DESIGN & DEVELOPMENT OF AN OPTIMIZED SENSOR SCHEDULING & TASKING PROGRAM FOR TRACKING SPACE OBJECTS

David Shteinman, Mark Yeo, Alex Ryan, Scott Dorrington
Industrial Sciences Group, Sydney NSW Australia

James Bennett, Michael Lachut

EOS Space Systems Pty Ltd, Queanbeyan NSW Australia

Space Environment Research Centre, Mount Stromlo ACT Australia

ABSTRACT

With a large and ever-growing number of resident space objects (RSOs) in orbit around the Earth, the efficient tasking of sensors is critical to track objects and maintain reliable state estimates of objects across a catalog. This paper describes a scheduler developed to task sensors in a way that maximizes the total utility of a sensor network, measured in terms of information gain, or the reduction in Rényi α-divergence of object state covariances. The program contains several features such as object prioritization, customizable propagators, and the capability to schedule both optical and laser sensors. The program has been fully implemented in C++ and can schedule a catalog containing over 20,000 objects (building up to 100,000 objects) with up to 6 sensors (building up to 72 sensors) in real-time. The scheduler is currently in use for catalog maintenance by the Space Environment Research Centre at its Mt Stromlo Facility in Canberra, Australia.

1. INTRODUCTION

The efficient scheduling of sensors is important for the field of Space Situational Awareness (SSA). Sensor scheduling assists the maintenance of catalogs containing orbital data on resident space objects (RSOs). The accuracy of these catalogs is imperative for issuing reliable conjunction and threat warnings, as well as developing better space situational and domain awareness.[1] The *Industrial Sciences Group (ISG)* and the *Space Environment Research Centre's (SERC)* Space Asset Management program collaborated on the development and implementation of mathematical models and software to optimize sensor scheduling and tasking for a network of sensors. The resulting program seeks to maximize the total information gain across the sensor network, which increases sensor utilization, and indirectly minimizes idle time. The program also facilitates catalog maintenance as it favors the scheduling of objects with greater covariances as demonstrated in Figure 1.

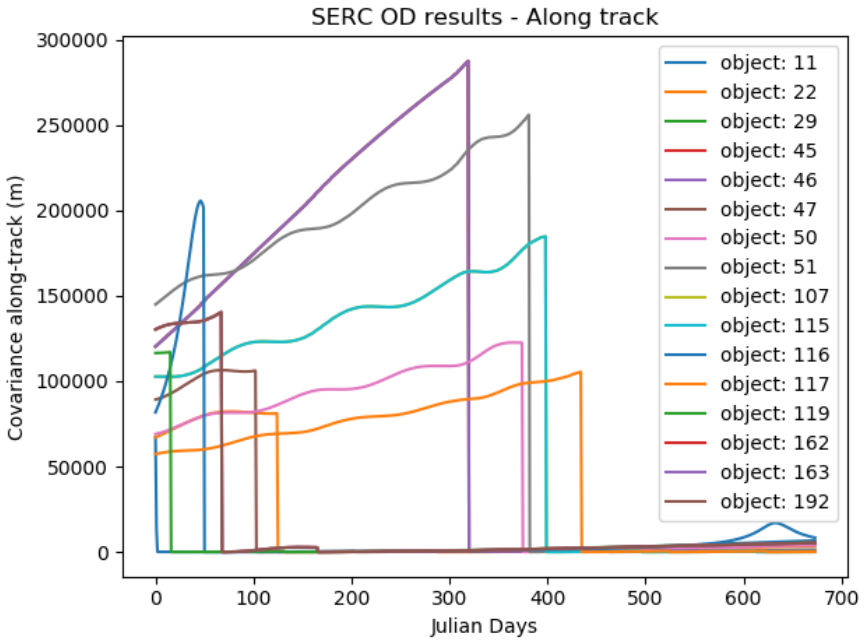

Figure 1. Object covariances over time. Drops in error are due to objects being measured.

The program uses as inputs a catalogue of space objects, with initial state vectors and covariances, and a set of active and passive optical sensors across Australia. Each object is propagated forward using an Unscented Kalman Filter (UKF) over a 12-hour period to determine visible passes that may be tracked by each sensor. The program outputs a schedule for each sensor and details the tracking times of selected objects over this period.

The program accounts for asynchronous assignment windows, computes the cumulative information gain from multiple observations, scales information gain for high priority targets, and adds constraints on laser measurements. The program optimizes sensor utilization by producing sensor-object assignments that maximize the information gain – that is, the reduction in object covariances – for each assignment window throughout the schedule. The program and algorithms were successfully tested by SERC on a catalogue of 20,000 objects with 6 sensors. It is now being used by SERC to produce a daily schedule.

