MEXEC: An Onboard Integrated Planning and Execution Approach for Spacecraft Commanding

Martina Troesch, Faiz Mirza,
Kyle Hughes, Ansel Rothstein-Dowden, Robert Bocchino, Amanda Donner, Martin Feather,
Benjamin Smith, Lorraine Fesq, Brian Barker, Brian Campuzano
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

Abstract

The traditional form of spacecraft commanding is with sequences that specify when commands should execute based on a schedule generated on the ground. Some sequences have control logic and event driven responses to increase flexibility, but it is limited. An approach to increase autonomy is to use goal-based planning and commanding. Using this paradigm, intention and behavior is modeled on board the spacecraft. In this paper we describe MEXEC (Multi-mission EXECutive), a multi-mission, task-based, onboard planning and execution software designed specifically to be used as flight software. As a path to infusion for future flight projects, we describe two experiments performed on the ASTERIA CubeSat and testbed that demonstrate that MEXEC can be integrated and used for spacecraft operations and increase robustness and science return compared to the standard sequences that were being used.

Introduction

The prevalent form of spacecraft commanding is through the use of sequences. As the name implies, sequences generally define a sequence of commands to execute at absolute or relative times, but no information is maintained about why a specific command is needed or why a command is scheduled in a particular way to achieve a desired effect (Gat and Pell 1998). Although some sequences allow for control logic, such as *if* statements or loops, and some even allow for event driven responses (Grasso and Lock 2008), there is limited flexibility in how sequences are executed, which limits the autonomous behavior that can be achieved. Furthermore, when unexpected states or events happen, the spacecraft response is often to go into safe mode, which may prevent the spacecraft from performing future tasks that could have been safe to execute (Gat and Pell 1998). Because the spacecraft is not able to autonomously recover from the unexpected state, scheduled science gains are potentially lost.

An alternative approach is to use task- or goal-based planning and commanding on board to increase autonomous behavior by maintaining intentions and effects on board the spacecraft. A goal can be described through tasks, which model spacecraft behavior and constraints, where the behavior is the expected change of state by executing the task and the constraints are the states that are required to successfully carry out the task. A set of tasks makes up a task network. The tasks are activities to be accomplished and the modeled behavior and constraints on each task enforce establishment and protection of required states. An onboard planner can use a task network with the most up-to-date state information to generate conflict-free schedules and effectively use time and resources, while an executive can perform real-time constraint checking on executing tasks. In case any unexpected events occur, the planner can re-plan a new schedule, and the executive can continue to execute any tasks that are safe to do so, based on constraints defined in the network.

We follow this approach in MEXEC (Multi-mission EXECutive), a multi-mission, goal-based, onboard, integrated planning and execution software that uses task networks (Verma et al. 2017; Troesch et al. 2019). With MEXEC, since an operator specifies a goal where the intent is maintained through a network of tasks, it is possible to increase science return and improve robustness compared to sequences, as well as respond to anomalies without safing the spacecraft.

Although increased autonomy is an enabler for future missions where human-in-the-loop commanding is not possible, it is important to methodically prove out autonomous capabilities and allow users to gain confidence in new software. To that end, we have developed MEXEC in a systematic way to be flight-ready, including performing incremental flight and testbed experiments on the Arcsecond Space Telescope Enabling Research in Astrophysics (ASTERIA) CubeSat.

ASTERIA was a 6U CubeSat developed as a collaboration between the Jet Propulsion Laboratory (JPL) and the Massachusetts Institute of Technology (MIT). The flight software was built in F' (Bocchino et al. 2018) and nominal operations were performed with sequences. It was deployed from the International Space Station on November 20, 2017 for a 90-day prime mission to demonstrate precision photometry technology. It successfully achieved all of its primary mission goals of pointing stability, thermal stability, and photometric capability (Smith et al. 2018). Af-

ter that, ASTERIA had three extended missions, the third of which focused on using ASTERIA as a "testbed in the sky" for technology demonstrations (Fesq et al. 2019). As part of those demonstrations, MEXEC was flown to demonstrate nominal science operations using task networks. A second in-flight experiment was planned, but was converted to a testbed-only demonstration when the spacecraft stopped communicating three months shy of the expected end of mission (Jet Propulsion Laboratory, California Institute of Technology 2020).

In the rest of this paper, we will begin by reviewing some related work, then describe MEXEC and two experiments that were performed - one on the ASTERIA spacecraft and one on the testbed. We will finish with a look ahead to future work and present conclusions.

Related Works

Sequencing languages such as the Virtual Machine Language (VML) can provide higher-level programming capabilities compared to traditional sequences. This was demonstrated successfully on Spitzer (Peer and Grasso 2005), where less conservative schedules could be generated by taking advantage of relative timing instead of conservative estimates, which allowed pre-identified observations to be added from a list whenever possible. However, the resulting execution is still based on a sequence and no projection or re-planning is performed.

A move toward more intelligent, autonomous systems through goal-based commanding instead of sequence-based commanding was seen on Remote Agent, which was demonstrated on Deep Space One (DS1) (Muscettola et al. 1998). Remote Agent flew on DS1 for 48 hours and used model-based programming and onboard search with goal-based, closed-loop commanding to achieve more autonomous behavior.

