AUTONOMOUS PERFORMANCE MONITORING SYSTEM: Monitoring and Self-Tuning (MAST)

Chariya Peterson
Computer Sciences Corp. 7700 Hubble Drive,
Lanham-Seabrook, MD 20706, cpeters5@csc.com

Nigel A. Ziyad
NASA/GSFC Code 588,
Greenbelt, MD, 20771, Nigel.Ziyad@gsfc.nasa.gov

Abstract

Maintaining the long-term performance of software onboard a spacecraft can be a major factor in the cost of operations. In particular, the task of controlling and maintaining a future mission of distributed spacecraft will undoubtedly pose a great challenge, since the complexity of multiple spacecraft flying in formation grows rapidly as the number of spacecraft in the formation increases. Eventually, new approaches will be required in developing viable control systems that can handle the complexity of the data and that are flexible, reliable and efficient. In this paper we propose a methodology that aims to maintain the accuracy of flight software, while reducing the computational complexity of software tuning tasks. The proposed Monitoring and Self-Tuning (MAST) method consists of two parts: a flight software monitoring algorithm and a tuning algorithm. The dependency on the software being monitored is mostly contained in the monitoring process, while the tuning process is a generic algorithm independent of the detailed knowledge on the software. This architecture will enable MAST to be applicable to different onboard software controlling various dynamics of the spacecraft, such as attitude self-calibration, and formation control. An advantage of MAST over conventional techniques such as filter or batch least square is that the tuning algorithm uses machine learning approach to handle uncertainty in the problem domain, resulting in reducing over all computational complexity. The underlying concept of this technique is a reinforcement learning scheme based on cumulative probability generated by the historical performance of the system. The success of MAST will depend heavily on the reinforcement scheme used in the tuning algorithm, which guarantees the tuning solutions exist.

1. Introduction

Some of the problems encountered during the development of a control system are the uncertainty in the application domain and the balancing between efficiency and complexity of the system. In a large and complex problem such as distributed spacecraft, the accuracy of the control software system depends on how much information about the problem can be modeled into the system. The more information taken into account, the more complex the system becomes, leading to higher computational cost. Moreover, the task of maintaining long-term performance of the control software onboard spacecraft can be a major factor in the operation cost of future multiple spacecraft missions.

In the control of distributed spacecraft flying in formation, conventional control algorithms are very complex due to large number of variables and the interaction among individual control systems in the formation. This makes formation maintenance and control of future constellations

or distributed spacecraft a great challenge, since the complexity of a spacecraft formation grows non-linearly as the number of spacecraft in the formation. New approaches are required that result in viable control systems that can handle the complexity of the data and that are flexible, reliable and efficient.

In this paper we propose the Monitoring and Self-Tuning (MAST) algorithm that aims to reduce the complexity of onboard software by dealing appropriately with uncertainty. MAST uses an approach based on the reinforcement learning scheme that can be applied to various dynamics of spacecraft such as onboard attitude self-calibration, or formation keeping. This type of machine learning approach has a much wider operational range than the conventional batch least square or filter techniques. This is simply because; the learning system can be designed to automatically accumulate and reuse its past activities, which will enable the system to react and adapt to changes in the environment. This approach is therefore appropriate for problems with large degree of uncertainties. Moreover, this technique is not critically dependent on the detailed knowledge of the software being tuned. As a result, some of the technical restrictions generally required in conventional techniques such as linearity, or conditions on process and measurement noises are not required if a learning algorithm is being used.

MAST is an extension of a project at NASA/Goddard Space Flight Center (GSFC): Autonomous Model-based Trend Analysis System (AMTAS) [1]. The objective of AMTAS is to monitor the health and safety of spacecraft hardware and subsystems. MAST extends this objective to dynamic applications by proposing to apply techniques developed in AMTAS to onboard flight software, which control the dynamics of spacecraft. In general, the performance of flight software can be meaningfully defined as a measure of the closeness between the observed and the predicted state of the systems. These quantities are usually referred to as *residuals*. Understanding the uncertainty underlying the residuals, identifying its controlling factors, and quantifying the propagation of these factors through the model for the system can lead to an improvement in the performance of the software.

