Agent Control of Cooperating Satellites*

N.K.Lincoln, S.M.Veres†, L.A.Dennis, M.Fisher and A.Lisitsa‡

Abstract

A novel, hybrid, agent architecture for (small) swarms of satellites has been developed. The software architecture for each satellite comprises a high-level rational agent linked to a low-level control system. The rational agent forms dynamic goals, decides how to tackle them and passes the actual implementation of these plans to the control layer. The rational agent also has access to a MatLabmodel of the satellite dynamics, thus allowing it to carry out selective hypothetical reasoning about potential options. This hybrid architecture has been implemented on simulated Earth orbiting satellites and hardware at Southampton's satellite hardware simulation lab.

While the deployment of satellites in formation within an Earth orbit is clearly both interesting and useful, this paper takes the spacecraft swarm further afield. In particular, we here investigate the rational agent autonomy problem, including collision avoidance, fault diagnosis and recovery, and cooperative behaviour of such spacecraft deployed to explore groups of Trojan Asteroids.

1 Introduction

"Trojan Asteroids" are space objects "caught" in particular regions of Jupiter's orbit as a result of dynamical/gravitational interactions between Jupiter, the Sun and the third body object. These regions are known as the L4 and L5 libration (or Lagrange) points of the three body problem. It is estimated that the Jupiter L4 point holds 160-240,000 asteroids with diameters larger than 2 km and about 600,000 with diameters larger than 1 km. Although this equates to a mass of about 1/5th the Asteroid Belt, these asteroids are concentrated in a much smaller volume. In addition, the Trojan Asteroid environment is highly complex and dynamic, consisting of closely packed families, contact binaries and an indeterminate number of small bodies trapped in a complex oscillating motion and liable to regular collisions.

In this paper we model and implement (the software for) a complex mission aiming to catalogue the asteroids at selected Trojan points, primarily as a precursor to mining etc. Since Trojan asteroid belts are even more distant than the asteroid belt itself, the full autonomy provided by the rational agents is not only essential, but also allows us to carry out sophisticated exploration and analysis activities. The specification for this mission focuses on the following key aspects:

- There are over-arching "mission goals", typically a list of observations, sorted by priority, that the ground controllers would like made. These can change dynamically, for instance a message may be received from Earth that may change the mission task, based on data received from the spacecraft group on mission. To coordinate actions and for safety, spacecraft should always be in contact with each other and share results of observations.
- In the light of the mission goals the satellites need to decide autonomously which satellite will make which observations at which asteroids. Based on the results, they need to jointly assess what could be the best-value mission tasks to continue with and send back results to Earth for approval. Mission tasks need then to be jointly planned.
- The satellites have different capabilities (i.e. different equipment) and these capabilities can change dynamically (i.e., as equipment breaks). Similarly, some equipment can make only a finite number of observations. These are to form part of the planning process for high value mission tasks.
- The various observations include ones which take place at a single point in time, ones which involve some continuous monitoring of an object, and ones which require triangulation and so the co-operative behaviour of at least two satellites. Similarly, some of the observations involve the satellites flying "close" to an asteroid and therefore some rational assessment of risk, and reasoning about risk versus priority, is required.
- There are unexpected hazards (e.g. solar flares or uncharted asteroids) which require the satellites to drop

*Supported by EPSRC grant numbers: EP/F037201/1 and EP/F037570/1

†N.K.Lincoln and S.M.Veres are within the School of Engineering Sciences, University of Southampton, UK

‡L.A.Dennis, M.Fisher and A.Lisitsa are within the Department of Computer Science, University of Liverpool, UK

their current goal and take evasive action. The team needs to be able to predict such events and distribute warnings to all spacecraft members.

- Sometimes the result of one observation implies that another needs to be taken (this might not be planned in advance). This is typical for science and mining interests and the system needs to re-plan and execute such investigations until the results are satisfactory.

2 Previous Work

We have produced a hybrid system embedding existing technology for generating feedback controllers and configuring satellite systems within a decision making part based upon a high-level agent programming language [Dennis *et al.*, 2010c]. Such languages assume an underlying imperative programming layer in which an agent's actions are executed; hybrid control systems appears to be a natural fit for this style of programming in which a decision making layer is combined with a lower level dynamic execution layer.

