Adavnaced Algorithm Implementation: D* and Informed RRT*

Shiva Kumar Tekumatla
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester,MA
stekumatla@wpi.edu

***Abstract*—Traditional discrete planning based algorithms fail to perform in the dynamic environments.For eample, A* path can be computed assuming each state of the change in environment as static , and then recompute every time the environment or obstacles are changed. However, this is a very slow process ,and by the time a feasible path is found by A* , the dynamics of the environment may change. This is where D* algorithm helps. D* is known as dynamic A * , which uses local re-planning to reconstruct the damaged path. In this paper, I discuss the implementation of D* algorithm. In sampling based methods, RRT and RRT* are well known and can be used to find the feasible path in static environments. RRT* simply uses rewiring method to optimized the path computed by RRT. In RRT* , while rewiring , the samples are randomly formed again in an entire work space. However, in informed RRT*, the samples are formed in a locally formed ellipse with start and goal positions as foci. Informed RRT* is faster compared in RRT* and can result in better and optimized paths. In this paper, I discuss the implementation of informed RRT*, and compare the results with RRT star.**

Index Terms*—Discrete planning,D*, Sampling based planning, RRT and RRT*, and Informed RRT

I. Introduction

D star, also known as dynamic A *, is an incremental search algorithm that works on free space assumption. This algorithm can be used when the robot has to navigate in an unknown terrain. Robot initially assumes that there are no obstacles in the unknown terrain forms a path. Robot follows this path , and observes the environment for the changes. If the robot detects any changes in the environment, for example, a new dynamic obstacle, it adds this information to the map , and then recompute the path if necessary by repairing the pre-computed disturbed paths. This process is repeated until robot reaches the goal.However, in the practical implementation this computation should be fast to make sure the change in dynamics is not faster than the robots reaction rate. D* has many variants: 1) Original D* 2) Focused D* 3) D* Lite. All these algorithms are faster than repeated A* searches.

Informed RRT* is a variant of RRT* algorithm. As we have discussed before in the standard algorithm implementation, RRT* is an extension of RRT algorithm.Rapidly exploring Random Tree(RRT) is an algorithm that can efficiently search non-convex spaces by randomly sampling the maps. The tree is grown from start until the goal point is found using one of the extend or connect methods. As each node is sampled, a connection is made to the nearest available nodes in the already constructed tree. This process is repeated until the max number of predetermined nodes is reached or the goal is reached. Each step in constructing the node is defined beforehand. This step can be determined as simply the euclidean distance. The main difference between RRT and RRT * is the rewiring function. Once the path is found using RRT, we try to optimize it by sampling more points in the entire work space. However, the path quality depends on the number of sampling we do. More the samples can result in more optimized path. But, once the path is found, we may not need to sample the entire work space again. Simply by focusing on the area more close to the already found path, we can improve the time complexity of the algorithm .This exact method is followed in informed RRT* algorithm. Once a feasible path is found, an ellipse is formed based on the minimum distance , that is the distance between start and goal, and the best distance that was found by RRT*. Next sampling is fully done inside this shrinking ellipse , which over the time can result in more optimized path compared to RRT*. Figure 1 shows the ellipse that is used as sampling region in informed RRT*.

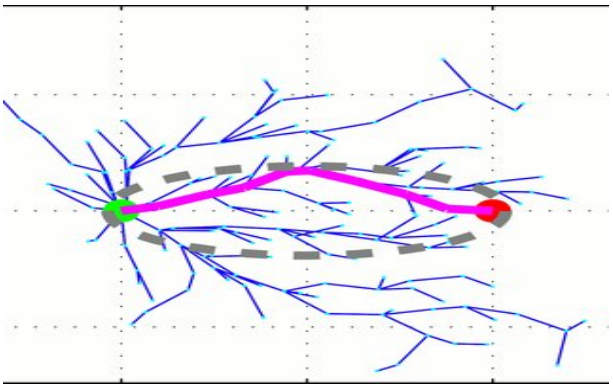

Fig. 1. Ellipse region that is used in Informed RRT*

In this paper, I discuss about the each step involved in the implementation of D star and Informed RRT* . For D * , I present the cost modification based on the dynamic obstacle information, repairing the damaged paths , and re planning based on the available environment information. Further , I explain the differences between RRT* and Informed RRT* and

give a detailed explanation of the implementation of Informed RRT*.

