. Brain
Erik B. Kovel
John R. MacLean

METECS
Houston, Texas 77058
Email: tbrain@metecs.com, bkovel@metecs.com, jmaclean@metecs.com

Leslie J. Quiocho[*]

NASA Johnson Space Center
Software, Robotics, & Simulation Division
Houston, Texas 77058
Email: leslie.j.quiocho@nasa.gov

ABSTRACT

Pong is a new software tool developed at the NASA Johnson Space Center that advances interference-based geometric contact dynamics based on 3D graphics models. The Pong software consists of three parts: a set of scripts to extract geometric data from 3D graphics models, a contact dynamics engine that provides collision detection and force calculations based on the extracted geometric data, and a set of scripts for visualizing the dynamics response with the 3D graphics models. The contact dynamics engine can be linked with an external multibody dynamics engine to provide an integrated multibody contact dynamics simulation. This paper provides a detailed overview of Pong including the overall approach and modeling capabilities, which encompasses force generation from contact primitives and friction to computational performance. Two specific Pong-based examples of International Space Station applications are discussed, and the related verification and validation using this new tool are also addressed. [1]

INTRODUCTION

Human spaceflight operations associated with various programs, ranging from Shuttle to International Space Station (ISS) to Exploration, often involve complex multibody and contact dynamics [1–4]. Examples include spacecraft docking and undocking, payload/vehicle capture and release by robotic manipulators, payload/vehicle berthing and unberthing by robotic manipulators, and future surface operations. While some static scenarios exist, in many cases, crew members must interact with so-called free-flyers or objects in motion, making situations even more difficult. The Pong contact dynamics modeling software was initially developed in response to an increasing need to rapidly simulate these type of operations. The capability was originally conceptualized in 2007 when the NASA Johnson Space Center (JSC) training community was preparing for HTV-1 Exposed Pallet (EP) extraction and insertion operations with the JAXA HTV Unpressurized Logistics Carrier (ULC). This scenario is shown in Figure 1. Unique to this particular robotic operation was the use of the Space Station Remote Manipulator System (SSRMS) Force Moment Accommodation (FMA) and the guide rail and roller wheel mechanism that allows the EP to slide properly in and out of the ULC [5]. The action is similar to opening and closing a desk drawer.

Moreover, since this EP extraction/insertion operation involved the initial use of FMA which was identified for several

[*]Address all correspondence to this author.
[1]This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States. Approved for public release; distribution is unlimited.

FIGURE 1. SSRMS GRAPPLES HTV EXPOSED PALLET

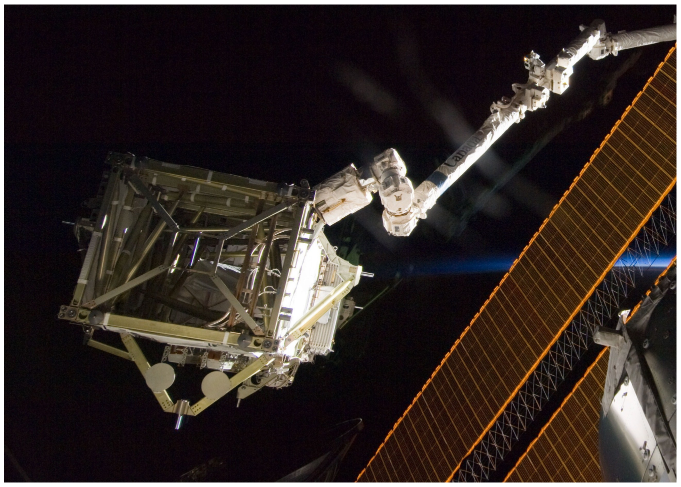


FIGURE 2. SSRMS MANUEVERS S6 TRUSS

robotics operations on the ISS, it was critical to Verify and Validate (V&V) the NASA FMA simulation capabilities against data from the Original Equipment Manufacturer (OEM) of the SSRMS (i.e., the Canadian Space Agency (CSA) and it's prime contractor McDonnell-Detweiler and Associates (MDA)). After developing contact test cases to support this effort that included point-plane tripod geometry and peg-in-hole insertion geometry, an integrated simulation with the SSRMS and EP wheel/rail contact model was developed. The Pong-based model was correlated against a high-fidelity EP contact model which was validated against data provided by JAXA.