The Autonomous Sciencecraft Experiment (ASE) on Earth Observing One (EO-1) (Chien et al. 2005) also demonstrated higher-level commanding based on goals with robust execution by responding to events and anomalies at execution time. ASE used the Continuous Activity Scheduling Planning Execution and Replanning (CASPER) (Chien et al. 2000) software to perform onboard planning and communicate with an execution system. ASE flew for over 12 years (Ellis 2017), aiming to maximize science return by taking data, processing it to create new goals, and replanning on board.

The idea of a spacecraft executive has been used in other software on many other missions, such as the Spacecraft Commanding Language (SCL) on ASE and TACSAT-3 (Chien et al. 2005; Mackey et al. 2010), Remote Agent Executive on DS1 (Pell et al. 1997), and VML on Spitzer and Dawn (Grasso and Lock 2008). MEXEC shares many characteristics with the planner and executive on Remote Agent as well as CASPER on ASE, such as having a separate planner and executive and performing constraint-based, periodic planning in a limited scheduling window. MEXEC also inherits the use of a commit window from CASPER (described later in this paper). However, one of the major differences is that MEXEC provides a more consistent representation of behavior modeling at planning and execution time and has a tighter coupling between the planner and executive.

Looking ahead to future missions, the M2020 Onboard Planner (Rabideau and Benowitz 2017) and MEXEC have many similarities. In fact, they use the same timeline library to search for valid intervals to place tasks during planning. Both use a planner to schedule tasks and an executive to perform real-time constraint checking. However, MEXEC is multi-mission, whereas the Onboard Planner has a specialized planning algorithm with limited choice points or options during scheduling including what states can be represented in the timeline library.

MEXEC

MEXEC is a multi-mission, onboard planning and execution software that uses task networks to generate and execute conflict-free schedules to achieve goals. It consists of three modules: a *planner*, an *executive*, and a *timeline library* which is used by the planner to search for valid intervals to place tasks. MEXEC was designed specifically to be used for flight software and to have a consistent, cooperative design between the planner and executive. It also allows operators to control the level of autonomy as is appropriate for their needs.

Figure 1 shows a diagram of the MEXEC modules and their interactions with the ground and other flight components. MEXEC assumes that there is a *state database* that reports system state and a *command dispatcher* that dispatches commands. The ground sends a task network to the spacecraft, which is read by the planner. The planner schedules the tasks with the help of the timeline library and the system state from the state database. Once sufficiently close to the start time of a task, the planner passes it on to the executive, which performs real-time constraint checking to ensure safe execution of the task. Task execution updates are sent from the executive to the planner to keep the planner informed in case re-planning is necessary.

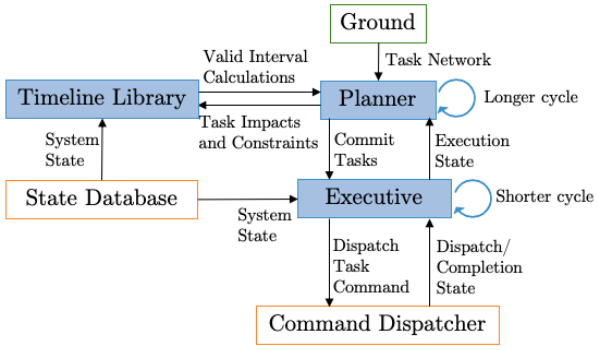


Figure 1: Diagram of the interactions between components in MEXEC. MEXEC components are shown in blue.

Tasks and Templates

Tasks represent a desired change to the system. Besides a unique ID and a name to identify them and a ground-specified priority to inform scheduling, tasks include a com-

mand to execute (which may be no operation or even be a sequence) as well as expected effects of executing the task based on spacecraft behavior (*impacts*) and the conditions required for execution (*constraints*). Impacts and constraints can be defined at the beginning (*pre*), during (*maintenance*), or at the end (*post*) of a task. Conditions to set limits on task deviation at execution time, such as timeouts for waiting on constraints to be met or when to skip tasks, can be defined as *control conditions*. In the case of execution failure, tasks can also specify *contingencies*, which are actions to take on the task network by the planner, such as adding new tasks from templates. Templates are tasks that are available to the planner, but that are not explicitly requested to be included in the schedule. An immediate response by the executive in the case of task failure can also be defined in the form of a command to execute.

Timelines

The timeline library supports planning by calculating valid intervals for tasks. Many of the concepts outlined in (Chien et al. 2012) for common capabilities for different timeline representations are supported by this timeline library, including support for analyzing schedules and constraints for state and resource timelines, which we separate into atomic, state, claimable, cumulative and cumulative rate timelines. Each resource or state that is referred to in a constraint or impact is represented as its own timeline and can either portray values reported by the spacecraft system or values managed internally by MEXEC, also known as *internal states*. Impacts from tasks are placed on the timelines to change the timeline value and are projected into the future to predict future state to compare to constraints and determine conflicts. Impacts can assign a value, a change in value, or a change in rate to a timeline. The aggregation of impacts and their projections provide a timeline *result*, or expected value, at any given time. A constraint is in conflict if the result during the constraint is not within the prescribed values. System state updates from the state database are also applied to the timelines to keep the latest spacecraft values synchronized. This allows the planner to perform its perioidic constraint checking on all states with up-to-date values.

During scheduling, the planner uses the timeline library to look for valid intervals in which to place a task. A valid interval is one in which a task can be placed without creating conflicts. Any previously scheduled tasks have their impacts and constraints placed and projected on the relevant timelines. To find valid intervals for a task, the timeline library systematically places the task on the timelines around result times and checks for conflicts.