II. Implementation of D*

In this section, the implementation of D* is discussed.Same as Djikstra or A* , D* algorithm also maintains an open list. Each node in the environment has different states such as: NEW, OPEN, CLOSED,RAISE and LOWER. Based on the state of each node, that are processed using a processing operation. The cost of the path is computed generally using euclidean distance method, but based on the changes this cost is modified. This is discussed in the modify cost sub section. All the affected nodes or the path is repaired using prepare repair method that is discussed in the prepare-repair sub section. The path repairing is discussed in the repair re-plan sub section.

A. Process State

In process state function , we compute the optimal path to the goal. We initially set goal's heuristic as zero and then insert into the open list. We keep calling the process state function until the robot's state is removed from the open list. Figure 2 shows the pseudo code for the process state function.

```
X = MIN-STATE( )
if  X= NULL then return –1
k_old = GET-KMIN( );  DELETE(X);
if k_old< h(X) then
   for each neighbor Y of X:
            if t(Y) ≠ new and  h(Y) <= k_old and h(X) > h(Y) +
c(Y,X) then
              b(X) = Y;  h(X) = h(Y)+c(Y,X);
if k_old= h(X) then
   for each neighbor Y of X:
            if t(Y) = NEW or
            (b(Y) =X and h(Y) ≠ h(X)+c (X,Y) ) or
            (b(Y) ≠ X and h(Y) > h(X)+c (X,Y) ) then
            b(Y) = X ; INSERT(Y, h(X)+c(X,Y))
else
   for each neighbor Y of X:
            if t(Y) = NEW or
            (b(Y) =X and h(Y) ≠ h(X)+c (X,Y) ) then
               b(Y) = X ; INSERT(Y, h(X)+c(X,Y))
            else
               if  b(Y) ≠ X and h(Y) > h(X)+c (X,Y) then
                  INSERT(X, h(X))
               else
                  if  b(Y) ≠ X and h(X) > h(Y)+c (X,Y) and
                  t(Y) = CLOSED and h(Y) > k_old then
                  INSERT(Y, h(Y))
Return GET-KMIN ( )
```

Fig. 2. Pseudo-code for process state method

We start the process state by selecting the minimum node from the open list. We then find the neighbors of this node . Before finding the raise or lower states of the node, we get the minimal k value from the open list. Then we compare this k values with the heuristic of the selected node. If the heuristic value is greater than the k value then it is considered as the raise state. If the found state is raise then we perform the operations that are shown in figure 2. Same way lower state is found and corresponding operations are performed as shown in the figure 2. Finally the minimum k value is returned which is further reduced in the upcoming steps.

B. Modify cost

This method is called when the robot detects am error in the cost function , which is nothing but the robot found a new obstacle. In this method , we change the cost of the effected states. Pseudo-code for this is given in figure 3.

$MODIFY-COST(O,X,Y,cval)$
1: $c(X,Y) = cval$
2: if $t(X) = CLOSED$ then
3: $INSERT(O,X,h(X))$
4: end if
5: Return $GET-KMIN(O)$

Fig. 3. Pseudo code for modify cost

C. Repair Re plan

In this method, we find the minimum k value of the process state by calling process state method until the open node is completely empty or k value is greater than the heuristic of corresponding node. The pseudo code for this method is given in figure 4.

$REPAIR-REPLAN(O,L,X_c,G)$
1: repeat
2: $k_{min} = PROCESS-STATE(O,L)$
3: until $(k_{min} \geq h(X_c))$ or $(k_{min} = -1)$
4: $P = GET-BACKPOINTER-LIST(L,X_c,G)$
5: Return (P)

Fig. 4. Pseudo code for repair re-plan

D. Prepare Repair

In this method, we find the neighbors of a given node and check if it is an obstacle or dynamic obstacle. If the neighbor is a dynamic obstacle , then we find the neighbors of each neighbor and then modify their cost. Pseudo code for this given in figure 5.