It soon became apparent that the simplified contact dynamics approach taken to solving the above V&V requirements could be extended to address more complex contact scenarios. This has resulted in Pong being used for crew and flight controller training of other ISS operations, pre-flight and post-flight analysis such as grappling payloads and free-flyers with the SSRMS Latching End-Effector (LEE), and simulation of ISS Orbital Replacement Unit (ORU) changeout. One such scenario involved a rapid prototype of a particular contingency operation. For ISS Flight 15A, the S6 truss potentially needed to be reberthed by the SSRMS into the Shuttle cargo bay via the Payload Retention Latch Assembly (PRLA) alignment guides (refer to Figure 2). Shortly prior to on-orbit operations, an independent simulation analysis predicted uncommanded motion of the S6 payload during the reberth. To address this anomaly, a Pong contact model of the PRLA alignment guides was created in less than a day to investigate the unexpected behavior. The Pong model demonstrated that not only was the uncommanded motion in the high-fidelity PRLA contact model correct but also that this motion was in fact due to motor stall in one of the SSRMS joints.

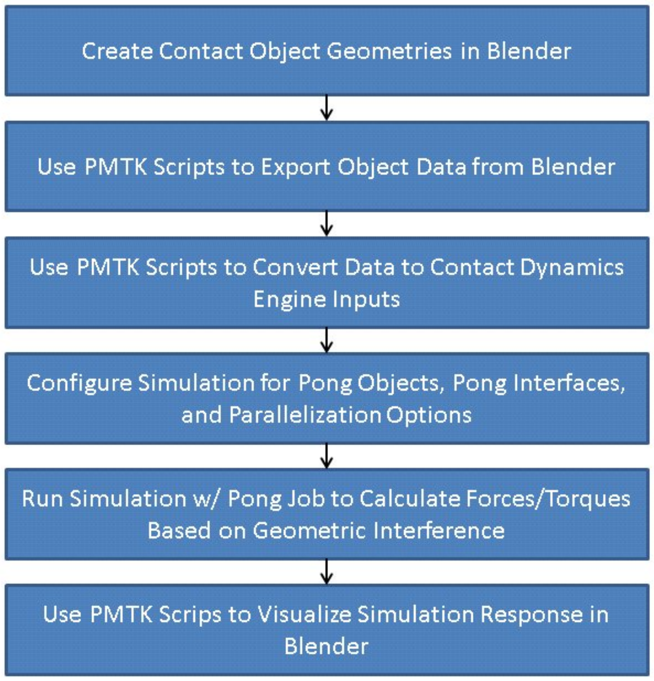
Pong has also been used to prototype contact models associated with ORU changeout, including worksite models of a Remote Power Controller Module (RPCM), the Special Purpose Dexterous Manipulator (SPDM) gripper mechanism, and its associated Micro-Conical Fixture. Using the Pong tool it was possible to reproduce a jamming condition from the RPCM contact interaction with the worksite observed in ground test hardware which was used to replicate an on-orbit anomaly in 2010.

This paper provides a detailed overview of JSC's Pong contact dynamics package including the overall approach and modeling capabilities. Two specific example applications utilizing Pong are discussed along with the related V&V of these examples.

APPROACH

Pong is a software tool that implements interference-based geometric contact dynamics from 3D graphics models. The Pong software consists of three parts: a set of scripts to extract geometric data from 3D graphics models, a contact dynamics engine that provides collision detection and force calculations based on the extracted geometric data, and a set of scripts for visualizing the dynamics response with the 3D graphics models. The contact dynamics engine can be linked with an external multibody dynamics engine to provide an integrated multibody contact dynamics simulation. The workflow for creating and using a Pong model is illustrated in Figure 3.