Timeline results are also adjusted when feedback from the executive notifies the planner that task execution has deviated from its originally scheduled time. In this case, the task's impacts and constraints are moved to reflect the actual execution of the task.

Additional information about the timeline library can be found in (Rabideau and Benowitz 2017) as described for the prototype for the Mars 2020 Rover.

Planner

The planner, with the valid interval calculations provided by the timeline library, generates and maintains conflict-free schedules given an input task network. Every planning cycle, the planner takes the following actions in order:

1. Commit tasks to the executive
2. Consolidate impacts
3. Schedule tasks (configurable)
4. Repair conflicts (optional)
5. Optimize the schedule (optional)

The planner makes use of various timing windows to inform decisions. The first is the *plan process interval*, which defines the frequency at which the planner cycle is run. This interval must be longer than the worst case duration of the planner cycle. The plan process interval also influences the duration of the *commit window*, which is used to determine which tasks should be passed on, or committed, to the executive, first defined in (Chien et al. 2000). Any scheduled task with a start time before the end of the commit window should be committed. Since tasks are only committed at the beginning of the planning cycle and tasks should be committed before their start time, the commit window must at a minimum be as long as the plan process interval.

Another important window is the *scheduling window*. Unscheduled tasks with a preferred start time before the end of the scheduling window are considered for scheduling by the planner during the planning cycle, allowing the planner to work on a subset of the problem at a time. The planner then schedules the tasks within the *plan horizon*, which starts after the commit window, to prevent tasks from being scheduled before they can be committed, and extends to the *plan end time*, which defines the time before which all tasks in the task network should be scheduled. During scheduling, the planner checks for conflicts during the *conflict checking window*, which starts after the commit window and goes until the maximum time allowed on timelines. The conflict checking window starts after the commit window since any tasks already committed to the executive should not be changed by the planner since they may have started executing already, and any conflicts that occur at execution time will be handled by the executive. A summary of the timing windows and intervals is shown in Figure 2.

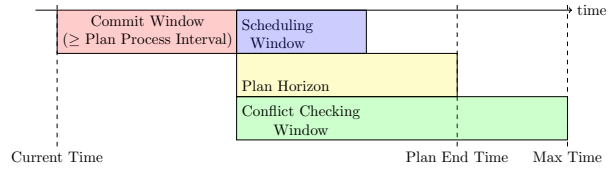


Figure 2: Diagram of the timing windows and intervals used by the planner.

After committing tasks, the planner consolidates impacts on the timelines. This step reduces the memory necessary to represent a timeline by combining past impact results into a

single result. The memory footprint of MEXEC in general is adjustable based on various configuration values.

Next, the planner schedules any tasks in the scheduling window. One feature of MEXEC is that it is easy to plug in different scheduling algorithms that are most appropriate for the application in which it is being used. In fact, we used different scheduling algorithms for our two experiments.

If enabled by the configuration, two additional capabilities can be used after scheduling. The first is to find conflicts and attempt to repair them using iterative repair techniques, and the second is to optimize the schedule by shifting tasks closer to their preferred times, if possible. Neither of these capabilities was used for our experiments, but are supported by MEXEC.

The planner also handles incoming information when it is not executing its repeated cycle. This includes reading task network files, handling state updates from the state database to be placed on timelines (as mentioned in the Timelines section), and interpreting task execution updates from the executive. Task networks are uploaded to the spacecraft from the ground in the form of a binary file. The planner reads the file and performs input validation before storing the information to ensure that the incoming timelines and tasks will fit within the available memory and satisfy any assumptions made by the tasks. Both state database updates and task updates are used by the planner to update timelines, which provide information to the planner in the case that re-planning is required. Task updates also notify the planner of task completion status, including if and why a task failed. This information is used to determine if any contingency actions need to be taken.

Executive

The executive takes care of task execution and handles adjustments needed for execution deviation. The MEXEC executive converts constraints and control conditions into boolean expression trees to be evaluated at each stage of task execution on a per task basis. The evaluation is based on system state reported from the state database or from internal state propagation. Each transition to a new stage and any faults are reported to the planner.

In the executive, all constraints are monitored separately for each task. Pre constraints, including the start time of the task, must be achieved before the task will start. If the constraints are not achieved within a defined timeout interval, the task is considered failed. Once the constraints are achieved, the executive dispatches the command associated with the task and starts monitoring any maintenance constraints in the task. The executive expects a command response to indicate that the command was dispatched and completed successfully. If the command response comes back with an error, or the command response does not come back within a defined timeout interval, the task fails. Additionally, if any of the maintenance constraints do not hold true before the end of the task, the task fails. If the command response returns with success, and the maintenance constraints hold, the task completes with success after the duration of the task has been reached.

Experiment 1

The goal of the first experiment was to demonstrate that MEXEC could be integrated with the existing ASTERIA flight software and that it is sufficiently mature to fly and be used for operations. To show this, MEXEC was used to replicate the behavior of ASTERIA's standard sequences. A *standard sequence* is a binary file that can be run onboard by the ASTERIA sequencer. It is generated from an *input sequence*, which is simply a textual specification of a sequence of spacecraft commands to be executed. This input sequence is also used to generate binary task networks that are run by MEXEC and behave like the standard sequences generated by the same input. Although using MEXEC to replicate standard sequences does not exercise the majority of MEXEC's capabilities, it is an important, incremental step in proving out the MEXEC software.