Once these base functions are defined , I use the method given in the figure 6. In this method,the main procedure of D star algorithm is implemented. Initially, the heuristic of goal is set to zero. Goal node is added to the open node and basic path finding algorithm such Djikstra or A* is used to find a feasible path from goal to the start. Once the path is found, the robot is moved along the path while taking the sensor inputs from thr sensors. At each state , the nodes are repaired based on the affected nodes and paths and parents of each affected node is evaluated. This process is repeated until the robot reaches the goal if a feasible path is found.

E. Bresenham's Algorithm

III. Results of D*

A. Map-1

Figures 7, 8 and 9 show the results of D* implementation on a relatively simple map. Here initially a path is computed , and

$PREPARE - REPAIR(O, L, X_c)$

for each state $X \in L$ within sensor range of X_c and X_c do
  for each neighbor Y of X do
    if $r(Y,X) \neq c(Y,X)$ then
      $MODIFY - COST(O, Y, X, r(Y,X))$
    end if
  end for
  for each neighbor Y of X do
    if $r(X,Y) \neq c(X,Y)$ then
      $MODIFY - COST(O, X, Y, r(X,Y))$
    end if
  end for
end for

Fig. 5. Pseudo code for prepare repair

Input: List of all states L
Output: The goal state, if it is reachable, and the list of states L are updated so that the backpointer list describes a path from the start to the goal. If the goal state is not reachable, return NULL.

for each $X \in L$ **do**
  t(X) = NEW
end for
$h(G) = 0$
$INSERT(O, G, h(G))$
$X_c = S$
$P = INIT - PLAN(O, L, X_c, G)$
if P = NULL **then**
  Return (NULL)
end if
while $X_c \neq G$ **do**
  $PREPARE - REPAIR(O, L, X_c)$
  $P = REPAIR - REPLAN(O, L, X_c, G)$
  if P = NULL **then**
    Return (NULL)
  end if
  X_c = the second element of P {Move to the next state in P}.
end while
Return (X_c)

Fig. 6. Pseudo code for modify cost

the robot starts to traverse that path, which is given in figure 7. But after starting, the robot senses an obstacle and changes the path, which is given in figure 8. Then robot continues to follow this path until goal position is reached. Figure 9 shows the robot reaching the goal.

B. Map-2

Map-2 is relatively a complex map , and has many dynamic obstacles. This map has a total of three dynamic obstacles and we can see the robot changing path after encountering the second and third obstacles. As the first obstacle did not effect the path , the path remains the same. We can see the results in figures 10,11,12,13 and 14.

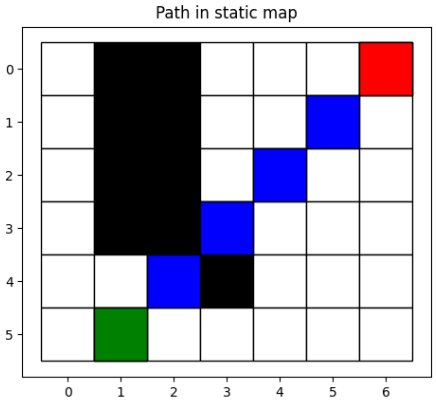

Fig. 7. D star algorithm implementation on map-1. Here , initially generated static path is shown in the given map