To build the 3D graphics model, Pong interfaces with the open-source graphics suite called Blender [6]. Using the Blender editing tools, 3D meshes are first constructed to represent the desired contact geometry. Next, a set of in-house developed Python scripts, collectively known as the Pong Modeling Toolkit (PMTK), are used to identify these primitives and set contact dynamics parameters such as stiffness and damping coefficients. Once the geometry is created and contact parameters are set, a parenting tree is constructed. The parenting tree corresponds to the bounding volume tree of the model and each node of the tree



Create Contact Object Geometries in Blender
Use PMTK Scripts to Export Object Data from Blender
Use PMTK Scripts to Convert Data to Contact Dynamics Engine Inputs
Configure Simulation for Pong Objects, Pong Interfaces, and Parallelization Options
Run Simulation w/ Pong Job to Calculate Forces/Torques Based on Geometric Interference
Use PMTK Scrips to Visualize Simulation Response in Blender

FIGURE 3. PONG WORK FLOW

is referred to as a bounding volume, reference frame, or group. Each group can contain other groups or primitives but not both. To distinguish the difference between pieces of the model tree (i.e. groups and primitives) that are static with respect to its parentage and pieces of the model that may move despite the parentage, each group has a flag to specify if it is ”movable” in the PMTK interface. If it is movable, then the group and all of its children are considered fully-independent bounding volume trees from the rest of the model. In terms of dynamic state, a movable group is considered completely detached from its parent and assumes no kinematic constraints or relative motion. Instead, it relies completely on the user's dynamics engine to provide a full dynamic state to the movable group. The root of the entire tree is a special group called an *Object*. An *Object* contains the entire tree despite ”movable” designations and is a movable group itself. While one can separate ”movable” groups into their own *Objects*, continuing to parent them to a top-level *Object* group allows Pong to architecturally treat the entire tree as one contact model. To interface with a Pong model, one must integrate each movable group of an *Object* to an instance of the Interface class. The Interface class provides a means to transfer dynamic state data to Pong and contact forces and moments back to dynamics. The Interface class is what connects each movable group to a user-supplied external dynamics package. Finally, the groups that contain primitives are called leaf groups or leaf bounding volumes. The parentage tree is illustrated in Figure 4. The model

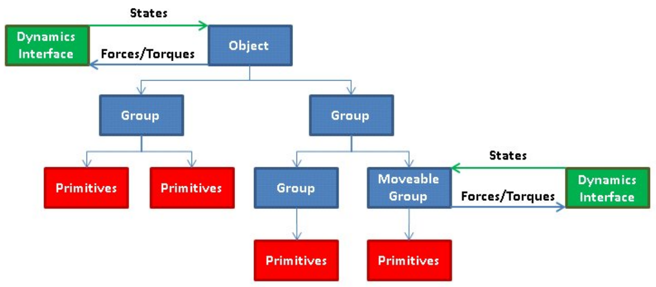


FIGURE 4. PONG PARENTING

developers choice of the contents of the leaf bounding volume has a significant impact on the run-time computational performance of the contact engine. To create an efficient contact model, great care must be taken for specifying the primitives to be contained in each leaf bounding volume. Smart modeling choices based on experience can lead to desired/optimal run-time performance while other poor choices can bog down even relatively simple models. After the geometry, tree, and contact parameters have all been specified, PMTK is used to export a model configuration file which is parsed by the engine contact dynamics engine.

The contact engine allows the creation of multiple *Object* instances that use the same model configuration file. The engine will duplicate the geometry information in independent *Object* instances, each with their own set of interfaces to dynamics. To complete a configuration pertaining to geometry, the user then specifies which *Objects* can contact by calling an API function to name two compatible *Object* instances. If *Object* instances are not paired together, they are not tested in any form or fashion.

