Mission-Directed Behaviour-Based Robots For Planetary Exploration

Andrei M. Rotenstein, Albert L. Rothenstein, John K. Tsotsos

Centre for Vision Research and Department of Computer Science, York University
4700 Keele Street, Toronto, Ontario, M3J 1P3, Canada
{andrei,albertlr,tsotsos}@cs.yorku.ca

Keywords: Autonomy, Behavior-Based Robotics, Planetary Exploration, Robot Control Architectures

Abstract

This paper argues that the S* robot control architecture of Tsotsos [1997] is well-suited to intelligent control of planetary rovers. Behaviour-based systems operate well in unknown environments, but cannot support deliberative functionality. Hybrid systems resolve this issue, but introduce the potential for other problems. S* is a non-hybrid approach that supports deliberation but also attentive vision. As well, S* includes a mission-specification language that permits behaviour re-configuration on the fly.

1. Introduction

This paper describes our preliminary efforts in developing the S* robot control architecture of Tsotsos [1997], and argues that it is a behaviour-based approach well-suited to intelligent control of planetary rovers.

Our approach is conceptual, primarily, and so this paper discusses how the behaviours of an S* controller interact and not details about how they ought to be implemented. We begin with an overview of behaviour-based systems. Following this, we present some of the issues in designing planetary rover control systems in general. Then, we present our architecture, focusing first on the structure of an S* controller and then on a mission specification language. We conclude with comparisons to other work and a discussion of our future direction.

2. Behaviour-Based Systems

A behaviour-based controller essentially consists of noncommunicating processes, whereby each accepts input from sensors and computes candidate actuator commands so as to fulfill some 'hard-coded' goal. An arbitration module combines the candidate outputs to yield the action the robot is to take. All of this occurs in real-time, and are therefore considered reactive, and also referred to as reactive systems. The behaviour-based approach emerged in the mid-1980's, most notably with Rodney Brooks' Subsumption architecture [Brooks, 1986]. It was a radical departure from the pervasive view of robot control, which held that a controller should organize functional modules into an open-feedback loop, where one module senses the environment, another applies the sensor data to a model, another generates a plan from the model, and another executes the plan by performing actions in the environment [Nilsson, 1980; Brooks, 1991], and is known as the sense-model-plan-act (SMPA) cycle. Because behaviour-based systems are not bogged down by having to maintain representations or by generating plans, they act more quickly than SMPA systems, navigating dynamic and unstructured environments with relative ease. See [Arkin, 1998] for an extensive overview of behaviour-based controllers.

Behaviour-based systems are known to operate well in unknown environments, and assure liveness and safety reasonably well. However, they are generally unable to support deliberative functionality (generating plans from goals, or any other kind of decision-making based on explicit representations of tasks and other knowledge) as do SMPA systems. Consequently, they can neither execute operator-specified missions nor guarantee operator-specified constraints, rendering them unsuitable for planetary exploration applications. Clearly, to justify the enormous cost of planetary exploration, rovers ought to be able to both execute different missions autonomously and be able to do so safely in an unstructured environment. Therefore, space applications require architectural support for both deliberation and reactivity.

Hybrid systems [Arkin, 1998] are the most common approach to integrating deliberation and reactivity, in which conventional deliberative and reactive systems are mated using various interface strategies. In the AuRA architecture [Arkin, 1998] for example, for each step of a plan, the deliberative component activates the behaviours from a pool suitable for completing that step. It monitors the state of the environment to determine when tasks are completed. In ATLANTIS [Gat, 1992], a component called the Sequencer requests plans from another component that performs planning and maintains model representations, and controls a reactive "skills" component. In the case of failures, the Sequencer can ignore the plan and choose actions that are more appropriate given the situation. TCA/TDL [Simmons 1994; Simmons & Apfelbaum, 1998] has a central controller that generates task trees (plans) that other distributed components expand and execute on-the-fly. The controller also performs monitoring of task completion. Gat [1998] describes these as Three-layer Architectures: a middle layer maintains the internal state of the agent which in turn informs the functioning of both a deliberative layer and a reactive layer, thus interfacing the two other layers.It is important here to recognize that it's necessary for such systems to perform and monitor task execution and to recognize failures.