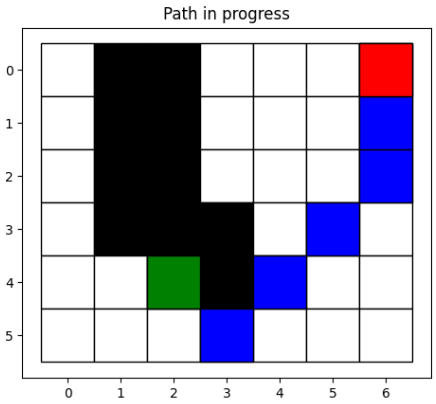

Fig. 8. Change in path once an obstacle is encountered

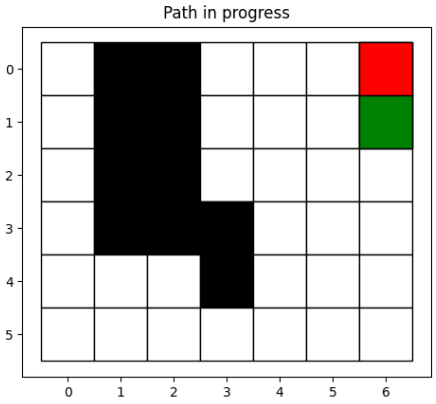

Fig. 9. Robot reaches the goal in map-1

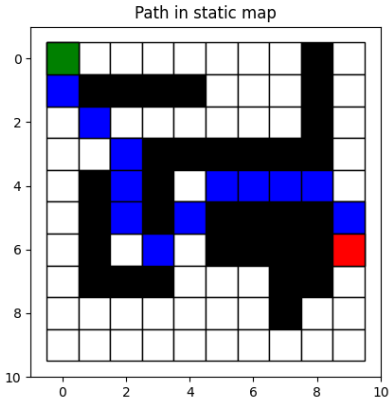

Fig. 10. Static path in map-2

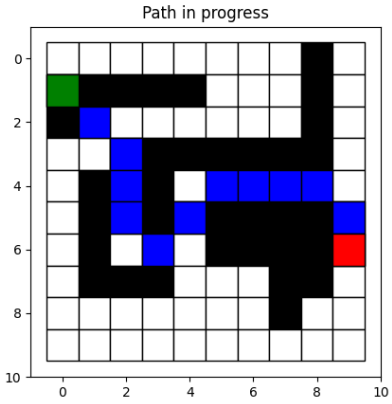

Fig. 11. Path is not effected as the dynamic obstacle has not disturbed the previous path

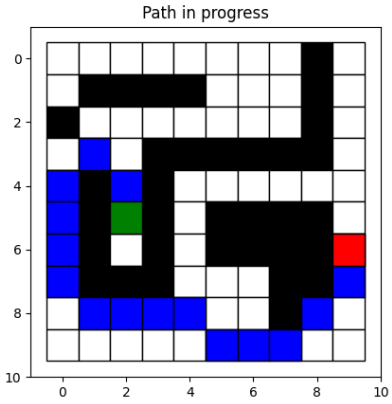

Fig. 12. Path is changed after the second obstacle is encountered

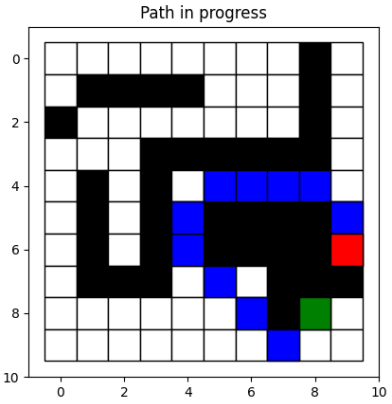

Fig. 13. Path is changed after the third obstacle is encountered

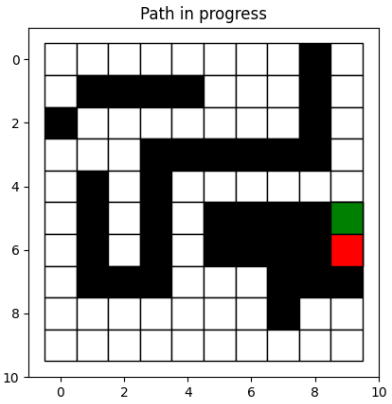

Fig. 14. Robot reaches the goal

C. Map-3

Map 3 is similar to map 1. However, in the end the dynamic obstacle completely covers the goal which makes goal unreachable to the robot. These results are given in figure 15 and 16.

