Path Planning in Non-Stationary Environment using RRTs

Sai Chaitanya
Guide: Dr. Balaraman Ravindran

May 6, 2015

Path Planning

- Generic problem in many domains , Robotics, CAD, Computer Graphics, Computational Biology etc.

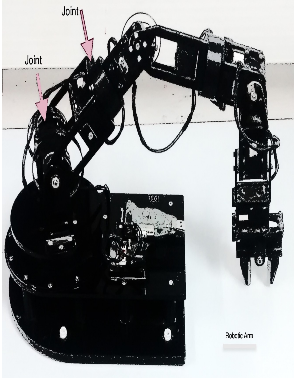


Figure : Left: screenshot of a game, courtesy: supercell games; Right: 6-D0F Arm

Cont.

Different types

- Known environment
- Unknown environment
- Static/Dynamic obstacles
- Changing environment

Robot entering a room: Continuous state/action spaces, unknown non-stationary environments.

Configuration space

- Description of the geometry of the moving agent or robot.
- Description of the geometry of the workspace.
- Description of robot's degrees of freedom.
- A start and goal configuration(can be set of configurations also) for the robot.

A solution is now a path connecting points in configuration space.

- Kino-dynamic constraints
- Non-holonomic robot

Approaches.

- Search based - Dijkstra, A*, D* ,etc.
  - Discretize the space into grid of fixed resolution and work on it.
- Potential field based methods
  - Potential field across the space and gradient descent.
  - Local minima. Artificial Randomized Potential Field method.
  - Complex environments, large number of obstacles , keep track of all obstacles
- Sampling based methods - PRM, RRT
  - Probabilistic Road Maps - multi query not suited for dynamic environment
  - Rapidly exploring random trees - single shot, space filling, probabilistically complete.

RRT

- Sample a point x_{random}
- Extend the tree towards the point.
  - Find out the "nearest" node to x_{random}
  - Move towards x_{random} by ϵ-distance

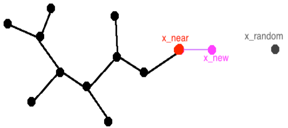


Figure : RRT extend

RRT Cont.

- "Rapidly" exploring into voronoi regions.
- If large area is unexplored probability of RRT extending into that region is higher than other areas.

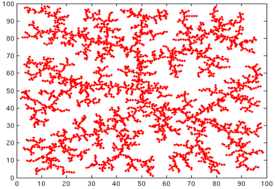


Figure : A RRT grown from start point [50;50]

Problems? Motivation!

- Takes long time to reach goal.
    - goal biasing, RRT-Connect
- Sub-optimal solutions
    - Heuristic Cost function , RRT*, LQR RRT* , RRT++
    - But difficult to determine in non-holonomic, high dimensional setups.
- Non-stationary environment
    - Replanning - E-RRT, D-RRT, MP-RRT.
    - Potential fields + RRTs. But again issues with potential fields.

Reinforcement Learning

- We want to learn the distance metric used in RRT procedure using experience.

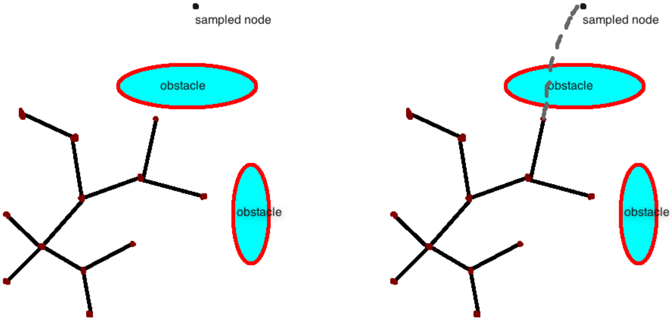


Figure : RRT: Choose the next point from the tree!

RL

- Markov Decision Process (MDP) from RL allows us to learn values of states which are sampled as a chain.
- MDP M is a tuple $< S, A, T, R, \gamma >$ where $S \in R^n$ is the n-dimensional state space, A is the action space.
- Transition function $T(s, a) = s'$. Also $\sum_{\forall s' \in S} T(s, a, s') = 1$
- $R(s, a, s')$ is the expectation of the real valued rewards for action $a \in A$
- For a trajectory $s_o, a_o, r_1, s_1, a_1.r_2, s_2, a_2, r_3, s_3 \ldots$

  Return $= \sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the reward at the time step t
- A policy π is a mapping from state space S to action space A $\pi : SxA \rightarrow [0, 1]$ and $\pi(s) = a$

Solving MDP

- State value function $J^{\pi}(s)$ is the expected return from any state s
- Optimal Policy: $\pi, \forall s \in S,\ J^{\pi}(s) < J^{\pi^*}(s)$,
- Bellman equation

$$J^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma J^{\pi}(s')]$$

- T and R are not available readily.