Apart from supporting deliberation and reactivity, a clear advantage of hybrids is that it easy to compose existing pure-deliberative and pure-reactive systems this way. However, it is centralized, assumes an artificial distinction between the two types of processing, and consequently, this limits our ability to add new components. Non-hybrid strategies for supporting deliberation in a behaviour-based system are therefore an improvement. As an example of such a system, Nicolescu and Mataric [2002] augment behaviours with operational preconditions and connect them together into a behaviour network representing a flexible plan with many possible executions. Then, a behaviour can determine if the actions taken by others satisfy its preconditions, and if so, activates itself until it completes a task, thus executing a plan generated in a manner analogous to that of STRIPS [Nilsson, 1980], but in a distributed manner.

Often, designers treat the role of sensory and perceptual systems as ancilliary to the control problem, and are usually implemented externally. Having discussed behaviour-based robotics, we turn our attention to the problem of enabling planetary exploration with architecture, in which it becomes important to consider architectural support for sensing modalities.

3. What must rovers do?

There is no argument that the key to enabling unmanned planetary exploration is “onboard autonomy” [Wyatt, 2001] – the ability of the rover to handle complex tasks independently of external intervention or supervision. Given a high-level description of a scientific exploration mission complete with goals, a autonomous rover should be capable of multiple missions and should achieve a high degree of science return by being able to complete science goals on a best-effort basis. For example, if a rover discovers it cannot reach some geological formation specified as a goal due to intraversible terrain or insufficient power, then it should be able to discard that goal and perform another. As well, if a rover discovers that another geological formation specified as low priority is much more easily reached than one of higher priority, it should be able to focus its efforts on achieving the former goal first [Knight, 2001]. This requirement implies that the rover should perform planning and decision-making without sacrificing liveness and safety: clearly, integrating deliberation and reactivity into the control architecture is therefore necessary.

However, this is not enough. It is also necessary to consider the role of sensing modalities, particularly robot vision, since it is capable of capturing vast amounts of information about the robot’s environment – much more than any other. As well, the fact that humans use vision a great deal in performing tasks is a strong case for using it to the fullest extent possible in an intelligent agent.

It is not without its difficulties, however: the simple task of visual search is intractable in general [Tsotsos, 1990]. As described in [Tsotsos, 1995], active-vision (using multiple viewpoints to resolve ambiguities in scene construction) and attention (constraining processing to a subset of the visual information that is somehow relevant to the task at hand) are necessary mechanisms for rendering the task tractable [Tsotsos, 1990]. We believe these mechanisms should be supported by any robot control architecture used for this problem domain. They require the use of explicit representations of goals, task information, models, plans, and so on. In a behaviour-based system in particular, which starkly opposes representation, they are nevertheless necessary and must be augmented to include it [Tsotsos, 1995].

4. Our own efforts towards control technology

The S* architectural approach [Tsotsos, 1997] was conceived to support not only deliberation and reactivity but also attentive processing mechanisms. In subsumption-style behaviour definitions, a behaviour acts directly on the physical world. In S*, however, this notion is generalized: the 'World' on which a behaviour may act may be an internal (logical) representation or an external (physical) representation. The World is added to the sense-model-plan-act cycle to yield the sense-model-plan-act-world (SMPA-W) cycle. It should be noted that, although the SMPA concept is a part of this definition, it is used as a means to decompose individual behaviours into component parts, not to organize the robot’s functional components into a centralized, hierarchical strategy.