The success of this project required a multi-disciplinary team with expertise in astrodynamics, statistics, information theory and software engineering. With the addition of multithreading capabilities, the software is fully implemented and tested in C++ and can schedule a catalog with over 100,000 objects (all known space debris objects) with a network of 72 sensors in real time.

The program implements Gehly's design of a multi-sensor information-gain-based scheduler, which utilizes the Rényi α-divergence to quantify the amount of information gain that would be obtained by making by a measurement.[2] The program additionally uses an auction algorithm described by Bertsekas[4] which uses an economics-based bidding approach to maximize the total reward, or information gain, made by the sensor network.

2. CHALLENGES

Processing Requirements

The practical component of scheduling imposes several additional requirements to the project, particularly when considering real-time scheduling requirements on a full catalog of objects. To generate a schedule in real-time, the program's processing speed must be faster than the time interval it simulates. Real-time schedule generation allows live measurement data to be integrated into the latest state estimates, but also involves additional overhead costs associated with sensor data processing and transfer time. Additionally, data for every object in the catalog is required to be simultaneously held in memory, as schedule generation requires the whole catalog to be processed at the same time. Additionally, it was highly preferable to meet these requirements without resorting to distributed computing, which would significantly increase the complexity of the project. Our approaches to solve these problems are outlined below.

Choice of Programming Language. The original version of the program was written in Python. However, the run-time was prohibitively slow even when utilizing multithreading and code optimization. Thus the decision was made to rewrite the program in C++. Run-time comparisons between the two versions showed that the C++ version ran roughly 100-200 times faster than the Python code, due to the inherent differences between the two languages and the ability for C++ to perform speed optimizations at compilation time.

Code Profiling and Optimization. Code profiling was performed to identify key 'problem areas' of the program that were taking up significant amounts of execution time, with optimization efforts being focused on these sections. Significant reductions in run-time were achieved by passing data structures by reference (instead of copying the whole data structure for each function call), rewriting slower methods with more efficient algorithms (particularly important with large catalog sizes), and choosing data formats with lower write-time speeds (which contributed significantly to the program run-time).

Multithreading. Multithreading was performed in sections of the code where each object was processed independently of the others (e.g. calculating object rewards over a single assignment window), which allowed for the program run-time to be significantly lowered. Measures were put in place to ensure that threads were synchronized before executing serial code (e.g. for the auction algorithm), and to ensure that the code was thread-safe.

Memory Management using Recursive Filtering. Each object within the catalog generated a substantial amount of data during run-time, which needed to be exported for later use. Retaining all data for the entire catalog would require exorbitant amounts of memory (RAM), so alternative approaches were considered. It was noted that the Kalman filter is an *optimal recursive* filter, in that the filter incorporates *all* information that is provided to it, and that the filter does not require all past data to be reprocessed each time that an estimate is made.[5] Thus, by using a UKF to estimate object states, only the most recently calculated estimates were required, so at the end of each assignment window the program could clear old data from memory without impacting future estimation. This slightly increased program run-time but allowed the program to process full object catalogs (100,000+ objects) without requiring specialized hardware.

Positive-Definite Matrix Requirements

The UKF uses the Cholesky decomposition to generate sigma points due to its numerical efficiency.[5] However, this decomposition is only stable on positive-definite (PD) matrices, which cannot be guaranteed when importing data, or throughout the duration of the program. This issue was addressed by implementing a correction algorithm on non-PD matrices to find the nearest PD matrix,[7] which allowed the program to be robust to non-PD covariance matrices.

3. PROGRAM OVERVIEW

The program takes in object state data and produces a schedule with sensor-object assignments that maximize the total network information gain. The program is split into two main parts – the Scenario Generator, which imports and pre-processes object and visibility data, and the Scheduler, which generates the schedule. This process is described in greater detail below and summarized in Figure 2.

Data Input

At the start of the program, the Scenario Generator takes in the following data:

- Object state and covariance data from the user's catalog
- Object two-line element (TLE) data
- Satellite Catalog (SATCAT), for RCS and operational status
- Naval Observatory Vector Astrometry Software (NOVAS) Bulletins, for time system conversions

Object state data is transformed into the Earth-centred inertial (ECI), true-of-date (TOD) coordinate frame and stored in the program in an object catalog. Objects with missing data are initialized using TLE data from publicly available databases, and objects with missing covariance data are given large, 'default' covariances. SATCAT and NOVAS Bulletin data are stored for later use throughout the program.