Scenario

The ASTERIA sequencer supports absolute and relative timed commands. An absolute timed command will start no earlier than the given absolute time and a relative timed command will start no earlier than a specified relative offset from the previous command. Both command types may start late if previous commands run long.

For the first experiment, we simply performed nominal operations sequences of setting up passes and taking science observations, with the expectation that everything would execute nominally. The major parts of our development included flight software integration, MEXEC flight preparation, and task network design.

Design and Implementation

Several steps were required to perform an MEXEC onboard experiment, the first of which was to integrate MEXEC with F', the flight software used on ASTERIA (Bocchino et al. 2018). This involved designing and implementing several new components to control and interact with MEXEC. Wrapper components were created to send/receive messages for the MEXEC libraries and hook them into ASTERIA's commanding, telemetry, and fault protection. Although this integration required careful design, it showed how MEXEC is readily adapted to fit within software rather than being custom designed. It was decided that as a first step, a conservative approach to off-nominal behavior and fault protection would be implemented. In the case that fault protection is triggered, MEXEC would clear out any outstanding tasks and stop running. For this first experiment, it was expected that all tasks would schedule sequentially; therefore, as a further precaution, if any tasks could not be scheduled or any parts of planning or execution were not nominal, this would trigger a fault response to clear out and stop MEXEC. A state database was also needed to store spacecraft state values to be tracked by the planner and executive, and the Sequencer needed to be modified to send the current task network sequence count to the state database. Finally, a manager component was needed to control timing and the interaction of MEXEC with fault protection.

Careful planning and timing considerations were taken into account when designing the manager component. The

command for loading a task network into MEXEC is relatively straightforward: load the task network file, and when it succeeds, start running the planner and executive at their respective rate groups. To clear and stop MEXEC, a new command was implemented, which can be called either by the ground or by fault protection; however, it is quite complex due to potential timing risks. For instance, if fault protection were to first shut down the planner, a command from the executive could theoretically load another task network and restart the planner. Another risk is if the planner were busy planning, but took an extended period of time to finish and respond to the shutdown command. Cases like this required some verification and validation work, as well as careful design to handle.

Memory limitations were also considered. Obviously, ASTERIA was not designed with MEXEC in mind, and therefore did not have any spare memory for MEXEC, so buffers needed to be shrunk to make room. It was agreed that MEXEC would fit within 2 MB of memory, which limited task networks to 100 tasks or fewer. Since ASTERIA sequences can have thousands of commands, clearly a one-to-one mapping of tasks to commands would not work, so we allowed the task network to issue commands that start standard sequences.

Writing the task networks for this experiment entailed translating input sequences written by the operations team to task networks. Some of the commands were grouped into sequence files which were referenced by tasks in the task network to execute. This required creating new tools to translate and write task networks. For the first experiment, task network writing was a relatively straightforward translation from sequences. One complication was that the ASTERIA commands did not have a defined or documented duration, so the duration of each task was interpreted as the relative delay between the previous task and the current task.

As this first experiment was to replicate ASTERIA's standard sequence behavior, the constraints and impacts modeled in the task networks were simple. Two states, represented as timelines, were used: one to monitor the task sequence counter, the other to monitor whether a standard sequence was currently running. Each task had a pre constraint that the sequence counter be one higher than the previous task, as well as a post impact that increases the sequence counter by one. This enforced a strict ordering of the tasks: they needed to be scheduled and executed in sequential order. In addition, each task had a pre constraint that there was not a standard sequence running. This prevented commands issued by a task network from running concurrently with standard sequences. Figure 3 illustrates how the impacts and constraints relate to each other to enforce the sequential ordering of the tasks. It is clear that the sequence count impact from each task satisfies the constraint for the next task, enforcing the ordering.

One of the properties of the MEXEC planner is that it is built to be multi-mission and therefore made to be easy to plug in different scheduling algorithms based on the needs of the mission. For this experiment, part of our effort was designing and implementing a naive algorithm that scheduled the tasks sequentially. MEXEC's planning capability

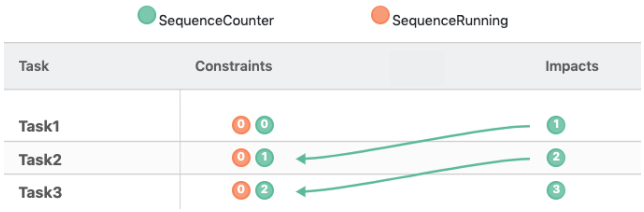


Figure 3: An example of the impacts and constraints used for the onboard experiment showing how they enforce a sequential order. This figure was created using a prototype of a tool developed to visualize the relationship between impacts and constraints in a task network.

was not seriously exercised. Beyond that, significant testing, input validation, static analysis, memory optimizations, speed optimizations, and other changes were made to prepare MEXEC for its first flight experiment. A description of the assurance techniques that were used can be found in (Smith et al. 2020).

Results

MEXEC was demonstrated on board the ASTERIA CubeSat from September 4 - 20, 2019, and was used to perform observations of HD219134, New York City, the Moon, and the Vesta asteroid. All uploaded task networks were scheduled and executed as expected, with some caveats. Figure 4 shows an excerpt of the plan for one of the uploaded task networks and the associated effects on the timelines from the planned schedule.