Solving MDP

- Sampling trajectories $\{s_o, a_o, r_1, s_1, a_1, r_2, s_2, ...s_M\}_{i=1}^{N}$
- TD(0) update rule

$$J^{\pi}(s_t) = (1-\alpha)J^{\pi}(s_t) + \alpha(r_t + \gamma J^{\pi}(s_{t+1}))$$

α step size and γ discount factor

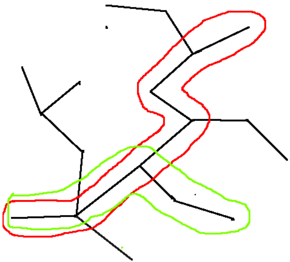


Figure : RRT: TD(0) on RRTs

DYNA

- In dynamic domains, agent should quickly repond to changes
- Integrate planning and execution
- Action taken is used to update J function as well as maintain a model and do simulated planning.

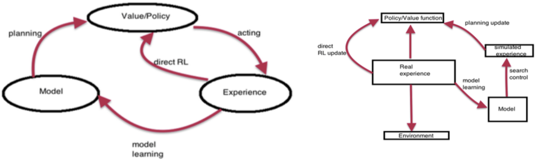


Figure : Relationship between Value function , Experience and Models and DYNA framework

DYNA-RRTPI

We present algorithms for Episodic, Online, Dynamic Replanning settings.

Algorithm *DYNA-RRTPI(N)*
(∗ Episodic offline version ∗)
1. G$\leftarrow$ Build a vanilla RRT using Euclidean distance metric
2. $J_o \leftarrow$ NN-TD(G) i.e use the vanilla RRT over the space to initialize the J values
3. n $\leftarrow$ 0
4. **while** n $<$ N
5. $\quad G_n \leftarrow$ Episode(S,$||.||_{J_{n-1}}$)
6. $\quad J_n \leftarrow$ NN-TD(G_n , J_{n-1})
7. $\quad$ n $\leftarrow$ n+1
8. End of Algorithm

Policy Improvement

Algorithm *Episode(S, $||.||_J$)*

(∗ Build a RRT using value functions in dyna framework ∗)

V(G) $\leftarrow x_{start}$; E(G) $\leftarrow \phi$
while goal is not reached
    $x_{rand} \leftarrow$ Sample(S);
    $x_{near} \leftarrow$ Modified-Nearest(x_{rand}, V(G) , $||.||_J$);
    $(x_{new}, a, r) \leftarrow$ Extend(x_{near}, x_{rand}, S);
    if Not-Colliding(x_{new}, x_{near},S)
        then Connect x_{new} to x_{near}
            V(G) $\leftarrow$ V(G) U x_{new}
            E(G) $\leftarrow$ E(G) U (x_{new}, a, r)
            $J(x_{near}) \leftarrow (1-\alpha)J(x_{near} + \alpha(r + \gamma J(x_{new})))$
    J $\leftarrow$ RRT-Planning(G, J, number of iteration)
return G
End of Algorithm

Policy Evaluation

We estimate the value function from the trajectories using TD-learning

Algorithm *NN-TD(G, J)*

(∗ Estimates the value function J from the trajectories ∗)

1. Let Y_n be the set of trajectories starting at $x_s tart$ till leaf nodes in G.
2. **for** each trajectory $(s_o, a_o, r_1, s_1, a_1, r_2, s_2, ...)$ in Y_n
3. **for** each pair $(s_i, a_i, r_{i+1}, s_{i+1})$
4. $J(s_i) = (1 - \alpha)J(s_i) + \alpha(r_i + \gamma J(s_{i+1})$
5. return J

6.End of Algorithm

Nearest Point

- The basic nearest procedure in does not work well.
- We modified the procedure to work with any intialization. We use euclidean distance to sort the points whose J values are close.

Algorithm *Modified-Nearest*(x_{rand}, $V(G)$, $||.||_J$)
(∗ Finds a point nearest to given point in the graph ∗)
1. $X_{near} \leftarrow$ subset of vertices in V(G) such that $x_{near} = arg \max_{x \in V(G)} (||x - x_{rand}||)$
2. return $x_{near} \in X_{near}$ that is closest to the goal when compared using euclidean distance
3. End of Algorithm

$||x - y||_J = J(x) - J(y)$ where x, y $\in$ S.