MAST algorithm consists of two main parts: a *predictor* and a *tuner*. The predictor is a real-time dynamic system that performs the monitoring task, coupling with the software it is monitoring, taking as input the states of the software at regular time intervals. The step size of the sampling time varies depending on the parameters being monitored. The state of the predictor represents the performance of the software. When the software performance is found to approach a given limit, the tuner will be activated. The tuner is a closed-loop learning algorithm guided by a reinforcement scheme, which is generated by an uncertainty handler. The goal of the tuning process is to minimize a *cost function.* During each cycle the values of the model parameters being tuned are increased or decreased, depending on the outcome of the previous few cycles. With the adjusted parameters, the software performance is recalculated and the next cycle begins. The rate of convergence of the tuning process depends on the reinforcement scheme used to score how successful the adjusted parameters are towards the tuning goal. If the reinforcement scheme is completely impartial, then the learning algorithm is simply a random search. On the other extreme, a reinforcement scheme that always scores perfectly is equivalent to the conventional gradient (steepest descent) method. It should be noted that the tuner is an off-line algorithm running in parallel and isolated from the routine operation of the flight software. Not until the tuning goal has been reached, that the software will be updated with the new values for the model parameters. Hence, the tuner may be performed on the ground or on an onboard computer.

In this paper we will discuss two *possible* applications of MAST: the attitude monitoring and self-calibration (ASCAL), previously proposed in [2] and an application of MAST to formation control. In the first application, the accuracy of attitude software shall depend on, among other things, how accurate its sensor models are. Sensor models are generally a function with parameters representing relevant uncertainties such as bias, scale factor or misalignment. In the

beginning, these parameters are set at certain pre-calibrated values and are manually tuned and updated periodically throughout the life of the spacecraft. Some tuning processes are routine activities, while others are elaborated and performed on ground by attitude specialists. In this proposed application, MAST will automatically monitor and tune a set of sensor parameters. For further readings on standard attitude calibration procedures, see for instance [3-6].

In the second example, we propose an application of MAST to the maintenance of a future mission of large formation. The task of controlling a number of spacecraft to fly in formation is more complicated than controlling a single spacecraft. One problem that may be encountered in the development of formation control algorithms for large formation is the complexity that arises from the high degree of freedom of the system. In practice, the conventional state-space representation approach is manageable only for formation of a small number (2-3) of spacecraft. The complexity becomes very high in a large formation, which makes the control algorithm computationally intensive. Moreover, uncertainties in the system models or from environmental disturbances can be propagated and magnified. To correct these errors the control system has to be tuned often and regularly. Hence, the task of keeping the formation intact requires continuous monitoring and adjusting the position of each individual spacecraft. Hence it is more desirable to perform this task onboard, and hence, efficient and fast algorithms for the real-time solution of such a large-scale optimization problem are needed.

The organization of this paper is as follows. Section 2 describes the architecture of MAST including the interface between onboard software, the predictor, and the tuner. Section 3 describes the formulation of the monitoring mode including the predictor and its interface with input software being monitored. Section 4 describes the tuning mode. Section 5 describes the formulation of the learning system and its reinforcement scheme. Section 6 discusses the two examples: ASCAL and a formation maintenance methodology using MAST.

2. MAST Architecture

There are two different modes in MAST: The *monitoring mode* and the *tuning mode*. The monitoring mode consists of the control software being monitored and the predictor, both running in real time. Figure 1 demonstrates the connection between the predictor and the software. The predictor is the part of MAST that is dependent on the software being monitored. It is necessary that, in order to monitor and diagnose the problem accurately, the predictor must understand the nature of the software it is interacting with. A model for the predictor is described in the next section.