As previously mentioned, the Interface and Interface-derived classes allow the integration of a dynamics engine (e.g. MBDyn) with Pong [4]. The Interface class requires the dynamics engine to provide position, orientation, and translational and rotational velocities relative to a common frame. Note that not every contact *Object* instance must have the same coordinate reference if they will never be tested against each other. After processing the dynamic states, the contact dynamics engine detects collisions between the paired contact *Objects* and, when a collision exists, calculates forces and moments for each colliding primitive. The forces and moments are summed up and transformed to each movable group. These forces and moments are placed as outputs in the Interface class to be consumed by the external multibody dynamics engine. The forces and moments are applied at and coordinated in the movable group's reference frame.

For the most part, the algorithms for the primitive contact tests and the bounding volume intersection tests are relatively simple. Most of the current models are made using points, line segments, and convex polyhedra to produce fairly complex mod-

Primitive Type	Point	Plane	Line Segment	Sphere	Triangle	Ellipsoid	Polyhedron
Point	No	Yes	No	Future	No	Future	Yes
Plane	Yes	No	Yes	Future	No	Future	Future
Line Segment	No	Yes	No	Future	Future	Future	Yes
Sphere	Future	Future	Future	Future	Yes	Future	Yes
Triangle	No	No	Future	Yes	Future	Yes	Future
Ellipsoid	Future	Future	Future	Future	Yes	Future	Yes
Polyhedron	Yes	Future	Yes	Yes	Future	Yes	Future

TABLE 1. PONG PRIMITIVE TABLE

els covering a wide range of contact models. By keeping the base algorithms simple, debugging complex models requires little effort to understand the base algorithms that generate the contact forces. The trade-off is that more primitives are required to accurately describe certain geometries. In response to this, there has been significant efforts to improve the computational performance of Pong. Algorithms have been improved to eliminate no-contact situtations rapidly. Code has been optimized to reduce function call costs and temporary variables as well as other improvements. Bounding volumes were introduced and implemented as mentioned above. And, finally, Pong can perform contact tests in parallel on a multi-core computer. The specific capabilities are addressed in the next section.

MODELING CAPABILITIES

The capabilities of Pong can be divided into modelling and performance categories. Primitives and friction models all play a part in determining the forces generated whereas bounding volumes, parallelization, and memory pools improve computational performance.

Primitives are essentially definitions of the smallest pieces of geometry required to do geometric interference detection. Collision detection engines all implement their own sets of primitive types in accordance with their requirements and Pong is no different. Pong has a mixture of dimensionless, 2D, and 3D primitives to choose from, each with their own uses. Each primitive will contain information about its geometry as well as physical properties such as surface stiffness and/or friction parameters. Pong currently has Point, Plane, LineSegment, Sphere, Triangle, Ellipsoid, and Planar Polyhedron primitives implemented. The interactions between these primitive types are listed in Table 1. The architecture of Pong allows for extension of the Primitive class into more primitive types if desired. Pong's design approach allows for primitive definitions to exist and yet not every primitive can contact every other primitive type.

For collision detection, Pong utilizes the polymorphic properties of the ContactTester class. A contact tester is the actual collision algorithm between two primitives and is also a container class for the results of the collision. A contact tester must be implemented for each paired type allowed to contact. Any primitive type pairs not implemented in a contact tester will simply be ignored. For example, a Point has contact testers for Point-Plane, Point-Polyhedron, Point-Sphere, and Point-Ellipsoid. Any contact object containing a point will be tested against the polyhedrons in another object. However, if the second object contains both a polyhedron and a triangle, only the polyhedron will be tested against the point. While this requires the user to understand the various primitive types that can contact and explicitly decide what to use in the model, it also gives flexibility to implement primitive types and their corresponding collision algorithms without having to invent algorithms that do not make sense. For instance, a point-triangle algorithm is not practical because the only collision that can be detected is if the point is on the triangle itself. No penetration depth can be determined to generate forces and one has to decide what determines intersection via a numerical tolerance. The Pong contact tester concept allows us to avoid implementing impractical algorithms or dummy code to satisfy the engine's requirements. The current contact testers implemented are: Ellipsoid-Polyhedron, Ellipsoid-Triangle, LineSegment-Plane, LineSegment-Polyhedron, Point-Plane, Point-Polyhedron, Ray-Polyhedron, Ray-Triangle, Sphere-Polyhedron, and Sphere-Triangle.