D. D* vs A* or Djikstra

When the environment is dynamic, A* or Djikstra try to find the path by repeating the whole process. But in D* , the repairs done locally. This means that D* is comparatively faster than A* or Djikstra. The main reason for choosing D* is that it is an incremental based algorithm. Mainly, when the initial path is disturbed, D* can take advantage of the previous path. Dijkstra or A* cannot use the previous computation and have to start from the beginning. It needs to recompute everything from start.

IV. Implementation of Informed RRT*

Overall, the informed RRT* implementation is same as RRT* except one main difference. That is , instead of sampling the entire work space, the sampling is done only in the ellipse

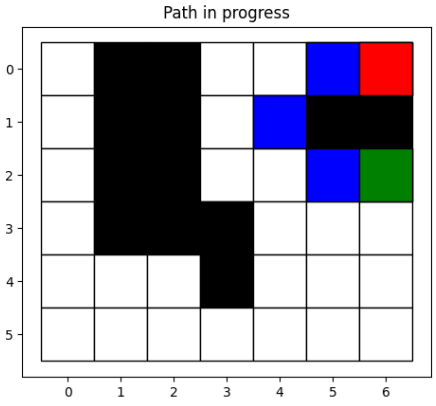

Fig. 15. Change in path after the dynamic obstacle is detected

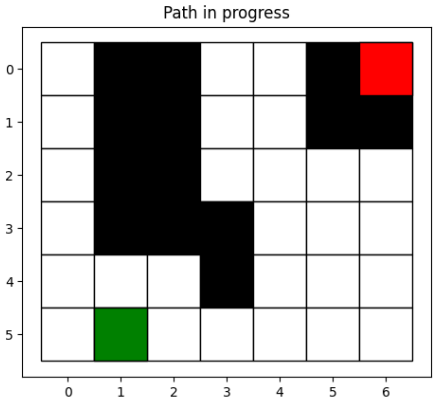

Fig. 16. Dynamic obstacle completely covers the goal, which makes the goal unreachable to the robot

that is formed around the start and goal position.RRT* is similar to RRT,but it can give more optimized path.There is mainly one extra step in RRT* implementation, that is rewiring. Once a new node is sampled, all the nearest nodes to it are rewired to reduce the cost of their paths. This step results in more refined path. However, for RRT* the terminating condition must be defined. usually it is the maximum number of sampling iterations.

For implementing informed RRT* , I created a random point generator inside a given ellipse as start and goal positions as foci, and used best path length and minimum path lengths to form minimum and maximum radii of the ellipse. An example of 1000 points generated inside an ellipse is shown by figure 15.

Figure 17 explains the parametric form that is used to generate a random point inside the ellipse .

Over the time, the radii of the ellipse shrink to form the most optimal path. Figure 18 shows the path generated by the informed RRT* algorithm. Figure 19 shows the path generated by informed RRT* algorithm.