Schedule Pre-Processing

The Scenario Generator then preprocesses object and visibility data to be used later in the program during schedule generation. This increases the Scheduler run-time speed to allow it to run in real-time if required. Firstly, all objects in the catalog are 'brought forward' in time to the start of the schedule using a UKF predict method (see Section below '*Unscented Kalman Filter*'). Secondly, object states are propagated throughout the schedule in 60-second steps, and sensor-object visibilities are calculated at each interval. The following parameters are considered when calculating sensor-object visibility:

- Sensor elevation/range limits
- Apparent magnitude (based on RCS and Sun angle (sun-satellite-station), assuming the satellite is a diffuse sphere)
- Sky darkness (based on time of day and sensor location)
- Phase angle (sun-station-satellite)
- Sun angle
- Moon angle (moon-station-satellite)
- Radii < Earth's equatorial radius
- Laser link budget (laser sensors only)
- Sensor slew rate – based on previously-assigned satellite and starting position of current satellite

Schedule Generation

The Scheduler simulates measurements of all valid sensor-object pairs and generates a schedule which maximizes the total information gain made by the sensor network. Firstly, sensor-object visibilities at each time step are used to predict object passes. For each pass, measurements are simulated to predict the information gain expected if the object was measured by the associated sensor for that pass. Information gain is calculated by the decrease in object state covariance size, as measured by the reduction in Rényi α-divergence (see Section "*Information Gain*").[2]

After information gain rewards are calculated for all visible sensor-object pairs, an auctioning algorithm is utilized to assign a unique object to each sensor, such that the total reward of the sensor network is

maximized. Finally, assigned tasks are executed and the object state estimate and covariances are updated to reflect the new data.