An S* behaviour only performs processing when there has been a relevant change to the subset of the World that it pays attention to (termed its *event window*) a daemon monitors the World and, upon detection of a relevant change, triggers the behaviour to perform one iteration of the SMPA cycle. S* behaviours accept sensory input from this event window by reading representations, and act upon the world by writing a relevant subset of representations in that world (termed its *action window*). The event windows can be changed to include a different subset of the world; this corresponds to a shift of attention. Behaviours may be reading from representations of robot sensors and writing to representations of robot actuators and may also read and write internal representations that include plans, task knowledge, maps and models, and so on. Each behaviour’s computation can be as complex as an entire SMPA process, or as simple as a mapping between sensing and action (equivalent to having no modeling and planning). S* behaviours can therefore include deliberative mechanisms that operate and maintain representations, and yet retain the reactivity.

4.1 Behaviour Networks

The relationships between behaviours and representations in the World constitute what we term a Behaviour Network. A designer can create behaviours and representations and organize them into networks, to solve a particular control problem.

For example, suppose we would like a differential drive robot platform with a panning stereo camera, bumpers, and proximity detectors to navigate a terrain, visiting interesting objects in its environment (uniquely identified by colour) sequentially in a list (e.g. green, red, blue). The world consists of a flat terrain with obstacles. The diagram in Figure 1 is of a behaviour network that performs such a task. Rectangles depict representations, hexagons depict behaviours, and arrows to/from behaviours and representations indicate read/write relationships. This network has a number of representations:

- **BUMPERS, PROXIMITY DETECTORS, STEREO CAMERA IMAGES.** Contain data from sensors.
- **MOTORS, CAMERA MAST.** Hold actuator signal for wheel motors and desired position for camera panning motor.
- **HEADING.** Contains a command indicating a desired heading and travel direction (forwards/backwards) to move in.
- **MAP.** A low-resolution *a priori* map of the world.
- **NAVIGATION TYPE.** Represents the current mode of navigation.
- **PERCEPT.** Represents the image locations of centroids in the stereo images of colour blobs having the same colour as the desired target.
- **TASK QUEUE, NEXT TASK.** Contains a queue of tasks to perform (objects to locate), and a flag indicating the next task should be performed, respectively.
- **POSE.** Contains the compass heading and co-ordinates of the robot in the environment (assuming sensors can determine this).

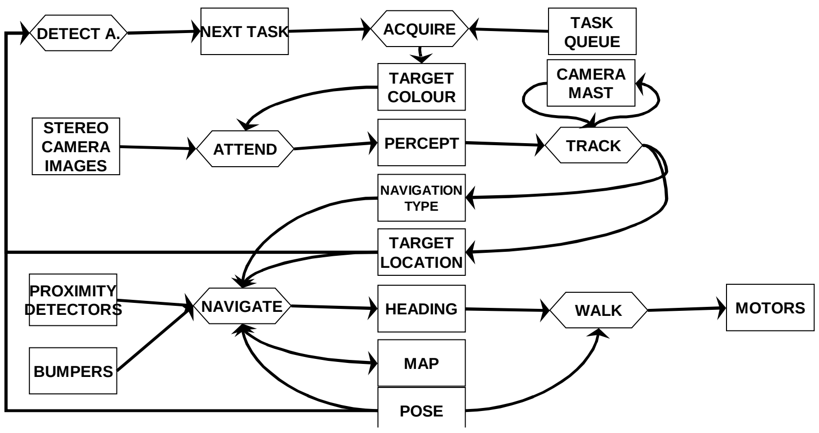

Figure 1: Example Behaviour Network

This network has six behaviours. Their functionality is summarized here:

- **Walk.** Commands the motors such that the platform is oriented towards the desired heading and moving forwards or backwards, as specified in *HEADING*.
- **Navigate.** Ensuring obstacles are avoided, determines appropriate direction to move in so as to either navigate towards a target location or to perform a random walk, depending on the navigation mode. Performs path planning from current position to desired position, based on information in *MAP*.
- **Attend.** Locates centroids of largest regions of specified colour of target in stereo image, and stores their image locations in *PERCEPT*.
- **Track.** Manipulates camera mast so that stereo camera is directed towards the colour target in the environment (i.e. the centroids are centered horizontally in the stereo image), and estimates their location as best as possible, and the navigation mode in NAVIGATION TYPE is set so that the Navigate behaviour tries to move towards the target location. If no centroids are found, the camera is simply panned, and the navigation mode is set so that the rover performs a gradual random walk through its environment.
- **Detect Acquisition.** Raises a flag in *NEXT TASK* if the robot is in the vicinity of the target, to indicate that the current task is complete and to move on to the next task, or otherwise lowers it.
- **Acquire.** When the previous task was completed, selects a task from the queue and stores task information (colour of the target) in *TARGET COLOUR.*