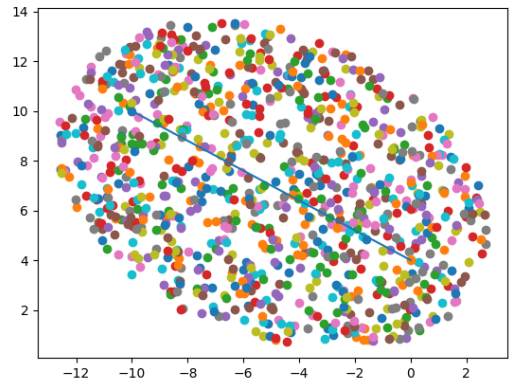

Fig. 17. An example of random point generator inside an ellipse

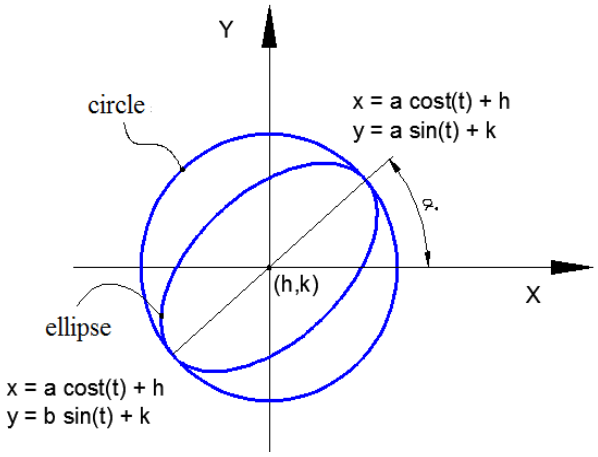

Fig. 18. Parametric form of an ellipse

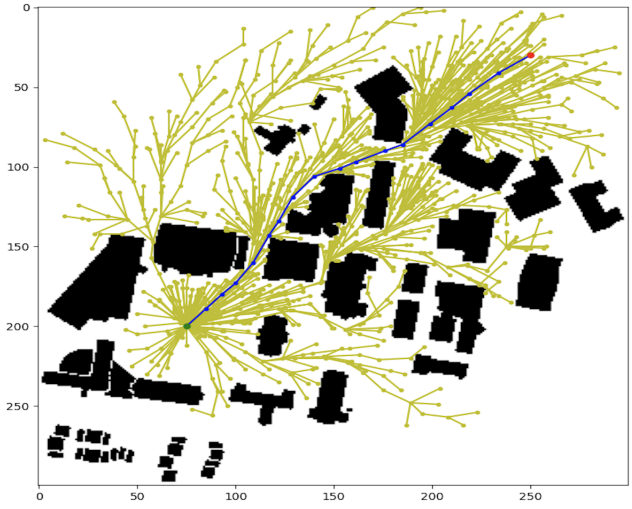

Fig. 19. Path generated by Informed RRT* algorithm

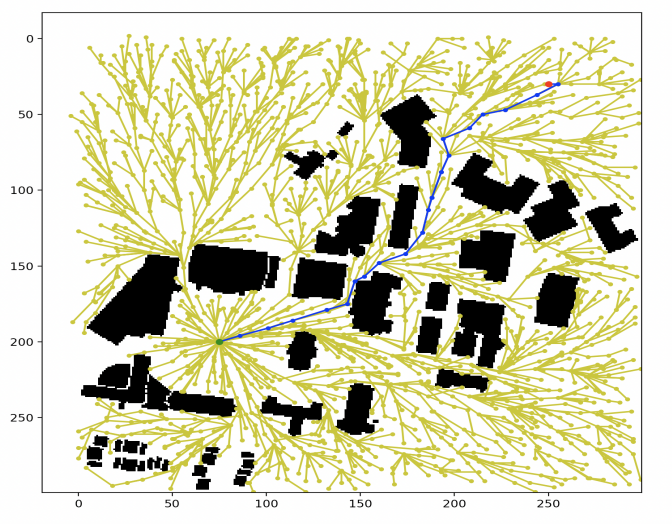

Fig. 20. Path generated by RRT* algorithm

Figure 20 shows the path generated in the same environment by RRT* algorithm . The RRT* algorithm resulted in a path with a length of 282.29 after 2000 nodes , but the informed RRT* results a path that is 248.93 , and with 2000 nodes.

A. *Informed RRT* vs RRT**

Informed RRT* results in probabilistic guarantee on completeness and optimality as RRT* while resulting the better convergence rate and final solution quality.Informed RRT* is a simple modification to RRT* .Informed RRT* outperforms RRT* in rate of convergence, final solution cost, and ability to find difficult passages while demonstrating less dependence on the state dimension and range of the planning problem. Informed RRT* focuses on sampling only in a local region where as RRT* tries to sample the whole work space. This may not be required as the initially generated path can simply be modified locally to get better solutions as in informed RRT. Therefore it is always preferred to use informed RRT* over RRT* algorithm.

V. Conclusions

In this paper, I explained the implementation of D* , results of it in a dynamic environment and advantages of it when compared to A* or Dijkstra algorithms. Overall is D* is a better algorithm compared to the other algorithms, and can result in better performance in dynamic environments.

I also explained the informed RRT* implementation process, and its differences compared to RRT* algorithm. With the results and theoretical explanation I proved that informed RRT* is a better algorithm and can result in more optimal and faster solutions.