Decision-making tends to rely on *discrete* information (e.g. a thruster is broken) while system control tends to rely on *continuous* information (e.g. thruster fuel pressure is 65.3). Thus, it is vital to be able to *abstract* from the dynamic system properties and provide discrete abstractions for use by the agent program; see Figure 1. It is for this reason that we have an explicit *abstraction layer* within our architecture that translates between the two information styles as data flows around the system. This *Abstraction Engine* generates a stream of incoming sensor and action abstractions for use by the Rational Engine.

Figure 1 describes the architecture of our system. Real time control of the satellite is governed by a traditional feedback controller drawing its sensory input from the environment. This forms a *Physical Engine* (Π). This engine communicates with an agent architecture consisting of an *Abstraction Engine* (A) that filters and discretizes information. To do this A may a use a *Continuous Engine* (Ω) to make calculations involving the continuous information. Finally, the *Rational Engine* (R) contains a "Sense-Reason-Act" loop. Actions involve either calls to the Continuous Engine to calculate new controllers (for instance) or instructions to the change these controllers within the Physical Engine. These instructions are passed through the Abstraction Engine for reification.

In this way, R is a traditional BDI system dealing with discrete information, Π and Ω are traditional control systems, typically generated by MatLab/Simulink, while A provides the vital "glue" between all these parts.

The agent programming language within the Rational Engine encourages an engineer to express decisions in terms of the facts an agent has to hand, what it wants to achieve and how it will cope with any unusual events. This reduces code size so an engineer need not explicitly describe how the satellite should behave in each possible configuration of the system, but can instead focus on those facts that are relevant to particular decisions [Dennis *et al.*, 2010b]. The key aspect of *deliberation* within agent programs allows the decision making part of the hybrid system to adapt intelligently

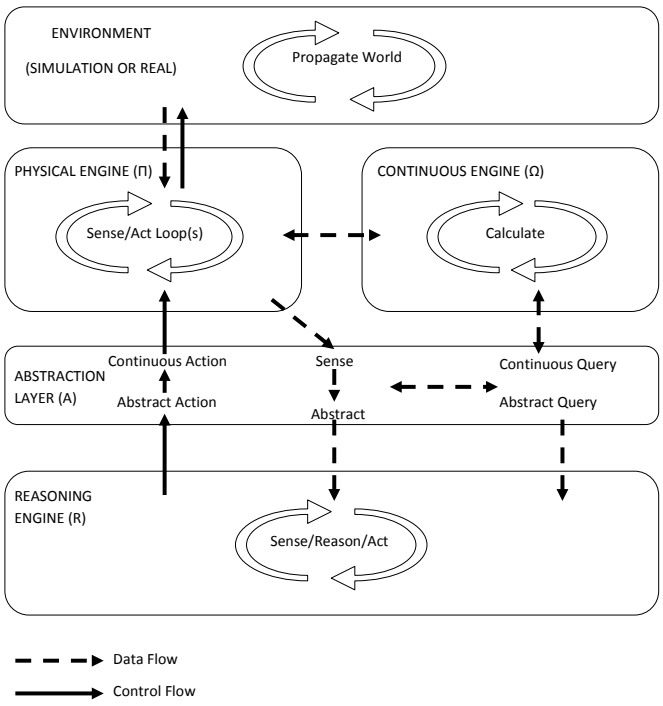

Figure 1: The Agent System Structure

to changing dynamic situations, changing priorities, and uncertain sensors.

This architecture has already been used successfully to investigate simple scenarios involving the deployment of groups of satellites in geostationary and low Earth orbits: the VR displays of these are given within Figures 2 and 3 respectively. Within these environments, the applied architecture was shown to be

- *resilient*, in that if a satellite develops faults or leaves its assigned path it can dynamically be moved back (by reconfiguring the control system) to achieve its tasks,
- *cooperative*, in that the satellites together organise a particular formation and, during their mission, can dynamically re-organise into a different formation,
- *concise*, in that the code required to describe and implement this hybrid architecture is significantly reduced [Dennis *et al.*, 2010b], and
- *transparent*, in that the agent is able to explain, at a high-level, what its aims were in choosing a certain direction.