The tuning mode consists of three components connected in a closed-loop: an off-line copy of the software being monitored, the evaluator, and the tuner. Their interface is demonstrated in Figure 2. The evaluator measures the convergence of the tuning solutions and the tuner makes appropriate adjustment to certain model parameters of the software guided by a reinforcement learning scheme. In general, the reinforcement learning scheme can be generated by various uncertainty handling technique. In MAST, the scheme is based on the Local Dempster-Shafer theory (LDS) which is a modification of the Dempster-Shafer theory of belief and evidence [7,8]. LDS was originally developed for AMTAS diagnosis process [1,9]. It is specifically developed for problems with a large number of variables. As opposed to the predictor, the evaluator and the tuner are generic processes that do not require in-depth knowledge of the software being tuned. Their basic requirements are a set of software parameters to be tuned and an appropriate cost function that measures the inaccuracies of the software. The evaluator evaluates and scores the result of each cycle by monitoring the effect of the parameter adjustment on the cost function. Based on this score, the tuner continues to adjust the parameters until the process converges.

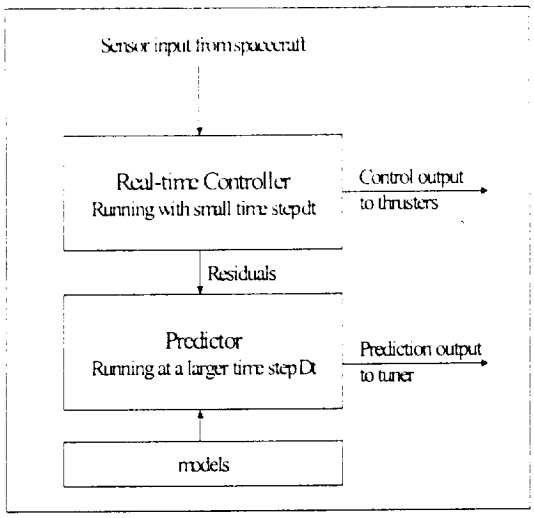

Figure 1. Monitoring Mode

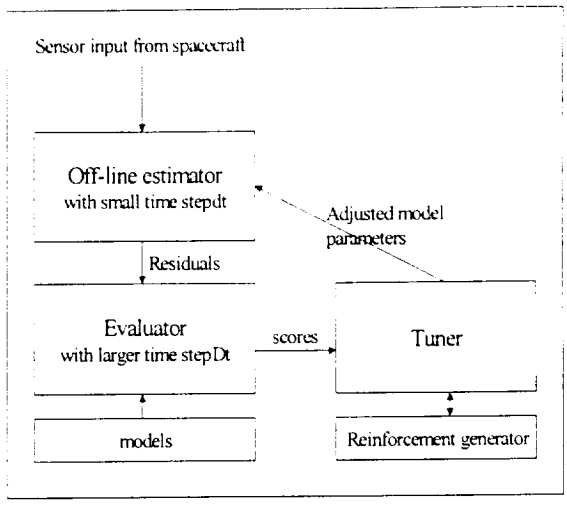

Figure 2. Tuning Mode

3. Monitoring mode

This mode is performed during normal operation. Let x denotes the state vector estimated by the software and s denotes the vector of sensor parameters being monitored and calibrated. Assume that an expected state vector x_a is given. x_a may be obtained in various ways depending on the software and on sensors and parameters being monitored. Let the software be driven by the dynamic system

$$\begin{aligned} \dot{x}(t) &= f(x(t)) + u(t) \\ z_{r,k} &= G(s_r, x(t_k)) + w(t_k) \end{aligned} \tag{1}$$

where $z_{r,k}$ is the measurement for sensor r at time t_k, and s_r is the parameter vector associated with the model of measurement r. The process noise u and measurement noise w is assumed to be uncorrelated white Gaussian with zero mean. During the monitoring mode (normal operation) the parameter vectors s_r are constant.