The network works as follows. The system starts with a sequence of tasks to perform on the task queue (i.e. visit "red", "green", "blue"). All behaviours are triggered by default, so Acquire dequeues the first task and stores the related colour in *TARGET COLOUR*. Meanwhile, Track pans the camera. As well, it sets the navigation mode causing Navigate to control the heading so that Walk has the robot randomly walking through its environment. At some point, Attend picks up the centroids of blobs with the target colour, and Track starts manipulating the camera so that it centers on the target as best as possible, as well as estimating the target's location in the environment and changing the navigation mode to cause the robot to travel towards the target location. At this point, Navigate uses the *a priori* map information to plot a shortest path to the target location. Obstacles are avoided along the way. Eventually the robot reaches the vicinity of the target locations, Detect Acquisition raises the flag in *NEXT TASK*, and Acquire selects the next task from the queue, starting the process over again.

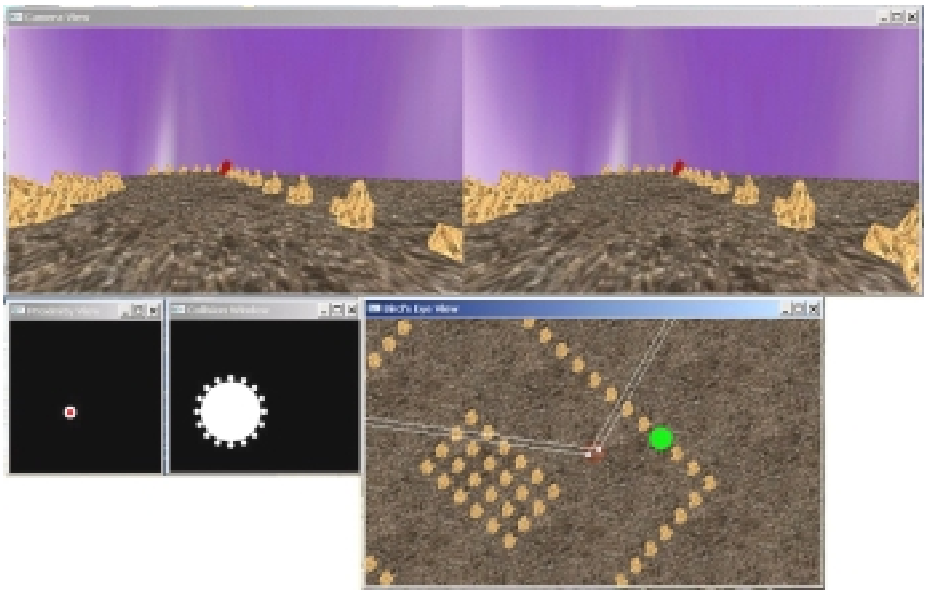
Figure 2: Example screen shot of controller in simulation

This controller provides an example of task-directed perception, where the Attend behaviour uses task information (colour) to perform the required image processing. Note how the controller is decomposed based on dependence on both the platform and task capabilities: that both Navigate and Walk are tied very closely to the sensors and actuators, whereas behaviours like Detect Acquisition and Acquire relate more to the task-at-hand. In this manner, "low-level" behaviours are platform-oriented, while "high-level" behaviours are task-oriented.

An object-oriented software framework was developed in C++ that allows a controller designer to focus strictly on rapid development of S* behaviours and representations, and the framework takes care of all interactions between them, as well as access control to representations and the interface with the environment. The framework is cross-platform, but at present, it has only been tested on an MS Windows 2000 platform. It was applied to implementing this controller, and was tested in simulation. Full details of the software framework and a detailed discussion of behaviour networks are available in [Rotenstein, 2003].