Further details regarding the application of the system architecture to these scenarios may be found in [Dennis *et al.*, 2010c; 2010a; 2010b; Lincoln *et al.*, 2010].

The implemented scenarios are arguably trivial, relating to the somewhat benign dynamics of Earth's orbit; this was especially true for the geostationary example that dealt only with the issues of internal reconfiguration and controlled motion. Although the "low earth orbit" (LEO) example increased in complexity through more active dynamics and the need for collision avoidance to perform motion to specific

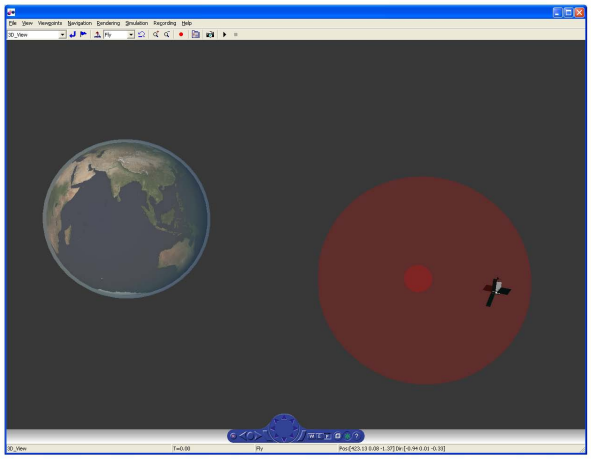

Figure 2: A geostationary single agent scenario

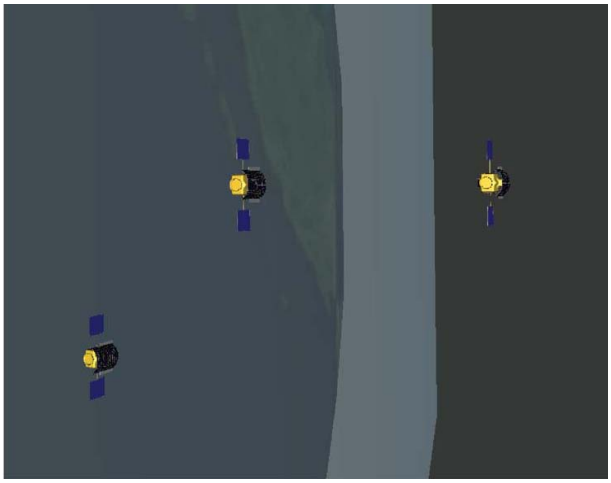

Figure 3: A low Earth orbit multi-agent scenario

configurations, the environment remained highly predictable, with the only consideration of consequence being the motion of companion agents. Such trivial examples were useful in testing the functionality of the system and demonstrating autonomy in proximity to the Earth: this work reports the extension of this system to a more complex scenario.

3 Investigated Scenario

The scenario investigated within this paper is that of the cooperative action of four autonomous satellites operating within an asteroid field environment, tasked with cataloguing the asteroid numbers and composition. Only a small subset of the asteroids present within the environment are initially known and are only observable if they are not occluded by other, more proximate, asteroids. Additionally, agents may only communicate with other agents that are not occluded by an asteroid[1]. This results in the requirement to commence operation in a partially known environment and to develop complete knowledge of this environment through cooperative action. Asteroid collisions may occur, resulting in random asteroid motion; should an agent collide with an object in the environment then it will incur some form of damage, dependent upon the severity of the impact. Whilst hardware failure may occur as a result of a collision instance, it may also occur as a random hardware "gremlin" that the agent must be tolerant to.

The complete agent system is a multi-software system in which the abstraction process and agent reasoning are performed in Java and the spacecraft agent hardware is modeled in Simulink. The asteroid environment, in which the asteroids and spacecraft agents may move and interact, is implemented using a Java port of the Bullet Physics Library [Web,]. VR output is performed in OpenGL, and a screen capture of this is given within Figure 4.

