Automated Resource-Constrained Science Planning for the MiRaTA Mission

Andrew Kennedy, Anne Marinan, Kerri Cahoy, James Byrne, Timothy Cordeiro, Zachary Decker, Weston Marlow, Stephen Shea
Space Systems Laboratory, Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA 02139; 713-824-8499
AKENNEDY@MIT.EDU

William J. Blackwell, Michael DiLiberto, R. Vincent Leslie, Idahosa Osaretin, Erik Thompson
Lincoln Laboratory, Massachusetts Institute of Technology
244 Wood Street Lexington, MA 02420-9108
WJB@LL.MIT.EDU

Rebecca Bishop
The Aerospace Corporation
2310 E. El Segundo Blvd. El Segundo, CA 90245-4609
REBECCA.L.BISHOP@AERO.ORG

ABSTRACT
We present a novel algorithm for science planning for the Microwave Radiometer Technology Acceleration (MiRATA) CubeSat mission that reasons about onboard resource limitations and automatically produces timelines for onboard activities with minimal human involvement. The Resource-Aware SmallSat Planner (RASP) attempts to maximize total science data acquisition time while also maximizing onboard energy and data storage margin. RASP was demonstrated over a representative 24 hour simulation of MiRATA’s orbit with 19 timing and resource critical science acquisition opportunities. We showed that RASP successfully plans the science opportunities over varying planning horizon lengths, achieving at most 16 and at least 12 of the opportunities. Average onboard resource usage margins were examined, and ranged between 69.0 and 72.5% of the total range for data storage and 60.5 and 71.1% of the total range for energy storage. We examined the effect of ignoring resource storage margin, and found that energy margin dips as low as 42.3% and data margin as low as 33.6%. Finally, we found that RASP takes on the order of 10 seconds to create a feasible plan for the length of one orbit, suggesting that the algorithm is suitable for adaptation to a more computationally constrained onboard processor system.

INTRODUCTION
We first introduce the Microwave Radiometer Technology Acceleration (MiRATA) mission and science concept of operations, then discuss the benefits of automated planning for MiRATA’s science operations.

The MiRATA Mission
MiRATA is a 3U CubeSat being developed with the support of the NASA Earth Science Technology Office (ESTO) for a late 2016 launch [1,2]. Microwave radiometry and GPS radio occultation (GPSRO) measurements of all-weather temperature and humidity provide key contributions toward improved weather forecasting. The MiRATA mission will validate new technologies in both passive microwave radiometry and GPS radio occultation: (1) new ultra-compact and low-power technology for multi-channel and multi-band passive microwave radiometers, and (2) new CubeSat-scale GPS receiver and patch antenna array technology for GPS radio occultation retrieval of both temperature-pressure profiles in the atmosphere and electron density profiles in the ionosphere. MiRATA will also validate (3) a new radiometer calibration approach for spaceborne microwave radiometers that uses collocated GPS radio occultation measurements. The MiRATA passive microwave radiometer payload features three bands for sensing all-weather temperature (50-58 GHz), water vapor (175-191 GHz), and cloud ice (203-206 GHz).

Science Concept of Operations
In order to achieve its mission goal of radiometer calibration through GPSRO measurements, MiRATA performs a slow pitch-up to allow its radiometer and GPS receiver payloads to sound overlapping volumes of Earth atmosphere where sensitivity, calibration, and dynamic range are optimal [3]. The spacecraft will periodically slew from a radiometer nadir pointing...
attitude, Local Vertical Local Horizontal (LVLH), to a 90-105° pitch angle and back. A diagram of this maneuver is shown in Figure 1. MiRaTA’s 3-axis stabilized attitude determination and control system (ADCS) controls this pitch maneuver. During steps 1 and 2 in Figure 1, the radiometer passes over, or sounds, a volume of Earth’s atmosphere. In steps 3 and 4, the onboard GPS receiver - Compact TEC (Total Electron Content)/Atmosphere GPS Sensor (CTAGS) - antenna beam points toward the Earth’s limb and receives signals from one or more satellites in the GPS constellation as they appear to “set” behind (are “occulted by”) the Earth from MiRaTA’s perspective (Figure 1, steps 3 and 4). The volumes of atmosphere sounded in steps 1 to 2 and 3 to 4 should overlap as much as possible to achieve a good calibration. The sequence of radiometer sounding, then pitch-up, GPSRO collection, and subsequent pitch-down maneuver is expected to last approximately 22 – 32 minutes.

**Figure 1: The science maneuver for MiRaTA: 1) Start from LVLH-stabilized attitude. 2) Pitch up at 0.5°/s to scan the radiometer field of view through the limb. 3-4) GPSRO is enabled and directed through the same atmosphere. 5) Pitch down to nominal attitude [3]**

**Automating Science Planning**

One of MiRaTA’s mission goals is to successfully perform at least 100 of these science calibration maneuvers over the course of a 60-day primary mission operations period. The timing for the maneuvers depends on orbital geometry, particularly achieving the desired alignment between the radiometer field of view and the location of the GPSRO measurements. Given the relatively infrequent opportunities for ground contacts (a representative day contains two sets of three short ground passes separated by about 90 minutes, with the two sets separated by about 10 hours), the spacecraft must be capable of performing these maneuvers without human-in-the-loop supervision during the actual maneuver. We plan to perform the maneuvers in a scripted fashion, with operations scripts derived in advance on the ground from predicted orbital parameters and uplinked to MiRaTA during ground contacts with the mission’s ground station at NASA Wallops. In order to ensure the successful execution of multiple maneuvers over long periods without ground contact, these scripts must also manage onboard resources effectively, including energy stored in the spacecraft batteries and storage of the science and engineering telemetry data collected. The same needs are common to many types of Earth observation missions. For this reason, we decided to design an algorithm (the Resource-Aware SmallSat Planner, RASP) that could build these scripts in an automated way by reasoning about the timing of maneuver opportunities and other onboard activities as well as the spacecraft resource states at any given time.