An example screen-shot of this is shown in Figure 2. The robot is seen from overhead in the bottom-right window. The two white dots and related white lines depict the stereo camera and its fields of view, respectively. The stereo image can be seen on the top window, with a red target in the distance. Full details of the virtual environment are available in [Rothenstein, 2002].

4.2 Mission Language Specification

S* introduces a language for mission plan specification that permits the integration of deliberative and reactive specifications and behaviour re-configuration on the fly.

Intelligent agents must be capable of executing a variety of missions, and thus face a much more difficult task than single-mission systems. Generally, planners provide sequences of steps that must be executed, taking into account motion and sensor uncertainty by including localization and correction steps explicitly in the plan. Attempts to capture and integrate reactivity in this kind of plan (or, conversely, to structure behaviour-based systems in a way that is compatible with planning) have had varying degrees of success. Modern mission specification languages, facilitate the development of robotic systems that are: able to do reasoning and planning; able to react to the environment; able to monitor mission execution; integrate reactive and deliberative behaviours; programmed in high-level languages.

In general the systems developed have two characteristics: there is a clear separation between the deliberative and reactive parts of the system, and special modules do the monitoring. The former is in general a reasonable approach, but it introduces a split in the definition and capabilities of the system, which has as a consequence the fact that usually the reactive part of the system can not be controlled by mission programs, and the systems generally have high complexity, leading to engineering problems, especially when it comes to extending the capabilities of the system with new modules. The latter characteristic means that monitoring is implemented outside the normal flow of the mission and not under mission developer control. Since it is very likely that different stages of complex missions operate under different constraints, and thus require different monitoring, this aspect should be under user control in a flexible and uniform manner. The language proposed in [Tsotsos 1997] and implemented in [Rothenstein 2002] addresses these two issues in the context of the S* framework, while at the same time providing the normal temporal constraints that characterize programming languages used in robotics control.

We will introduce the mission specification language gradually. In its simplest form, a mission is a series of steps that have to be executed in sequence, and could be defined at any level of abstraction, but the simplest and most obvious thing to do is to use actual behaviour names in the language. This has a number of advantages, ranging from the readability of the program to the ease of implementation.

This simple temporal sequence is obviously sufficient only for the simplest of missions, so temporal grouping constructs are needed to enhance the expressive power of the language, especially since S* permits all behaviours to be active in parallel. We need to be able to express parallelism and to connect events in time. The default behaviour connector is the semicolon and it implies that two steps (behaviours) are to be executed one after the other.

The binary connectors used are:

- **starts with**: the two behaviours start together
- **within**: the first behaviour starts after and ends before the second one
- **during**: the first behaviour starts during the execution of the second one
- **starts after <time>**: the first behaviour starts after the end of the second one (after an optional time delay)
- **ends with**: the two behaviours end at the same time
- **ends during**: the first behaviour ends during the execution of the second one

Also, an arbitrary number of behaviours aligned by start time or end time are represented as `[b1, b2, …)` and `(b1, b2, …]` respectively.

A grouping construct is introduced, called a *phase*, used to label sequences of actions for reuse, similar to a function or procedure call in a traditional programming language.

As mentioned before, the aspect that sets the language proposed in [Tsotsos 1997] apart is that it allows the programmer or planner to specify aspects of the environment of which the robot must be vigilant during the execution of various stages of the plan. We include a second grouping construct, called a '**while-ensure**' primitive, so that for a collection of plan steps, actions may be associated with a clause to ensure that particular conditions are met (e.g. "ensure that the position is within a certain tolerance," "keep a safe distance from obstacles," etc.). These grouping constructs can be nested. The nesting permits for sub-phases of a mission to proceed with localized constraints, tightening or relaxing the **ensure** clauses. The **while** part of the **while-ensure** construct is identical to a phase, in that it is nothing more than a grouping construct. The **ensure** part can contain three types of information:

- Ensure behaviours: behaviours that need to execute for the duration of the **while** construct, i.e. in parallel with all the behaviours included in the **while**
- Prohibited behaviours: behaviours that must not execute at the same time as the **while** construct, identified by a preceding "!", similar to the "not" logical operator used in traditional programming languages
- Boolean clause: a Boolean clause that needs to be periodically evaluated for the duration of the **while** construct, the frequency of evaluation being specified in an optional argument.