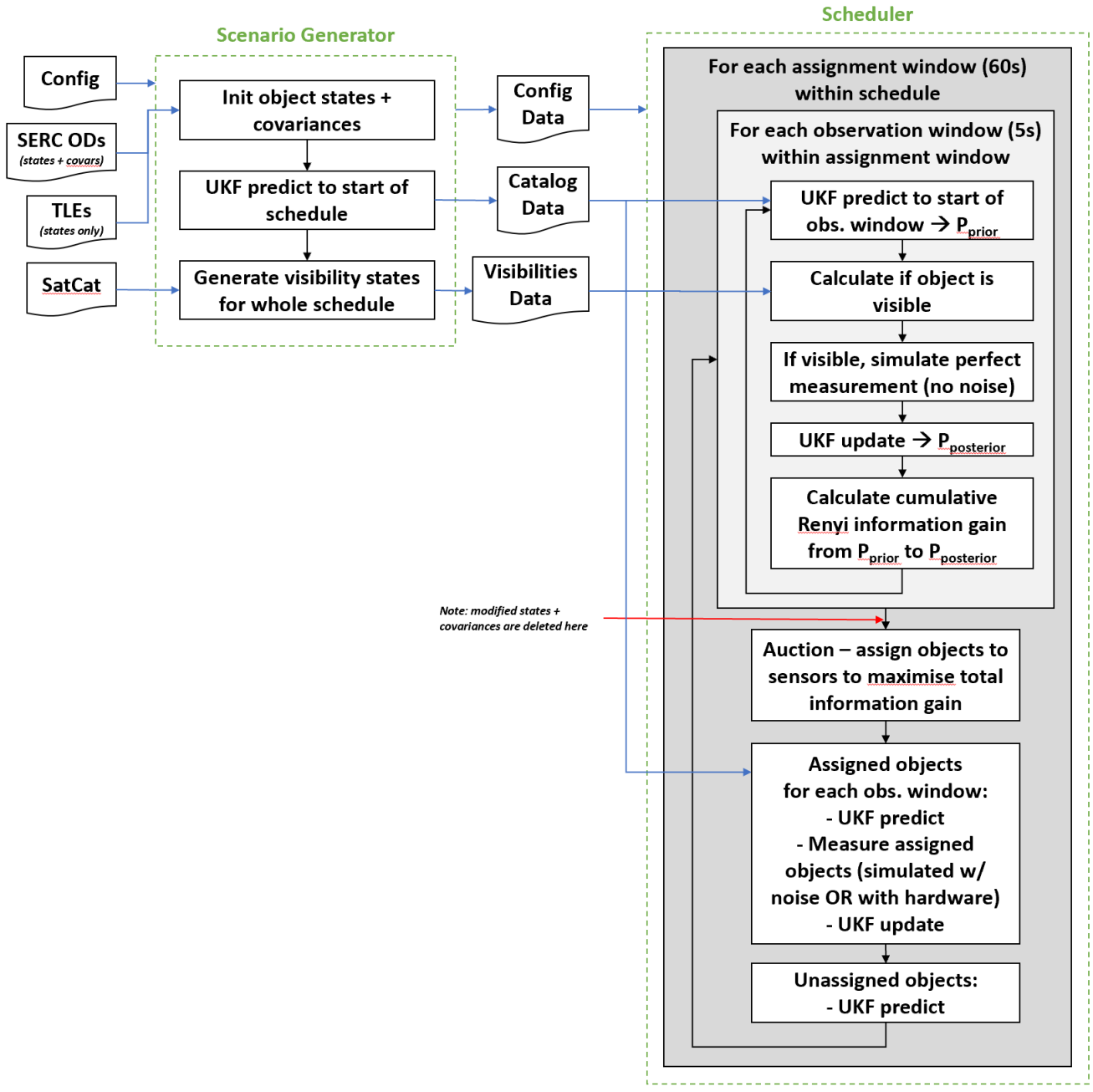

Figure 2. Flowchart of the Program.

4. PROGRAM FEATURES

The Scheduler generates a schedule that maximizes the total information gain of the sensor network. This requires the use of several features, detailed below.

Asynchronous Tasking

To minimize idle time caused by sensors having to 'wait' for other sensors to complete their tasks, a method of asynchronous tasking was developed, involving a combination of unit time-slots and larger windows to consider consecutive visibilities that would occur during a pass.

The schedule starts when the first visibility occurs (i.e. when an object is visible to any sensor for first time) and ends at the final visibility of the night. The Scheduler divides the entire schedule into 5 second *observation steps*, which represent the time resolution of the schedule. Each observation step represents a schedule slot in which a sensor can be assigned an object to measure continuously for 5s.

Rather than tasking at each observation step, selections are made over 120-second *assignment windows*. This allows passes that last longer than a single observation window to be measured continuously, thus reducing sensor slew time required to switch between different objects. There are several variables used in asynchronous tasking:

- *window_start* – Start of assignment window, set at the earliest sensor starting time
- *ti_max* – Stop time of assignment window = window_start + max_task_length
- *ti_sensor* (Unique to each sensor) – Start of when the sensor is available to start tracking within assignment window; given by the stop time of its previous task
- *min_task_length* – Minimum task length (30s). Passes shorter than this length are not considered to be valid.
- *max_task_length* – Maximum task length (120s)
- *max_sync_time* – Maximum time sensor can be ahead of first sensor (240s)

A common stop time is used for all sensors, at a fixed interval from the window start time, so sensors that start later will have a shorter duration over which to consider tracking objects. This is done to keep the scheduling of the sensors from becoming too unsynchronized. If the duration between a sensor's start time *ti_sensor* and the end time *ti_max* is less than the minimum pass duration (*min_task_length*), no passes are valid. The sensor would not select an object during this assignment window – instead it would wait for the other sensors to "catch up".

To help keep the sensors synchronized, a check is added to specify when a sensor is too far 'ahead' of the others. If in the previous assignment window, Sensor A's start time was ahead of any of the stop times of the other sensors, then tasking would be skipped for Sensor A in the current assignment window. An example of this can be found in Figure 3.

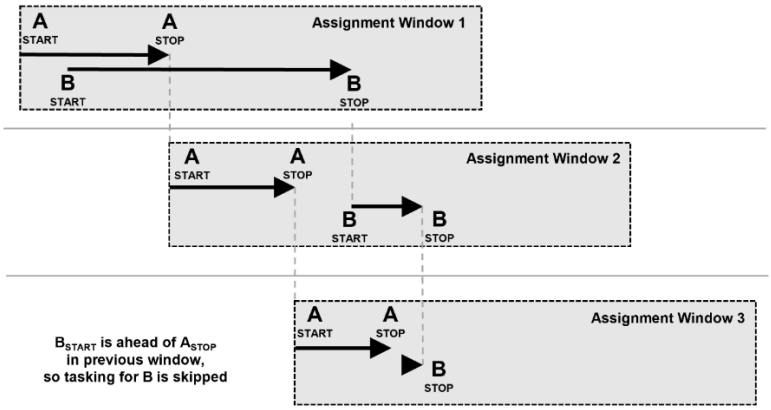

Figure 3. Diagram of synchronization constraint.

Unscented Kalman Filter

A UKF is used to calculate object states and covariances throughout the schedule, as well as optimally incorporating measurements into the current state estimates. The use of a UKF allows orbit non-linearities to be handled accurately.[8]

There are a few trade-offs in using a UKF. Firstly, calculation of covariances is performed by using a minimal set of sample points around the mean. These sigma points are individually propagated forward in time when performing the UKF predict method, which adds to the run-time required to process each object at each time-step. This was addressed by using C++, code optimization, multithreading and by restructuring the UKF predict process to calculate covariances with a faster propagator (see Section below "*Propagators*").