**Organization**

We first discuss the operational constraints of the MiRaTA mission in detail and create a simple, representative model of the spacecraft activities. We then describe the algorithm (RASP) developed for autonomously scripting MiRaTA’s activities based on this model. Finally, we evaluate the algorithm’s performance over a 24-hour simulation run and analyze the effects of changing various parameters in the algorithm.

**MODELING MIRATA OPERATIONS**

We developed a simple but representative model of MiRaTA’s operations that can be used for automated planning. We first discuss the details of MiRaTA’s operations and then introduce this model.

**Orbit**

The design orbit for the MiRaTA mission is an elliptical sun-synchronous orbit - 450 km x 810 km altitude at 97.2 degrees - with local time of ascending node 13:25. The target launch is in November 2016, and deployment from the International Space Station is held as a potential backup option.

**Calculation of Maneuver Times**

There are specific times when the maneuver activity can be performed, given MiRaTA’s orbit and positioning relative to GPS satellites. To simulate the GPSRO accesses that would overlap with the radiometer field of view, a scenario was set up using Analytical Graphics, Inc’s (AGI) Systems Toolkit (STK). The satellite was placed in the reference orbit described above, and line-of-site accesses to each satellite in the GPS constellation were computed. From each access, we were interested in the range, azimuth, and elevation of each GPS satellite relative to MiRaTA.

The science maneuver was modeled assuming the spacecraft was at a 100 degree pitch-up angle (relative
to LVLH) across the entire orbit (rather than changing the pitch-up angle based on altitude for each maneuver). The radiometer sensor field of view is assumed to be a 2 degree full angle, offset from the spacecraft orbital plane by +5 degrees. We identified the set of GPS occultations that overlapped with this radiometer field of view (174°-176° azimuth, -20° to 0° elevation) and that passed through the area of highest gain on the patch antenna feeding MiRaTA’s GPS receiver to determine the frequency and duration of viable GPS occultation opportunities. A representative set of GPSRO accesses from a 24-hour period is shown in Figure 2.

Figure 2: GPSRO Accesses for one GPS satellite over 24 hours. Overlapping radiometer and GPSRO events shown in red.

Preliminary results indicate that over a 24-hour period an average of 2-3 setting occultations will occur that overlap with the radiometer field of view, and the overlap will last for 5-7 minutes. For the purposes of demonstrating automated science planning in this work, we relaxed the radiometer field of view elevation restriction and considered all GPS satellites that passed through the radiometer’s azimuth range. This relaxation results in more maneuver access times being considered by the algorithm and a more thorough assessment of the algorithm’s planning robustness.

**Onboard Energy Production and Storage**

MiRaTA has two sets of double-deployed solar panels that can generate up to 24.8 W in total when fully illuminated at normal incidence as shown in Figure 3.

Figure 3: Zenith face of MiRaTA

When the spacecraft is in LVLH orientation, the spacecraft is in an overall energy negative situation. To stay power positive, the ADCS system will implement a sun-tracking attitude in which the zenith face of the spacecraft is pointed within 20 degrees of the sun vector. There are, however, several circumstances that can prevent the spacecraft from maintaining a default sun-tracking attitude:

- All science maneuvers must start from an LVLH orientation
- The spacecraft must be in LVLH when communicating with the groundstation
- The LVLH attitude is the lowest-drag configuration, which is especially necessary for low-altitude orbits (e.g. ISS deployment)

The day-to-day (and orbit-to-orbit) operations of the satellite must take into account these constraints as well as a requirement to keep the battery (20 W-Hr lithium polymer) above a 30% depth of discharge at all times.

**Onboard Data Production, Storage, and Downlink**

MiRaTA produces a large amount of science data during its science maneuver, greater than 60 Megabits for a maneuver with 15 minutes of high rate GPS signal tracking. It also continually produces both spacecraft bus and payload housekeeping data. To meet the requirement to get all this data to ground, the spacecraft has a radio for high data rate downlink. The radio nominally operates at an effective 2.6 Mbps data rate over a link with the mission’s dedicated ground station at NASA Wallops at latitude and longitude 37.86° and -75.51°, respectively (data rate value from personal
communication with Erik Stromberg at Utah State University Satellite Dynamics Laboratory).

**Spacecraft Attitude Control Model**

The MiRaTA design includes a dedicated ADCS package with a sensor and actuator suite, capable of achieving sub-degree level pointing accuracy [4]. Attitude determination is achieved with three earth horizon sensors, six coarse sun sensors, and a 3-axis magnetometer. Actuation is achieved with three reaction wheels and magnetorquers. The system is sized to achieve MiRaTA’s science maneuver and maintain pointing in LVLH and sun tracking modes during nominal operations without saturating the reaction wheels. Continual desaturation via the magnetorquers aids this process.

**Operational State Machine and Activities**

Figure 4 presents a state machine representation of the operational activities (modes) for MiRaTA. During operations the spacecraft is assumed to be in one and only one of these modes at any given time. Each mode corresponds to an activity that the spacecraft is performing, and the spacecraft can transition between activities along any of the arrows in the state machine. The science maneuver is represented by “Maneuver”, at the top. The Recharge activity corresponds to the dedicated ADCS mode when the spacecraft tracks the sun. The Downlink activity corresponds to a downlink to the ground station. The Slew activity is when the spacecraft uses the ADCS actuator suite to change its attitude. We make the conservative assumption that a slew must occur between any maneuver and resource management activity, including two of the same activity in a row. The Idle activity occurs when no other activities are ongoing.

![Figure 4: State machine representation of the operational activities for MiRaTA](image)

Resource usage is broken down by activity in Table 1. The energy usage values in the table were calculated including all power draws from activated components by mode. We assume here that energy is only produced in recharge mode, when the spacecraft maintains an attitude that maximizes solar power produced (~24.8W). The MiRaTA spacecraft will in actuality have solar power input during other modes, depending on orbital geometry. The restriction to recharge mode here simplifies the model while maintaining a conservative estimate of power production.