Examples of ensure behaviours would be obstacle avoidance behaviours, tracking behaviours, or position validation behaviours. Similarly, prohibited behaviours could be, for example, yielding the right of way in a situation where continuous motion is critical. Boolean conditions can be expressions that include comparisons

```
mission <- start phase_def while_def behaviours end
phase_def <- null
           | phase phase_def
while_def <- null
           | while phase_def while_def
phase <- phase phase_id {behaviours}
while <- while while_id {behaviours}ensure
            (time predicate)
behaviour_args <- null
           | individual_arg
           | behaviour_args , individual_arg
individual_arg <- representation_id
           | number
           | symbol
behaviour_atom <- behaviour_id ( behaviour_args )
           | phase_id
           | while_id
behaviours <- behaviour_atom
           | behaviour_list
           | behaviours connector behaviours
behaviour_list <- [b_list)
           | (b_list]
b_list <- behaviour_atom
           | b_list, behaviour_atom
connector <- ;
           | starts with
           | within
           | during
           | starts after time
           | ends with
           | ends during
time <- null
           | number
predicate <- behaviour_preds
           | boolean_preds
           | behaviour_preds && boolean_preds
behaviour_preds <- behaviour_id (behaviour_args)
           | ! behaviour_id ()
boolean_preds <- boolean_preds && boolean_preds
           | boolean_preds || boolean_preds
           | (boolean_preds)
           | individual_arg = individual_arg
           | individual_arg < individual_arg
           | individual_arg <= individual_arg
           | individual_arg > individual_arg
           | individual_arg >= individual_arg
```

Figure 3: Mission Specification Language Grammar

between the values of various representations and other representations or mission specified values, such as, for example, the minimum distance to obstacles should not go below a certain value, the speed should not exceed a maximum value, the battery charge should not decrease below a certain threshold.

To make the language and the missions that it can define more flexible, it is important to provide the flexibility to allow behaviours to act on various representations, rather than having the link hard-coded in the design stage, as is the case in most behaviour-based systems. A formal definition in the style of most traditional programming languages is simple and flexible enough to be a good candidate, resulting in the

```
start

phase MAKE_PANORAMA {
           raise mast();
           take snapshot()
           turn camera (45);
           take snapshot();
           turn camera (45);
           take snapshot();
           turn camera (45);
           take snapshot();
           turn camera (45);
           lower mast(); }

phase WHERE_AM_I {
           raise mast();
           find landmark (A);
           find landmark (B);
           localize();
           lower mast(); }

phase APPROACH_TARGET {
           goto landmark (A) within
               track landmark (A); }

phase ANALYZE_TARGET {
           localize();
           MAKE_PANORAMA }

while THE_MISSION {
           while W1{
                       WHERE_AM_I
                       MAKE_PANORAMA
           } ensure (speed==0)
           APPROACH_TARGET
           while W1{
                       ANALYZE_TARGET
           } ensure (speed==0)
} ensure ()

end
```

Figure 4: Example mission specification.

grammar shown in Figure 3, which adds parameters to behaviours in the form of lists of representations, numbers and symbols.

Figure 4 presents an example mission that consists of the following high-level steps: robot localization, image acquisition in the form of a panoramic picture, target approach and analysis.

5. Conclusions and Future Direction

In this paper, we introduced the S* architecture as a behaviour-based approach to robot control that supports deliberation, without relying on hybrid techniques. As well, an example of a behaviour network was given that demonstrates some of the mechanisms of S*. A mission specification language is also presented that supports monitoring of constraints. A software framework has been developed, and both it and the language have been tested in a simulated environment.