A major component of generating contact forces is the friction model. There are many techniques for calculating friction forces but very few that can be considered generic for every application. At this time, Pong's friction model is based on a modified reset-integrator bristle model in and along a plane (2-Dimensional) [7]. The friction forces are calculated and applied to each contact point determined in the collision algorithms. While this model is currently implemented as an internal fixed structure that cannot be changed, there are plans to make the friction model class extensible for the situation where the default model is insufficient for the desired application.

Bounding volumes are a very common and useful method to reduce the number of calculations required to determine contact [8]. The approach for bounding volumes is to encapsulate portions of geometry with a very simple geometric shape. By doing so, one can rely on much faster intersection testing of the simple geometric shapes to eliminate primitives that are not close enough to each other to execute the much slower complicated collision algorithms. Perhaps the most common bounding volumes are the Sphere and the Axis-Aligned Bounding Box (AABB), both of which Pong provides. Similar to the primitive and contact tester class, the bounding volume class can be extended for more bounding volume types. Pong currently has AABB, Sphere, and Oriented Bounding Box (OBB) types.

Similar to contact testers, Pong uses bounding volume testers to implement intersection tests for two bounding volume types. This is where Pong diverges from a majority of the contact dynamics engines. Typically, bounding volumes in a contact engine are all of the same type. AABB is possibly the most common. Pong allows one to mix and match bounding volumes for better modelling choice and control. For example, a model with a sphere bounding volume will be tested against an AABB bounding volume on the opposing object as well as AABBs on both sides. In order to accomplish this, the Pong engine must make no assumptions about the nature of the bounding volume types or the bounding volume tree associated with it. This means that Pong cannot take advantage of specific bounding volume tree structures (e.g., a binary tree) and there is a slight performance penalty for the logic required to determine which bounding volumes types are in need of testing. However, the flexibility is well worth the cost as one can make design choices for choosing the best bounding volume according to the geometry of the specific model. Like contact testers, a bounding volume definition may exist but not include an algorithm for determining intersection between two types. Unlike contact testers, incompatible bounding volume types are not ignored and a run-time error is provided if an incompatibility is found.

Pong is also capable of executing its algorithms in parallel by leveraging off of a small NASA JSC developed software package called Critical Threads (also known as CThreads). Basically, CThreads uses a combination of spin loops, triggers, and polymorphism to perform high-frequency parallel phases while avoiding the cost of sharing a CPU with another process or thread. Utilizing CThreads, Pong is able to split up the bounding volume testers and contact testers among the various threads, performing intersection and collision detection repeatedly until all the testers required for the dynamic step are complete. After parallelization is finished, the contact testers are looped over serially to sum up all of the forces and apply the results through the interface to dynamics.

As discussed previously, the contact tester class is also a container class for the forces due to contact generated from two primitives. This was done so that each primitive pairing between two objects is an independent calculation and container from all the other primitive pairs. As one might realize, this implies that a contact tester must exist for every possible primitive pair that can contact, however, if every possible primitive pair is allocated, memory usage becomes a severe problem for intermediate to large models. If contact testers are allocated during run-time, the cost of multiple calls to allocate (malloc or new) and deallocate (free or delete) them will negatively impact realtime performance. To solve this problem, Pong utilizes a common technique to reduce this overhead called a memory pool. A memory pool allocates large chunks of memory at a time and provides pieces of it to its users. Some memory pools are fixed size where others may grow as needed. Pong uses a variable size memory pool to dole out smaller chunks of memory for the allocation of contact testers. Rather than hundreds of individual allocation calls per contact tester, there are vert few depending on the number of primitives in an object and the number of bounding volumes that intersect at any given point in time. Using memory pools allows Pong to keep its memory usage small while adding little overhead for run-time allocations.