The data production values in the table include the production of spacecraft bus housekeeping and payload housekeeping telemetry. For the maneuver mode, the nominal GPSRO data production rate (63.2 kbps, tracking three GPS satellites at 50 Hz sampling rate) is added in, and assumed to occur over the entire duration of the maneuver. Downlink uses the nominal radio downlink data rate, 2.6 Mbps.

<table>
<thead>
<tr>
<th>Table 1: Activity Resource Usage Broken Down by Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Energy (ES)</td>
</tr>
<tr>
<td>Data (DS)</td>
</tr>
<tr>
<td>Minimum Duration</td>
</tr>
</tbody>
</table>

**RESOURCE-AWARE SMALLSAT PLANNER (RASP) ALGORITHM**

The RASP algorithm was developed to autonomously plan and schedule activities onboard a resource-constrained small satellite. Our discussion of the algorithm follows its previous introduction by Kennedy and Cahoy [5], with more detail added. The algorithm has some similarities with the ASPEN/CASPER algorithms developed at NASA JPL [6] in that it evaluates the feasibility of performing activities based on onboard resource usage, but it a) uses a simpler model focused specifically on a resource-constrained satellite and b) it constructs an entire activity sequence in a single algorithm, as opposed to creating an initial high level sequence for later onboard refinement.

Activity planning constitutes the selection of a set of activities (an “activity sequence”) from the operational state machine (Figure 4) that allows the satellite to execute as many science maneuver activities as possible while maintaining onboard resources within constraint limits. Scheduling is the assignment of a set of start and end times to every activity in the plan (an “activity timeline”) such that an overall score function is
maximized as well as the determination of acceptable trajectories for onboard resource states. Figure 5 illustrates two example resource trajectories that are kept within resource bounds.

Figure 5: Example data storage and energy storage trajectories over an activity timeline

The RASP algorithm finds a suboptimal but acceptable activity timeline within a given planning horizon ($t_h$) given a set of initial maneuver access windows. In order to limit the required computational time it does not attempt to determine the optimal timeline. The planner is used in a receding horizon fashion; that is, the satellite plans a set of activities for itself within $t_h$, executes those for a certain time, and replans from that new time using an updated state. This repeats for the duration of the scenario.

The following subsections detail the main components of RASP. The first subsection discusses the inputs RASP uses to create an initial activity sequence, the second discusses the scheduling of an optimal activity timeline from a specified activity sequence, the third, fourth, fifth, and sixth discuss in detail RASP’s search mechanism for modifying this initial sequence to arrive at a feasible sequence, and the seventh discusses how RASP modifies the initial activity sequence if no feasible activity timeline was found during the search.

1. RASP Inputs

An STK simulation of MiRaTA’s orbit is used to derive a set of maneuver access windows, recharge windows, and downlink windows. Maneuver access windows are assumed to be non-overlapping. Downlink windows occur whenever the satellite is above a fixed elevation mask as viewed by the ground station. Recharge windows occur whenever the satellite is illuminated by the sun. Given the time windows over the specified planning window horizon, $t_h$, RASP constructs an initial activity sequence with a single maneuver activity during every maneuver access time. Figure 6 depicts a notional initial activity timeline over the course of an arbitrary orbit, $i$. Note that slew activities must occur between maneuvers in order to restore the desired attitude, and idle activities occupy the non-used times.

2. Activity Timeline Optimization

Given an activity sequence, the scheduler component of RASP attempts to find an optimal activity timeline. An activity timeline consists of an ordered list of timepoints $t_{a,i}^S$ and $t_{a,i}^E$, where $i \in [1,N]$, which represent the start and end times of each activity, respectively. $N$ is the number of activities. The symbol $a$ signifies a high-level activity, such that $a \in \text{Act} = \text{Man} \cup \text{Dlnk} \cup \text{Rech} \cup \text{Slew} \cup \text{Idle}$, where each set in the overall union contains all the maneuver, downlink, recharge, slew, and idle activity types, respectively. This optimization is formulated as a Mixed Integer Linear Program (MILP) [7]:

$$\text{max} \{ \sum_{M_{man}} (t_{a,i}^E - t_{a,i}^S) - w_d \sum_{D_{link}} d_j(t_{a,i}^E - t_{a,j}^S) + w_o \sum_{i=1}^{N} \sum_{j=1}^{i} c_j(t_{a,i}^E - t_{a,j}^S) \}$$

subject to:

$$t_{a,1}^S = 0, \quad t_{a,i}^S \leq t_{a,i}^E, \quad t_{a,N}^E = t_h; \quad 1 \leq i \leq N$$

$$t_{a,i}^E = t_{a,i}^S; \quad i \geq 1, j \leq N \mid j = i + 1$$

$$t_{a}^S = t_{a}^{SW}, \quad t_{a}^E \leq t_{a}^{EW}; \forall a \in \text{Obs}$$

$$t_{a}^S \geq t_{a}^{SW}, \quad t_{a}^E \leq t_{a}^{EW}; \forall a \in \text{Dlnk} \cup \text{Rech}$$

$$t_{a}^E - t_{a}^S \geq \text{MinDur}_a; \forall a \in \text{Act}$$

and

$$R_{S_{i,1}} + r_1(t_{a,1}^E - t_{a,1}^S) \leq UB_{RS} + M(1 - z_{RS,1})$$

$$R_{S_{i,1}} + \frac{2}{i} \sum_{i=1}^{2} r_i(t_{a,i}^E - t_{a,i}^S) \leq UB_{RS} + M(1 - z_{RS,1})$$

$$...$$

$$R_{S_{i,1}} + \frac{N}{i} \sum_{i=1}^{N} r_i(t_{a,i}^E - t_{a,i}^S) \leq UB_{RS} + M(1 - z_{RS,N})$$