The performance of (1) is observable from the deviation of certain quantities, such as state residuals $x - x_a$, or sensor residuals, $z_{r,k} - G(s_r, x_a(t_k))$. Let ξ represents the vector of the desired residual observations. The monitoring process is then defined via a tracking process, i.e. the linear dynamic of ξ and its slope $\dot{\xi}$:

$$\begin{aligned} \xi(t_{K+1}) &= \xi(t_K) + \Delta t \cdot \dot{\xi}(t_K) + \tfrac{1}{2}\Delta t^2 \nu(t_K) \\ \dot{\xi}(t_{K+1}) &= \dot{\xi}(t_K) + \Delta t \cdot \nu(t_K) \end{aligned}$$

where ν is a zero mean white Gaussian acceleration noise. The time step $\Delta t = t_{K+1} - t_K$ for residual samplings may be larger than the time step of the input system (1). Let $\hat{x} = [\xi \quad \dot{\xi}]^T$. Then the state-space representation of the predictor can be written as

$$\begin{aligned} \hat{x}(t_{K+1}) &= A \cdot \hat{x}(t_K) + V \cdot v(t_{K+1}) \\ \hat{z}_K &= H \cdot \hat{x}(t_K) + \gamma_k \end{aligned} \tag{2}$$

where $A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$, $V = [\Delta t^2/2 \quad \Delta t]$, $H = [1 \quad 0]$.

Note that, the measurement $\hat{z}_K$ represents the residual sampling while the state $\hat{x}(t_K)$ measures the level of performance of (1) during the time t_K. A propagation of $\hat{x}(t_K)$ predicts if and when the performance of (1) approaches an acceptable threshold. System (1) and the predictor (2) connect as shown in Figure 1. Higher order derivatives of the state residual can be included in $\hat{x}(t_K)$ in a similar way. In which case, we would have a higher order predictor. Higher order derivative may be crucial for software systems that are sensitive to uncertainties in measurement models, which is generally the case for a highly non-linear, chaotic or unstable systems.

4. Tuning Mode

The tuning process is a closed-loop algorithm composed of the software to be tuned, e.g. the dynamic estimator (1), an evaluator that evaluates the outcome of the tuner during each cycle, and a tuner, which is a learning system that adjusts model parameters based on the evaluation. The evaluator takes as input the estimated states of (1) and a nominal state given by a model. The evaluation is based on the effect of the tuner on a cost function, typically written as

$$J(\hat{x}) = \hat{x}^T \cdot M \cdot \hat{x}$$

where M is a symmetric positive definite quadratic form that provides the weight and relations among the residuals. This weight reflects the importance and sensitivity of each state variable in the tuning process.

Remark: For the tuning process to be fully independent of the application software there should be a preprocessor that properly initializes the tuner when it is activated. The preprocessor identifies and initializes the parameters, step size, and parameter ranges. For instance, the parameter ranges are chosen in such a way that the region is void of any singularity and at least one solution exists. This knowledge can be given *a priori* by human experts in terms of rules or a belief measure on the set of parameters, their ranges, and step sizes. Moreover, with proper learning capability, these values can be based on the past experiences of the tuner. For instance, these measure functions can be updated each time the system completes a tuning task, whether it is successful or not. This preprocessor is highly dependent of application domain and will not be discussed in this paper.

5. Reinforcement Learning System

Reinforcement learning is the type of learning that is popular among most current researches in machine learning and statistical pattern recognition. Other popular type of learning systems such as artificial neural network, requires *a priori* training from examples provided by an experienced supervisor. Such systems are not quite appropriate for problems involving learning from interaction. In interactive problems it is often impractical to obtain examples of desired behavior ahead of time, which are both correct and representative of all the situations to which the system has to react. In an unknown situation, where learning is most beneficial, the system must be able to learn proactively from its own experience.