EXAMPLE APPLICATIONS

Pong contact models have been created for specific and general space-related systems for both analysis and training applications at the NASA JSC. These include the previously mentioned Shuttle and ISS examples (i.e., HTV EP, SSRMS LEE, PRLA, and RPCM) as well as the Common Attach System (CAS) and Cosmic-Ray Energetics and Mass investigation (CREAM) payload. To address Exploration Program vehicles and future concepts, Pong has been used for an International Docking Standard (IDS) interface for Orion Multi-Purpose Crew Vehicle (MPCV), a rover driving on a surface/rocks, and a hexapod robot walking on a surface. The two most notable and detailed applications of Pong to date are the SSRMS LEE and the Common Berthing Mechanism (CBM), discussed in the following sections.

The SSRMS LEE model was developed to assist robotics analysis teams perform loads analyses of nominal and off-nominal Visiting Vehicle (VV) capture and release scenarios. For this use, the Pong contact model was integrated with a 3-cable snare model which captures and rigidizes various types of Grapple Fixtures (GFs), an electromechanical model which drives the LEE snare and carriage retract mechanisms, the LEE Control Software (LCS), a high-fidelity SSRMS model, and a dynamics package used to propagate the object states. The LEE/GF snare model response during a capture sequence can be seen in Figure 5.

The contact model is comprised of two Pong objects: the LEE interface and the GF interface. Renderings of the LEE and GF Blender models, which were used to create the 3D Pong geometries, and the hardware interfaces they represent are shown in Figure 6 and Figure 7. Since this model was developed for analysis, the geometry consists of a larger amount of primitives than typically seen in training models. The contact engine accepts the inertial states of each interface from the dynamics engine, performs collisions tests, calculates the force/torque at each object, and feeds this information back to the dynamics engine.

The LEE/GF model, including the contact model, has been validated against the OEM's truth model simulation [9,10]. Over a course of six validation exercises with MDA in association with the CSA, confidence has been built in the model such that it can be used for NASA flight analysis. These validations from 2010 to date continue to ensure the NASA LEE model response is consistent with the OEM response. The timeline of these validations can be seen in the LEE Model Validation Timeline table.

In 2012, the Flight Operations Division (FOD) at the NASA JSC began the development of the Training Systems for the 21st Century (TS21) program which supports many ISS operations

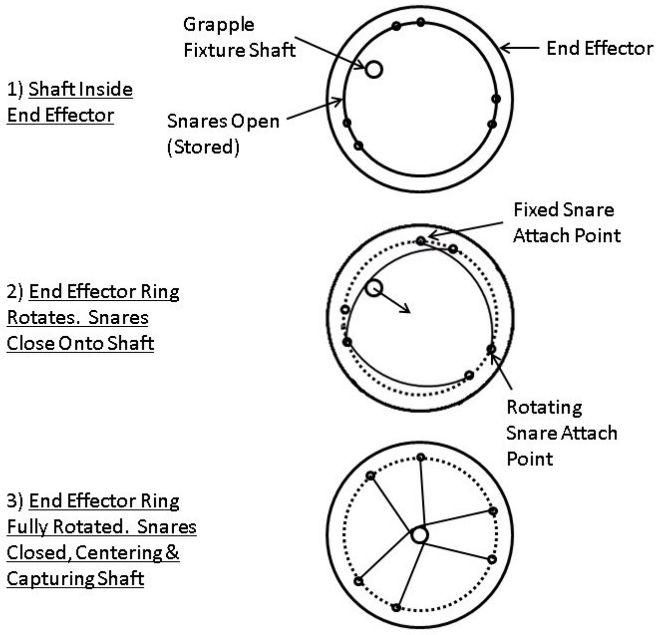

Grapple Fixture Shaft
End Effector
1) Shaft Inside End Effector
Snares Open (Stored)
Fixed Snare Attach Point
2) End Effector Ring Rotates. Snares Close Onto Shaft
Rotating Snare Attach Point
3) End Effector Ring Fully Rotated. Snares Closed, Centering & Capturing Shaft

