Navigation Templates for PSA

Li Tan & David P. Miller

School of Aerospace and Mechanical Engineering

University of Oklahoma

Norman, OK 73019 USA

litan@ou.edu & dpmiller@ou.edu

Abstract

Navigation Templates (NaTs) provide a method for constructing highly flexible navigation plans which capture the essence of the navigation situation (i.e., the task and relevant environmental constraints). This paper presents NaTs as applied to a dimensional simulator of NASA's Personal Satellite Assistant the PSA. The specifics of this application required that the basic NaT algorithm be supplemented with certain memory additions in order to function reliably with the narrow beam range sensors with which the PSA simulator is equipped. These modifications and the results are presented.

Keywords:

Navigation templates, navigation plan, robot navigation

1. Background

The PSA is a new intra-vehicular robot designed by NASA for use inside the International Space Station. Each PSA will be equipped with a camera, display, microphone , speakers and a full micro-gravity mobility system. The PSAs will be used to keep track of object/personnel whereabouts in the station, teleconferencing and monitoring of experiments [1,2]. Because the Space Station is a dynamic environment, the PSA needs to be able to find its way from location to location even when objects are moved contrary to its internal map and even when objects are moving through the same space at the same time in the area it is trying to traverse. PSAs must never run into an astronaut, and should make every possible effort to get out of the astronaut's way.

The experiments described in this paper were done for the PSA 2D simulator. This prototype PSA robot floats over an air-bearing table. Its motion is restricted to X, Y and rotational movement. The usable area of the table is approximately two meters on a side.

2. Overview

When given a task, PSA must be able to quickly plan a plausible route that would allow it to meet its goals. Once PSA decides to attempt a task, it must be able to dynamically plan its path through the actual space. It must be able to modify its actual path from the planned route as needed, while projecting the consequences of those actions throughout the entire plan. There are actually two programs that would be used for PSA navigation. The first we will call the route planner. This system would use a stored model of the environment. The second program will be the path planning system, and real-time issues will dominate the decisions of the planner.

The Navigation Templates (NaTs) algorithm is used to cover both of these needs. Two independent programs are developed, NaTs algorithm with known objects and NaTs algorithm with sensor navigation. Simulation programs have also developed based on Open Inventor [3] C++. in order to test the programs.

3. Navigation Templates

Navigation Templates (NaTs) [4,5] are primitive building blocks for constructing highly flexible navigation plans which capture the essence of the navigation situation (i.e., the task and relevant environmental constraints). There are two types of Navigation templates: those that are used to characterize the basic local navigation task being pursued (Substrate Navigation Templates, S-NaTs), and

those used to model known environmental constraints and characterize the relationship of constraints to the navigation task (Modifier Navigation Templates, M-NaTs). Once a navigation plan has been built from a set of Navigation Templates, a powerful heuristic is employed to isolate the currently critical aspects of the plan and quickly generate guidance for the robot's low-level control system. Figure 1 shows an example of a NaTs-based navigation plan. NaTs have been used for a variety of robot navigation applications, some of which are described in [6].

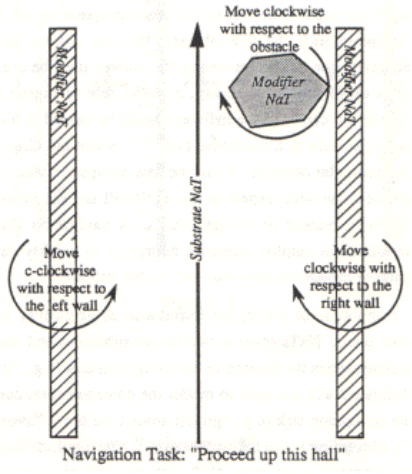

Figure 1: An example of NaTs based navigation plan (from [4])

4. Coordinate System

There are one global and two local coordinate systems in the simulation program. The global system's center is at the center of the air-bearing table, and its X-axis is Open Inventor's X-axis that points to the right of the window and its Y-axis is Open Inventor's Z-axis that points out of the window. The global system is used to process sensor or known object data and part of the parameters of the M-NaTs and vectors are created based on this system.