The work described here is ongoing. At present, we are extending the capabilities of our simulation environment to achieve greater physical and visual realism, so that we can perform further testing.

Until now, our efforts have focused on developing and validating the control infrastructure. As a consequence, we have neglected fundamental issues such as sensor error and related problems such as localization, and we have not investigated the role of active and attentive vision with realistic vision problems. In future, we intend to construct a robot controller for a planetary rover testbed [Earon, 2001] complete with stereo vision, a suite of sensors, and a six-wheel differential drive system with active suspension, and eventually extend this research to multi-agent systems.

Acknowledgements

We gratefully acknowledge the generous financial support of the Centre for Research in Earth and Space Technology (CRESTech) and of McDonnell Dettweiler Space and Advanced Robotics, Ltd. (MDRobotics). In particular, we are particularly grateful to Dr. Piotr Jasiobedzki of MDRobotics, for his valuable mentorship.

References

Arkin, R. “Integrating Behavioral, Perceptual and World Knowledge in Reactive Navigation.” *Robotics and Autonomous Systems* 6 (1990) pp 105—122.

Arkin, R. *Behavior-Based Robotics*, MIT Press, 1998.

Brooks, R.A. “A Robust Layered Control System for a Mobile Robot.” *IEEE Journal on Robotics and Automation*, vol RA-2, no. 1, March 1986.

Brooks, R.A. “Intelligence without representation.” *Artificial Intelligence* 47, 1991. pp 139 – 160.

Earon, E.J.P., Barfoot, T. D., D'Eleuterio, G.M.T., "Development of a Multiagent Robotic System with Application to Space Exploration", *Advanced Intelligent Mechatronics*, Como, Italy, 8-11 July, 2001.

Gat, E. “Integrating Planning and Reaction in a Heterogenour Asynchronous Architecture for Controlling Mobile Robots” *Proc of the Tenth National Conference on Aritificial Intellgence (AAAI)*, 1992.

Gat, E. “On Three-Layer Architectures” in D. Kortenkamp et al. eds. *AI and Mobile Robots*. AAAI Press, 1998.

Knight, R. Fisher, F., Estlin, T., Engelhardt, B., Chien S., “Balancing Deliberation and Reaction, Planning and Execution for Space Robotic Applications” *Proc. 2001 IEEE Intl Conf on Intelligent Robotics and Systems*, Maui, Hawaii, USA, Oct. 29 – Nov. 03, 2001 pp 2131 – 2139.

Nicolescu, M., Mataric, M.J., "A Hierarchical Architecture for Behavior-Based Robots", *Proceedings, First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 227-233, Bologna, ITALY, July 15-19, 2002

Nilsson, N.J., *Principles of Artificial Intelligence*, Palo Alto: Tioga. 1980.

Rothenstein, Albert L. “A Mission plan specification language for behaviour-based robots” M.Sc. Thesis, Department of Computer Science, University of Toronto, 2002.

Rotenstein, Andrei M. “Supporting Deliberation Within Behaviour-Based Systems.” M.Sc. Thesis, Department of Computer Science, York University, 2003.

Tsotsos, J.K., "Analyzing Vision at the Complexity Level", *Behavioral and Brain Sciences 13-3*, p423 - 445, 1990.

Tsotsos, J.K., "On Behaviorist Intelligence and the Scaling Problem", *Artificial Intelligence 75*, p 135 - 160, 1995.

Tsotsos, J.K., "Intelligent Control for Perceptually Attentive Agents: The S* Proposal", *Robotics and Autonomous Systems 21-1*, p5-21, July 1997.

Wyatt, E.J., Nilsen, E., “Spacecraft Autonomy Needs Analysis for Planetary Exploration Missions.” *Proceeding of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space: i-SAIRAS 2001*, Canadian Space Agency, St-Hubert, Quebec, Canada, June 18-22, 2001.