FIGURE 5. SNARING SEQUENCE

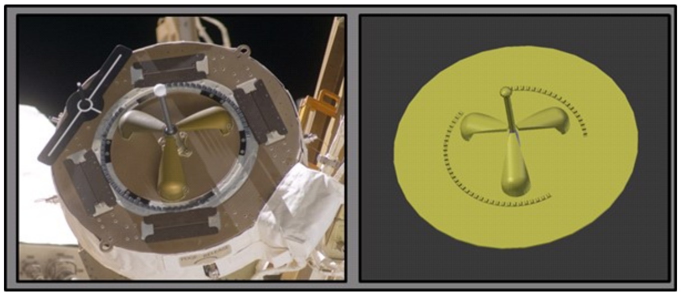


FIGURE 6. GRAPPLE FIXTURE CONTACT MODEL

TABLE 2. LEE Model Validation Timeline

Validation Scenario	Component Validated	Completion Date
Rigidization	Snare and Contact Models	2010
Vehicle Capture	Snare, Contact, and Motor Models	2011
Vehicle Capture/Release	Curvic Coupling Contact	2014
Carriage Push-Off	Carriage Contact and Motor Model	2015
3D Snare Enhancements	Snare, Contact, and Motor Models	2016

[11]. As part of this program, a CBM model was built which, in addition to Pong contact, includes integration with Flight Software (FSW) and motor models to drive the four latches. Each latch is a four-bar mechanism attached to the Active CBM (ACBM) interface with one constraint point. The latches are driven by a motor at the base, and can contact the Passive CBM

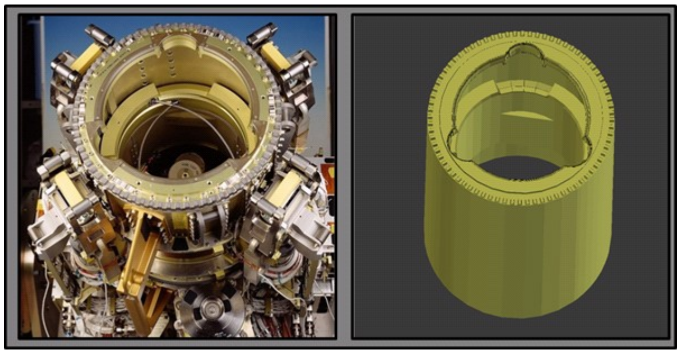


FIGURE 7. LATCHING END-EFFECTOR CONTACT MODEL

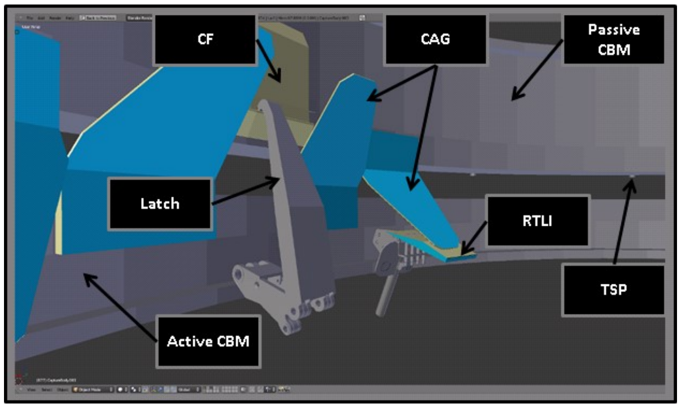


FIGURE 8. CBM COMPONENTS

(PCBM) interface at their tips. Each latch includes Pong contact at the tip and these moveable groups interface with MBDyn which models each link of the mechanism. In addition, the ACBM interface include four Ready-to-Latch Indicators (RTLIs) which are sensors attached to a rotational paddle and translational plunger mechanism. RTLIs let the operator know that the interfaces are within tolerance to command the latches to drive. These mechanisms also include Pong contact under moveable groups linked to the dynamics engine. Additional contact elements rigidly attached to either the ACBM or PCBM include Coarse Alignment Guides (CAGs), Capture Fittings (CFs), Thermal Standoff Plungers (TSPs), Striker Plates (SPs), and Duckhead Bumpers (DBs). These subsystem elements are shown in Figure 8.