During the tuning process, the parameter adjustment is based on the rate of convergence (or divergence) of the residuals during the previous two (or more) cycles. Assume there are n sensor parameters to be adjusted, and each parameter can be increased or decreased by a fixed quantity. This corresponds to $H = \sum_{i=0}^{n} 2^i \binom{n}{i}$ possible ways of adjustment. Each choice is a set of parameters with a + or – sign to denote if the parameter is being increased or decreased. For instance, an increase in parameter a and a decrease in parameter b is represented by the "signed" set $\{a_+, b_-\}$. During each loop K, the set H of all possible choices is indexed by a cumulative probability distribution p_K which is computed using the Local Dempster-Shafer (LDS) theory. The learning process in the tuner is precisely the mechanism that adapts p_K to obtain the new index p_{K+1} for the next cycle.

Due to space limitation, we will describe a simpler algorithm based on the Dempster-Shafer (DS) theory, which we modified to suit our tuning problem. For a more in-depth discussion of the LDS theory see [2]. DS theory is defined on a set of n elements. Recall that, H is a set of all possible ways of modifying model parameters being tuned. A mass function on H is a probability function that assigns a degree of belief to each of its element. The mass function satisfies the following conditions

$$\sum_{A \subseteq H} m(A) = 1, \text{ for } A \neq \varnothing \quad \text{and} \quad m(\varnothing) = 0$$

Two mass functions m_1 and m_2 on H can be combined into a single mass function $m_1 \otimes m_2$ by the Dempster composition rule:

$$m_1 \otimes m_2(A) = \sum_{B \cup C = A} m_1(B) m_2(C) \Big/ (1 - \sum_{B \cup C = \varnothing} m_1(B) m_2(C)), \qquad \text{for } A \neq \varnothing$$

$$m_1 \otimes m_2(\varnothing) = 0.$$

These mass functions are used to generate the degree of belief associated to each element of H. A belief function generated by a mass function m is defined as:

$$p : H \rightarrow [0,1];\ b(A) = \sum_{B \subseteq A} m(B)$$

where the union between two signed sets is obtained by "adding" all elements in the two sets according to their sign. This way, every subset of the form $\{a_+, a_-\}$ will all be cancelled out. In statistical terms, the belief function is a cumulative probability on H.

During each tuning cycle, the belief function is evaluated and used to index the set H. If the resulting residuals are found to decrease with a faster rate or increase with a lower rate, the tuner will re-compute the next belief vector p_{K+1} by applying a positive learning algorithm described in [1,9]. The new index will strengthen the performance in the previous cycle. Conversely, if the residuals performed in the opposite manner, then the negative learning algorithm will be applied, resulting in lessen the degree of belief on the failed action.

The learning process discussed above is the simplest application of the (modified) DS theory to the tuner. In practice this algorithm can be enhanced in various ways to increase the performance and robustness of the tuner. First, the localization of the DS theory on H defined in [1,9] will reduce the size of search space. Second, the size of parameter increment may be decreased as the residuals begin to converge. Third, the use of hierarchical or multilevel learning systems

accelerates the learning process (more so for the initial rate of learning) and simplifies the structure of the tuner in each layer.

Remark: In some situation, the dynamics driven the software may have a hierarchy structure. A typical example: in a formation with complex topology, it may be more convenient to partition the system into layers of homogeneous sub-formation. In which case, the control algorithm will have to be partitioned accordingly. Hence, the set H will also be required to have a hierarchical structure to support the hierarchy of the control software. A hierarchical version of DS theory can be defined in a natural way, and the parameter tuning is performed in a sequence of steps. First, the highest level in the set H is selected, following by a lower level. This procedure continues until the last level is reached. This hierarchical structure will reduce the size of the search space in each layer, and hence enhance the performance of the tuning system. The third component in the tuning mode is the evaluator. Its important task is to diagnose the problems that predictor predicted. This corresponds to determining, based on the residual data alone, which parameters in the software need adjustment, and what are the "safe" ranges that these parameters may vary. Such information must be determined prior to the tuning process. Usually, expert knowledge can be encoded in some form, such as rules.