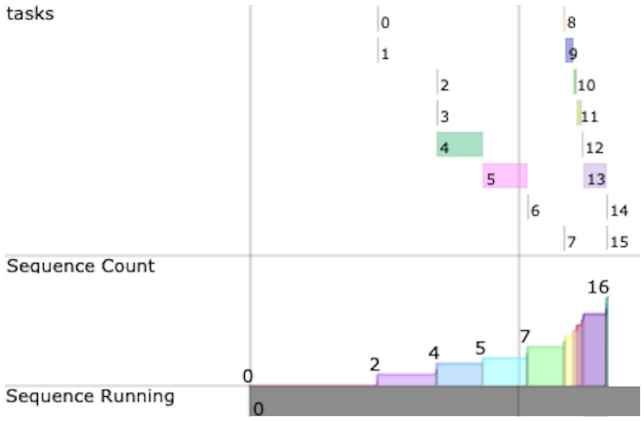


Figure 4: Plan excerpt of an uploaded task network showing effects on timelines.

ASTERIA had some idiosyncrasies that guided both our design choices and the results we encountered. As discussed, each task had a pre constraint that there be no standard sequences running, which might not seem necessary, since all tasks schedule sequentially. However, running a standard sequence in ASTERIA reported completion when the command to run the sequence had been dispatched, not when all commands in the sequence were complete. When the command dispatching the standard sequence completed, the task

network sequence count was incremented, even though the standard sequence had not completed yet. ASTERIA command durations were implied by relative sequence command times. In most cases, this calculation was correct, but not always. If a task had an incorrect duration and a sequence ran longer than expected, nothing prevented the next task from dispatching while the first was still running. This behavior appeared in our testbed runs before the actual experiment, and was caused by the combination of premature command response and lack of concrete duration information about tasks. We prevented this from being an issue with the pre constraint on running sequences so that no task could start executing if a standard sequence was already running.

Another issue encountered during both testing and the in-flight experiment was an imperfect system timer. The ASTERIA flight software clock, like all clocks, has a limit to its precision. The planner and executive use the current system time, truncated to the nearest second, to evaluate whether it is time to commit, dispatch, or otherwise process a task. Once in a while, assumptions made by the planner and executive about the precision of the timer caused a task to dispatch late or commit late by a second. These timing issues and other experiences during the first experiment informed many of the changes and designs for the second experiment and beyond. Taking into account the imprecise timer, all task networks in the first experiment succeeded in testing and the onboard experiment by scheduling and executing in order and starting at or after the scheduled time, like a standard sequence would.

Experiment 2

The goal of the second experiment was to demonstrate MEXEC's ability to react to events both in the planner and in the executive. This required more complex impacts and constraints in the task network as well as using contingencies, control conditions, and responses by the executive. Furthermore, we wanted to highlight a scenario where MEXEC could be useful and improve operations for ASTERIA. The second experiment had to be performed on the ASTERIA testbed due to loss of contact with the spacecraft.

Scenario

The motivating scenario for the second experiment was momentum management. ASTERIA used the fleXible Attitude Control Technology (XACT) from Blue Canyon Technologies (BCT) (Blue Canyon Technologies) as its attitude control system, which has 3-axis attitude control with three reaction wheels and three torque rods as described in (Mason et al. 2017) from its first flight on the MinXSS-1 CubeSat and in (Pong 2018; Pong, Sternberg, and Chen 2019; Smith et al. 2018) on ASTERIA. External torques caused momentum buildup, especially since ASTERIA had a large residual magnetic dipole moment. Too much buildup causes attitude control loss (Pong, Sternberg, and Chen 2019). To protect the spacecraft, ASTERIA fault protection had a momentum magnitude threshold that, if surpassed for some persistence, triggered the spacecraft to go into safe mode and reset. Resetting the spacecraft not only caused any upcoming observations to be lost, but also required reset recovery procedures during the following passes, which meant that the passes could not be used for useful downlink of data. When scheduling observations, operators used ground tools to predict momentum buildup and reaction wheel zero crossings, which led to lower quality science data. If the momentum magnitudes were too high, or the zero crossings were at unfavorable locations, the observations were rescheduled (Pong, Sternberg, and Chen 2019). Observations were scheduled conservatively to avoid tripping a fault and resetting the spacecraft.

MEXEC can be used to autonomously dump momentum and resume safe observations before resetting the spacecraft and losing all upcoming observations and pass downlinks. Since the risk of resetting the spacecraft is reduced, observations can be scheduled less conservatively, which could allow more observations to be scheduled and increase observation quality. MEXEC can monitor the momentum magnitude on board and react by sending commands to dump momentum and skip upcoming observations before the reset threshold is reached. Once an acceptable momentum magnitude has been reached, MEXEC can resume observations.

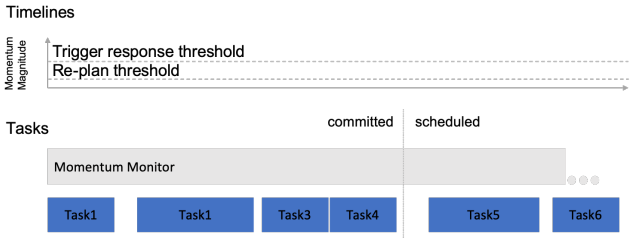
Design and Implementation

In order to monitor the momentum during the scenario, the momentum magnitude, calculated from telemetry values, was added to the state database.