(7)
The score function in Equation 1 attempts to maximize three items: the sum of all observation durations in the activity timeline (summation 1), the total amount of data downlinked over the activity timeline (summation 2), and the average amount of ES margin over the course of the activity timeline (double summation). The $d_i$ and $e'_i$ terms correspond to the DS usage rate and ES usage rate for activities $i$ and $j$, respectively. ES margin here refers to the difference between the ES state at the end of an activity and the ES lower limit. The outer summation ($i = 1$ to $N$) accounts for the ES margin at the end of all activities in the activity sequence, and the inner summation propagates the ES state forward through the activity timeline by accounting for ES changes over all activities $j$ up to activity $i$. The weighting terms $w_d$ and $w_e$ are calculated as:

$$w_d = u_{DS} / (UB_{DS} - LB_{DS})$$

$$w_e = u_{ES} / (UB_{ES} - LB_{ES}) / N$$

Equation 9 expresses that the total amount of data downlinked over an activity timeline is normalized by the range between DS bounds (where $UB_{DS}$ and $LB_{DS}$ represent the upper and lower bounds respectively) and multiplied by a unitless “urgency factor”, $u_{DS}$, which effectively tunes the algorithm’s preference for downlinking data. If this factor is set to 0, RASP will not care at all about downlinking data outside of its necessity to keep DS within bounds. Equation 10 is a similar expression, except that the additional normalization by the number of activities, $N$, means that the algorithm minimizes average ES margin.

Equations 2 enforce a planning window from 0 to $t_e$ and ensures that the end of every activity follows its start. Equations 3 force activity $j$ to follow activity $i$. Equations 4 and 5 force the activities which have time windows (i.e., not slew or idle) to fall within those windows; $t^{SW}_{i,j}$ and $t^{EW}_{i,j}$ signify the start and end of the relevant time window. Equation 6 enforces activity minimum durations. The $N$ equations in 7 and 8 enforce resource constraint upper bounds and lower bounds, respectively; the RS signifies that these equations hold for both resource types: ES and DS. We use the “Big M” method to select whether specific constraints will or will not be enforced [7]; hence, $M$ is a large integer and $z^B_i \in \{0,1\}$ is a variable that decides whether the constraint for the given activity number is enforced.

3. Activity Sequence Construction Through Greedy Search

RASP uses the selective enforcement of constraints in Equations 7 and 8 as a mechanism for determining where to add downlink and recharge activities to arrive at a final plan with all constraints enforced. At the highest level the algorithm performs a Depth-First Search through a tree of modified activity sequences constructed from the initial activity sequence. Children activity sequences are created by adding a single resource management activity- an activity of type $dlnk$ or $rech$ - at a time to the parent activity sequence. Slew and idle activities are added as necessary to maintain conformity to the semantics of the operational state machine. This process of search through incremental activity sequence modifications is shown in Figure 7.

Figure 7: Illustration of Depth First Search process used to find feasible activity timeline. The top activity sequence (1) cannot produce a consistent timeline, due to resource usage limit violations. The algorithm attempts to add recharge or downlink activities at various locations in sequence (2), progressively enforcing resource constraints it searches through the tree. Eventually a feasible timeline is found (3), consisting of the original sequence plus the recharges and downlinks added.

Adding these activities allows the algorithm to progressively enforce more of the driving constraints (DS $UB$ and ES $LB$), pushing towards the goal state of having all constraints enforced. When a new activity is added, the algorithm attempts to solve the MILP with the appropriate resource constraint set enforced up to the location where the activity was added. The scores of the children activity sequences produced along the way...
inform the algorithm’s choice of the next node to expand.

A heuristic function is used to push the algorithm to progressively enforce more constraints, while also trying to increase the score for the timeline. The function favors downlinks first because of their small, rare time windows, followed by recharges. When a timeline is found that satisfies all the constraints in the MILP, it is returned. For practical purposes, we limit the search to a timeout period (25 seconds), after which a reduction is made to the input activity sequence (the problem is simplified) and RASP is run again. The RASP algorithm as currently implemented is non-optimal and non-complete, but we strongly believe it is sound (if it returns what it believes to be a solution, that solution is in fact correct and reliable).

The following sections discuss in detail the algorithms used in RASP.

4. RASP High Level Algorithm: Depth-First Search

The high-level search algorithm is presented in Figure 8. Lines 2 and 3 set the non-driving resource constraints and driving constraints to be on and off, respectively. Initially fixing the non-driving constraints on reduces the size of the search space and still provides acceptable performance. Line 4 initializes the root node of the search tree with the initial activity sequence $A_{init}$ as well as all the enforced constraints for this activity sequence from the previous lines, $z^*$. A node can store many values describing its activity sequence; more values will be introduced in the lower level algorithms. Lines 7 through 11 are a standard search formulation; the best child, Next, is popped from the search queue, $Q$, tested to see if it’s a Goal state (all constraints enforced), and if not, then its children are added to the search queue. Detailed discussion of algorithms PopBest() and GetChildren() follows.

![Figure 8: RASP Depth-First Search Algorithm](image)

5. GetChildren: Find Children Through One-Step Modification

The heart of the RASP algorithm lies in the GetChildren() procedure in Figure 9. GetChildren produces all possible one-step modifications to the parent activity sequence and returns the modified activity sequences as nodes for addition to the search queue. A one-step modification consists of replacing an idle in the parent activity sequence with a resource management activity (of type dlknk or rech), as well as any transition activities necessary to make the resulting modified sequence valid.
The GetChildren() procedure first grabs all the idle activities from the parent activity sequence, in Line 4. It then loops through all driving constraint types (Line 5) and all parent idle activities (Line 6) and replaces the idle with the corresponding management activity (Lines 7 to 9). In lines 10 and 11, it attempts to find a valid activity timeline from the modified activity sequence by solving a relaxed version of the MILP in Equations 1 to 10. It does this while enforcing a) all its parents’ constraints from constraint sets other than RS, and b) all constraints in set RS up to the activity that was just replaced. If a valid activity timeline cannot be found, then this modification is likely not useful, and no child is created for this particular management activity - idle location combination. If a valid timeline is found, then the algorithm attempts to produce a valid timeline with all constraints enforced from set RS (Lines 13 and 14). Attempting to enforce all constraints in RS drives the algorithm towards the goal state more effectively. If no all-enforced timeline can be created, a new child node is added with enforcement up to the replacement location and with a record of the evaluated score function, Score, for this new timeline (Lines 26 and 27).