Secondly, the use of a UKF places additional requirements on the propagator used, by the fact that the propagator must have 'state-to-state' functionality – i.e. it must be able to take in an object state (in ECI TOD) and output the propagated object state (in ECI TOD). This requirement is due to the way that sigma points are handled – the initial covariance is used to generate thirteen sigma point states, each point is propagated forward, and the modified states are combined into a covariance using the unscented transform. This process requires the propagator be able to take in object states in ECI-TOD and output object states in ECI-TOD. A unique problem was posed when using propagators without this characteristic – for example, SGP4. The Section "*Propagators*" outlines the approach taken to work around this restriction imposed by the UKF.

Propagators

The UKF predict method contains a system model that uses propagators to specify how the system behaves over time. Modularity and configurability were emphasized with regards to propagator integration with the program. This was done to facilitate future program maintenance and to help balance accuracy with run-time speed. For example, a simple analytical propagator would deliver poor accuracy but a fast run-time, whereas a complex numerical propagator would deliver the opposite. Careful configuration allows these characteristics to be leveraged in sections where speed is more critical than accuracy, and vice versa.

Also, the UKF predict method has been modified to allow two propagators to be used in parallel. The UKF predict method uses a fast 2-body propagator to calculate the new covariance, while the new state is calculated using a slower, but more accurate propagator – as shown in Figure 4 and Figure 5. This modified setup reduces program run-time but also preserves accuracy of calculated states. Additionally, this setup allows for a 'non-state-to-state' propagator (such as SGP4) to be used (Figure 4).

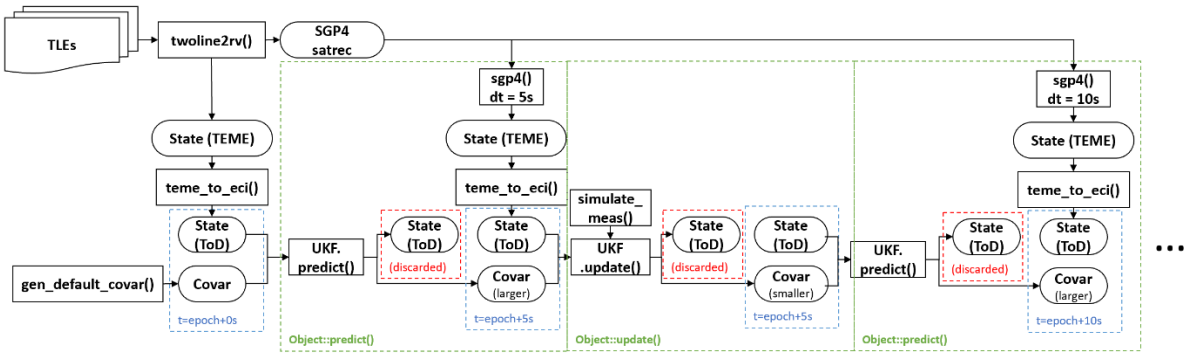

Figure 4. Flowchart of the Scheduler UKF, using SGP4 as a propagator.

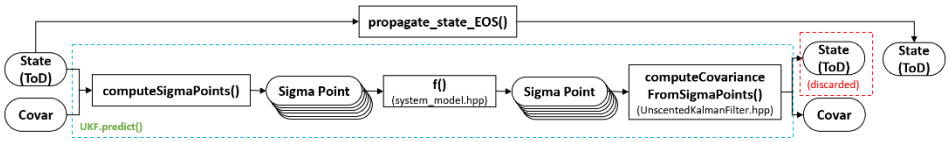

Figure 5. Flowchart of the UKF Predict Method, with Added State Recalculation.

Reward Function

Within each assignment window, the reward for each object-sensor pair is calculated from the cumulative information gain over each observation step within a pass. At each observation step, the UKF-predict method is performed to predict the *prior* state and covariance at the current time (prior to measurement). A perfect observation is then simulated, using from the predicted state vector transformed by the measurement function h(). This measurement is then used within the UKF-update step to compute the *posterior* state and covariance. The information gain is then calculated as a function of the prior and posterior covariances

Information gain. Information gain is calculated using the Rényi α-divergence between the prior and posterior state covariances. An assumption is made that the Rényi function has an additive property, such that the total information gained over several observations is the cumulative sum of the incremental information gains from each observation. The program implements the equations described by Gehly[2] and represents the reduction in covariance due to the fusion of measurement data into the current state estimate.