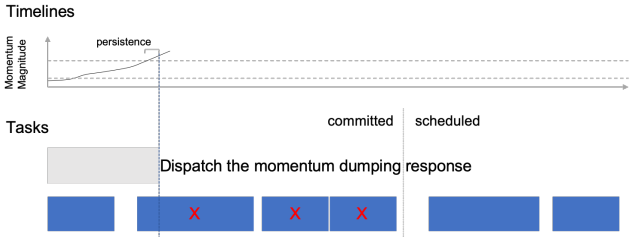
The task network was generated based on the scheduling of three observations of 55 Cancri using the traditional sequence operations ground tools. All tasks had constraints and impacts based on their behavior instead of simply a sequence ordering count, as was seen in the first experiment. Each task started either a single command or a standard sequence of commands. Any tasks that started a standard sequence had a constraint that they not execute while another task that started a standard sequence was running. A distinction was made between tasks that supported science observations and those that were required for setting up and taking a pass. Science observations had the restriction that they respect momentum management, whereas pass tasks should execute regardless of momentum magnitude (unless fault protection had been triggered).

To manage momentum and enforce the restriction on observation tasks, we used a long-running momentum monitor task. This task had a maintenance constraint on momentum magnitude. If the momentum magnitude went above the trigger threshold (which is lower than the fault protection threshold), the momentum monitor task would fail, triggering an immediate response in the executive to start a standard sequence to dump momentum and return to a known state, as well as triggering a contingency response in the planner. The contingency response was to re-schedule a momentum monitor and an observation setup task when the momentum magnitude had dropped to a safe level. All science observation tasks had a constraint that they should execute only if the momentum monitor was running. Therefore, these tasks fail when the momentum monitor fails. Additionally, the science observation tasks had a control condition to skip the task if the momentum monitor was not running. This sequence of

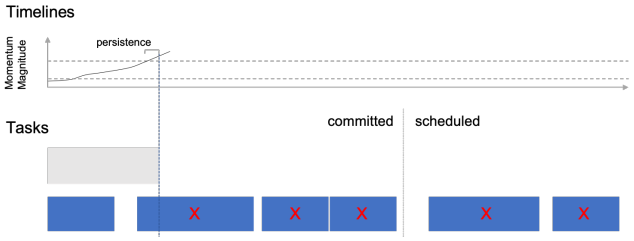
events is shown in Figure 5 with science observation tasks. Figure 6 shows a simplified version of what a subset of the



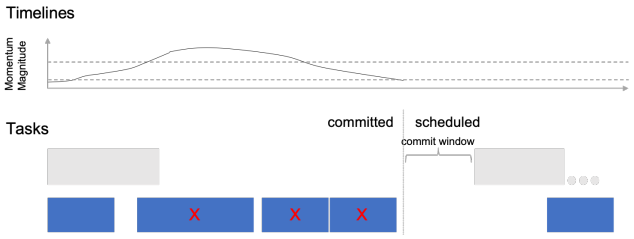
(a) Tasks have been scheduled and some have been committed. The higher momentum magnitude trigger response threshold is shown in relation to the lower threshold required for re-planning.



(b) The momentum magnitude has increased and is above the trigger threshold for some persistence. This violates the momentum monitor task's maintenance constraint, causing it to fail, which then dispatches a command to dump momentum and return to a known state. It is expected that the upcoming tasks will be skipped.



(c) The planner also removes scheduled tasks from the schedule.



(d) When the momentum magnitude has decreased to the re-planning threshold, the planner schedules another momentum monitoring task, as was added by the contingency, and any other tasks that can be scheduled.

Figure 5: This figure shows the sequence of events that happen after scheduling and committing tasks and responding after the momentum magnitude goes too high. The momentum monitor task is shown in grey and all other tasks are science observation tasks shown in blue. The Momentum Magnitude timeline is also shown.

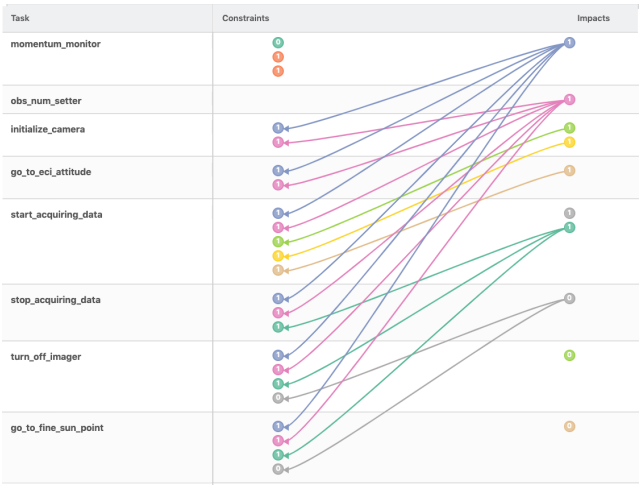


Figure 6: Simplified excerpt of what the constraints and impacts for some of the tasks in the second experiment could look like. This figure was created using a prototype of a tool developed to visualize the relationship between impacts and constraints in a task network.

tasks, constraints, and impacts look like with the relationships between them.

This also highlights some differences from the first experiment. In this experiment, we expected that there would be times when tasks do not schedule or fail during execution, so the fault protection responses had to be updated to reflect this change. The more complex constraints and re-planning needs for this scenario also required a different scheduling algorithm.

Since we could not fly this experiment on the ASTERIA spacecraft, and were instead using the ASTERIA testbed, we used the Real-time Dynamics Processor (RDP), designed and built by BCT, to simulate sensor input to the XACT (Pong, Sternberg, and Chen 2019). We selected thresholds that reflected the behavior simulated by the RDP.