The CBM model has been verified against a validated high-fidelity CBM model created at the Marshall Space Flight Center (MSFC) with respect to all contact interfaces besides the latches. The latches were designed such that their motion and motor torques required to drive them match hardware responses as verified by JSC ISS trainers and flight controllers. The major difference between the two CBM models listed here is that only the Pong CBM model is sufficiently efficient such that it can execute in the real-time TS21 ISS simulator.

CONCLUDING REMARKS

NASA's human spaceflight program has numerous contact dynamics operations, including spacecraft docking and berthing, manual or robotic change-out of avionics boxes or equipment, and interaction between robotics end-effectors and their environment during vehicle or satellite servicing. Moreover, future applications will include robotic systems interacting with planet, lunar, or even asteroid terrain, also involving contact. These types of operations have driven contact and multibody dynamics modeling requirements at the JSC that include rapid prototyping, detailed engineering analysis, and crew and flight controller training. Pong has successfully met this challenge by providing a cost effective means to generate contact surfaces from 3D graphics models and visualize the resulting outputs from its contact dynamics engine.

ACKNOWLEDGMENT

The work described in this paper was performed entirely within the Simulation and Graphics Branch of the Software, Robotics, and Simulation Division of the NASA JSC Engineering Directorate.

REFERENCES

[1] Quiocho, L.J., Huynh, A., and Crues, E.Z, 2005, "Application of Multibody Dynamics to On-Orbit Manipulator Simulations", *ASME 2005 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, DETC 2005-85545, Long Beach, CA.

[2] MacLean, J.R., Huynh, A., and Quiocho, L.J., 2007, "Investigation of Boundary Conditions for Flexible Multibody Spacecraft Dynamics", *ASME 2007 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, DETC 2007-35511, Las Vegas, NV.

[3] Ghosh, T.K., and Quiocho, L.J., 2013, "Development and Evaluation of an Order-N Formulation for Multi-flexible Body Space Systems", *EUROSIS 11th Annual Industrial Simulation Conference*, Ghent, Belgium.

[4] Huynh, A., Brain, T.A., MacLean, J.R., and Quiocho, L.J., 2016, "Evolution of Flexible Multibody Dynamics for Simulation Applications Supporting Human Spaceflight", *ASME 2016 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, DETC 2016-60108, Charlotte, NC.

[5] Japan Aerospace Exploration Agency, 2009, HTV-1 Mission Press Kit (Revision A). See also URL: www.global.jaxa.jp/countdown/h2bf1/pdf/presskit_htv_e.pdf.

[6] Blender Home Page. See also URL: www.blender.org.

[7] Haessig, D. and Friedland, B., 1991, "On the Modeling and Simulation of Friction", *Journal of Dynamic Systems, Measurement, and Control*, 113, September, pp. 354-362.

[8] Erickson, C., 2005, *Real Time Collision Detection*, Morgan Kaufman/Elsevier, Amsterdam, Netherlands.

[9] Shi, J-F. and Ulrich, S., 2014, "A Direct Adaptive Control Law Using Modified Rodrigues Parameters for ISS Attitude Regulation During Free-Flyer Capture Operations", *65th International Astronautical Congress*, IAC-14-C1.4.2, Toronto, Canada.

[10] Ma, O., 2000, "CDT - A Generic Dynamics Toolkit", *31st International Symposium on Robotics (ISR 2000)*, pp. 468-473, Montreal, Canada.

[11] Williams, C., 2012, "Plan, Train, Fly (21st-Century Style)", *Lyndon B. Johnson Space Center Roundup*. See also URL: www.jsc.nasa.gov/roundup/online/0612_rev.pdf.