Object Prioritization. Object prioritization is performed using a "Tactical Importance Function" τ (tau) that can be used to scale the reward function for different objects based on their importance or priority, also described by Gehly.[3] This allows objects to be prioritized by their attributes (orbital regime, size), custom prioritizing (e.g. particular objects) or a combination of these categories. The τ (tau) function also acts as a scaling factor to adjust the gradient of prioritization, i.e. how much prioritization scales the original rewards.

Auction-style Task Selection

Once all rewards have been calculated, object-sensor pairing is performed by an auction algorithm, implementing the algorithm described by Bertsekas.[4] This algorithm uses a series of bidding rounds to maximize the total reward of object-sensor pairs, producing a schedule that maximizes the total information gain made

by the sensor network. These selections are then executed by sensors, resulting in improvements in the accuracy of object state estimations, as shown above (Figure 1).

5. PROGRAM PERFORMANCE

This section outlines the program's run-time performance when running a full schedule, with relation to the number of objects processed and the number of sensors within the sensor network. Testing was performed on two machines with varying specifications to characterize the program's run-time performance, summarized on Table 1.

Table 1 – Test machine specifications

	Machine 1 (Consumer Grade Laptop)	**Machine 2 (Commercial Desktop)**
CPU	Intel Core i7-8650U	Intel Core i7-8700
Clock Speed	1.90GHz	3.2GHz
# Cores	4	6
Memory (GB)	8	32

Run-time Variable Dependence

As expected, the program increases in run-time linearly with increasing number of objects, as shown in Figure 6. Additionally, run-time increases slightly more than linearly as the number of sensors is increased, as shown in Figure 7. These values were calculated with Machine 1.

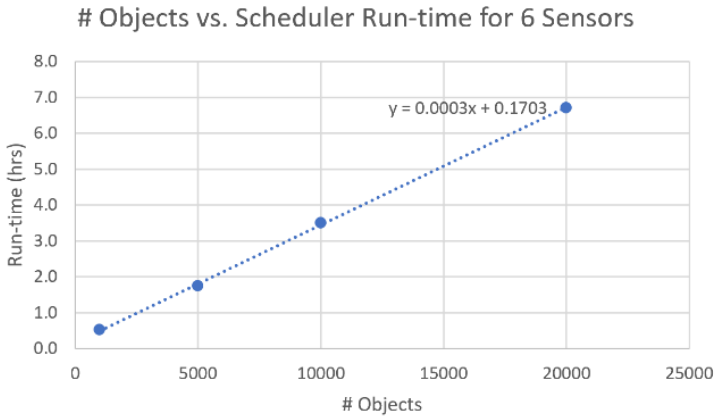

Figure 6 – Scheduler run-time for an increasing number of objects (Machine 1)

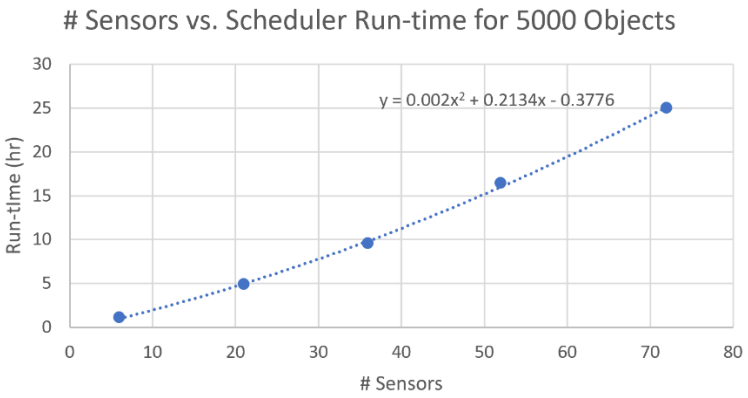

Figure 7 – Scheduler run-time for an increasing number of sensors (Machine 1)

Multithreading

The program involves processing each object in the catalogue in sequence and independently of each other, which is classified as a High Throughput Computing (HTC) problem. HTC problems can be parallelized with multithreading and/or distributed computing. At present only multithreading is implemented, but distributed computing may be considered in the future for further performance enhancements.

To implement multithreading for processing the object catalogue, a thread pool was created with its size dependent on the number of hardware processing cores available and whether hyperthreading was available. Each object was placed in the thread pool queue, and then each thread would take an object off the queue and execute the necessary processing, independently of each other. Once all the objects were processed, the threads would end and the program would continue as normal.

The following graphs show the decreasing run-time of the scheduler as the number of threads increases, where “Ideal” refers to the theoretical minimum run-time based on the increased resources. This is inversely proportional to the number of threads, such that 2 threads halve the original run-time, 3 threads use a third the original run-time and 4 threads is a quarter of the original run-time. When increasing the number of objects, the improvement in run-time thread speed-up is increased, as shown across Figures 8-9. Similarly, when increasing the number of sensors, the same effect can be seen, as shown across Figures 9-10. This is due to the fact that the ratio of serial vs. parallel code decreases as the number of objects and sensors increase.