Lines 16 to 24 serve a special purpose in driving the algorithm towards a high quality solution, by determining the usefulness of adding another resource management activity corresponding to set RS even if all the constraints in RS have already been enforced. The intuition here is that a certain placement of management activities may barely satisfy constraints, without any margin between resource usage and resource upper/lower bounds, so adding more activities may introduce more margin and give the flexibility to achieve a higher quality solution. Each node stores the value NumTimesUseful for each driving constraint set RS, which tells the algorithm how many times set RS has been “usefully” fully enforced along this particular branch of the search tree. These three values are initially set to 0. The first time RS is fully enforced, its value is set to 1 (Lines 18 and 19). For every subsequent time RS is enforced, this value is incremented if the corresponding activity replacement caused an increase in score function from the parent.

Algorithm 2 Find all consistent single-add activity sequences

```plaintext
1: procedure GetChildren(Parent)  
2:     Children ← Ø  
3:     A ← GetActSeq(PARENT)  
4:     indices ← RankIdles(A)  
5:     for all RS ∈ {ES LB, DS UB} do  
6:         for all k ∈ indices do  
7:             switch RS do  
8:                 case ES LB : Atemp ← REPLACE(A, k, rech)  
9:                 case DS UB : Atemp ← REPLACE(A, k, dlkn)  
10:             end switch  
11:             zRS ← 1 ∀ 1 ≤ i ≤ k  
12:             if Solvable then  
13:                 zRS ← 1 ∀ 1 ≤ i ≤ N  
14:                 if Solvable then  
15:                     ∆Score ← Score − Parent.Score  
16:                     NumTimesUseful ← TIMESUSEFUL(Parent, RS)  
17:                     if NumTimesUseful = 0 then  
18:                         NumTimesUseful = 1  
19:                     end if  
20:                 else if ∆Score > 0 then  
21:                     NumTimesUseful = NumTimesUseful + 1  
22:                     end if  
23:                     NewChild ← MakeNode(Atemp, Parent.z*, zRS, Score, NumTimesUseful)  
24:                     Children ← Children ∪ NewChild  
25:                 end if  
26:                 NewChild ← MakeNode(Atemp, Parent.z*, zRS, Score)  
27:                 Children ← Children ∪ NewChild  
28:             end if
29:         end for  
30:     end for  
31: end procedure
```

Figure 9: Algorithm for finding children based on one-step modifications to parent activity sequence
activity sequence (Lines 20 to 21). This value is used to
guide the selection of the next child to expand in
Algorithm 3. The TimesUseful() procedure on Line 17
merely returns NumTimesUseful for RS from the given
node. After the update check, a new child node is added
in Lines 23 and 24 with the updated value.

6. PopBest: Select Best Child to Expand

The PopBest() procedure in Figure 10 selects the best
child to expand next in the search tree. It uses a series
of filters, FilterQueue(), to eliminate candidates from
the search queue and hone in on this best child. Each
successive filter only grabs those children from Q with
the maximum value of the second argument. PopBest
was designed to cause the RASP algorithm to first
enforce all constraints in set DS UB, then those in set
ES LB, because of the increased ease of meeting time
windows for rech activities.

Algorithm 3 Remove best child from Q, return child and modified Q
1: procedure PopBest(Q)
2: $Q_{temp} \leftarrow \text{FilterQueue(} \text{max, TimesUseful(} \text{DS UB})\text{)}$
3: $Q_{temp} \leftarrow \text{FilterQueue(} \text{max, TimesUseful(} \text{ES LB})\text{)}$
4: $Q_{temp} \leftarrow \text{FilterQueue(} \text{max, z}_{UB}\text{)}$
5: $Q_{temp} \leftarrow \text{FilterQueue(} \text{max, z}_{ES}\text{)}$
6: $Q_{temp} \leftarrow \text{FilterQueue(} \text{max, Score}\text{)}$
7: end procedure

Figure 10: Algorithm to find best next child to expand in search tree

The first set of filters in Lines 2 to 3 causes the
algorithm to prefer adding as many of each resource
management activity as appears useful, before
attempting to achieve the goal state. The filters in Lines
4 to 5 push the algorithms towards the actual goal state
of enforcing all constraints. Line 6 selects from the
filtered children the child with the best score.

7. Modification of Maneuver Activities Selection

If it is found that the input activity sequence with its set
of maneuver access times cannot be solved through the
search process, a maneuver access time is subtracted
from the input sequence to make the problem more
feasible. Multiple access times can be progressively
removed if the search process fails more than once.
RASP attempts to remove the most time-restricted
maneuver access; that is, the access time with the
minimum amount of time between the start of the
access before it and the end of the access after it. This is
a proxy for how much demand that maneuver places on
the activity sequence as a whole. Maneuver activities
are particularly constraining because there are often
many of them, and they have a large minimum time
duration. Note that the first activity in the input
sequence will never be removed because it is the
satellite’s current activity. If all possible maneuvers are
removed and RASP still cannot solve the MILP,
resource bounds are progressively relaxed until a
solution is found. In the current algorithm, there is no
method to reconcile relaxing a bound past a physical
limit, so resource bounds should in general be set
tighter than physical bounds.

OPERATIONAL SIMULATION AND ALGORITHM PERFORMANCE

The RASP algorithm was implemented in the
MATLAB programming language from MathWorks,
using the linprog() function and dual-simplex optimizer
for linear program solution. A software simulation
program was developed in Python to evaluate the
performance of the algorithm over a 24-hour
operational period. The simulation environment keeps
track of a global clock, propagates the satellite’s
resource states forward, and calls RASP to replan for
the spacecraft at regular intervals (fixed at 20 minutes
in this work). Plans are generated from the satellite’s
current time to the current time plus the planning
horizon, $t_h$. The global clock was configured to run
with 1 second ticks. The resource bounds for the spacecraft
were set to $[0, 100]$ MiB for DS ($1\text{MiB} = 1024^2\text{Bytes}$)
and $[14, 20]$ W-hr for ES (a 30% depth of discharge).
Note that the 100 MiB lower limit is lower than amount
of data MiRaTA can actually store onboard, but this
low limit forces the RASP algorithm to try to downlink
data to limit data latency.