Extend

Algorithm *Extend*(x_{near}, x_{rand}, S)

(∗ Finds a point close to x_{rand} that is in $\epsilon-$neighbourhood of x_{near} ∗) respecting the kinodynamic constraints

1. $x_{new} \leftarrow arg \max_{a \in A} ||x - x_{rand}||_J$ and $||x_{new} - x_{near}||$ is within kinematic bounds and action a_{max} which generated x_{new} is within the acceleration bounds
2. $(r, x_{new}) \leftarrow$ MDP(x_{near},a)
3. return $(x_n ew, a, r)$
4. End of Algorithm

Planning

Algorithm *RRT-Planning(G, $||.||_J$, number of iterations)*
(∗ samples trajectories using RRTs ∗)
1. n $\leftarrow$ 0
2. **while** n $<$ number of iterations
3. Randomly sample a point x_{begin} from V(G)
4. Find a trajectory Y with x_{begin} as the root
5. **for** each pair $(s_i, a_i, r_{i+1}, s_{i+1}) \in Y$
6. $J(s_i) = (1-\alpha)J(s_i) + \alpha(r_i + \gamma J(s_{i+1})$
7. return J
8. End of Algorithm

Replanning in DYNA-RRTPI

Algorithm *Replan-DYNA-RRTPI( )*

1. F $\leftarrow$ initialize to empty set of forest of disconnected trees.
   while Graph G doesnt contain goal
2. **if** Graph is not empty
3. **then** Prune(G,F)
4. **else** Intialize V(G) $\leftarrow \{x_{start}\}$
5. $x_{rand} \leftarrow$ Modified-Sample(S,F);
6. $x_{near} \leftarrow$ Modified-Nearest(x_{rand}, V(G) , $||.||_J$);
7. $(x_{new}, a, r) \leftarrow$ Extend(x_{near}, x_{rand}, S);
8. **if** Not-Colliding(x_{new}, x_{near},S)
9. **then** Connect x_{new} to x_{near}
10. V(G) $\leftarrow$ V(G) U x_{new}
11. E(G) $\leftarrow$ E(G) U (x_{new}, a, r)
12. $J(x_{near}) \leftarrow (1-\alpha)J(x_{near} + \alpha(r + \gamma J(x_{new})))$
13. J $\leftarrow$ RRT-Planning(G, J, number of iteration)

Prune

Algorithm *Prune(Graph G, Forest F)*

(∗ Perfoms collision checks on all nodes and removes nodes ∗)

1. **for** each node q ∈ V(G) and F
2. **if** Not-valid (q)
3. **then** remove q from G and F
4. split the tree at q and add subtrees to F
5. **if** G is empty
6. **then** Initialize G and add x_{start} to V(G)
7. End of Algorithm

Sampling

Algorithm *Modified-Sample(S,F)*
(∗ Returns a point to which the RRT would try to expand to ∗)

1. prob ← Random(0,1)
2. **if** prob $< p_{goal}$
3. **then** $x_{result} \leftarrow x_{goal}$
4. **else** **if** prob $< p_{goal} + p_{forest}$
5. **then** $x_{result} \leftarrow$ root of a randomly sampled tree from F , the forest of disconnected trees.
6. **else** $x_{result} \leftarrow$ Randomly sample a point from the state space
7. return x_{result}
8. End of Algorithm

Continuous Domain

- We test our algorithm against RRTPI in continuous puddle world domain.
    - continuous space (0,0) to (50,50). Start at (5,5) and goal is to reach above (45,45). one puddle of radius 5 at (30,10)

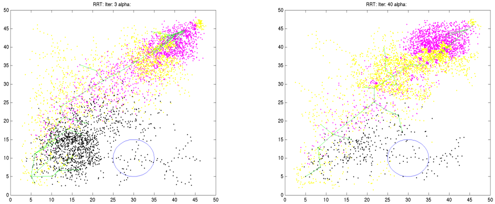


Figure : The value function distribution for the puddle world task. The green line is the path learnt in RRT grown in that episode. The left graph is for 3 episodes of DYNA-RRTPI and right graph for RRTPI.

Cont.

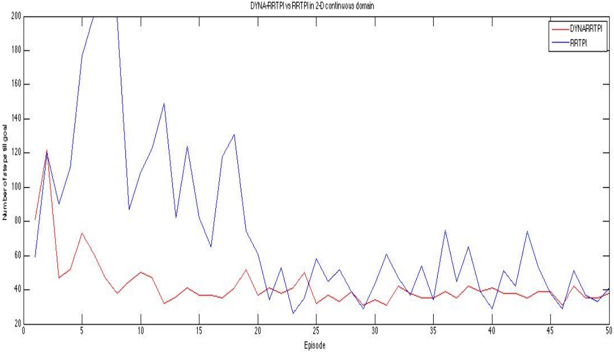


Figure : DYNARRTPI vs RRTPI in 2-D continuous puddle world from (0,0) to (50,50) with puddle of radius 5 at (30,10)

Dynamic Environment

- We increased the puddle radius after the algorithms have estimated the value functions.

