Please read me first!

Thanks for using my source code to your studying. Good luck to you. If you need more information, please feel free to send a message to my email:

Nguyenchanhtruc2005@gmail.com

A. Instruction for the project

1. In my source code, I have developed A*, D*, and D*Lite. I am not a professional UI designer, so I made 3 Forms for the three algorithms separately for being easy to test. Although I have fix a lot of bugs, there still remain many bugs related to UI. In my opinion, there are no bugs about algorithm logic.
   The performance of these algorithms is: A* < D*Lite < D*. D* is the best one, although some articles mentioned that D*Lite has been used for the Robot to explore Mars. D*Lite is the most painful algorithm to fully understand. I hate it much! D* is easier D*Lite in programming, but the easiest one is A*.

2. To run one of these Forms:
   a. Open Visual C#
   b. Open the project
   c. Open Program.cs file
   d. You will see these codes lines:

```
4       static void Main()
5       {
6           Application.EnableVisualStyles();
7           Application.SetCompatibleTextRenderingDefault(false);
8           //Application.Run(new DStarForm());
9           //Application.Run(new A_Star_Form());
0           Application.Run(new DStarLiteForm());
1       }
```

   Just remove "//" at the Form you want to run, and add "//" at the others. As you see, the Form of DstarLiteForm algorithm is chosen, the others (D star and A star) are set as comment lines. Then, compile!

B. Instruction about the algorithms

To understand the source code quickly, I think you should know my spinal ideas. However, the below instruction is only a hint, not fully explained.

1. **Grid map**: I design a grid map as below:

   In Figure 1, there are totally 25 nodes identified by IDs in decimal numbers (from 0 to 24). Beside the IDs, nodes' coordinates are determined in a 2-D Cartesian coordinate system. Beginning at the origin, the node 0 has coordinate (0, 0) and the node 24 has the coordinate (4, 4). This pair of data, Coordinate - ID, is the most basic and important information of a node for further conveniences of path planning algorithms.

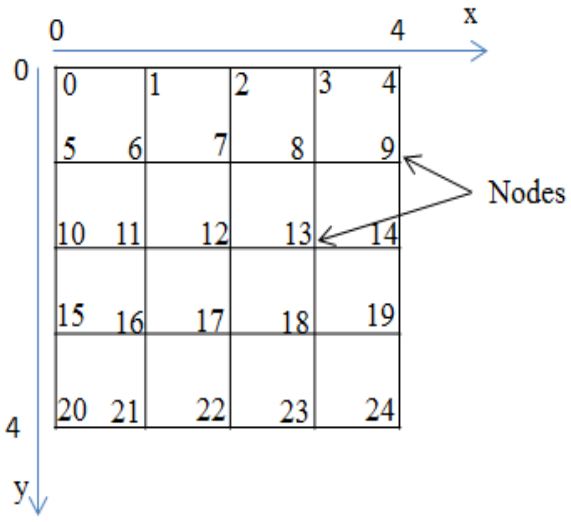

Fig.2 Nodes on a grid map

2. **You should understand static environment and dynamic environment. If not understand yet, read the below explanation. If you have already known before, just skip this section.**

- There are two types of working environments, static environment and dynamic environment. The static environment assumes that it is completely known by the robot and unchanged over the time. This means that after searching the shortest path, the robot will move on the path and reach its target without collision with new obstacles which appear suddenly and unexpectedly in the map. In reality, the terrain can be changed anytime because of thousands of reasons. For example, in Figure 1a, there

are a number of random obstacles which are known coordinates before, a robot on the top, and a target on the bottom of the map. As shown in the Figure 1b, the robot uses a path planning algorithm to search for the shortest path such as the A* algorithm. Then the robot begins moving from its starting coordinate to the target on the found path. However, a new obstacle appears and obstructs the path as in Figure 1c. The robot keeps getting closer to the obstacle until its sensors detect the unexpected obstacle. Normally, a solution in this case is updating the obstacle to the map, then running A* algorithm again to search a new path from the current coordinate to the target (Figure 1d). By this way, the algorithm must be repeated each time the robot is blocked until the robot reaches the target. (Figure 1e, and Figure 1f).

Repeating a path planning algorithm is not an efficient method for a large environment. It was measured that when the A* algorithm was simulated in a 200 x 200 grid map using a computer with CPU core i5, 4GB RAM, the consumed time was about 30 seconds for one time operating. It would take more time if the robot has to do again the A* algorithm for searching a new path.