The first local system's origin is the PSA simulator's center and its X-axis and Y-axis are parallel to those of global system. The PSA simulator's normal is identical to this Y-axis.

The second local system (PSA system's) origin is the PSA simulator's center and its Y-axis is the current S-NaTs. The X-axis is in the direction that follows Right-Hand-Rule and its Z-axis points into the window.

In the simulation program, the PSA simulator does not change its orientation. However, the PSA simulator's orientation would be changed to follow S-NaT. Thus, in practice, only the second local system is used.

These three coordinate systems are illustrated as Fig 2.

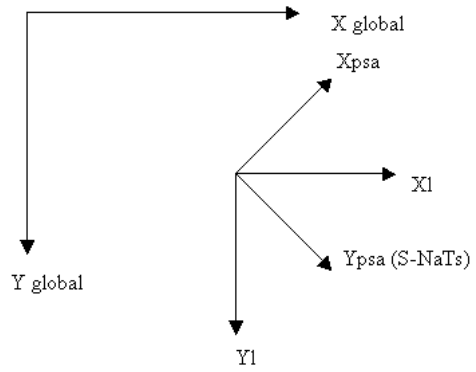

Figure 2: Coordinate Systems

5. NaTs Algorithm Program

Two NaTs algorithm programs were developed. One is with known objects and another one is with sensor navigation and object memory.

5.1, NaTs Algorithm Program With Known Objects

In this program, all objects are defined by the user prior to the robot's traverse. The location and shape of all of the objects in the environment are available to the NaTs system. The parameters of objects are obtained through a function processing input objects. The objects in the simulation are defined by their center, orientation and length. A function is then called to convert the above inputs to end points and distances, then another function creates obstacle structures which convert known object parameters to those structures NaTs algorithm need, such as spin, Relevant Primary Bound and so on. The algorithm goes over all the effective objects (objects whose position might effect the robot's path) and finally outputs a navigation path for PSA simulator. The output is in the form of an instantaneous gradient of direction for a particular position. To calculate a path,

1. start the robot at its original position
2. calculate the gradient for that position based on the NaTs of known objects (M-NaTs) and the NaT from the goal (S-Nat).
3. move the robot (in simulation) through a time step along that gradient

4. calculate the robot's new position

5. got to step 2 and repeat until the robot's calculate position is sufficiently close to the goal.

5.2. NaTs Algorithm Program With Sensor Navigation

In this program, the PSA simulator is assumed to navigate by range sensors. The sensors on the PSA are an array of SHARP GP2D12 distance sensors which output an analog value corresponsding to the distance to the object within the field of view of the sensor. The viewing angle of each sensor is significantly less than a degree. The effective range of the sensors is from approximately 6cm to one meter.

The algorithm only processes the objects (points) that the simulator can sense. The objects consists of points that are obtained directly from sensors, so there is no limitation on the shape of the objects in the model. Sensed points are clustered to the same object if they are within sufficient distance. This distance is determined by the size of the robot – if the robot could not fit between two sensed points then for all practical purposes those two points form a single obstacle around which the robot must navigate.

A function is called to create the obstacle structures which NaTs algorithm need such as spin, Relevant Primary Bounds, and so on. The algorithm goes over all effective objects and finally outputs a navigation path for the PSA simulator. The algorithm also provides object memory whose function is to compare objects in memory with currently sensed objects and assure that current objects have the same spin direction to previously sensed objects that have similar locations in space.

Without this memory, slight changes in viewing angle could cause the currently viewed Nat to have the opposite spin of the Nat that was viewed in a similar location moments previously. This could cause the robot to reverse course, causing the next observation set to be from a previous viewpoint and thereby causing the newly viewed NaT to go back to the previous spin, leading to the robot getting stuck oscillating between views. Because the only sensory data coming in to the robot from the outside objects is a sparse set of range points, it is not possible to absolutely identify any of those points as belonging to the same object as previously observed points.