RASP inputs were ingested from the STK orbit
simulation. Maneuver windows were derived by
determining GPSRO access times (as explained in the
introduction) and adding an additional 9 minutes in
front of the access time to account for the expected
radiometer data collection period, indicated during the
pitch up phase in Figure 1. If multiple GPSRO access
times overlapped or were separated by less than 9
minutes, they were all combined together into a single
longer time window, and the radiometer time added
afterwards. This is a simple assumption useful for
demonstrating RASP’s execution in this work, but does
not necessarily reflect final mission operations. There
were 19 such combined maneuver windows found over
the 24-hour period investigated, with an average length
of 18.1 minutes. Downlink windows were determined
based on when the satellite is above a 10 degree
elevation mask as viewed by the Wallops ground
station. Recharge windows occurred whenever the
spacecraft was illuminated by the sun.

Figure 11 shows an example execution of RASP, where
an activity timeline is successfully created for a
planning window of $t_h = 120$ minutes, at a time point 19
hours and 20 minutes into a 24 hour simulation. This
was the 58th of 72 such plans generated over the whole simulation (a number fixed by the simulation duration and replan time).

![Activity Timeline - MiRaTA SV 58](image)

**Figure 11:** Example consistent activity plan generated for simulation run with $t_h = 120$ mins, $u_{ES} = 1$, $u_{DS} = 1$. The upper plot shows the RASP-derived activity timeline with the heights of the black activity line matching to the activity labels on the left or right y-axis. The dashed color lines represent activity windows: green for maneuver, blue for downlink, and magenta for recharge. The lower three plots show corresponding resource states during this timeline. (1 MiB = 1024² Bytes)

The execution illustrates the performance of maneuver activities as well as all the use of downlinks and recharges to stay within resource bounds. The dashed color lines represent activity windows (when an activity can be performed) and the solid black lines represent the activity timeline actually chosen. For this work, RASP was configured to find and select the best (highest score function value) of 10 consistent solutions for every input initial activity sequence (Figure 11 represents the best of 10), and was set to timeout after 25 seconds of not finding a consistent solution.

**Algorithm Performance Assessment Organization**

We analyze the performance of the RASP algorithm running multiple simulations with the same set of inputs from this 24-hour period. We first look at how varying the planning window length affects the planning and scheduling of activities by the algorithm, and maneuver activities in particular. We then investigate how resource usage performance varies with planning window length. Next we focus specifically on the effects of varying the ES and DS urgency factors ($u_{ES}$ and $u_{DS}$, respectively, in Equations 9 and 10). Finally, we assess the time performance of RASP.

**Dependence of Activities Executed on Planning Window Length**

Table 2 details the number of activities performed by the spacecraft over the course of the 24-hour simulation, with varying $t_h$. Note that these are the activities both planned and executed by the spacecraft; if RASP planned an activity during one planning window then later decided on a different plan without that activity, it is ignored. The simulations were all run with the nominal weighting of ES and DS urgency factors. Note that the maximum number of maneuvers to execute for all simulations was 19. We see that the number of maneuvers changes with planning window length in a counterintuitive way. We would expect a longer planning window length to do better at achieving maneuvers because the algorithm can plan resource usage further in advance to achieve the maneuvers. This dependence is probably due to the interdependence of the three terms in the cost function in Equation 1, and the choice of resource weightings.

**Table 2: Number of Activities Performed for varying $t_h$, with $u_{ES} = 1$, $u_{DS} = 1$**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Planning Window Length (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td>Maneuver</td>
<td>15</td>
</tr>
<tr>
<td>Recharge</td>
<td>97</td>
</tr>
<tr>
<td>Downlink</td>
<td>5</td>
</tr>
<tr>
<td>Slew</td>
<td>106</td>
</tr>
<tr>
<td>Idle</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 3 shows the total time spent in these activities over the whole simulation, for the same cases as Table 2. We see that the satellite actually spends much more time performing maneuvers in the $t_h = 60$ minutes scenario. Interestingly, the amount of time spent in slew mode increases with $t_h$ until it drops off suddenly at $t_h = 240$ mins. Overall it appears that the $t_h = 60$ simulation runs the most effectively, spending the most time in maneuver mode, and the least time in idle mode.
Maneuver Execution Performance as Function of Planning Window Length

Figure 12 shows which maneuvers are actually executed for each of the simulations with different $t_h$. We see that all simulations executed most of the 19 maneuver windows, and the executed ones were well distributed across the windows. The $t_h = 120$ and $240$ simulations both had a period from about 7 hours to 11 hours where they didn’t execute any maneuvers.

Table 3: Total Time Spent in Each Activity (minutes) for varying $t_h$ with $u_{ES} = 1$, $u_{DS} = 1$

<table>
<thead>
<tr>
<th>Mode</th>
<th>Planning Window Length (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td>Maneuver</td>
<td>168.7</td>
</tr>
<tr>
<td>Recharge</td>
<td>509.0</td>
</tr>
<tr>
<td>Downlink</td>
<td>7.8</td>
</tr>
<tr>
<td>Slew</td>
<td>318.0</td>
</tr>
<tr>
<td>Idle</td>
<td>436.5</td>
</tr>
<tr>
<td>Total</td>
<td>1440</td>
</tr>
</tbody>
</table>

Figure 12: Executed maneuvers (green) and maneuver access windows (grey) for $t_h = 30$, 60, 90, 120, and 240 mins, $u_{ES} = 1$, $u_{DS} = 1$. Fewer maneuvers are executed for larger $t_h$. Table 4 shows the “occupation” percentages for the maneuvers. For example, out of the maneuver windows that were executed for the $t_h = 30$ simulation, an average of 78.6% of the window length was actually executed. We see that the $t_h = 30$, 60, 240 simulations did the best at filling their maneuver windows.