6. Example 1: ASCAL

During an attitude sensor calibration, where both states and model parameters are simultaneously solved for, it is natural to consider extended state vectors consisting of both attitude and sensor parameters. However, including sensor parameters as part of the state will introduce additional non-linearity into the system, making it more complex and too costly to run onboard. An alternative approach is to apply MAST to adjust these parameters incrementally. During each cycle, sensor parameters are adjusted and attitude and sensor residuals are computed. Using different combination of sensors and gyroscope, two or more attitudes are estimated. The predictor monitors and predicts the values of the residuals using conventional prediction algorithms such as the dynamic predictor given in Section 3, or standard regression and extrapolation. When it is discovered that the residuals will exceed a given threshold sometime in the future, implied by an inconsistency in the estimated attitudes, the tuning mode will be activated. In the tuning mode, the evaluator will diagnose the inconsistencies and create one or more calibration goals, usually expressed as "which measurement parameters are needed to adjust the ranges for the appropriate calibration algorithm". The tuning process is then planned and scheduled. In a spacecraft where one or more sensors need regular calibration, or where computing resource is stringent, the predictor may be replaced by a fixed schedule or by a cron table.

The calibration process is an iterative process, where sensor parameters believed to be in error are adapted on the basis of the system experience with a goal that the mean of all residuals converge to zero. The calibration procedure depends on the types of sensors available onboard. If there are sufficient number of redundant sensors, a standard technique is to compare the attitude determined by the measurements from a set of sensors including the sensor to be calibrated with those determined from a different set of sensors with at least equal or higher accuracy. On the other hand, if there are no redundant sensors of high enough accuracy, then the procedure usually involves more in-depth analysis. In this paper, we assume there is at least one accurate sensor such as a CCD. Typically, CCD is chosen as the standard frame of reference and generally does not need calibration. In this case, we may calibrate other sensors by comparing the resulting estimated attitude and sensor residuals with that determined from the CCD. Any inconsistency that occurs indicates that there are errors in one or more model parameters.

For current missions, the gyro scale factor calibration task has to be done manually and regularly by attitude specialists. MAST can be applied to this problem if there is sufficient planning

capability on board. The gyro scale factor parameter is calibrated by inspecting changes in attitude during a planned maneuvering. Having an autonomous planner and scheduler onboard will enable the system to piggyback gyro scale factor calibration during routine spacecraft maneuvering.

7. Example 2: Formation Keeping

Formation control architectures are being developed for various future missions and several approaches are being investigated. One of the research efforts in this area at the Goddard Space Flight Center is the formation flying for the New Millennium Program [10, 11] designed for Earth Orbitor 1 (EO-1) spacecraft flying in formation with the Earth Observing System-AM1 (EOS-AM1). The formation of EO-1 and EOS-AM1 involves position maintenance of the two spacecraft relative to measured separation errors. This involves the use of an active control scheme to maintain the relative positions of EO-1 (chaser) with respect to EOS-AM1 (target). This formation structure is specifically designed for the EO-1/EOS-AM1 formation, which involves only two spacecraft. With care, conventional control algorithms can be used effectively in such a small formation. For a large formation, the complexity rises very rapidly and eventually conventional algorithm will break down and new approach will be needed. GSFC and Stanford have form a partnership to develop the Autonomous Control System (AutoCon) architecture which employs innovative use of fuzzy logic and natural language to resolve multiple conflicting constraints and autonomously plan, execute and calibrate routine spacecraft orbit maneuvers. The underlying control algorithm is a robust autonomous closed-loop three-axis system. However, it is still not clear if AutoCon will be feasible for the control of a large formation. Our main objective in this application of MAST is to improve on our machine learning approach to work with or integrate into AutoCon environment. See also [12] for another approach to formation control.

Currently, there are two major approaches in spacecraft formation control and maintenance: the slave and master architecture, and the decentralized formation architecture. In the slave and master approach, one of the spacecraft, designated the center of the formation, performs all the necessary computation to determine control requirements for itself and for the rest of its crew. The master spacecraft has two-way communication with each of the slave spacecraft. In the decentralized approach, all spacecraft in the formation are peers. They transmit necessary attitude, position, velocity and control information among each other. A decentralization algorithm with minimal exchanged information has been developed by R. Carpenter [13]. His technique is based on the Linear-Quadratic Gaussian Control algorithm [14]. Another approach to the control of a large formation is to use synchronization algorithm introduced by Pecora and Carroll [15,16].