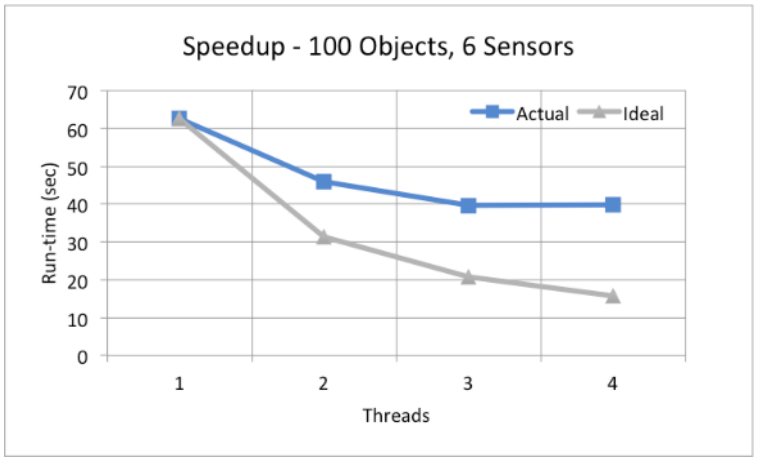

Figure 8 – Scheduler run-time speedup for 100 Objects and 6 Sensors running up to 4 threads

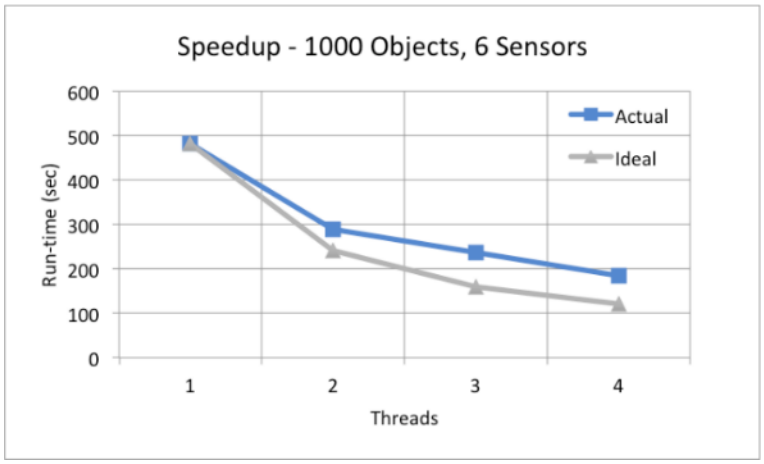

Figure 9 – Scheduler run-time speedup for 1000 Objects and 6 Sensors running up to 4 threads

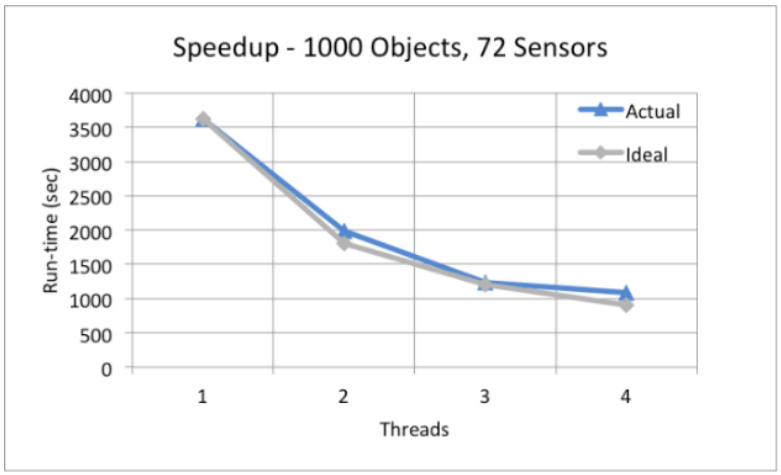

Figure 10 – Scheduler run-time speedup for 1000 Objects and 72 Sensors running up to 4 threads

The run-times on Machine 2 are shown in Table 2. Estimated values are shown in *italics* and are estimated based on the speed-up ratios calculated in Figures 8-10.