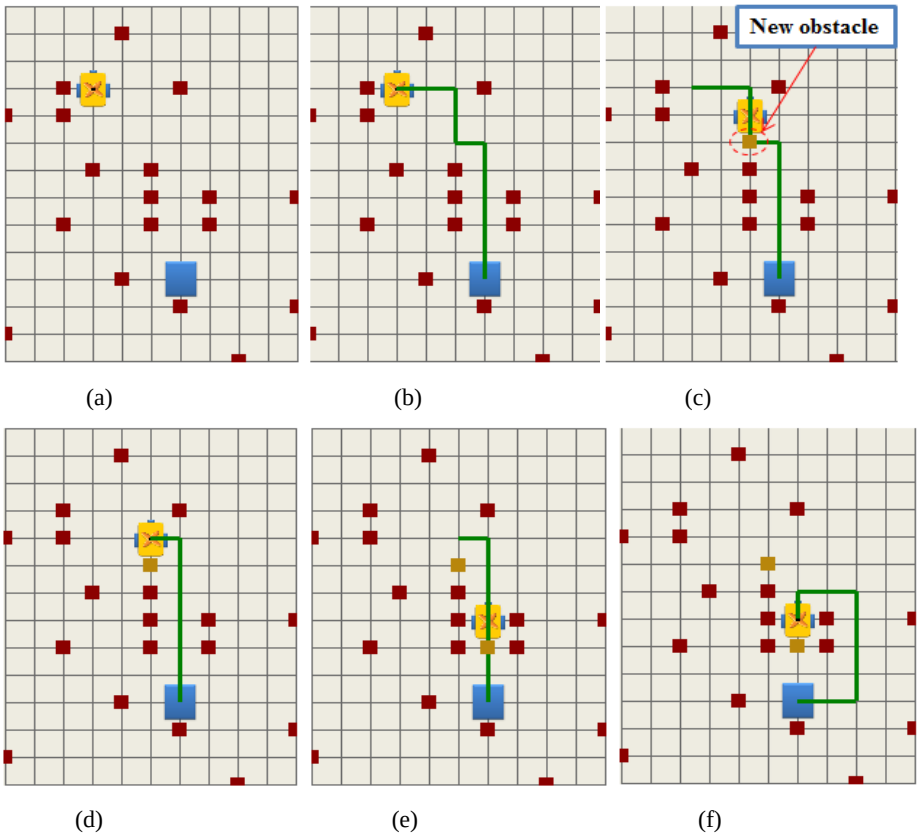

Fig.1 Path planning in the dynamic environment

Therefore, the robot needs an incremental search algorithm which is faster and more efficient than A* algorithm. The simplest way is moving around the obstacle until the robot meets the old path, and then continues the path to arrive the target. This method is fast because it does not need much computing. However, it is clearly not optimal since there are probably paths shorter than going around the perimeter. Another method is updating the new obstacle and also path costs of associated nodes instead of calculating again whole things. By this way, the time period of the next search would be much shorter, and the found path always is the shortest. This method is called incremental search algorithm.

3. **Nodes in algorithms:**
   - The number of node in path planning algorithms must be equal to the number of node in a grid map. Each node has a coordinate(x,y).
   - Nodes are defined in class: Node_Define.cs, you can find all nodes' attributes in this file.
4. **Important note for D*Lite (just read this section after you have a deep reading the paper: http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf.)**
   a. My D*Lite is not 100% as the original version. First, because my U_List is sorted by a different way, so I do not need to use K_m to optimize the sorting process.

   The paper does not mention how to sort Nodes in U_List, it just gives you a standard to sort. So I use my own compare method. For example:

   You have an empty array (in this case, U_List), Someone gives you a number to place to the array. After placing, he gives you another one, and you must add these numbers to the array from small to large. I do as below:

   + Receive **a**, Add node **a** to the array

   + Receive node **b**, compare it to node a. If **b** > **a**, **b** should be placed behind **a**. If **b** <= **a**, it should be placed before **a**.

+ Receive node **c**. Remember, **a** and **b** are already sorted from smallest to largest. So, compare c to **a, b** to see which is larger than **c**, then place **c** before this number, if there is no number larger than **c**, place **c** at the end of array.

+ receive **d**, because **a, b,** and **c** are already from smallest to largest. So, compare **d** to **a**, **b, c** to see which is larger than **d**, then place **d** before this number, if there is no number larger than **d**, place **d** at the end of array.

+ do the same for other numbers….

b. Because my D*Lite uses only perpendicular directions, the result usually is a zigzag line, so I have optimized the path in Smallest_Rhs_Node function, which does not exists in original paper.
c. About Predecessors and Successors, they have made me confused and had headache for a long time. I hope that the below explanation can help you to avoid buying mental medicine! Please have a quick look below:

D* Lite manages nodes in a map by classifying them as Successor and Predecessor. Successor is a vertex which is the one already checked in the previous iteration (parent nodes). Predecessor is a vertex which is checked in the next iteration (children nodes).

In Figure 2, vertexes A, B, C can go to S. S and G can go to E and F. A, B, C have been calculated before, so they are successors of S, while E and F are Predecessors. Similarly, E and F are also predecessors of G. The number of successors and predecessors of a vertex can vary from 0 to 4 since the maximum number of neighbours of a vertex is 4.

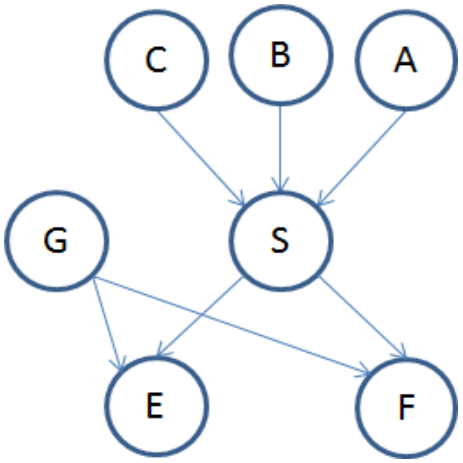

Fig.3 Predecessor and Successor illustration

5. **Note for D* (just read after you finish reading the paper of this algorithm: http://www.frc.ri.cmu.edu/~axs/doc/icra94.pdf)**

The path also has the same phenomena as D*Lite (zigzag). So I optimized it by adding an extra number to **Arc cost.**

----------END----------