Table 4: Average Maneuver Access Window Occupation for varying $t_h$ with $u_{ES} = 1$, $u_{DS} = 1$

<table>
<thead>
<tr>
<th>Mode</th>
<th>Planning Window Length (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td>Occupation (%)</td>
<td>78.6</td>
</tr>
</tbody>
</table>

Resource Usage Performance as Function of Planning Window Length

Figures 13 and 14 provide more context for understanding the activity execution behavior of the different $t_h$ value simulation runs. Figure 13 shows how well the $t_h = 30$ run managed its resources. We see that it performed well at keeping the DS state below the limit, but poorly at keeping the ES state above its limit. There were two occasions when ES dipped to around 12.5 W-Hr due to RASP relaxing resource bound limits, representing approximately a 40% depth of discharge (DoD). The desired limitation was 30% DoD.

Figure 13: Resource usage (blue) and lower/upper bounds (red) for simulation run with $t_h = 30$ mins, $u_{ES} = 1$, $u_{DS} = 1$ (1 MiB = 1024² Bytes). Multiple energy storage lower limit violations are seen.

Figure 14 shows that with a longer planning window, $t_h = 120$, these limit violations are eliminated. Note that the slight appearance of spikes above the upper ES limit are an artifact of the graphing program and don’t represent real limit violations. Also note that with the longer planning horizon, the planner tends to wait longer to downlink; the large downlinks are shifted later than in Figure 13.
Table 5: Average Resource Margin for varying $t_h$, with $u_{ES} = 1$, $u_{DS} = 1$

<table>
<thead>
<tr>
<th>Mode</th>
<th>Planning Window Length (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td>Average DS</td>
<td>71.8</td>
</tr>
<tr>
<td>Margin (%)</td>
<td></td>
</tr>
<tr>
<td>Average ES</td>
<td>60.5</td>
</tr>
<tr>
<td>Margin (%)</td>
<td></td>
</tr>
</tbody>
</table>

The maneuver and resource usage performance over varying planning window lengths suggests that there is a tradeoff in RASP between maneuver performance and resource usage performance. With shorter $t_h$ values, more maneuvers are executed but average ES margin is reduced and we see multiple ES lower limit violations. With the longer $t_h$ values, less time is spent in maneuver mode, but we see higher margin and few to no limit violations.

**Planning Performance as Function of Energy Storage Urgency Factor**

We now focus on the dependence of maneuver and resource usage performance on the energy storage urgency factor, $u_{ES}$. Figure 15 shows the maneuvers executed for three different $u_{ES}$ values for a planning window of 120 minutes. The middle plot is the same as the one from Figure 12. We see that when $u_{ES} = 0$ (i.e. RASP doesn’t factor in any reward for maintaining ES margin) only 8 maneuvers are executed. When the urgency factor is increased to 10, we see one fewer maneuver.

Table 6 summarizes the average DS and ES margins in these modes. Average ES margin bottoms out at 42.3% with $u_{ES} = 0$, but doesn’t change much between the higher factors. DS margin appears to climb a little higher with $u_{ES} = 0$, possibly because the reduced emphasis on keeping ES margin allows more time for downlink.

<table>
<thead>
<tr>
<th>Mode</th>
<th>ES Urgency Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_{ES} = 0$</td>
<td>$u_{ES} = 1$</td>
</tr>
<tr>
<td>Average DS</td>
<td>74.9</td>
</tr>
<tr>
<td>Margin (%)</td>
<td></td>
</tr>
<tr>
<td>Average ES</td>
<td>42.3</td>
</tr>
<tr>
<td>Margin (%)</td>
<td></td>
</tr>
</tbody>
</table>
Figure 16 adds insight into the subpar ES margin performance with $u_{ES} = 0$. We see multiple, large ES lower limit violations, which occur at around the same times that no maneuver windows are executed in Figure 15. This shows that without explicitly rewarding ES margin maintenance, RASP tends to be too shortsighted in its planning.

**Figure 16: Resource usage (blue) and lower/upper bounds (red) for simulation run with $t_h = 120$ mins, $u_{ES} = 0$, $u_{DS} = 1$ (1 MiB = 1024$^2$ Bytes). Large energy storage lower limit violations are seen.**

**Planning Performance as Function of Data Storage Urgency Factor**

Figure 17 shows how maneuver execution varies with varying the DS urgency factor, $u_{DS}$. The middle plot is again repeated from Figure 12. We see a similar behavior to varying $u_{ES}$; when the factor is set to zero, there are stretches of time when no maneuvers can be executed. In this case, this occurs towards the end of the simulation. When $u_{DS} = 10$, we see the same maneuver drop off as in $u_{ES} = 10$, which suggests that the period of the simulation from about 6 hours to 9 hours places a particular strain on resources. This also agrees with the variation seen in Figure 12.

Table 7 shows the average DS and ES margin results for varying $u_{DS}$. We see behavior similar to varying $u_{ES}$ in Table 6. When $u_{DS} = 0$, the average DS margin is very low. The DS margin is roughly the same for the other cases. ES margin improves slightly for $u_{DS} = 0$, probably because RASP doesn’t care as much about downlinking and instead can improve ES margin.