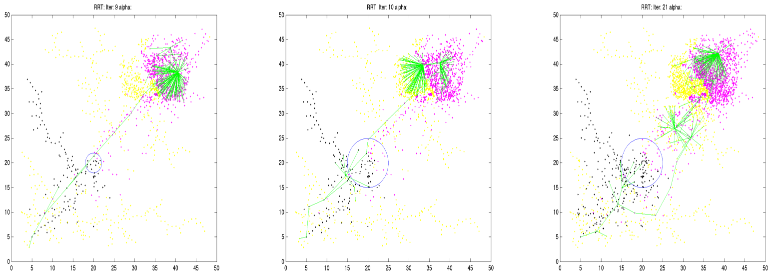


Figure : from left to right, RRTPI before puddle expansion, RRTPI in the episode after puddle expansion and RRTPI after 10 episodes

DYNA-RRTPI

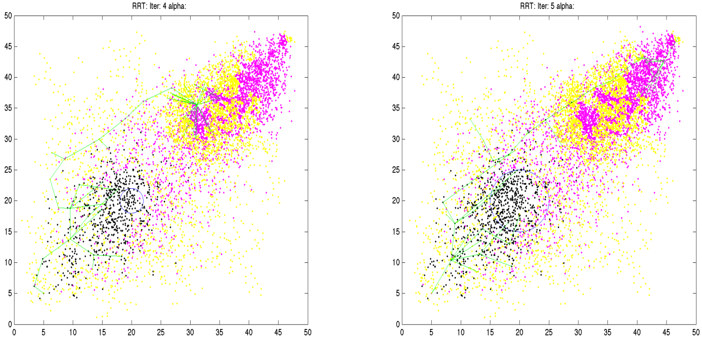


Figure : DYNA-RRTPI in successive episodes when puddle radius is increased.

Conclusion and Future work

- We present DYNA-RRTPI which gives near optimal paths combining good qualities of RRTs , RL, efficient replanning techniques.
- Responsive algorithm to dynamic changes
- Online algorithm - learn when its deployed
- Works in continuous space domains also and does not need explicit representation of enivironment and other obstacles
- Next, it can be tested on a real world robot navigation or a high dimensional problem like 6-DoF robotic arm.

Appendix

- Appendix

Online DYNA-RRTPI

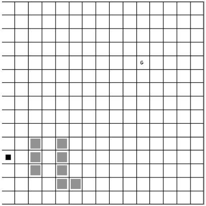
Algorithm *Online-DYNA-RRTPI( )*.

1. $G \leftarrow$ Build a vanilla RRT using Euclidean distance metric
2. $J_o \leftarrow$ NN-TD(G) i.e use the vanilla RRT over the space to initialize the J values
3. **while** goal is not reached
4. $\quad x_{rand} \leftarrow$ Sample(S);
5. $\quad x_{near} \leftarrow$ Nearest(x_{rand}, V(G) , $||.||_J$);
6. $\quad (x_{new}, a, r) \leftarrow$ Extend(x_{near}, x_{rand}, S);
7. $\quad$ **if** Not-Colliding(x_{new}, x_{near},S)
8. $\quad\quad$ **then** Connect x_{new} to x_{near}
9. $\quad\quad\quad$ V(G) $\leftarrow$ V(G) U x_{new}
10. $\quad\quad\quad$ E(G) $\leftarrow$ E(G) U (x_{new}, a, r)
11. $\quad\quad\quad$ $J(x_{near}) \leftarrow (1-\alpha)J(x_{near} + \alpha(r + \gamma J(x_{new})))$
12. $\quad$ J $\leftarrow$ RRT-Planning(G, J, number of iteration)
13. End of Algorithm

Discrete Domain Experiments

- Implemented on a discrete grid world domain. 15x15 grid, 100 episodes , maximum iterations 200 per episode and 50 per planning step. Both RRTPI and DYNA-RRTPI converge to the best solution.
- Increased the maximum iteration limit for RRTPI as it does not take time in planning step.

Continued



15x15 grid world

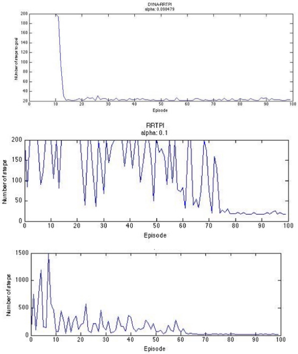


figure: Number of steps taken to reach goal vs Episode in DYNA-RRTPI and RRTPI