Table 2 – Multithreaded run-times on machine 2 (estimated values are in *italics*)

Threads	Objects	Sensors	Run-time (sec)	Run-time (hrs)
1	20,000	6	4,356	1.21
1	100,000	6	18,728	5.20
4	100,000	6	*7,154*	*1.99*
1	20,000	72	52,853	14.68
4	20,000	72	*15,846*	*4.40*

Future Improvements & Applications

To further improve the run-time speed of the program, several approaches may be made. Firstly, simply using a faster server class machine with more cores would provide a faster run-time, particularly when processing catalogs with many objects or using a large sensor network. If further improvement in run-time is required, then further parallelization of the code could be performed to enable distributed computing across multiple machines. This should only be attempted once all avenues of optimization and parallelization have been exhausted in the multithreaded program.

A future improvement to the Scheduler will include *dynamic scheduling*. This will allow the schedule to be updated and changed in real time due to changes in weather conditions effecting visibility (for example).

A significant application of the Scheduler *Visibility Module* will occur in late 2019 when SERC tests laser manoeuvring of space objects. This a key goal of the Space Asset Management program in SERC, (Research Program 3) "*At SERC the goal is to contribute to the mitigation of the debris environment by using high powered continuous wave lasers to apply photon pressure to perturb objects on orbit so that they avoid a collision. To demonstrate this goal, several operational components are needed to ensure a debris object is not moved into a less favorable trajectory…This requires knowledge of all objects and their behavior in the vicinity of the demonstration."* [1] Using a set of 50 target objects chosen by SERC the visibility module will be used in simulation (before the laser campaign) and during the campaign on actual data. The objective is to use tracking data of the visible objects before/after the laser firing to provide a *statistically valid confirmation and quantification* of manoeuvre detection. We will also optimize the targeting campaign by quantitatively describing effect of **inputs** (laser power, duration and geometry of engagement, number of passes, ballistic coefficient of object) on the **response** (change in orbital parameters in an object due to photon pressure).

6. CONCLUSION

A Case study has been presented of the design and development of an optimised sensor scheduling and tasking program for tracking space objects. The program can schedule a catalog containing over 20,000 objects (building up to 100,000 objects) with up to 6 sensors (building up to 72 sensors) in real-time. The scheduler is currently in use for catalog maintenance by the Space Environment Research Centre (SERC) at its Mt Stromlo Facility in Canberra, Australia.

We have described the major design features of the scheduler that tasks sensors to maximise the total utility of the sensor network measured in terms of information gain or the reduction in Renyi divergence of object state covariance. We also described the main features of the program such as object prioritisation, customizable propagator, and the capability to schedule both optical and laser sensors.

Due to the practical nature of the project, several unique challenges were faced, particularly with regards to processing a full object catalog while meeting real-time scheduling requirements. These challenges were solved through a combination of different approaches, including the choice of programming language, program optimisation and structuring the program to be able to be multithreaded. This speed advantage from parallelisation has been quantified, showing that multithreading gives increased benefits as both catalog size and sensor network size grows.

Future work and applications are described including the use of the visibility module of the Scheduler for a laser targeting campaign to occur in late 2019 soon to test manoeuvring of space objects by laser photon pressure.

7. ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of the Cooperative Research Centre for Space Environment Management (SERC Limited) through the Australian Government's Cooperative Research Centre Program. The authors give special thanks to SERC senior management David Ball and Stephen Gower for their support and consultations throughout the project. We also thank Alex Pollard of Electro Optical Systems for his advice on software improvements.

REFERENCES

[1] Bennett, J. et al, *Progress in a new conjunction and threat warning service for Space Situational Awareness*, in Advanced Maui Optical and Space Surveillance Technologies Conference. 2018: Maui, Hawaii.

[2] Gehly, S. and J. Bennett, *Distributed Fusion Sensor Networks for Space Situational Awareness*, in 68th International Astronautical Congress. 2017: Adelaide, Australia.

[3] Gehly, S. and J. Bennett, *Incorporating Target Priorities in the Sensor Tasking Reward Function*, in Advanced Maui Optical and Space Surveillance Technologies Conference. 2015: Maui, Hawaii.

[4] Bertsekas D., 2001, *Encyclopedia of Optimization*, Vol. I. Kluwer, Dordrecht.

[5] S. J. Julier and J. K. Uhlmann, *New extension of the Kalman filter to nonlinear systems*, in Signal Processing, Sensor Fusion, and Target Recognition VI, vol. 3068 of SPIE Proceedings Series, pp. 182–193, July 1997.

[6] Maybeck, Peter S. *Stochastic models, estimation and control*, Academic Press New York, 1979.

[7] Higham, N.J., 1988. Computing a nearest symmetric positive semidefinite matrix. LinearAlgebra Appl. 103, 103–118.

[8] S.J. Julier and J.K. Uhlmann. *A New Extension of the Kalman Filter to Nonlinear Systems*. In Proc. Of AeroSense: The 11th Int. Symp. On Aerospace/Defence Sensing, Simulation and Controls.,1997.