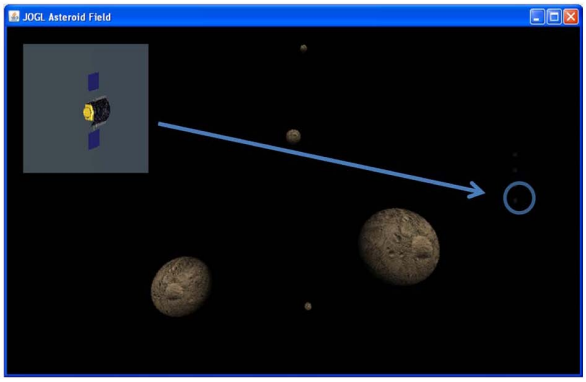

Figure 4: A Trojan asteroid multi-agent scenario.

The asteroid types vary in composition, including: pure rock, nickel-iron, rare Earth metals and ice. The agent is unaware of the absolute asteroid composition at the mission start.

Collision impacts between all system bodies may occur. Asteroids are only effected in that they may have their trajectories altered, however the collision of a spacecraft agent has obvious repercussions: should a spacecraft agent impact an object, then the agent is affected depending upon the magnitude of the impact. Small impacts may result in only a disturbance to their trajectory, however as the impact magnitude(s) increases the resultant damage becomes greater and may result in fuel line ruptures, total loss of control truster(s), loss of sensor payload and even complete agent loss.

3.1 Current Status

Agent Abstractions

Abstraction is a two-stage process within the agent architecture. The physical (Π) engine sends a subset of sensor data to the abstraction engine which then filters and in some cases further discretizes the data based on the current situation.

Π **Abstractions** Basic abstractions are taken from both the world environment and the spacecraft model; these are passed

[1] It is possible for agents to relay information between agents, provided a suitable link exists.

by the physical engine, Π, to the abstraction engine. The information provided to the abstraction engine includes:

A) Propulsion System Data Data relevant to the operation of propulsion system hardware is passed to the abstraction engine, this information includes but is not limited to: pressure information (main pressure vessel and fuel lines), valve activation status and current/voltage data for internal systems.

B) Control Performance Data This relates to output control requests that are sent by the control system and actual output responses observed by onboard systems; differences in these may enable the agent to infer the effectiveness of a particular control system and also augment investigations into faulty control hardware.

C) Kinematic Data High level kinematic state information is derived from the onboard sensors that monitor the world environment and abstracted into quantities relating to: orbital acquisition, path following status and regional bound information relating to asteroid approach.

D) Payload Status The status (health) of the agent payload, which in this case is primarily that of sensors, is available to the abstraction engine.

E) Asteroid Field Updates The internal navigation/mapping system of the agent may flag to the abstraction engine if a previously unknown asteroid is detected within the field.

A Abstractions and Reifications The abstraction engine is currently implemented using BDI style plans. A further abstracts the data received from Π and sends it to the rational engine (R). It also reifies instructions from R which are passed on to Ω and Π.

A's abstractions currently are:

A) Thruster Malfunction A determines from the Propulsion System Data whether a thruster is working or not.

B) Orbit Acquisition A determines from the Kinematic Data whether an agent has entered an intended orbit.

A's reifications closely match the Π and Ω abilities described below. In most cases A adds a few low level details to the request that are unimportant to the deliberations of the rational engine and manages housekeeping related to communication between the various engines.

Agent Plans and Abilities

The behaviour of any given agent using the architecture presented within Figure 1 is governed by its rational decision making processes and its capabilities.

Rational decision making is based on the use of *plans* and *reasoning rules* that are implemented in the rational engine R.

The capabilities may be divided into Π and Ω *abilities*. Π abilities, or skills, relate to specific physical actions the agent may invoke on the world environment; Ω skills are those related to complex queries that may be used to assist rational decision making occurring within R or specific Π skills.

Π Abilities Each agent is endowed with the ability to control its physical hardware. In the instance of an autonomous spacecraft agent, this relates to the ability to output required forces and torques for desired motion. This entails interaction with various systems at various levels of complexity: each agent has access to various discrete time closed loop control solutions and may interact directly with the propulsion system[2]. Sensor systems are assumed to be available for the internal control routines. Whilst the physical agent body is to be controlled by appropriate force output, damaged hardware systems may result in spurious control outputs. The agent may request the following actions to occur on the spacecraft:

A) Position Regulation The agent may invoke discrete time sliding mode control applied to a fixed inertial point in 3-Space. This control routine occurs as a closed loop process inside hardware on the spacecraft and whilst the agent may invoke the control regime, it does not have direct access to the internal procedures.