At the time this paper was written, none of the approaches to formation flying known to the authors have been fully developed and tested. Nevertheless, we will discuss the possibility of applying MAST algorithm to formation control and maintenance problems. As opposed to the attitude sensor calibration where sensor parameters are adjusted to achieve desired attitude accuracy goal, in formation maintenance application, the control vectors are adjusted to achieve desired position (and attitude) of each spacecraft in the formation. In the decentralized formation control, each spacecraft in the formation performs local closed-loop control using input from its local sensors in addition to information transmitted from other spacecraft in the formation. In the monitoring mode, relative position and attitude of each spacecraft is monitored against a formation model. When sizable drifts are predicted, MAST tuning mode will be activated. An example of this mode is demonstrated in Figure 3. In this mode, an extended Kalman filter is used in the position estimation, while MAST tuning process is used to adjust control parameters.

The tuner will take as input past measurements of position residuals and attempt to adjust control parameters based on the results of previous cycles. Of course, MAST tuning process should be done offline (to save fuel). Not until the system has accumulated sufficient information in terms of cumulative probability distribution), or the solutions are nearly converging, then MAST may be switched to a real-time tuning process.

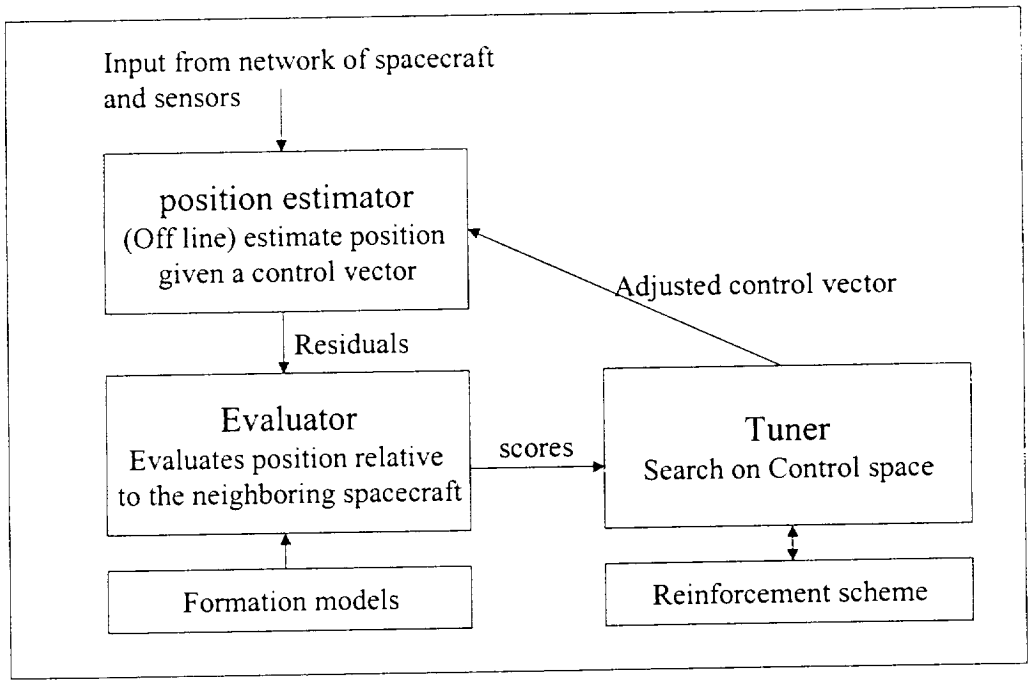

Figure 3. An example of using MAST in Formation Maintenance Application

For future investigation, extended Kalman filter in the position estimator will also be replaced by or coupling with MAST's tuning algorithm in order to reduce the computational cost even further.