5.3. Object Memory

PSA simulator could stall when navigation plan is created through range sensor information. This situation is illustrated as follows. In (a), A, B, C, and D are points PSA can sense, and they are clustered to two objects with two different spin directions (PSA does not know anything about the two objects except for the sensed points); in (b), as PSA moves forwards in the direction of arrow, it senses A', B', C', D', E' and F' points. Since these points are close, they form only one object with one spin direction. A' and B' are so close that PSA cannot go through, thus, it moves backward and the situation returns to the one shown in Figure 3(a). Then, PSA moves forward again since it senses two different objects again, starting an oscillation. PSA is stuck.

In order to solve this problem, object memory is introduced. Object information, such as center, contact point, left/right most point and spin direction are saved in this object memory.

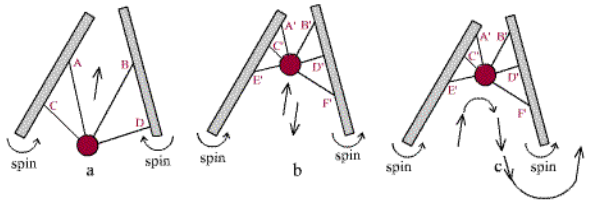

Figure 3: Spin Oscillations from Changes in Observation Position

PSA always compares currently clustered objects with objects in the memory. Distances between centers, contact points, and left/right most points are evaluated. The current object is assigned with the same spin direction as objects in the memory if any of them falls into the specified range. This distance range is usually decided according to the dimension of PSA plus a certain margin. The information for the objects in the memory is updated after the comparison.

The effect of object memory is shown in (c). Every time after PSA checks memory, sensed objects are forced to coordinate spin direction with those in memory, thus PSA can move round the objects, and oscillations are avoided.

5.4. Object Memory and Dynamic Objects

While object memory is useful for avoiding oscillations, it cannot be used indiscriminately. Two problems occur if object memory persists forever.

First, if objects persisted in memory indefinitely then NaTs would grow in size with dead-reckoning errors. A small object that is observed multiple times will have the observation points spread out over space as a function of the observation errors and absolute positioning error of the robot, in addition to the object's size. Assuming that the errors never exceed the robot's dimension between observation, each observation will grow the size of the Nat. Should a large error occur, then a new separate Nat will be created for the erroneous observation and the robot will navigate around phantom objects.

A second issue with memory persistence is dynamic objects. If the PSA encounters an astronaut in the Space Station, the robot should move around the astronaut. But if the astronaut is moving as well, and if memory persists, then the astronaut will form a barrier that is the thickness of an astronaut and the length of the observed movement.

To overcome both of these problems, it is important to have object memories fade over time. The optimal memory decay rate depends on the amount of dynamics in the environment, the speed of the robot and the absolute positioning error. If all are small then memory can persist for long times. If they are large them old object memories must be purged after only a few observation cycles.

6. NaTs Simulation

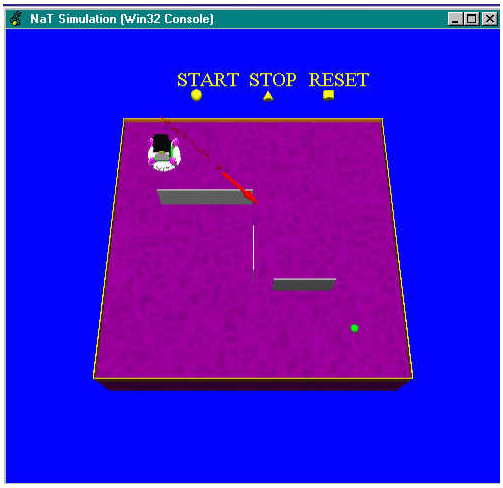

Figure 4: Known Objects Simulation

6.1. Overview

The simulation is based on Open Inventor C++. Open Inventor is the de facto standard for the development, management, and interchange of 3D graphics. It is an object-oriented developer's toolkit for C++. In order to simulate and visualize NaTs algorithm, two Win32 console simulation programs are developed, one is for NaTs algorithm with known object and another one is for NaTs with sensor navigation.