Results

The task network that we ran on the testbed surpassed the momentum threshold four times and recovered successfully each time to resume observations. Figure 7 shows when and how the momentum changed. The times with the grey background show when science observations were scheduled. The momentum threshold was exceeded in between tasks and such that the momentum dumping response returned the momentum magnitude to an acceptable level for all tasks to execute.

The results of this scenario show that the MEXEC software successfully responds to onboard state information and takes action both in the executive and the planner. The executive successfully enforced the momentum monitoring task's maintenance constraint, failed the task when the constraint was violated, and dispatched the specified momentum dumping response. The momentum magnitude successfully decreased after the momentum dumping response. The fact that the task failed was communicated by the executive

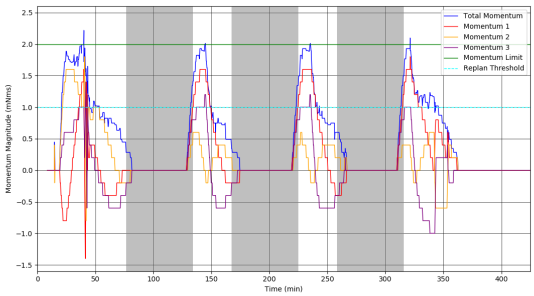


Figure 7: The evolution of the momentum during the experiment. The times of the science observations are shown with the grey background.

to the planner. With this information, the planner added the contingency tasks to the task network. The planner also replanned, removing any tasks that required momentum monitoring from the schedule, only re-scheduling them when the momentum magnitude reported by the state database satisfied the constraints for the momentum monitor. The testbed did not trip any faults and it did not go into safe mode.

If this had been a flight situation where the ground tools had not correctly anticipated the momentum buildup, and the momentum would have exceeded the fault protection threshold, the use of this task network with MEXEC could have prevented the spacecraft from going into safe mode and may have saved some science observations. This also potentially would allow operators to be less conservative with respect to the momentum prediction in the ground tools therefore possibly increasing science quality.

Looking Ahead

We will continue to develop MEXEC to align with the needs of potential future missions to help infusion and adoption of MEXEC. Some additional experiments that are planned on the ASTERIA testbed include experiments that integrate MEXEC with AutoNav (Riedel et al. 2000), which had a standalone experiment on board ASTERIA (Fesq et al. 2019), as well as with the Model-based Off-Nominal State Detection and Identification (MONSID) system (Kolcio 2016), which had planned an experiment on board ASTERIA before communication was lost. MEXEC is also being integrated with the Robotics Operating System (ROS) as part of a prototype for the Europa Lander Autonomy concept (Dooley 2019; Wang et al. 2020). There are many ways in which we plan to expand the capabilities of MEXEC for these tasks, including improving the hierarchical task structure, adding more scheduling algorithms and optimization functions, changing priorities of tasks onboard based on events or mission phase, and working with more complex task plans to make MEXEC more powerful.

Conclusions

The experiments that were performed on the ASTERIA CubeSat and testbed show that MEXEC can be integrated into existing flight software and be used for operations, as well as enable robustness and fail-operational commanding. The onboard experiment demonstrated successful integration of the MEXEC software into the ASTERIA flight software and replication of the behavior of the standard ASTERIA sequences for operations. The second experiment, performed on the ASTERIA testbed, demonstrated how MEXEC could benefit the ASTERIA mission's momentum management by monitoring momentum, dumping momentum when necessary, and recovering without reaching fault protection limits. This would prevent the spacecraft from going into safe mode, therefore allowing it to recover and possibly continue science observations as well as take passes without reset recovery procedures. This would also potentially allow operators to generate less conservative observation schedules to increase the number of observations or increase observation quality. Additionally, had contact not been lost with ASTERIA, operations would have continued as the spacecraft lost altitude and therefore also attitude control. The ground tools would have been less reliable in predicting momentum for observations and MEXEC could have prevented unnecessary occurrences of entering safe mode.

Acknowledgements

We would like to thank everyone on the ASTERIA team. We would not have been able to do these experiments without their expertise and help. In particular, we would like to acknowledge Dr. Matthew Smith, Dr. Christopher Pong, Dr. Terry Huntsberger, Aadil Rizvi, Michael Starch, and Carolyn Maynard. Additionally, we would like to thank Dr. Patricia Beauchamp for her leadership and support during the extended mission.

We would also like to thank Gregg Rabideau and Dr. Steve Chien for their feedback and mentorship in the development of MEXEC. We also thank Ellen Van Wyk for her work on visualization tools.

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

References

Blue Canyon Technologies. 2020. Blue Canyon Technologies. https://www.bluecanyontech.com. [Online; accessed February 25, 2020].

Bocchino, R.; Canham, T.; Watney, G.; Reder, L.; and Levison, J. 2018. F Prime: an open-source framework for small-scale flight software systems. In *Proceedings of the AIAA/USU Conference on Small Satellites*. Advanced Technologies II, SSC18-XII-328. https://digitalcommons.usu.edu/smallsat/2018/all2018/328/.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using Iterative Repair to Improve Responsiveness of Planning and Scheduling. In *International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davis, A.; Mandl, D.; Frye, S.; Trout, B.; et al.

2005. Using autonomy flight software to improve science return on Earth Observing One. *Journal of Aerospace Computing, Information, and Communication* 2(4):196–216.