B) Trajectory Following Control The agent may apply discrete time sliding mode control to follow a prescribed trajectory in 3-Space. Alike the point regulatory control, the routine occurs as a closed loop process inside hardware on the spacecraft and the agent does not have direct access to the internal procedures.

C) Fuel Valve Configuration The agent may toggle various fuel valves used to supply the thrusters with propellant. The fuel system is modeled as being doubly redundant, meaning that two fuel lines exist with multiple valves dictating the flow through these lines: the agent may change the status of these valves, though some valves may be faulty.

D) Thruster Power Configuration The agent may enable and disable propulsive devices by routing power to the system.

E) Sensor Operation The agent may operate (activate/deactivate) the payload sensors to achieve a particular imaging task.

F) Communication The agent may communicate with other agent(s) via direct or indirect radio-communications. Indirect communication is only possible if a communication route exists between two occluded agents via the agent community.

Ω Abilities These abilities relate to complex tasks that are required to support reasoning with the R engine and control routines within the Π engine. R is "interested" in eventualities of implemented action and is concerned with the fact that specific control routines require specific data sets to be generated prior to their implementation. It is also the agent that dictates specific motion with the asteroid system: its motion is directed by generation of non-intersecting trajectories to target destinations that the internal control systems are then directed to follow.

[2]This includes valve switching and power routing to enable contingencies for failure

A) Path Generation The agent may call for the generation of a path, or trajectory, in 3-Space for a specific control routine to use internally. The path may be to a specific point or the continuous orbit of a nominated (observable) asteroid.

B) Path Intersection The agent may call for the phased intersection of an existing trajectory (being followed by another agent) to enable joint observation of a nominated asteroid.

C) Point Selection The agent may may call for the determination of a ideal point to observe a particular asteroid or set of asteroids given the desired data output and the available sensor hardware specific to the agent.

D) Motion Prediction The agent may search ahead of time to investigate possible world scenarios involving asteroid motion and agent action; as the predictive (temporal) horizon increases, the accuracy of the results decrease.

E) Evaluation of Sensor Data Abstracted data returned from the payload sensors may be returned to the agent; these results may or may not be conclusive.

R Plans and Rules The agent deliberation cycle relates to the desire to fulfill the asteroid cataloguing mission using the abstractions and abilities listed previously. In addition to carrying out the cataloguing mission, the agent must be tolerant to internal hardware failure(s) such that the agent may continue the mission, even if this entails continuation with some degraded performance.

Complete cataloguing of the asteroid field requires traversal of all (localised) asteroids to enable investigation of the asteroids using the spacecraft sensors. The sensors may or may not return a conclusive result or may necessitate the revisit of the asteroid from another spacecraft agent that has a specialist sensor.

Agent (rational) action is dictated and prescribed through use of a specialised rational agent (BDI) language based upon the Gwendolen programming language [Dennis and Farwer, 2008]. The execution of the rational and abstraction engines are based primarily upon the use of *plans* for action and *rules* for reasoning about facts. At present the rational engine has implemented plans for:

A) Asteroid Selection The rational engine selects an asteroid to examine by requesting distance information from the continuous engine for a selection of target asteroids, and negotiating with other agents to avoid multiple agents surveying the same asteroid. It then instructs Π to orbit that asteroid.

B) Thruster Repair The rational engine selects a suitable course of action for compensating for a damaged thruster, based on the propulsion system data, and instructs Π to reconfigure its hardware appropriately.

C) Summoning Assistance If the rational engine detects that sensor equipment is required that it doesn't possess, it calculates the closest agent with the correct equipment and summons it for assistance.

The engine has implemented reasoning rules for:

A) Closest Unexamined Asteroid The rational engine can request distance information from Ω and use this, together within information about the intentions of other satellites, to determine the nearest asteroid which no other satellite plans to examine.

