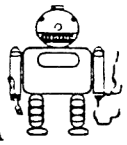
ROBOT BUILDER



Robotics Society of Southern California December 1990

Upcoming Events Calendar.

December 4	RSSC December Meeting, MTI College: Topic - Batteries 7-9 PM.
December 8	RSSC Robot Project Workshop, The Robot Company 10-12 AM.
December 25	**Merry Christmas** to all.
January 1	**Happy Newyear.**
January 8	RSSC Meeting at MTI College: 7-9 PM. Note 2nd Tuesday in January.
January 12	RSSC Robot Project Workshop, at Jerry Burton's Laboratory.

November 6th RSSC MEETING

We had a well attended meeting. We need all members to pay their dues to carry us into the next year. If you have not paid your dues, please bring them to the next meeting.

The Topic for the evening was *MOTORS* and present by Tom Carroll. Excerpt from C & H surplus catalog was distributed and discussed. This catalog, has many surplus motors for sale. Tom explained which motors were best for robotic applications based on years of robotic experience.

The meeting concluded with the demo of the club's robot. The unit was aloud to wonder the halls of MTI College. Thanks to Mark Frank for his work in building this fine unit.

o

November 10 Work Shop

The group that gathered played with the club's robot and then collected to discuss methods of data transmission to the robot. A RF modem is needed transmit the data to and from a remote station.

o

MERRY CHRISTMAS

AND

HAPPY NEWYEAR

The Roboteer
by Jerry Burton

Last month I discussed the E_DEF module and the services it needed to provide. In reviewing the article it wasn't apparent that for the E_DEF module to allow the user to modify the E_MAP and P_MAP that it must provide a graphical interface. It should show as minimum the tiles that it thinks are empty, occupied, and which are designated a path tiles.

When the user selects certain tiles they should be highlighted in some fashion, so the user can tell what subset a given action applies to. The society robot has an EGA card so a rather good picture should be able to be generated. Since normally we use a monochrome monitor we can use bright, normal, low, and blinking attributes to show the various states of each tile.

This month I'll cover the PLANNING module. For the sake of explanation, let's assume that you have a complete map of your environment and have defined a task point BEER, and a task to be executed at that location (assume we entered GETBEER, which somehow gets the beer once the robot is positioned correctly), in the class KITCHEN using the E_DEF module. Also assume the knowledge base can recognize the command "GET me a BEER" (you only have to train GET and BEER, since the word GET triggers the recognition and BEER completes the command, the words in between are ignored). The speech subsystem puts GET BEER in the keyboard buffer. The knowledge base then calls the PLANNING module and supplies the action GET and the destination BEER.

The action GET implies a round trip, whereas the action GO implies a one-way trip.

We must also assume that the robot knows where it currently is. The first thing the PLANNING module has to do is search through the task structure (table, or list, or ?) to get the global coordinates for the destination BEER. Each entry must contain the task point name (BEER), its location (global X,Y and class KITCHEN), and the task to be executed once the destination is reached (GETBEER).

The next step is to determine how to get from where the robot is to the final destination. This requires that the PLANNING module search through the P_MAP to find an appropriate path. The direct method is to search through P-MAP to find the shortest path from the initial point to the destination point. There are a number of tree searching algorithms defined in the AI literature to solve this type of problem., however, they can be time consuming and are best handled by LISP or PROLOG type languages.

In order to speed up this process I use the concept of a class transition matrix.

For example, assume we have 5 classes A-E, with 5 exit points, the transition matrix is as follows :

	A	B	C	D	E
A		1			
B	1		1		
C		1		1	
D			1		1
E			1	1	

The table has a 1 if a transition from one class to another is possible. This means that the robot can go from A to B, but not to C, D, E. From class B the robot can go to A or C, but not D or E, etc. The primary reason for using a class transition matrix is to limit the search of the P-MAP.

If it is required to go from a point p1 in A to a point p2 in E then the following set of commands would be generated. Go from point p1 in A to the AB-exit point, from B to C, C to CE-exit, and the CE-exit to the destination point p2. Another path is from point p1 in A to AB-exit, from B to C, C to D, D to DE-exit, and DE-exit to point p2 in E.

The search algorithm should determine the shortest path of all possible paths available. Note that the initial move, from p1 in A to AB-exit and DE-exit to p2, are identical so the only part of the P-MAP that needs to searched is in the intermediate classes. If the initial point and the destination are in the same class then a direct path can be calculated and the class transition matrix does not need to be checked. If the initial and destination classes are adjacent then the transition class matrix gives an immediate solution. If there are intervening classes then the transition matrix gives the allowable transitions.

It is possible to determine a set of sub-goals to transition from one class to another so the intermediate paths do not have to be calculated each time. This is accomplished by calculating transition paths for all classes that have multiple exits.

For example, since class B has 2 exit points, a precalculated path would give a path from the AB-exit to the BC-exit point. Since class C has 3 exit points, there are 3 paths: 1) from the BC-exit to DC-exit, which provides a transition from B to D via C; 2) from the BC-exit to CE-exit, which provides a transition from B to E via C; from CE-exit to DC-exit, which provides a transition from D to E via C. In general, there are n times (n-1) divided by 2 transition paths for a class with n exits.

Using this strategy cuts down the amount of searching required. We only have to build paths from the initial point to an exit point and from the final exit point to the destination point. The simplest way to do this is traverse the P-MAP from the exit point to the initial/destination point and count the number of tiles in the path generated. Since there may be multiple paths (depending on the width of the P-MAP), we keep generating paths until all the tiles in the P-MAP within the class have been checked. The path with the lowest number represents the shortest path.

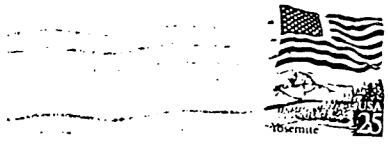
The output from the PLANNING module is a set of X,Y points that the NAVIGATION module is to move the robot through. My current implementation is to enter the NAVIGATION module with a stack containing X,Y points. The NAVIGATION module then calculates any heading changes and distances required to go from the robots current position to the next point in the stack. Once the stack is empty the destination point has been reached and NAVIGATION returns with a completion flag set TRUE. If the completion flag is FALSE, then PLANNING with calculate another path to the destination.

The only reason the NAVIGATION module returns a FALSE completion flag is that it encountered an obstacle and could not get around it. This implies that the NAVIGATION module updates the E-MAP and P-MAP on the fly, so that the PLANNING module will calculate a new path.

Next month I'll go through the NAVIGATION module which will complete the overall discussion of the Navigation system. After that I'll start presenting more detail of each of the modules and what functions they have to contain, and eventually the C++ code which implements them.

o

Robotics Society of Southern California
1125 Birch Ave.
Orange, CA 92714



Mr. Scott MacGillivray
4436 Ostrom Ave.
Lakewood, CA 90713