Chien, S.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; and Policella, N. 2012. A generalized timeline representation, services, and interface for automating space mission operations. In *SpaceOps 2012*.

Dooley, J. 2019. Direct-to-Earth Mission Concept for a Europa Lander. In *2019 IEEE Aerospace Conference*.

Ellis, E. G. 2017. The Journey of NASA's Smartest Satellite Finally Comes to an End. https://www.wired.com/2017/03/say-farewell-eo-1-nasas-smartest-satellite. [Online; accessed February 25, 2020; posted March 24, 2017].

Fesq, L.; Beauchamp, P.; Donner, A.; Bocchino, R.; Kennedy, B.; Mirza, F.; Mohan, S.; Sternberg, D.; Smith, M. W.; and Troesch, M. 2019. Extended Mission Technology Demonstrations Using the ASTERIA Spacecraft. In *2019 IEEE Aerospace Conference*, 1–11. IEEE.

Gat, E., and Pell, B. 1998. Smart executives for autonomous spacecraft. *IEEE Intelligent Systems and their Applications* 13(5):56–61.

Grasso, C., and Lock, P. 2008. VML sequencing: Growing capabilities over multiple missions. In *SpaceOps 2008 Conference*, 3295.

Jet Propulsion Laboratory, California Institute of Technology. 2020. Tiny satellite for studying distant planets goes quiet. https://www.jpl.nasa.gov/news/news.php?feature=7568. [Online; accessed January 3, 2020; posted January 3, 2020].

Kolcio, K. O. 2016. Model-Based Fault Detection and Isolation System for Increased Autonomy. In *AIAA SPACE 2016*. 5225.

Mackey, R.; Brownston, L.; Castle, J. P.; and Sweet, A. 2010. Getting diagnostic reasoning off the ground: maturing technology with TacSat-3. *IEEE intelligent systems* 25(5):27–35.

Mason, J.; Baumgart, M.; Woods, T.; Downs, C.; Hegel, D.; Rogler, B.; Stafford, G.; Solomon, S.; Chamberlin, P.; Caspi, A.; Palo, S.; Jones, A.; Kohnert, R.; Li, X.; Moore, C.; and Rouleau, C. 2017. MinXSS-1 CubeSat On-Orbit Pointing and Power Performance: The First Flight of the Blue Canyon Technologies XACT 3-axis Attitude Determination and Control System. *Journal of Small Satellites* 6(3):651–662.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial intelligence* 103(1-2):5–47.

Peer, S., and Grasso, C. A. 2005. Spitzer Space Telescope Use of the Virtual Machine Language. In *2005 IEEE Aerospace Conference*, 1–10. IEEE.

Pell, B.; Gat, E.; Keesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1234–1239.

Pong, C. M.; Sternberg, D. C.; and Chen, G. T. 2019. Adaptations of guidance, navigation and control verification and validation philosophies for small spacecraft. In *Proceedings of the 42nd Annual AAS Rocky Mountain Section Guidance and Control Conference held January 31 to February 6, 2019, Breckenridge, Colorado*. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration.

Pong, C. 2018. On-Orbit Performance & Operation of the Attitude & Pointing Control Subsystems on ASTERIA. In *Proceedings of the AIAA/USU Conference on Small Satellites*. Poster Session 1, SSC18-PI-34. https://digitalcommons.usu.edu/smallsat/2018/all2018/361/.

Rabideau, G., and Benowitz, E. 2017. Prototyping an onboard scheduler for the mars 2020 rover. In *International Workshop on Planning and Scheduling for Space*.

Riedel, J.; Bhaskaran, S.; Desai, S.; Han, D.; Kennedy, B.; McElrath, T.; Null, G.; Ryne, M.; Synnott, S.; Wang, T.; et al. 2000. Using autonomous navigation for interplanetary missions: The validation of deep space 1 AutoNav.

Smith, M.; Donner, A.; Knapp, M.; Pong, C.; Smith, C.; Luu, J.; Pasquale, P.; and Campuzano, B. 2018. On-Orbit Results and Lessons Learned from the ASTERIA Space Telescope Mission. In *Proceedings of the AIAA/USU Conference on Small Satellites*. The Year in Review, SSC18-I-255. https://digitalcommons.usu.edu/smallsat/2018/all2018/255/.

Smith, B.; Feather, M.; Huntsberger, T.; and Bocchino, R. 2020. Software Assurance of Autonomous Spacecraft Control. In *66th Annual Reliability & Maintainability Symposium (RAMS)*.

Troesch, M.; Mirza, F.; Rabideau, G.; and Chien, S. 2019. Onboard re-planning for robust mapping using pre-compiled backup observations. In *11th International Workshop on Planning and Scheduling for Space (IWPSS)*, 168–175. Berkeley, California, USA.

Verma, V.; Gaines, D.; Rabideau, G.; Schaffer, S.; and Joshi, R. 2017. Autonomous Science Restart for the Planned Europa Mission with Lightweight Planning and Execution. In *Proceedings of the 10th International Workshop on Planning and Scheduling for Space (IWPSS2017)*. Pittsburgh, PA.

Wang, D.; Russino, J. A.; Basich, C.; and Chien, S. 2020. Using Flexible Execution, Replanning, and Model Parameter Updates to Address Environmental Uncertainty for a Planetary Lander. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation for Space*, i-SAIRAS'2020. Noordwijk, NL: European Space Agency.