B) Closest Satellite Similarly the engine can use information about other satellite's intentions, capabilities and position to select the closest satellite with a desired capability.

3.2 Initial Scenario

An initial test scenario has been investigated within the presented simulation environment in which the satellites are distributed throughout an asteroid field. The satellite agents negotiate in order to select the most suitable target asteroid for each agent; the satellites then move to and orbit their respective target asteroid, correcting any thruster malfunctions that may occur. Once in orbit one satellite detects that it needs additional sensor equipment and it summons the closest satellite with the correct equipment to assist it.

At present communication is assumed to be completely reliable and does not yet take into account the problem of occluding objects etc.

From a control perspective, the implemented discrete time (sliding mode) control methods were effective in following the generated optimal trajectories and placing each agent into a controlled orbit about a designated asteroid. From an agent systems perspective, the complete architecture was observed to perform the desired mission based upon the provided plans and rules, using the multiple-engine construct depicted within Figure 1.

3.3 Further Work

The presented work is to be extended in both depth and breadth to include more agent capabilities, increased dynamics within the asteroid field and a more accurate model of communication. In particular we want to extend the scenario to investigate all the aspects specified in Section 1. In particular we seek to implement the ability for multiple spacecraft agents to operate cooperatively in phased (synchronised) orbits about a single asteroid and to investigate functional roles of spacecraft agents that have suffered complete (sub)system failures, removing the ability to move, detect or sense. We also intend to increase the dynamics of the asteroid field in order to introduce asteroid collisions and the presence of more (smaller) unknown asteroids that will require dynamic collision avoidance and increased sharing of knowledge among the agents relating to the perceived environment.

All of the current agent actions rely upon pre-coded solutions existing within A, R, Ω or Π; In the longer term we would like to investigate the generation of appropriate actions in the presence of unplanned eventualities.

4 Conclusions

The authors have already reported on the ability of the presented agent architecture to represent the high-level decision making aspects of the program in a transparent and concise manner. This paper presents the initial work on a more

complex scenario that will allow us to explore the flexibility and robustness of the architecture in a dynamic environment which requires a high degree of autonomy.

The agent architecture was implemented in a high fidelity asteroid field scenario with a complex cataloguing mission to be performed by multiple, cooperating, spacecraft agents. The agent system was observed to cope with internal hardware failures whilst being able act cooperatively to perform the overriding mission goal using the provided Ω and Π abilities and A and R plans and rules; the scenario requires further elaboration and an investigation of the system resilience in the event of any unplanned events that have no pre-coded solution within A, R, Ω or Π.

References

[Dennis and Farwer, 2008] Louise A. Dennis and Berndt Farwer. Gwendolen: A BDI Language for Verifiable Agents. In Benedikt Löwe, editor, *Logic and the Simulation of Interaction and Reasoning*, Aberdeen, 2008. AISB. AISB'08 Workshop.

[Dennis *et al.*, 2010a] L. A. Dennis, M. Fisher, N. Lincoln, A. Lisitsa, and S. M. Veres. Declarative Abstractions for Agent Based Hybrid Control Systems. In *Proc. 8th International Workshop on Declarative Agent Languages and Technologies (DALT)*, 2010.

[Dennis *et al.*, 2010b] L. A. Dennis, M. Fisher, N. Lincoln, A. Lisitsa, and S. M. Veres. Reducing Code Complexity in Hybrid Control Systems. In *Proc. 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-Sairas)*, 2010.

[Dennis *et al.*, 2010c] L. A. Dennis, M. Fisher, A. Lisitsa, N. Lincoln, and S. M. Veres. Satellite Control Using Rational Agent Programming. *IEEE Intelligent Systems*, 25(3):92–97, May/June 2010.

[Lincoln *et al.*, 2010] N. Lincoln, S. M. Veres, L. A. Dennis, M. Fisher, , and A. Lisitsa. An Agent Based Framework for Adaptive Control and Decision Making of Autonomous Vehicles. In *Proc. IFAC Workshop on Adaptation and Learning in Control and Signal Processing (ALCOSP)*, 2010.

[Web,] JBullet- Java port of Bullet Physics Library.