**Table 7: Average Resource Margin for $t_h = 120$ mins, $u_{ES} = 1$, and varying $u_{DS}$**

<table>
<thead>
<tr>
<th>Mode</th>
<th>DS Urgency Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_{DS} = 0$</td>
<td>$u_{DS} = 1$</td>
</tr>
<tr>
<td>Average DS Margin (%)</td>
<td>33.6</td>
</tr>
<tr>
<td>Average ES Margin (%)</td>
<td>74.1</td>
</tr>
</tbody>
</table>

Figure 18 shows the resource usage behavior for $u_{DS} = 0$. We see that RASP does not even try to downlink until around 16 hours into the simulation, after DS has effectively saturated. This time corresponds directly to when maneuvers started being dropped in the $u_{DS} = 0$ plot in Figure 17.
Kennedy

The RASP algorithm’s execution was timed over several simulation runs, as summarized in Table 8. 72 feasible plans were generated over the course of each simulation, and the times to generate each of these plans were averaged together. Each planning instance includes the generation of 10 feasible plans via RASP’s Depth First Search and the selection of the best scoring plan from among them. These results were generated on Macbook Pro running a 2 GHz Intel Core i7 (quad core) processor, with 8 GB of RAM. We see that as the planning window increases in size, the time to generate a plan ramps up significantly, roughly proportional to the square of the planning window length. Low Earth Orbit periods tend to be around 90 minutes, planning for one orbit should take about 5 seconds on this hardware.

Table 8: Average Time to Create Successful Plan for various \( t_h \), with \( u_{ES} = 1 \), \( u_{DS} = 1 \). Time includes generation of 10 plans and selection of best.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Planning Window Length (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td>Average Time (seconds)</td>
<td>0.66</td>
</tr>
</tbody>
</table>

CONCLUSION

We present initial results from our simulation of an algorithm (RASP) for resource-constrained science planning for the MiRaTA CubeSat mission. The algorithm was demonstrated over a representative 24 hour simulation of MiRaTA’s orbit over which 19 science maneuver windows were found. The algorithm produces, from a set of simple time window inputs, consistent activity plans that allow the spacecraft to schedule radiometer-GPSRO calibration maneuvers while keeping within constraints on onboard energy storage and data storage. No human-in-the-loop involvement is needed except for the initial production of time windows inputs.

We showed that RASP successfully balances the performance of maneuvers with onboard resource usage. We found that maneuver execution performance degrades slightly as the length of the RASP planning window increases; for a 60 minute planning window 16 out of 19 maneuver windows were executed and the total maneuver execution time was 176.7 minutes, and for a 240 minute planning window these decreased to 12 out of 19 and 143.36 minutes, respectively. The average maneuver window occupation (percentage of maneuver window executed over those maneuver windows which were executed) showed no consistent dependence on planning window length. It was shown that while shorter planning windows did lead to more maneuvers being executed, this happened at the cost of small resource usage limit violations. In the 60 minute planning window case, the average Energy Storage (ES) margin was 62.9%. This margin increased to 71.1% for a 120 minute planning window, representing the fact that the 120 minute case does better at keeping ES away from its lower bound. These results show that RASP plans well over a range of planning window lengths, and that if the mission is willing to take on more risk by allowing resources to occasionally exceed their pre-set limits, more science maneuvers can be achieved.

We examined RASP’s performance when changing urgency factors for ES and Data Storage (DS). We showed that when either of these factors is set to zero, meaning that RASP does assign a reward for increasing ES or reducing DS, the algorithm performs poorly in terms of science maneuver execution and resource management. With \( u_{ES} = 0 \), the average ES margin decreased to 42.3% and only 8 maneuvers were executed. With \( u_{DS} = 0 \), the average DS margin decreased to 33.6% and only 7 maneuvers were executed. This analysis shows that that the energy margin and data downlink reward terms in the MILP objective function (Equation 1) have a significant impact on the quality of planning over a long term, and that mission operators can effectively tune to an acceptable level of risk with the ES and DS urgency factors.
Finally, we looked at time performance of the algorithm, and found that planning for the duration of roughly one low-earth orbit (about 90 minutes) takes on the order of 10 seconds for a moderately capable laptop. This is a very small ratio of planning time to execution time, suggesting that it would be feasible to implement the planner onboard a small satellite with a less capable onboard computer but better optimized software. Moreover, the limited human involvement in the RASP algorithm (the production of initial inputs) provides significant latitude for an implementation of RASP entirely in onboard software.

**Limitations of Current Algorithm Implementation**

One important limitation of the current implementation of RASP is that it is only capable of scheduling a single onboard activity at a time. The assumption for the MiRaTA CubeSat operations model used here is that science maneuvers can only physically occur one-at-a-time. Significant development would be needed to generalize the algorithm to a simultaneous, multi-activity model while maintaining computational tractability.

The algorithm does not reason about the latency of the science data it has collected and is storing onboard; data is treated as a simple bulk item that is of equal value no matter how or when it is produced. This approach is of limited value operationally because in reality we do want to be able to treat time-sensitive science data and engineering telemetry differently.

Also, the algorithm does not model energy consumption and production separately, constraining it to a simple model of constant energy usage rate by spacecraft activity mode. We used a conservative assumption here, that energy can only be produced in recharge mode. In reality we could take advantage of solar power input during other modes as well.

**Future Work**

An important next step for work with the RASP algorithm is to perform more simulation runs over additional representative MiRaTA operational time periods. During spacecraft operations, we would like to run the planner over multiple days’ time without human input, so the algorithm should be simulated over multiple days. The good resource management results in this work show promise for multi-day simulations.

We plan to adapt the algorithm for operation in a CubeSat’s onboard flight software, so as to reduce the need for continuous ground interaction for planning. The planning windows used in this simulation (on the order of one LEO orbit in duration) and the sparse communications windows for a typical CubeSat would necessitate planning operations in flight software to achieve the best mission results. We plan to implement the RASP algorithm in the C language for more efficient operation.

We want to address the identified algorithm limitations: the restriction to single activities at a time, the lack of consideration for data latency, and the lack of a separate model for energy production and consumption. In addition, further characterization is needed of the operational performance of the MiRaTA downlink radio. It was assumed here that a constant 2.6 Mbps downlink would be available over the course of a whole downlink period, however the real data rate will be reduced by operational constraints like the time required to establish the communications link.

**Acknowledgments**

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This work is also supported by NASA Earth Science Technology Office grant number NNX14AC75G and NASA Space Technology Research Grant NNX12AM30H. This work was also sponsored by the National Oceanic and Atmospheric Administration under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and not necessarily endorsed by the United States Government.

**References**