Figure 5: Sensed Objects Simulation Window

6.2, Simulation of NaTs with Known Objects

The following picture shows simulation window for NaTs with known objects.

The object type is limited to very thin panel (linear type object). The PSA is placed at original start position, and green point indicates target position. The simulation program can be started, stopped and reset by press START, STOP and RESET buttons.

The input for the simulation is initial and target positions of PSA simulator, the number of linear type objects, and the parameters of objects in the format of X coordinate of center, Y coordinates of center, orientation (degree) and length.

After all parameters are entered, the Inventor simulation window comes up as shown in Figure 4.

6.3. Simulation of NaTs with Sensor Navigation

In this simulation, objects can constructed from cubes and cylinders. The objects are used by the sensor simulator to generate sensor readings, but the objects are not known to the navigation system. The input for the simulation is Initial and target positions of PSA simulator, the number and type objects, and the parameters of objects in the format of X coordinate of center, Y coordinates of center,

orientation (degree). Sensed points are marked in different color according to its spin directions.

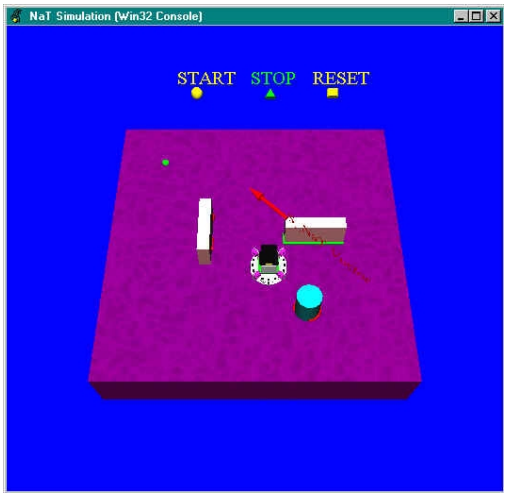

Figure 6: Sensed Points

7. Conclusion

We have presented an implementation of Slack's NaT algorithm tailored to the needs of NASA's PSA simulator. The NaT algorithm works very well for both known objects and sensor navigation in the simulation. Our memory extensions to Slack's original system allows reliable navigation with sparse point sensors. The object memory can avoid some oscillation situations during PSA simulator navigation. Improvements in memory efficiency are still desirable.

Acknowledgements

The authors wish to thank Adam Sweet and Greg Dorais for their help on this project. This work was supported in part under USRA Contract # 08008-002-008-001 and NASA Contract # NCC-2-8032

References

[1] The Personal Satellite Assistant, http://ic-www.arc.nasa.gov/ic/psa/.

[2] Yuri Gawdiak1, Jeff Bradshaw2, Brian Williams3, Hans Thomas, R2D2 in a Softball: The Portable Satellite Assistant. in the *Proceedings of the 2000 ACM Intelligent User Interfaces Conference*, New Orleans, January 2000.

[3] Josie Wernecke, "The Inventor mentor"

[4] Marc Slack, Dissertation title: *Situationally Driven Local Navigation for Mobile Robots*, Ph.D., Department of Computer Science, Virginia Tech, May 1990.

[5] Marc G. Slack, Navigation Templates: Mediating Qualitative Guidance and Quantitative Control in Mobile Robots, *IEEE Trans. On System, Man, and Cybernetics*, *Vol. 23, No. 2*, March, 1993

[6] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack, Experiences with an Architecture for Intelligent, Reactive Agents, *Journal of Experimental and Theoretical Artificial Intelligence, Vol. 9, No. 1*, 1997.