8. Conclusion

The proposed program MAST is designed with the following philosophy in mind: the dependency on the application domain lies entirely in the predictor component, while the tuning component is generic and independent of application domain. With this concept, new applications can be developed quickly by focusing on developing a predictor with full knowledge of the nature of the application domain enough to monitor and diagnose problems that may occur. The tuning mode, on the other hand, will only need information on the cost function that needs to be optimized, and parameters to be modified.

This study is part of our program to increase the level of autonomy of onboard flight software. A proof of concept of ASCAL, the first phase of the program is now being developed in MATLAB™.

References:

[1] C. Sary, C. Peterson, J. Rowe, K. Mueller, W. Truszkowski, T. Ames, N. Ziyad, *Automated Multimodal Trend Analysis System*, Proceedings of the AAAI Spring Symposium 1998.

[2] C. Peterson, J. Rowe, K. Mueller, N. Ziyad, *ASCAL: Autonomous Attitude Sensor Calibration*, proceedings of the Flight Mechanics Symposium, NASA/GSFC May 1999.M. D. Shuster, D. M. Chitre, and D. P. Niebur, *Inflight Estimation of Spacecraft Attitude Sensor Accuracies and Alignments*, J. of Guidance and Control, 5, 4, 1982.

[3] M. D. Schuster, *In-Flight Estimation of Spacecraft Sensor Alignment*, Advances in the Astronautical Sciences, 72, 1990.

[4] M. D. Shuster, D. M. Chiter, and D. P. Niebur, *Inflight estimation of Spacecraft Attitude Sensor Accuracies and Alignments*, J. of Guidance and Control, 5, 4, 1982.

[5] G. J. Bierman and M. D. Schuster, *Spacecraft Alignment Estimation*, 27th IEEE Conference on Decision and Control, Austin, TX, Dec. 7-9 1988.

[6] J. E. Keat, *Gyro Calibration Analysis for the High Energy Astronomy Observatory-A (HEAO-A)*, Computer Sciences Corporation, CSC/TM-77/6082, June 1977.

[7] G. A. Shafer, *Mathematical Theory of Evidence*, Princeton U. Press, 1976.

[8] A. P. Dempster, *Upper and Lower Probabilities Induced by Multivalued Mappings*, Annals of Math. Stat. 38, pp. 325-329, 1967.

[9] C. Peterson, *Local Dempster Shafer Theory*, CSC-AMTAS-98001, C.S.C Internal Report

[10] D. Folta, *Enhanced Formation Flying for the New Millennium and Mission to Planet Earth Program, Mission Design and Implementation of Satellite Constellation*, van der Ha (ed.) pp. 243-254, International Astronautical Federation, 1998.

[11] F. Bauer, J. Bristow, D. Folta, K. Hartman, D. Quinn, J. P. How, *Satellite Formation Flying Using an Innovative Autonomous Control System (AUTOCON) Environment*, AIAA Proceedings 1998.

[12] J.P. Diris, J. Fourcade, C. Jayes, T. Tournier, L. Lefebvre, J. Dulac, N. Dubernet, *Autonomous Orbit Determination and Control in Constellations of Satellites*, Mission Design and Implementationn of Satellite Constellations, 255-261, J. C. van der Ha (ed.), International Astronautical Federation, 1998

[13] J. R. Carpenter, *A Preliminary Investigation of Decentralized Control for Satellite Formation*, Proceedings of the IEEE Aerospace Conference, Big Sky, MT, March 2000.

[14] J. L. Speyer, *Computation and Transmission Requirements for a Decentralized Linear-Quadratic Gaussian Control Problem*, IEEE Trans. Automatic Control, AC-24, 2, pp. 266-269, 1979.

[15] T. L. Carroll, L. M. Pecora, *Synchronizing Chaotic Circuits*, IEEE Trans Circuits and Systems, 38, 4, pp. 453-456, 1991.

[16] L. M. Pecora, T. L. Carroll, *Synchronization in Chaotic Systems*, Physical Review Letters, 64, 8, pp. 821-824, 1990.